

사용자 가이드

Amazon EKS



Amazon EKS: 사용자 가이드

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 브랜드 디자인은 Amazon 외 제품 또는 서비스와 함께, Amazon 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

.....	xxi
.....	xxii
Amazon EKS란 무엇입니까?	1
Amazon EKS: 간소화된 Kubernetes 관리	1
Amazon EKS의 기능	2
관련 서비스	3
Amazon EKS 요금	4
일반 사용 사례	5
아키텍처	6
컨트롤 플레인	6
컴퓨팅	7
Kubernetes 개념	8
Kubernetes를 사용해야 하는 이유	9
클러스터	13
워크로드	17
다음 단계	22
배포 옵션	23
Amazon EKS 배포 옵션 이해	23
클라우드의 Amazon EKS	23
데이터 센터 또는 엣지 환경의 Amazon EKS	24
에어 갭 환경을 위한 Amazon EKS Anywhere	25
Amazon EKS 도구	25
설정	26
다음 단계	26
AWS CLI 설정	26
액세스 키 생성	27
AWS CLI를 구성하려면	27
보안 토큰을 받는 방법	28
사용자 ID를 확인하는 방법	28
다음 단계	29
kubectl 및 eksctl 설정	29
kubectl 설치 또는 업데이트	29
1단계: kubectl가 설치되어 있는지 확인	29
2단계: 설치 또는 kubectl 업데이트	30

eksctl 설치	43
다음 단계	43
빠른 시작	44
이 튜토리얼에서는	44
사전 조건	44
클러스터 구성	45
IngressClass 생성	45
2048 게임 샘플 애플리케이션 배포	46
Amazon EKS Auto Mode를 사용하여 데이터 유지	49
클러스터와 노드 삭제	51
Amazon EKS 알아보기	52
개요	52
Amazon EKS 워크샵	52
Amazon EKS 실습 클러스터 설정 자습서	54
Amazon EFS 샘플	55
AWS 자습서	55
개발자 워크숍	56
Terraform 워크숍	56
AWS Amazon EKS 교육	56
시작	57
클러스터 생성(EKS 자동 모드)	57
클러스터 생성(eksctl)	58
사전 조건	58
1단계: Amazon EKS 클러스터 및 노드 생성	59
2단계: Kubernetes 리소스 보기	60
3단계: 클러스터 및 노드 삭제	62
다음 단계	62
클러스터 생성(콘솔 및 CLI)	63
사전 조건	63
1단계: Amazon EKS 클러스터 생성	64
2단계: 클러스터와 통신하도록 컴퓨터 구성	66
3단계: 노드 생성	67
4단계: 리소스 보기	69
5단계: 리소스 삭제	70
다음 단계	70
EKS Auto Mode	72

Features	72
자동 구성 요소	73
구성	74
클러스터 생성	75
eksctl CLI	75
AWS CLI	77
관리 콘솔	85
기존 클러스터 활성화	87
마이그레이션 참조	88
EBS 볼륨 마이그레이션	89
로드 밸런서 마이그레이션	89
클러스터에서 활성화	90
Karpenter에서 마이그레이션	93
MNG에서 마이그레이션	97
Fargate에서 마이그레이션	98
워크로드 실행	103
워크로드 팅창 배포	104
로드 밸런서 배포	106
상태 저장 워크로드 배포	112
구성	117
노드 클래스 생성	119
노드 풀 생성	126
수신 클래스 생성	132
서비스 생성	138
StorageClass 생성	144
EKS Auto Mode 비활성화	152
Kubernetes 버전 업데이트	153
내장 노드 풀 검토	154
배포 제어	156
핵심 추가 기능 실행	158
네트워크 정책 사용	160
서브넷에 태그 지정	161
가속화된 워크로드 배포	163
CIS 보고서를 생성합니다.	170
루트 볼륨 암호화	173
작동 방법	176

관리형 인스턴스	177
자격 증명 및 액세스	180
네트워킹	186
문제 해결	189
노드 모니터링 에이전트	190
AWS EC2 CLI를 사용하여 EC2 관리형 인스턴스에서 콘솔 출력 가져오기	190
디버그 컨테이너와 kubectl CLI를 사용하여 노드 로그 가져오기	190
AWS 콘솔에서 EKS Auto Mode와 연결된 리소스 보기	192
AWS 계정에서 IAM 오류 보기	192
포드가 자율 모드 노드로 예약되지 않음 문제 해결	192
클러스터에 조인하지 않는 노드 문제 해결	193
포드 간 볼륨 공유	195
자동 모드에서 포함된 컨트롤러 문제 해결	196
릴리스 정보	196
2025년 6월 13일	196
2025년 4월 30일	196
2025년 4월 18일	196
2025년 4월 11일	197
2025년 4월 4일	197
2025년 3월 31일	197
2025년 3월 21일	197
2025년 3월 14일	197
클러스터	198
자동 클러스터 생성	199
사전 조건	44
클러스터 생성 - AWS 콘솔	200
클러스터 생성-AWS CLI	205
다음 단계	85
클러스터 생성	210
사전 조건	44
1단계: 클러스터 IAM 역할 만들기	211
2단계: 클러스터 생성	213
3단계: kubeconfig 업데이트	221
4단계: 클러스터 설정	222
다음 단계	85
클러스터 인사이트	223

클러스터 인사이트를 사용한 Kubernetes 버전 업그레이드 준비	224
클러스터 인사이트로 잘못된 구성 문제 해결	225
클러스터 인사이트 보기	225
Kubernetes 버전 업데이트	230
Amazon EKS Auto Mode 고려 사항	232
요약	232
1단계: 업그레이드 준비	232
2단계: 업그레이드 고려 사항 검토	233
3단계: 클러스터 제어 영역 업데이트	235
4단계: 클러스터 구성 요소 업데이트	237
Amazon EKS 클러스터에 대한 Kubernetes 버전 다운그레이드	238
클러스터 삭제	238
고려 사항	89
클러스터 삭제(eksctl)	239
클러스터 삭제(AWS 콘솔)	240
클러스터를 삭제(AWS CLI)	241
클러스터 엔드포인트 액세스	243
IPv6 클러스터 엔드포인트 형식	243
IPv4 클러스터 엔드포인트 형식	244
클러스터 프라이빗 엔드포인트	244
클러스터 엔드포인트 액세스 수정	245
프라이빗 전용 API 서버 액세스	247
엔드포인트 액세스 구성	248
Windows 지원 사용	251
고려 사항	89
사전 조건	44
Windows 지원 사용	253
Windows 포드 배포	255
Windows 노드에서 더 높은 포드 밀도 지원	255
Windows 지원 사용 중지	256
프라이빗 클러스터	256
Kubernetes 버전	260
추가 지원으로 사용 가능한 버전	260
확장 지원으로 사용 가능한 버전	261
Amazon EKS Kubernetes 릴리스 일정	261
AWS CLI를 사용하여 버전 정보 가져오기	262

Amazon EKS 버전 FAQ	263
Amazon EKS 추가 지원 FAQ	265
표준 지원 버전	267
추가 지원 버전	273
지원 기간 보기	277
업그레이드 정책 보기	277
추가 지원 활성화	279
추가 지원 비활성화	280
플랫폼 버전	281
Kubernetes 버전 1.33	282
Kubernetes 버전 1.32	283
Kubernetes 버전 1.31	285
Kubernetes 버전 1.30	287
Kubernetes 버전 1.29	290
Kubernetes 버전 1.28	295
Kubernetes 버전 1.27	299
Kubernetes 버전 1.26	305
현재 플랫폼 버전 가져오기	310
플랫폼 버전 변경	311
AutoScaling	311
EKS Auto Mode	311
추가 솔루션	311
구역별 교대에 대해 알아보기	312
포드 간 동서 네트워크 트래픽 흐름 이해하기	313
EKS 구역 교대 근무 요건	317
FAQ	325
추가 리소스	327
영역 전환 활성화	327
Amazon Application Recovery Controller란 무엇입니까?	328
영역 전환이란 무엇입니까?	328
영역 자동 전환이란 무엇입니까?	328
EKS는 자동 전환 중에 무엇을 합니까?	328
Amazon Application Recovery Controller(ARC)에 EKS 클러스터 등록(AWS 콘솔)	329
다음 단계	85
액세스 관리	330
일반적인 작업	330

배경	161
EKS Auto Mode 고려 사항	239
Kubernetes API 액세스	331
IAM 자격 증명을 Kubernetes 권한과 연결	332
클러스터 인증 모드 설정	333
권한 부여	334
aws-auth ConfigMap	372
OIDC 제공자 연결	382
OIDC 공급자 연결 해제	386
클러스터 리소스 액세스	387
필수 권한	388
kubectl을 사용하여 클러스터에 액세스	393
자동으로 kubeconfig 파일 생성	394
AWS에 대한 워크로드 액세스	395
서비스 계정 토큰	395
클러스터 추가 기능	397
Amazon Elastic Kubernetes Service 클러스터의 워크로드에 AWS ID 및 액세스 관리 권한 부 여	397
IRSA를 사용한 자격 증명	400
Pod Identity	423
컴퓨팅 관리	454
컴퓨팅 옵션 비교	454
관리형 노드 그룹	459
관리형 노드 그룹 개념	460
관리형 노드 그룹 용량	462
생성	465
업데이트	475
동작 세부 정보 업데이트	479
시작 템플릿	483
삭제	499
자체 관리형 노드	500
Amazon Linux	502
Bottlerocket	510
Windows	514
Ubuntu Linux	522
업데이트 방법	525

AWS Fargate	537
AWS Fargate 고려 사항	538
Fargate 비교 테이블	540
시작	542
프로필 정의	548
프로필 삭제	552
포드 구성 세부 정보	554
OS 패치 이벤트	556
지표 수집	559
로깅	561
Amazon EC2 인스턴스 유형	572
각 Amazon EC2 인스턴스 유형의 Amazon EKS 권장 최대 포드 수	574
EKS Auto Mode 고려 사항	239
사전 빌드된 최적화 AMI	575
AL2 AMI 사용 중단	576
Amazon Linux	580
Bottlerocket	589
Ubuntu Linux	595
Windows	595
노드 상태	679
노드 모니터링 에이전트	679
노드 자동 복구	679
노드 상태 문제	680
노드 상태 보기	690
노드 로그 가져오기	693
하이브리드 노드	695
Features	696
일반적인 고려 사항	697
사전 조건	697
하이브리드 노드 실행	758
구성	778
작동 방법	814
하이브리드 노드 nodeadm	855
문제 해결	869
앱 데이터 스토리지	889
Amazon EBS	889

고려 사항	889
사전 조건	890
1단계-IAM 역할 생성	890
2단계: Amazon EBS CSI 드라이버 받기	898
3단계: 샘플 애플리케이션 배포	898
Amazon EFS	898
고려 사항	899
사전 조건	899
1단계-IAM 역할 생성	900
2단계: Amazon EFS CSI 드라이버 받기	906
3단계: Amazon EFS 파일 시스템 생성	906
4단계: 샘플 애플리케이션 배포	907
Amazon FSx for Lustre	907
드라이버 배포	907
최적화(EFA)	914
최적화(비 EFA)	931
Amazon FSx for NetApp ONTAP	938
Amazon FSx for OpenZFS	939
Amazon File Cache	939
Mountpoint for Amazon S3	940
고려 사항	940
드라이버 배포	940
드라이버 제거	950
CSI 스냅샷 컨트롤러	952
네트워킹 구성	953
관리 콘솔에서 Amazon EKS 클러스터에 기존 VPC 서브넷 추가	953
VPC 및 서브넷 요구 사항	953
VPC 요구 사항 및 고려 사항	954
서브넷 요구 사항 및 고려 사항	955
공유 서브넷 요구 사항 및 고려 사항	961
VPC 생성	962
사전 조건	44
퍼블릭 및 프라이빗 서브넷	962
퍼블릭 서브넷만	965
프라이빗 서브넷만	966
보안 그룹 요구 사항	967

기본 클러스터 보안 그룹	968
클러스터 트래픽 제한	969
공유 보안 그룹	970
네트워킹 추가 기능 관리	971
내장 추가 기능	971
선택적 AWS 네트워킹 추가 기능	972
Amazon VPC CNI	972
대체 CNI 플러그인	1071
AWS 로드 밸런서 컨트롤러	1073
CoreDNS	1090
kube-proxy	1107
워크로드	1113
샘플 배포(Linux)	1113
사전 조건	44
네임스페이스 생성	1114
Kubernetes 배포를 생성합니다.	1114
서비스 생성	1115
생성된 리소스 검토	1116
포드에서 쉘 실행	1119
다음 단계	1120
샘플 배포(Windows)	1120
사전 조건	44
네임스페이스 생성	1114
Kubernetes 배포를 생성합니다.	1114
서비스 생성	1115
생성된 리소스 검토	1124
포드에서 쉘 실행	1119
다음 단계	1127
Vertical Pod Autoscaler	1128
Vertical Pod Autoscaler 배포	1128
Vertical Pod Autoscaler 설치 테스트	1129
Horizontal Pod Autoscaler	1133
Horizontal Pod Autoscaler 테스트 애플리케이션 실행	1134
네트워크 로드 밸런싱	1136
사전 조건	44
고려 사항	89

Network Load Balancer 생성	1140
(선택사항) 샘플 애플리케이션을 배포합니다.	1142
애플리케이션 로드 밸런싱	1145
사전 조건	44
수신 그룹에서 ALBs 재사용	1148
(선택사항) 샘플 애플리케이션을 배포합니다.	1150
서비스 외부 IP 제한	1152
리포지토리에 이미지 복사	1155
Amazon 이미지 레지스트리 보기	1158
Amazon EKS 추가 기능	1161
고려 사항	1162
Amazon EKS Auto Mode 고려 사항	1163
지원	1164
AWS 추가 기능	1165
커뮤니티 추가 기능	1180
Marketplace 추가 기능	1185
추가 기능 생성	1201
추가 기능 업데이트	1212
호환성 확인	1220
추가 기능 제거	1221
IAM 역할	1225
사용자 지정할 수 있는 필드	1231
컨테이너 이미지 확인	1235
클러스터 관리	1236
비용 모니터링	1236
포드별 비용 보기	1237
Kubecost 설치	1238
Kubecost 대시보드 액세스	1239
Kubecost에 대해 자세히 알아보기	1240
지표 서버	1248
고려 사항	89
Amazon EKS 추가 기능을 사용하여 커뮤니티 추가 기능으로 배포	1248
매니페스트를 사용하여 배포	1249
Helm을 사용하여 앱 배포	1250
리소스에 태깅	1252
태그 기본 사항	1252

리소스에 태깅	1253
태그 제한	1253
리소스에 결제용 태깅	1254
콘솔을 사용한 태그 작업	1255
CLI, API 또는 eksctl을 사용한 태그 작업	1256
Service quotas	1257
AWS Management Console를 사용하여 EKS Service Quotas 보기	1257
AWS CLI를 사용하여 EKS Service Quotas 보기	1258
Amazon EKS 서비스 할당량	1259
AWS Fargate 서비스 할당량	1259
보안	1261
모범 사례	1262
취약성 분석	1262
Amazon EKS에 대한 Center for Internet Security(CIS) 벤치마크	1262
Amazon EKS 플랫폼 버전	1263
운영 체제 취약성 목록	1263
Amazon Inspector로 노드 탐지	1264
Amazon GuardDuty로 클러스터 및 노드 탐지	1264
규정 준수 검증	1264
EKS에 대한 고려 사항	1265
인프라 보안	1265
복원성	1269
Kubernetes 고려 사항	1270
인증서 서명	1270
기본 역할 및 사용자	1273
비밀 암호화 활성화	1277
AWS Secrets Manager	1282
모든 Kubernetes API 데이터에 대한 기본 봉투 암호화	1282
EKS Auto 고려 사항	1289
API 보안 및 인증	1290
네트워크 보안	1290
EC2 관리형 인스턴스 보안	1290
리소스 관리 자동화	1292
보안 모범 사례	1292
IAM 참조	1293
대상	1293

ID를 통한 인증	1293
정책을 사용한 액세스 관리	1296
Amazon EKS 및 IAM	1298
ID 기반 정책	1302
서비스 연결 역할	1309
포드 실행 IAM 역할	1324
커넥터 IAM 역할	1329
AWS 관리형 정책	1333
문제 해결	1355
클러스터 IAM 역할	1358
노드 IAM 역할	1361
자율 모드 클러스터 IAM 역할	1367
자율 모드 노드 IAM 역할	1370
클러스터 모니터링	1374
Amazon EKS의 모니터링 및 로깅	1374
Amazon EKS 모니터링 및 로깅 도구	1375
관찰성 대시보드	1379
요약	1379
클러스터 상태	1379
컨트롤 플레인 모니터링	1379
클러스터 인사이트	1381
노드 상태 문제	1381
Prometheus 지표	1382
1단계: Prometheus 지표 켜기	1382
2단계: Prometheus 지표 사용	1384
3단계: Prometheus 스크레이퍼 관리	1384
Helm을 사용하여 배포	1385
컨트롤 플레인	1388
Amazon CloudWatch	1395
Amazon CloudWatch의 기본 지표	1396
Amazon CloudWatch Observability Operator	1403
컨트롤 플레인 로그	1404
컨트롤 플레인 로그 활성화 또는 비활성화	1405
클러스터 컨트롤 플레인 로그 보기	1407
AWS CloudTrail	1408
참조	1409

로그 파일 항목	1410
Auto Scaling 그룹 지표	1413
ADOT Operator	1413
Amazon EKS 대시보드	1413
Amazon EKS 대시보드란 무엇인가요?	1414
대시보드에서 AWS Organizations를 어떻게 사용하나요?	1414
AWS 콘솔을 사용하여 EKS 대시보드 활성화	1415
EKS 대시보드 보기	1416
대시보드 구성	1416
AWS 콘솔을 사용하여 EKS 대시보드 비활성화	1418
EKS 대시보드 문제 해결	1418
Organizations 구성	1419
다른 서비스와 함께 사용	1425
AWS CloudFormation	1425
Amazon EKS 및 AWS CloudFormation 템플릿	1425
AWS CloudFormation에 대해 자세히 알아보기	1426
Amazon Detective	1426
Amazon EKS와 함께 Amazon Detective 사용	1427
Amazon GuardDuty	1427
AWS Resilience Hub	1428
Amazon Security Lake	1429
Amazon EKS에서 Security Lake를 사용하여 얻을 수 있는 이점	1429
Amazon EKS에서 Security Lake 활성화	1429
Security Lake에서 EKS 로그 분석	1430
Amazon VPC Lattice	1430
AWS 로컬 영역	1430
문제 해결	1432
용량 부족	1432
노드가 클러스터 조인에 실패	1432
권한이 없거나 액세스가 거부됨(kubect1)	1434
hostname doesn't match	1435
getsockopt: no route to host	1435
Instances failed to join the Kubernetes cluster	1435
관리형 노드 그룹 오류	1435
Not authorized for images	1440
노드가 NotReady 상태임	1440

EKS 로그 수집기	1441
컨테이너 런타임 네트워크가 준비되지 않음	1442
TLS 핸드셰이크 시간 초과	1443
InvalidClientTokenId	1444
컨트롤 플레인을 업데이트하기 전에 노드 그룹이 Kubernetes 버전과 일치해야 합니다.	1444
많은 노드를 시작할 때 Too Many Requests 오류가 있습니다.	1445
Kubernetes API 서버 요청에 대한 HTTP 401 권한 없음 오류 응답	1445
Amazon EKS 플랫폼 버전이 현재 플랫폼 버전보다 두 버전 이상 뒤떨어져 있음	1446
클러스터 상태 FAQ 및 오류 코드(해결 경로 포함)	1448
Amazon EKS Connector	1454
Amazon EKS Connector 고려 사항	1454
Amazon EKS Connector에 필요한 IAM 역할	1455
클러스터 연결	1455
고려 사항	1455
사전 조건	1456
1단계: 클러스터 등록	1456
2단계: eks-connector 에이전트 설치	1459
다음 단계	1461
클러스터에 액세스 권한 부여	1461
사전 조건	1461
클러스터 등록 취소	1463
Kubernetes 클러스터 등록 취소	1463
Kubernetes 클러스터의 리소스 정리	1464
EKS Connector 문제 해결	1465
기본 문제 해결	1465
Helm 문제: 403 Forbidden	1466
콘솔 오류: 클러스터가 보류 중 상태에서 멈춰 있음	1467
콘솔 오류: 사용자 system:serviceaccount:eks-connector:eks-connector가 클러스터 범위에서 API 그룹의 리소스 사용자를 가장할 수 없습니다.	1467
콘솔 오류: [...] 금지됨: 사용자 [...]이(가) 클러스터 범위에서 API 그룹의 [...] 리소스를 나열할 수 없음	1468
콘솔 오류: Amazon EKS가 Kubernetes 클러스터 API 서버와 통신할 수 없습니다. 연결에 성 공하려면 클러스터가 ACTIVE 상태여야 합니다. 몇 분 후에 다시 시도하세요.	1468
Amazon EKS 커넥터 포드가 크래시 반복됨	1469
eks-connector 시작 실패: InvalidActivation	1469
클러스터 노드에 아웃바운드 연결이 없습니다.	1470

Amazon EKS 커넥터 포드의 상태가 ImagePullBackOff임	1470
자주 묻는 질문(FAQ)	1471
보안 고려 사항	1472
AWS 책임	1472
고객 책임	1472
AWSOutposts의 Amazon EKS	1474
각 배포 옵션을 사용해야 하는 경우	1474
배포 옵션 비교	1475
로컬 클러스터 실행	1477
지원되는 AWS 리전	1478
로컬 클러스터 배포	1478
EKS 플랫폼 버전	1488
VPC 및 서브넷 생성	1502
연결 해제 준비	1505
용량 고려 사항	1510
클러스터 문제 해결	1512
노드	1522
eksctl	1523
AWS Management Console	1525
EKS에서의 기계 학습	1531
AI/ML용 EKS를 선택해야 하는 이유	1531
주요 사용 사례	1531
사례 연구	1532
EKS에서 기계 학습 사용 시작	1533
ML 시작하기	1533
EKS에서 ML 계속	1534
ML 준비	1535
Linux GPU AMI 실행	1536
Windows GPU AMI 실행	1538
MNG용 GPU 예약	1543
SMN용 GPU 예약	1546
테인트 GPU 노드	1549
EFA를 사용하여 클러스터 훈련	1551
Inferentia 클러스터 준비	1562
EKS에서 ML 사용을 위한 자습서 사용	1568
EKS에서 생성형 AI 플랫폼 구축	1568

EKS에서 특수 생성형 AI 프레임워크 실행	1568
EKS에서 ML용 NVIDIA GPU 성능 극대화	1568
EKS에서 비디오 인코딩 워크로드 실행	1569
추론 워크로드의 이미지 로드 가속화	1569
ML 워크로드 모니터링	1569
EKS에서 ML 사용 관련 공지	1569
Amazon EKS와 관련된 프로젝트	1570
EKS에 배포된 소프트웨어 지원	1570
관리 도구	1571
eksctl	1571
Kubernetes용 AWS 컨트롤러	1571
Flux CD	1571
Kubernetes용 CDK	1572
네트워킹	1572
Kubernetes용 Amazon VPC CNI 플러그 인	1572
Kubernetes용 AWS 로드 밸런서 컨트롤러	1572
ExternalDNS	1572
기계 학습	1573
Kubeflow	1573
오토 스케일링	1573
Cluster autoscaler	1573
Karpenter	1573
Escalator	1574
모니터링	1574
Prometheus	1574
연속 통합/연속 배포	1574
Jenkins X	1574
새로운 기능 및 로드맵	1575
문서 기록	1576
기여	1623
단일 페이지 편집	1623
사전 조건	44
절차	92
풀 요청 개요	1625
GitHub에서 파일 편집	1626
사전 조건	44

절차	92
스타일 피드백 보기	1627
Vale 설치	1628
VS Code Vale 확장 기능 설치	1628
값 동기화	1629
VS Code에서 스타일 피드백 보기	1629
페이지 생성	1629
페이지 생성	1629
탐색에 페이지 추가	1630
링크 삽입	1631
EKS 사용 설명서에서 페이지 또는 섹션에 대한 링크	1631
AWS 설명서에서 다른 페이지에 대한 링크	1631
인터넷에 대한 외부 링크	1631
Amazon Q를 사용하여 생성	1632
VS Code를 사용하여 Amazon Q 설치	1632
Amazon Q에 로그인	1633
Amazon Q를 사용하여 콘텐츠 생성	1633
팁	1633
PR 미리 보기 확인	1634
미리 보기 확인	1634
미리 보기 제한 사항	1635
AsciiDoc 구문	1635
기본 형식 지정	1635
헤더	1636
Lists	1636
Links	1636
이미지	1636
코드 블록	1637
표	1637
주의 문구	1637
포함 내용	1638

이 페이지 개선에 도움 주기

이 사용자 가이드에 기여하려면 모든 페이지의 오른쪽 창에 있는 GitHub에서 이 페이지 편집 링크를 선택합니다.

Amazon EKS란 무엇입니까?

Amazon EKS: 간소화된 Kubernetes 관리

Amazon Elastic Kubernetes Service(EKS)는 Kubernetes 클러스터 운영의 복잡성을 제거하는 완전관리형 Kubernetes 서비스를 제공합니다. EKS를 사용하면 다음과 같은 이점을 얻을 수 있습니다.

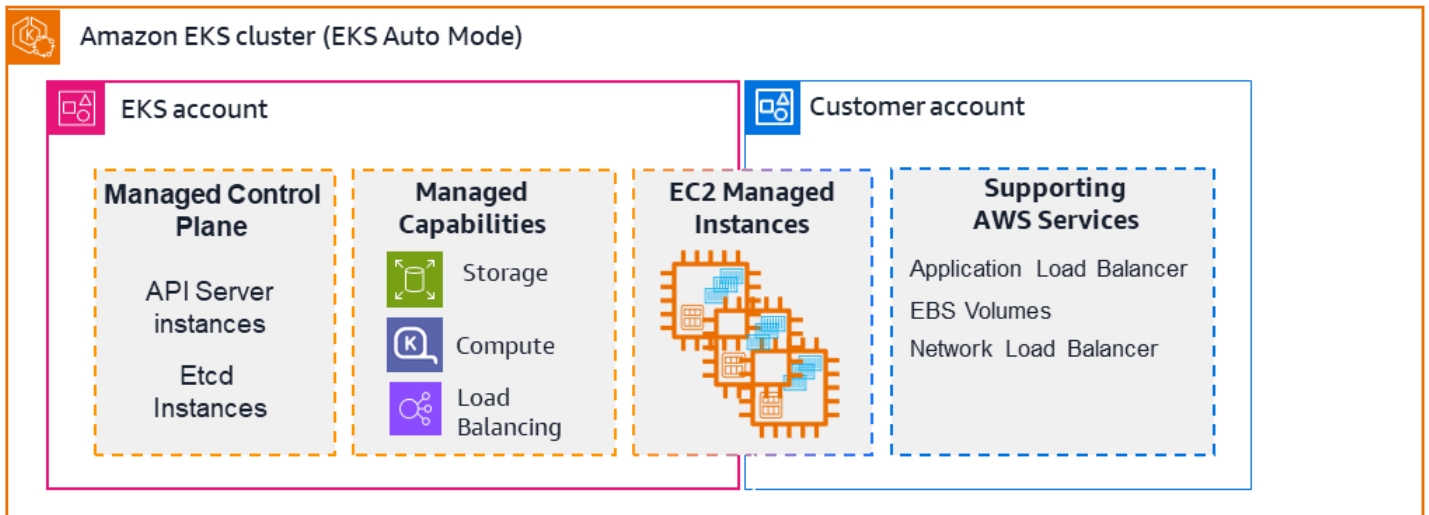
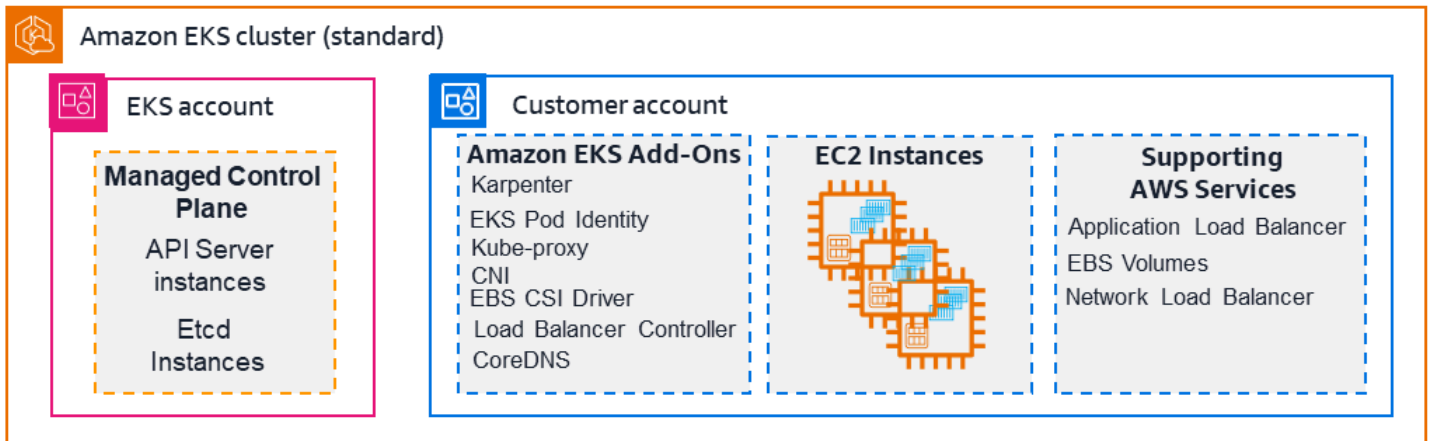
- 운영 오버헤드를 줄이면서 더 빠르게 애플리케이션 배포
- 변화하는 워크로드 수요에 맞게 원활하게 규모 조정
- AWS 통합 및 자동 업데이트를 통해 보안 개선
- 표준 EKS 또는 완전 자동화된 EKS Auto Mode 중에서 선택

Amazon Elastic Kubernetes Service(Amazon EKS)는 Amazon Web Services(AWS) 클라우드와 자체 데이터 센터([EKS Anywhere](#) 및 [Amazon EKS Hybrid Nodes](#)) 모두에서 [Kubernetes](#) 클러스터를 실행하는 데 있어 최적의 플랫폼입니다.

Amazon EKS는 Kubernetes 클러스터의 구축, 보안 및 유지 관리를 간소화합니다. 자체 데이터 센터를 유지 관리하는 것보다 피크 수요를 충족할 수 있도록 충분한 리소스를 제공하는 편이 비용 효율적일 수 있습니다. Amazon EKS 사용과 관련된 두 가지 주요 접근 방식은 다음과 같습니다.

- EKS 표준: EKS로 클러스터를 생성할 때 AWS가 [Kubernetes 컨트롤 플레인](#)을 관리합니다. 노드를 관리하고, 워크로드를 예약하고, AWS 클라우드와 통합하고, 컨트롤 플레인 정보를 저장하고 확장하여 클러스터의 가동을 유지하도록 하는 구성 요소가 자동으로 처리됩니다.
- EKS Auto Mode: [EKS Auto Mode](#) 기능을 사용하면 EKS는 [노드](#)(Kubernetes 데이터 플레인)도 관리하도록 제어를 확장합니다. 인프라를 자동으로 프로비저닝하고, 최적의 컴퓨팅 인스턴스를 선택하고, 리소스를 동적으로 확장하고, 비용을 지속적으로 최적화하고, 운영 체제를 패치하고, AWS 보안 서비스와 통합하여 Kubernetes 관리를 간소화합니다.

다음 다이어그램은 선택한 클러스터 생성 방법에 따라 Amazon EKS가 Kubernetes 클러스터를 AWS 클라우드와 통합하는 방법을 보여줍니다.



Amazon EKS를 사용하면 프로덕션 시간을 단축하고, 성능, 가용성 및 복원력을 개선하고, 시스템 보안을 강화할 수 있습니다. 자세한 내용은 [Amazon Elastic Kubernetes Service](#)를 참조하세요.

Amazon EKS의 기능

Amazon EKS는 다음의 대략적인 기능을 제공합니다.

관리 인터페이스

EKS는 AWS Management Console, Amazon EKS API/SDK, CDK, AWS CLI, eksctl CLI, AWS CloudFormation, Terraform을 포함한 클러스터를 프로비저닝, 관리 및 유지하기 위한 다양한 인터페이스를 제공합니다. 자세한 내용은 [시작](#) 및 [클러스터\(을\)](#)를 참조하세요.

액세스 제어 도구

EKS는 Kubernetes 및 AWS Identity and Access Management(AWS IAM) 기능을 모두 사용하여 사용자 및 워크로드의 [액세스를 관리](#)합니다. 자세한 내용은 [the section called “Kubernetes API 액세스”](#) 및 [the section called “AWS에 대한 워크로드 액세스”](#)(을)를 참조하세요.

컴퓨팅 리소스

[컴퓨팅 리소스](#)의 경우 Amazon EKS를 통해 모든 범위의 Amazon EC2 인스턴스 유형과 Nitro 및 Graviton과 같은 AWS 혁신 기술을 활용하여 워크로드에 맞게 컴퓨팅을 최적화할 수 있습니다. 자세한 내용은 [컴퓨팅 관리](#) 섹션을 참조하세요.

스토리지

EKS Auto Mode는 [EBS 볼륨](#)을 사용하여 스토리지 클래스를 자동으로 생성합니다. 컨테이너 스토리지 인터페이스(CSI) 드라이버를 사용하면 애플리케이션 스토리지 요구 사항에 맞게 Amazon S3, Amazon EFS, Amazon FSX 및 Amazon File Cache를 사용할 수도 있습니다. 자세한 내용은 [앱 데이터 스토리지](#) 섹션을 참조하세요.

보안

[Amazon EKS의 보안](#)과 관련이 있는 공동 책임 모델이 배포됩니다. 자세한 내용은 [보안 모범 사례](#), [인프라 보안](#) 및 [Kubernetes 보안](#)을 참조하세요.

모니터링 도구

[관찰성 대시보드](#)를 사용하여 Amazon EKS 클러스터를 모니터링합니다. 모니터링 도구로는 [Prometheus](#), [CloudWatch](#), [Cloudtrail](#) 및 [ADOT Operator](#)가 있습니다. 대시보드, 지표 서버 및 기타 도구에 대한 자세한 내용은 [EKS 클러스터 비용](#) 및 [Kubernetes 지표 서버](#)를 참조하세요.

Kubernetes 호환성 및 지원

Amazon EKS는 Kubernetes 호환 인증을 받았고, 그에 따라 리팩터링 없이 Kubernetes 호환 애플리케이션을 배포하고 Kubernetes 커뮤니티 도구 및 플러그인을 사용할 수 있습니다. EKS는 Kubernetes에 [표준 지원](#)과 [확장 지원](#)을 모두 제공합니다. 자세한 내용은 [the section called “Kubernetes 버전”](#) 섹션을 참조하세요.

관련 서비스

Amazon EKS와 함께 사용할 서비스

Amazon EKS를 사용하여 배포하는 클러스터에서 다른 AWS 서비스를 사용할 수 있습니다.

Amazon EC2

[Amazon EC2](#)를 사용하여 확장 가능한 온디맨드 컴퓨팅 용량을 확보합니다.

Amazon EBS

[Amazon EBS](#)를 사용하여 확장 가능한 고성능 블록 스토리지 리소스를 연결합니다.

Amazon ECR

[Amazon ECR](#)을 사용하여 컨테이너 이미지를 안전하게 저장합니다.

Amazon CloudWatch

[Amazon CloudWatch](#)를 사용하여 AWS 리소스와 애플리케이션을 실시간으로 모니터링합니다.

Amazon Prometheus

[Amazon Managed Service for Prometheus](#)를 사용하여 컨테이너화된 애플리케이션에 대한 지표를 추적합니다.

Elastic Load Balancing

[Elastic Load Balancing](#)을 사용하여 수신 트래픽을 여러 대상에 분산합니다.

Amazon GuardDuty

[Amazon GuardDuty](#)로 EKS 클러스터에 대한 위협을 탐지합니다.

AWS Resilience Hub

[AWS Resilience Hub](#)를 사용하여 EKS 클러스터 복원력을 평가합니다.

Amazon EKS 요금

Amazon EKS는 Kubernetes 클러스터 버전 지원, Amazon EKS Auto Mode 요금, Amazon EKS Hybrid Nodes의 vCPU 요금에 따라 클러스터당 요금을 책정합니다.

Amazon EKS를 사용하는 경우 Kubernetes 워커 노드에서 애플리케이션을 실행하는 데 사용하는 AWS 리소스에 대해 별도로 요금을 지불합니다. 예를 들어 Amazon EBS 볼륨 및 퍼블릭 IPv4 주소를 사용하여 Amazon EC2 인스턴스로 Kubernetes 워커 노드를 실행하는 경우 Amazon EC2를 통한 인스턴스 용량, Amazon EBS를 통한 볼륨 용량, Amazon VPC를 통한 IPv4 주소에 대한 요금이 부과됩니다.

자세한 요금 정보는 Kubernetes 애플리케이션에서 사용 중인 AWS 서비스의 각 요금 페이지를 참조하세요.

- Amazon EKS 클러스터, Amazon EKS Auto Mode, Amazon EKS Hybrid Nodes 요금은 [Amazon EKS 요금](#)을 참조하세요.
- Amazon EC2 요금은 [Amazon EC2 온디맨드 요금](#) 및 [Amazon EC2 스팟 요금](#)을 참조하세요.
- AWS Fargate 요금은 [AWS Fargate Pricing](#)을 참조하세요.
- Amazon EKS 클러스터에 사용되는 컴퓨팅에 절감형 플랜을 사용할 수 있습니다. 자세한 내용은 [Savings Plans 요금](#)을 참조하세요.

Amazon EKS의 일반 사용 사례

Amazon EKS는 컨테이너화된 애플리케이션을 최적화하도록 설계된 AWS에 강력한 관리형 Kubernetes 서비스를 제공합니다. 다음은 Amazon EKS의 가장 일반적인 몇 가지 사용 사례로, 특정 요구 사항에 맞게 제품의 강점을 활용하는 데 도움이 됩니다.

고가용성 애플리케이션 배포

[Elastic Load Balancing](#)을 활용하여 애플리케이션이 여러 [가용 영역](#)에서 높은 가용성으로 제공되도록 할 수 있습니다.

마이크로서비스 아키텍처 구축

[AWS Cloud Map](#) 또는 [Amazon VPC Lattice](#)에 Kubernetes 서비스 검색 기능을 사용하여 복원력이 뛰어난 시스템을 구축합니다.

소프트웨어 릴리스 프로세스 자동화

애플리케이션의 자동화된 구축, 테스트, 배포 프로세스를 간소화하는 지속적인 통합 및 지속적인 배포(CICD) 파이프라인을 관리합니다.

서버리스 애플리케이션 실행

Amazon EKS에 [AWS Fargate](#)를 사용하여 서버리스 애플리케이션을 실행합니다. 즉, Amazon EKS와 Fargate가 기본 인프라를 처리하는 동안 애플리케이션 개발에만 집중할 수 있습니다.

기계 학습 워크로드 실행

Amazon EKS는 [TensorFlow](#), [MXNet](#) 및 [PyTorch](#) 등과 같은 인기 기계 학습 프레임워크와 호환됩니다. GPU 지원을 통해 복잡한 기계 학습 작업도 효과적으로 처리할 수 있습니다.

온프레미스와 클라우드에서 일관적인 배포

온프레미스 환경에서 Kubernetes 실행을 간소화하기 위해 동일한 Amazon EKS 클러스터, 기능 및 도구를 사용하여 [AWS Outposts](#)에서 자체 관리형 노드를 실행하거나 자체 인프라에서 [Amazon](#)

[EKS Hybrid Nodes](#)를 사용할 수 있습니다. 독립형 에어 갭 환경의 경우 [Amazon EKS Anywhere](#)를 사용하여 자체 인프라에서 Kubernetes 클러스터 수명 주기 관리를 자동화할 수 있습니다.

비용 효율적인 배치 처리 및 빅 데이터 워크로드 실행

[스팟 인스턴스](#)를 활용하여 적은 비용으로 배치 처리 및 [Apache Hadoop](#), [Spark](#)와 같은 빅 데이터 워크로드를 실행할 수 있습니다. 이를 통해 미사용 Amazon EC2 용량을 할인된 가격으로 활용할 수 있습니다.

애플리케이션 보안 및 규정 준수 보장

[AWS ID 및 액세스 관리](#)(IAM), [Amazon 가상 프라이빗 클라우드](#)(Amazon VPC), [AWS 키 관리 서비스](#)(AWS KMS)와 같은 AWS 보안 서비스와 통합되는 Amazon EKS로 강력한 보안 관행을 구현하고 규정 준수를 유지하세요. 이를 통해 업계 표준에 따른 데이터 프라이버시 및 보호가 보장됩니다.

Amazon EKS 아키텍처

Amazon EKS는 Kubernetes의 일반 클러스터 아키텍처와 일치합니다. 자세한 내용은 쿠버네티스 문서의 [쿠버네티스 컴포넌트](#)를 참조하세요. 다음 섹션에는 Amazon EKS의 몇 가지 추가 아키텍처 세부 정보가 요약되어 있습니다.

컨트롤 플레인

Amazon EKS는 모든 클러스터가 고유한 Kubernetes 컨트롤 플레인을 보유하도록 보장합니다. 이 설계는 각 클러스터의 인프라를 분리하여 클러스터 또는 AWS 계정 간에 중복이 발생하지 않도록 합니다. 설정에는 다음이 포함됩니다.

분산 구성 요소

컨트롤 플레인은 AWS 리전 내 AWS 가용성 영역 3개에 걸쳐 최소 2개의 API 서버 인스턴스와 3개의 [etcd](#) 인스턴스를 배치합니다.

최적의 성능

Amazon EKS는 컨트롤 플레인 인스턴스를 능동적으로 모니터링 및 조정하여 최고의 성능을 유지합니다.

복원성

컨트롤 플레인 인스턴스가 불안정해지면 Amazon EKS는 필요한 경우 다른 가용 영역을 사용하여 신속하게 이를 대체합니다.

일반적인 가동 시간

여러 가용 영역에서 클러스터를 실행함으로써 안정적인 [API 서버 엔드포인트 가용성 서비스 수준에 관한 계약\(SLA\)](#)을 달성합니다.

Amazon EKS는 Amazon Virtual Private Cloud(Amazon VPC)를 사용하여 단일 클러스터 내의 컨트롤 플레인 구성 요소 간 트래픽을 제한합니다. 클러스터 구성 요소는 Kubernetes 역할 기반 액세스 제어(RBAC) 정책에 따라 권한을 부여받지 않은 경우를 제외하면 다른 클러스터 또는 AWS 계정의 통신을 보거나 수신할 수 없습니다.

컴퓨팅

컨트롤 플레인 외에 Amazon EKS 클러스터에는 노드라고 하는 작업자 컴퓨터 세트가 있습니다. 특정 요구 사항을 충족하고 리소스 사용률을 최적화하려면 적절한 Amazon EKS 클러스터 노드 유형을 선택하는 것이 중요합니다. Amazon EKS는 다음 프라이머리 노드 유형을 제공합니다.

EKS Auto Mode

[EKS Auto Mode](#)는 컨트롤 플레인을 넘어 데이터 영역을 포함하도록 AWS 관리를 확장하여 클러스터 인프라 관리를 자동화합니다. 컴퓨팅 오토 스케일링, 네트워킹, 로드 밸런싱, DNS, 스토리지, GPU 지원을 포함하여 핵심 Kubernetes 기능을 기본 구성 요소로 통합합니다. EKS Auto Mode는 향상된 보안 기능과 함께 변경할 수 없는 AMI를 사용하여 워크로드 수요에 따라 노드를 동적으로 관리합니다. 포드 중단 예산을 준수하면서 업데이트 및 업그레이드를 자동화하며, 추가 관리가 필요한 관리형 구성 요소를 포함합니다. 이 옵션은 일상적인 운영에 대한 AWS의 전문 지식을 활용하고, 운영 오버헤드를 최소화하고, 인프라 관리가 아닌 애플리케이션 개발에 집중하려는 사용자에게 적합합니다.

AWS Fargate

[Fargate](#)는 기본 인스턴스를 관리할 필요가 없는 컨테이너용 서버리스 컴퓨팅 엔진입니다. Fargate를 사용하면 애플리케이션의 리소스 요구 사항을 지정하고, AWS에서 인프라를 자동으로 프로비저닝, 규모 조정 및 유지 관리합니다. 이 옵션은 사용 편의성을 우선시하고 인프라 관리보다는 애플리케이션 개발 및 배포에 집중하고자 하는 사용자에게 적합합니다.

Karpenter

[Karpenter](#)는 애플리케이션 가용성과 클러스터 효율성을 개선하는 데 도움이 되는 유연한 고성능 Kubernetes Cluster Autoscaler입니다. Karpenter는 애플리케이션 로드 변화에 대응하여 적절한 크기의 컴퓨팅 리소스를 시작합니다. 이 옵션은 워크로드의 요구 사항을 충족하는 적시 컴퓨팅 리소스를 프로비저닝할 수 있습니다.

관리형 노드 그룹

[관리형 노드 그룹](#)은 Amazon EKS 클러스터 내 Amazon EC2 인스턴스 컬렉션 관리에 대한 자동화와 사용자 지정을 결합합니다. AWS는 노드 패치, 업데이트 및 규모 조정과 같은 작업을 처리하여 운영상의 부담을 덜어줍니다. 동시에 사용자 지정 kubelet 인수가 지원되므로 고급 CPU 및 메모리 관리 정책을 사용할 수 있습니다. 또한 클러스터별 별도 권한의 필요성을 우회하면서 서비스 계정에 대한 AWS ID 및 액세스 관리(IAM) 역할을 통해 보안을 강화합니다.

자체 관리형 노드

[자체 관리형 노드](#)를 통해 Amazon EKS 클러스터 내에서 Amazon EC2 인스턴스를 완벽하게 제어할 수 있습니다. 사용자는 노드 관리, 규모 조정 및 유지 관리를 담당하므로, 기본 인프라를 완전히 제어할 수 있습니다. 이 옵션은 노드를 세밀하게 제어하고 사용자 지정해야 하며 인프라 관리 및 유지 관리에 시간을 투자할 준비가 된 사용자에게 적합합니다.

Amazon EKS Hybrid Nodes

[Amazon EKS Hybrid Nodes](#)를 사용하면 온프레미스 및 엣지 인프라를 Amazon EKS 클러스터의 노드로 사용할 수 있습니다. Amazon EKS Hybrid Nodes는 환경 전반의 Kubernetes 관리를 통합하고 온프레미스 및 엣지 애플리케이션을 위해 Kubernetes 컨트롤 플레인 관리를 AWS로 오프로드합니다.

Kubernetes 개념

Amazon Elastic Kubernetes Service(Amazon EKS)는 오픈 소스 [Kubernetes](#) 프로젝트에 기반한 AWS 관리형 서비스입니다. Amazon EKS 서비스가 AWS 클라우드와 통합되는 방식에 대해 알아야 할 사항이 있지만(특히 Amazon EKS 클러스터를 처음 생성할 때), 일단 가동되어 실행되면 다른 Kubernetes 클러스터와 거의 동일한 방식으로 Amazon EKS 클러스터를 사용하게 됩니다. 따라서 Kubernetes 클러스터를 관리하고 워크로드를 배포하려면 최소한 Kubernetes 개념에 대한 기본적인 이해가 필요합니다.

이 페이지에서는 Kubernetes 개념을 [the section called “Kubernetes를 사용해야 하는 이유”](#), [the section called “클러스터”](#), [the section called “워크로드”](#)의 세 섹션으로 나누어 설명합니다. 첫 번째 섹션에서는 Kubernetes 서비스, 특히 Amazon EKS와 같은 관리형 서비스를 실행하는 것의 가치에 대해 설명합니다. 워크로드 섹션에서는 Kubernetes 애플리케이션의 구축, 저장, 실행 및 관리 방법을 다룹니다. 클러스터 섹션에서는 Kubernetes 클러스터를 구성하는 다양한 구성 요소와 Kubernetes 클러스터를 생성하고 유지 관리하는 사용자의 책임에 대해 설명합니다.

주제

- [Kubernetes를 사용해야 하는 이유](#)

- [클러스터](#)
- [워크로드](#)
- [다음 단계](#)

내용을 살펴보면서 여기에서 다루는 주제에 대해 더 자세히 알아보고 싶을 경우 링크를 통해 Amazon EKS 및 쿠버네티스 문서에 있는 Kubernetes 개념에 대한 추가 설명으로 이동할 수 있습니다. Amazon EKS가 Kubernetes 컨트롤 플레인 및 컴퓨팅 기능을 구현하는 방법에 대한 자세한 내용은 [the section called “아키텍처”](#) 섹션을 참조하세요.

Kubernetes를 사용해야 하는 이유

Kubernetes는 미션 크리티컬한 프로덕션 품질의 컨테이너화된 애플리케이션을 실행할 때 가용성과 확장성을 개선하도록 설계되었습니다. Kubernetes는 이러한 목표를 달성하기 위해 단일 머신에서 Kubernetes를 실행하는 것이 아니라(물론 가능하지만), 수요에 따라 확장되거나 축소될 수 있는 컴퓨터 세트에서 애플리케이션을 실행합니다. Kubernetes에는 다음과 같은 작업을 더 쉽게 수행하기 위한 기능이 포함되어 있습니다.

- 여러 시스템에 애플리케이션 배포(포드에 배포된 컨테이너 사용)
- 컨테이너 상태 모니터링 및 장애가 발생한 컨테이너 다시 시작
- 부하에 따라 컨테이너 확장 및 축소
- 새 버전으로 컨테이너 업데이트
- 컨테이너 간 리소스 할당
- 시스템 간 트래픽 균형 조정

Kubernetes가 이러한 유형의 복잡한 태스크를 자동화하므로 애플리케이션 개발자는 인프라에 대한 걱정 없이 애플리케이션 워크로드를 구축하고 개선하는 데 집중할 수 있습니다. 개발자는 일반적으로 원하는 애플리케이션 상태를 설명하는 구성 파일을 YAML 파일 형식으로 생성합니다. 여기에는 실행할 컨테이너, 리소스 제한, 포드 복제본 수, CPU/메모리 할당, 선호도 규칙 등이 포함될 수 있습니다.

Kubernetes의 속성

Kubernetes에는 목표를 달성하기 위한 다음과 같은 특징이 있습니다.

- 컨테이너화 - Kubernetes는 컨테이너 오케스트레이션 도구입니다. Kubernetes를 사용하려면 먼저 애플리케이션을 컨테이너화해야 합니다. 애플리케이션 유형에 따라, 이러한 컨테이너화는 마이크로 서비스 세트, 배치 작업 또는 다른 형태가 될 수 있습니다. 이제 애플리케이션에서 Kubernetes 워크

폴로를 활용할 수 있으며, 이 워크플로에는 컨테이너를 [컨테이너 레지스트리의 이미지](#)로 저장하고, Kubernetes [클러스터](#)에 배포하고, 사용 가능한 [노드](#)에서 실행할 수 있는 거대한 도구 에코시스템이 포함됩니다. Kubernetes 클러스터에 컨테이너를 배포하기 전에 로컬 컴퓨터에서 Docker 또는 다른 [컨테이너 런타임](#)을 사용하여 개별 컨테이너를 빌드하고 테스트할 수 있습니다.

- **규모 조정 가능** - 애플리케이션에 대한 수요가 해당 애플리케이션의 실행 중인 인스턴스 용량을 초과하는 경우 Kubernetes를 스케일 업할 수 있습니다. 필요에 따라 Kubernetes는 애플리케이션에 더 많은 CPU 또는 메모리가 필요한지 여부를 확인하고 사용 가능한 용량을 자동으로 확장하거나 기존 용량을 더 많이 사용하여 이에 대응할 수 있습니다. 애플리케이션의 더 많은 인스턴스를 실행하기에 충분한 컴퓨팅 용량을 사용할 수 있는 경우, 포드 수준에서 확장을 수행할 수 있고([수평적 포드 자동 확장](#)), 늘어난 용량을 처리하기 위해 더 많은 노드를 가져와야 하는 경우, 노드 수준에서 확장을 수행할 수 있습니다([Cluster Autoscaler](#) 또는 [Karpenter](#)). 용량이 더 이상 필요하지 않게 되면 이러한 서비스는 필요하지 않은 포드를 삭제하고 불필요한 노드를 종료할 수 있습니다.
- **사용 가능** - 애플리케이션 또는 노드가 비정상이거나 사용할 수 없게 되면 Kubernetes는 실행 중인 워크로드를 사용 가능한 다른 노드로 이동할 수 있습니다. 워크로드의 실행 중인 인스턴스 또는 워크로드를 실행 중인 노드를 삭제하기만 하면 문제를 강제로 해결할 수 있습니다. 여기서 중요한 것은 워크로드를 현재 위치에서 더 이상 실행할 수 없는 경우 다른 위치로 가져올 수 있다는 것입니다.
- **선언적** - Kubernetes는 활성 조정을 사용하여 클러스터에 대해 선언한 상태가 실제 상태와 일치하는지 지속적으로 확인합니다. 예를 들어, 일반적으로 YAML 형식의 구성 파일을 통해 클러스터에 [Kubernetes 객체](#)를 적용하면 클러스터에서 실행할 워크로드를 시작하도록 요청할 수 있습니다. 나중에 구성을 변경하여 최신 버전의 컨테이너를 사용하거나 추가 메모리를 할당하는 등의 작업을 수행할 수 있습니다. Kubernetes는 원하는 상태를 설정하기 위해 수행해야 하는 작업을 수행합니다. 이러한 작업에는 노드 가동 또는 중단, 워크로드 중지 및 재시작, 업데이트된 컨테이너 가져오기 등이 포함될 수 있습니다.
- **구성 가능** — 애플리케이션은 일반적으로 여러 구성 요소로 구성되므로 이러한 구성 요소 세트(대개 여러 컨테이너로 표시됨)를 함께 관리할 수 있어야 합니다. Docker Compose는 Docker를 사용하여 직접 이 작업을 수행하는 방법을 제공하지만 Kubernetes [Kompose](#) 명령을 사용하면 Kubernetes에서 이 작업을 수행할 수 있습니다. 이 작업을 수행하는 방법에 대한 예는 [Docker 작성 파일을 Kubernetes 리소스로](#) 번역하기를 참조하세요.
- **확장 가능** - 독점 소프트웨어와 달리 오픈 소스 Kubernetes 프로젝트는 필요에 따라 원하는 방식으로 Kubernetes를 확장할 수 있도록 설계되었습니다. API 및 구성 파일은 직접 수정할 수 있습니다. 타사에서 자체 [컨트롤러](#)를 작성하여 인프라 및 최종 사용자 Kubernetes 기능을 모두 확장하는 것을 권장합니다. [웹훅](#)을 사용하면 클러스터 규칙을 설정하여 정책을 적용하고 변화하는 조건에 맞게 조정할 수 있습니다. Kubernetes 클러스터를 확장하는 방법에 대한 자세한 내용은 [Kubernetes 확장](#)을 참조하세요.

- 이식 가능 - Kubernetes를 사용하면 모든 애플리케이션 요구 사항을 동일한 방식으로 관리할 수 있기 때문에 많은 조직에서 이를 기반으로 운영을 표준화하고 있습니다. 개발자는 동일한 파이프라인을 사용하여 컨테이너화된 애플리케이션을 빌드하고 저장할 수 있습니다. 그런 다음 이러한 애플리케이션을 온프레미스, 클라우드, 레스토랑의 POS 터미널 또는 회사 원격 사이트에 분산된 IOT 디바이스에서 실행되는 Kubernetes 클러스터에 배포할 수 있습니다. 오픈 소스라는 특성 덕분에 사용자가 이러한 특수 Kubernetes 배포판과 이를 관리하는 데 필요한 도구를 함께 개발할 수 있습니다.

Kubernetes 관리

Kubernetes 소스 코드는 무료로 제공되므로 자체 장비만 있으면 직접 Kubernetes를 설치하고 관리할 수 있습니다. 그러나 Kubernetes를 자체 관리하려면 심층적인 운영 전문 지식이 필요하며 시간과 노력을 들여 유지 관리해야 합니다. 이러한 이유 때문에 프로덕션 워크로드를 배포하는 대부분의 사용자는 자체에서 테스트한 Kubernetes 배포판과 Kubernetes 전문가 지원을 제공하는 클라우드 제공업체(예: Amazon EKS)나 온프레미스 제공업체(예: Amazon EKS Anywhere)를 선택합니다. 이렇게 하면 클러스터를 유지 관리하는 데 필요한 다음과 같은 힘든 작업을 대부분 제거할 수 있습니다.

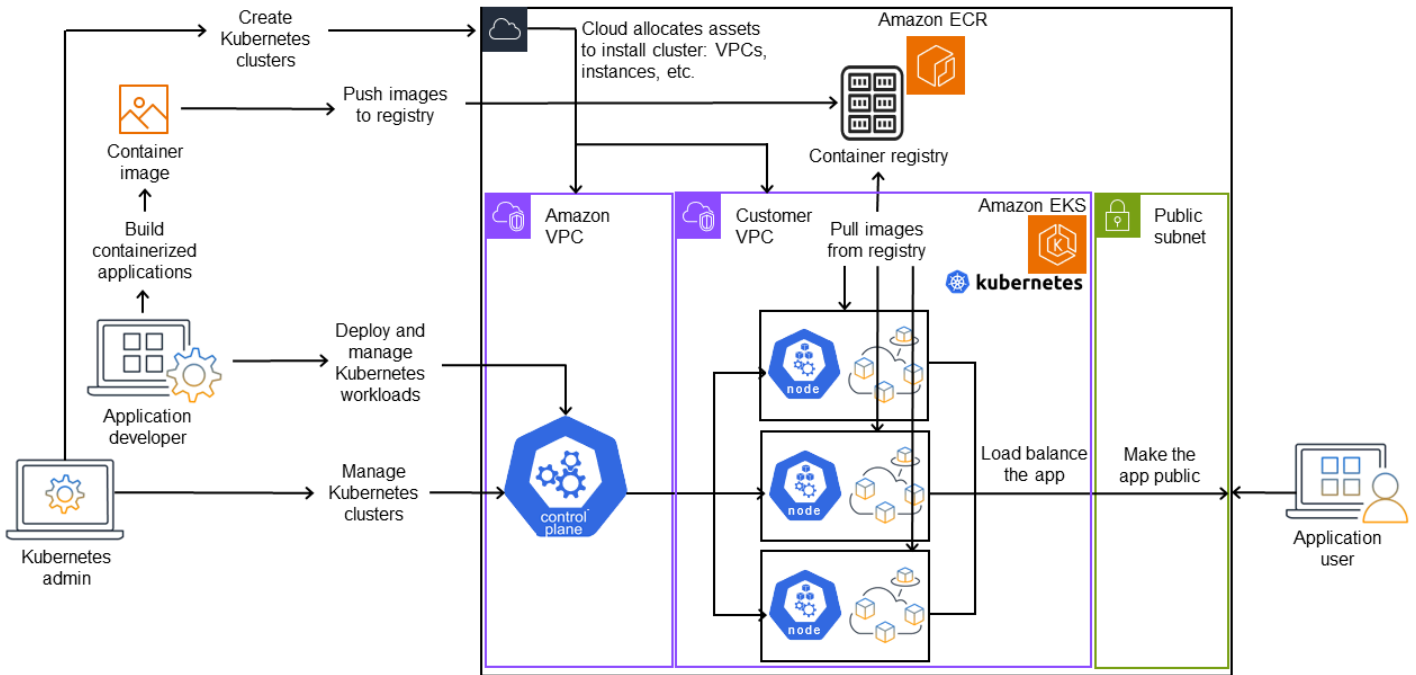
- 하드웨어 - 요구 사항에 따라 Kubernetes를 실행할 수 있는 하드웨어가 없는 경우 AWS Amazon EKS와 같은 클라우드 제공업체를 통해 초기 비용을 절약할 수 있습니다. Amazon EKS를 사용하면 컴퓨터 인스턴스(Amazon Elastic Compute Cloud), 자체 프라이빗 환경(Amazon VPC), 중앙 집중식 ID 및 권한 관리(IAM), 스토리지(Amazon EBS) 등을 비롯한 AWS에서 제공하는 최상의 클라우드 리소스를 사용할 수 있습니다. AWS가 컴퓨터, 네트워크, 데이터 센터 및 Kubernetes 실행에 필요한 기타 모든 물리적 구성 요소를 관리합니다. 마찬가지로, 수요가 가장 많은 기간 동안의 최대 용량을 처리하도록 데이터 센터를 계획할 필요도 없습니다. Amazon EKS Anywhere 또는 기타 온프레미스 Kubernetes 클러스터의 경우 Kubernetes 배포에 사용된 인프라를 관리할 책임은 사용자에게 있지만 여전히 AWS를 사용하여 Kubernetes를 최신 상태를 유지할 수 있습니다.
- 컨트롤 플레인 관리 - Amazon EKS가 컨테이너 예약, 애플리케이션 가용성 관리 및 기타 주요 태스크를 담당하는 AWS 호스팅 Kubernetes 컨트롤 플레인의 보안 및 가용성을 관리하므로 사용자는 애플리케이션 워크로드에 집중할 수 있습니다. 사용자의 클러스터가 중단되는 경우 AWS에 클러스터를 실행 상태로 복원할 수 있는 수단이 있어야 합니다. Amazon EKS Anywhere의 경우 컨트롤 플레인을 직접 관리할 수 있습니다.
- 테스트된 업그레이드 - 클러스터를 업그레이드할 때 Amazon EKS 또는 Amazon EKS Anywhere를 사용하여 Kubernetes 배포판의 테스트된 버전을 제공할 수 있습니다.
- 추가 기능 - Kubernetes를 확장하고 Kubernetes와 함께 작동하도록 구축된 수백 가지의 프로젝트가 있으며, 이를 클러스터의 인프라에 추가하거나 워크로드 실행을 지원하는 데 사용할 수 있습니다. 이러한 추가 기능을 직접 빌드 및 관리하는 대신 AWS는 클러스터와 함께 사용할 수 있는 [the section called “Amazon EKS 추가 기능”](#)을 제공합니다. Amazon EKS Anywhere는 여러 인기 있는 오픈 소스

프로젝트의 빌드가 포함된 [선별 패키지](#)를 제공합니다. 따라서 소프트웨어를 직접 빌드하거나 주요 보안 패치, 버그 수정 또는 업그레이드를 관리할 필요가 없습니다. 마찬가지로 기본 구성이 요구 사항을 충족하는 경우 일반적으로 이러한 추가 기능을 거의 구성할 필요가 없습니다. 추가 기능으로 클러스터를 확장하는 방법에 대한 자세한 내용은 [the section called “확장 클러스터”](#)을 참조하세요.

Kubernetes 적용 사례

다음 다이어그램에서는 Kubernetes 관리자 또는 애플리케이션 개발자가 Kubernetes 클러스터를 생성하고 사용하기 위해 수행하는 주요 활동을 보여줍니다. 이 프로세스에서는 AWS 클라우드를 기반 클라우드 제공업체의 예로 사용하여 Kubernetes 구성 요소가 서로 상호 작용하는 방식을 보여줍니다.

A Kubernetes cluster in action



Kubernetes 관리자는 Kubernetes 클러스터가 구축될 제공자의 유형에 맞는 도구를 사용하여 클러스터를 생성합니다. 이 예에서는 Amazon EKS라는 관리형 Kubernetes 서비스를 제공하는 AWS 클라우드를 제공업체로 사용합니다. 관리형 서비스는 클러스터를 위한 2개의 새 가상 프라이빗 클라우드 (Amazon VPC) 생성, 네트워킹 설정, 클라우드 자산 관리를 위한 새 VPC에 직접 Kubernetes 권한 매핑을 포함하여 클러스터를 생성하는 데 필요한 리소스를 자동으로 할당합니다. 또한 관리형 서비스에서는 컨트롤 플레인 서비스가 워크로드 실행을 위한 Kubernetes 노드로 0개 이상의 Amazon EC2 인스턴스를 실행 및 할당하는 위치를 지원함을 확인할 수 있습니다. AWS는 컨트롤 플레인에 대해 하나의 자체 Amazon VPC를 관리하며, 다른 Amazon VPC에는 워크로드를 실행하는 고객 노드가 포함되어 있습니다.

앞으로 Kubernetes 관리자의 많은 태스크는 `kubectl`과 같은 Kubernetes 도구를 사용하여 수행됩니다. 이 도구는 클러스터의 컨트롤 플레인에 직접 서비스를 요청합니다. 클러스터를 쿼리하고 변경하는 방식은 다른 Kubernetes 클러스터에서 수행하는 방식과 매우 비슷합니다.

이 클러스터에 워크로드를 배포하려는 애플리케이션 개발자는 여러 작업을 수행할 수 있습니다. 개발자는 애플리케이션을 하나 이상의 컨테이너 이미지로 빌드한 다음 Kubernetes 클러스터에서 액세스할 수 있는 컨테이너 레지스트리로 해당 이미지를 푸시해야 합니다. AWS는 이 용도로 Amazon Elastic Container Registry(Amazon ECR)를 제공합니다.

애플리케이션을 실행하기 위해 개발자는 레지스트리에서 가져올 컨테이너와 해당 컨테이너를 포드에 래핑하는 방법을 포함하여 클러스터에 애플리케이션 실행 방법을 알려주는 YAML 형식의 구성 파일을 생성할 수 있습니다. 컨트롤 플레인(스케줄러)은 컨테이너를 하나 이상의 노드에 예약하고 각 노드의 컨테이너 런타임은 실제로 필요한 컨테이너를 가져와 실행합니다. 또한 개발자는 Application Load Balancer를 설정하여 각 노드에서 실행 중인 사용 가능한 컨테이너에 대한 트래픽의 균형을 조정하고 공용 네트워크를 통해 외부에서 사용할 수 있도록 애플리케이션을 노출할 수 있습니다. 이 모든 작업이 완료되면 애플리케이션을 사용하려는 사용자가 애플리케이션 엔드포인트에 연결하여 액세스할 수 있습니다.

다음 섹션에서는 Kubernetes 클러스터 및 워크로드의 관점에서 이러한 각 기능에 대해 자세히 살펴봅니다.

클러스터

작업이 Kubernetes 클러스터를 시작하고 관리하는 것이라면 Kubernetes 클러스터가 생성, 강화, 관리 및 삭제되는 방법을 알아야 합니다. 또한 클러스터를 구성하는 구성 요소가 무엇인지, 이러한 구성 요소를 유지 관리하기 위해 어떤 작업이 필요한지도 알아야 합니다.

클러스터를 관리하는 도구는 Kubernetes 서비스와 기반 하드웨어 제공업체 간의 중복을 처리합니다. 이러한 이유 때문에 Kubernetes 제공업체(예: Amazon EKS 또는 Amazon EKS Anywhere)는 제공업체 전용 도구를 사용하여 이러한 태스크를 자동화하는 경향이 있습니다. 예를 들어, Amazon EKS 클러스터를 시작하려면 `eksctl create cluster`를 사용할 수 있고 Amazon EKS Anywhere에서는 `eksctl anywhere create cluster`를 사용할 수 있습니다. 이러한 명령은 Kubernetes 클러스터를 생성하지만 제공업체별로 다르며 Kubernetes 프로젝트 자체에 포함되지는 않습니다.

클러스터 생성 및 관리 도구

Kubernetes 프로젝트는 Kubernetes 클러스터를 수동으로 생성할 수 있는 도구를 제공합니다. 따라서 단일 시스템에 Kubernetes를 설치하거나 시스템에서 컨트롤 플레인을 실행하고 노드를 수동으로 추가하려는 경우 Kubernetes [Install Tools](#) 아래에 나열된 [kind](#), [minikube](#), [kubeadm](#) 등의 CLI 도구를 사용할 수 있습니다. 클러스터 생성 및 관리의 전체 수명 주기를 간소화하고 자동화하려면 Amazon EKS 또는

Amazon EKS Anywhere와 같이 확립된 Kubernetes 제공업체가 지원하는 도구를 사용하는 것이 훨씬 쉽습니다.

AWS 클라우드에서는 [eksctl](#)과 같은 CLI 도구나 Terraform([Amazon EKS Blueprints for Terraform](#) 참조) 같은 보다 선언적인 도구를 사용하여 [Amazon EKS](#) 클러스터를 생성할 수 있습니다. AWS Management Console에서 클러스터를 생성할 수도 있습니다. Amazon EKS의 제공 사항 목록은 [Amazon EKS 기능](#)을 참조하세요. Amazon EKS가 대신 수행하는 Kubernetes 책임은 다음과 같습니다.

- 관리형 컨트롤 플레인 — AWS는 자동으로 컨트롤 플레인을 관리하고 AWS 가용 영역 전체에서 사용할 수 있도록 만들기 때문에 Amazon EKS 클러스터의 가용성과 확장성이 보장됩니다.
- 노드 관리 - 노드를 수동으로 추가하는 대신 Amazon EKS에서 관리형 노드 그룹([the section called “관리형 노드 그룹”](#) 참조) 또는 [Karpenter](#)를 사용하여 필요에 따라 Amazon EKS에서 노드를 자동으로 생성할 수 있습니다. 관리형 노드 그룹은 Kubernetes [클러스터 오토 스케일링](#)과 통합됩니다. 노드 관리 도구를 사용하면 워크로드를 배포하고 노드를 선택하는 방법을 설정하는 [일정 예약](#) 기능을 사용하여 [스팟 인스턴스](#), 노드 통합, 가용성 등과 같은 작업에서 비용 절감 효과를 누릴 수 있습니다.
- 클러스터 네트워킹 - CloudFormation 템플릿을 사용하여 eksctl은 Kubernetes 클러스터의 컨트롤 플레인과 데이터 영역(노드) 구성 요소 간의 네트워킹을 설정합니다. 또한 내부 및 외부 통신을 수행할 수 있는 엔드포인트를 설정합니다. 자세한 내용은 [Amazon EKS 작업자 노드에 대한 클러스터 네트워킹 설명](#)을 참조하세요. Amazon EKS에서 포드 간 통신은 Amazon EKS Pod Identity를 사용하여 수행됩니다([the section called “Pod Identity”](#) 참조). 이 기능을 사용하는 경우 포드가 자격 증명 및 권한을 관리하는 AWS 클라우드 메서드를 활용할 수 있습니다.
- 추가 기능 - Amazon EKS를 사용하면 Kubernetes 클러스터를 지원하는 데 일반적으로 사용되는 소프트웨어 구성 요소를 빌드하고 추가할 필요가 없습니다. 예를 들어 AWS Management Console에서 Amazon EKS 클러스터를 생성하는 경우 Amazon EKS kube-proxy([the section called “kube-proxy”](#)), Kubernetes용 Amazon VPC CNI 플러그인([the section called “Amazon VPC CNI”](#)), CoreDNS([the section called “CoreDNS”](#)) 추가 기능이 자동으로 추가됩니다. 사용 가능한 추가 기능의 목록을 포함하여 이러한 추가 기능에 대한 자세한 내용은 [the section called “Amazon EKS 추가 기능”](#)을 참조하세요.

사용자가 자체 온프레미스 컴퓨터 및 네트워크에서 클러스터를 실행할 수 있도록 Amazon은 [Amazon EKS Anywhere](#)를 제공합니다. AWS 클라우드가 공급자가 되는 대신 자체 장비를 사용하여 [VMWare vSphere](#), [베어 메탈\(Tinkerbell provider\)](#), [Snow](#), [CloudStack](#) 또는 [Nutanix](#) 플랫폼에서 Amazon EKS Anywhere를 실행할 수 있습니다.

Amazon EKS Anywhere는 Amazon EKS에서 사용하는 것과 동일한 [Amazon EKS Distro](#) 소프트웨어를 기반으로 합니다. 하지만 Amazon EKS Anywhere는 다양한 [Kubernetes Cluster API\(CAPI\)](#) 인터페

이스 구현을 사용하여 Amazon EKS Anywhere 클러스터(예: vSphere의 경우, [CAPV](#), CloudStack의 경우, [CAPC](#))에 있는 시스템의 전체 수명 주기를 관리합니다. 전체 클러스터가 사용자 장비에서 실행되므로 컨트롤 플레인을 관리하고 해당 데이터를 백업해야 하는 추가적인 책임은 사용자에게 있습니다(이 문서 뒷부분의 etcd 참조).

클러스터 구성 요소

Kubernetes 클러스터 구성 요소는 컨트롤 플레인과 워커 노드라는 두 가지 주요 영역으로 구분됩니다. [컨트롤 플레인 구성 요소](#)는 클러스터를 관리하고 해당 API에 대한 액세스를 제공합니다. 워커 노드(간단히 노드라고도 함)는 실제 워크로드가 실행되는 위치를 제공합니다. [노드 구성 요소](#)는 각 노드에서 실행되어 컨트롤 플레인과 통신하고 컨테이너를 실행하는 서비스로 구성됩니다. 클러스터의 워커 노드 세트를 데이터 플레인이라고 합니다.

컨트롤 플레인

컨트롤 플레인은 클러스터를 관리하는 서비스 세트로 구성됩니다. 이러한 서비스는 모두 단일 컴퓨터에서 실행되거나 여러 컴퓨터에 분산되어 있을 수 있습니다. 내부적으로는 이를 컨트롤 플레인 인스턴스(CPI)라고 합니다. CPI 실행 방법은 클러스터 규모와 고가용성 요구 사항에 따라 다릅니다. 클러스터의 수요가 증가하면 컨트롤 플레인 서비스가 확장되어 해당 서비스의 인스턴스를 더 많이 제공할 수 있으며, 이때 요청은 인스턴스 간에서 로드 밸런싱됩니다.

Kubernetes 컨트롤 플레인의 구성 요소가 수행하는 태스크에는 다음이 포함됩니다.

- 클러스터 구성 요소(API 서버)와의 통신 - API 서버([kube-apiserver](#))는 Kubernetes API를 노출하므로 클러스터 내부와 외부 모두에서 클러스터에 요청을 보낼 수 있습니다. 즉, 클러스터의 객체(포드, 서비스, 노드 등)를 추가하거나 변경하는 요청은 포드를 실행하는 kubectl의 요청과 같은 외부 명령에서 발생할 수 있습니다. 마찬가지로, API 서버에서 클러스터 내 구성 요소로 요청을 보낼 수 있습니다(예: 포드 상태에 대한 kubelet 서비스의 쿼리).
- 클러스터에 대한 데이터 저장([etcd](#) 키 값 저장소) - etcd 서비스는 클러스터의 현재 상태를 추적하는 중요한 역할을 합니다. etcd 서비스에 액세스할 수 없게 되면 얼마 동안 워크로드가 계속 실행되더라도 클러스터의 상태를 업데이트하거나 쿼리할 수 없게 됩니다. 이러한 이유로 중요 클러스터는 일반적으로 로드 밸런싱된 여러 개의 etcd 서비스 인스턴스를 동시에 실행하고 데이터 손실 또는 손상에 대비하여 etcd 키 값 저장소를 주기적으로 백업합니다. Amazon EKS에서는 기본적으로 이 모든 작업이 자동으로 처리된다는 점에 유의하세요. Amazon EKS Anywhere는 [etcd 백업 및 복원](#)에 대한 지침을 제공합니다. etcd에서 데이터를 관리하는 방법을 알아보려면 [etcd Data Model](#)을 참조하세요.
- 노드에 포드 예약(스케줄러) - Kubernetes에서 포드를 시작하거나 중지하기 위한 요청은 [Kubernetes 스케줄러\(kube-scheduler\)](#)로 전달됩니다. 클러스터에는 포드를 실행할 수 있는 노드가 여러 개 있을

수 있으므로 포드를 실행할 노드(복제본의 경우, 여러 노드)를 선택하는 것은 스케줄러입니다. 요청된 포드를 기존 노드에서 실행할 수 있는 가용 용량이 충분하지 않은 경우 다른 프로비저닝이 수행되지 않는 한 요청이 실패합니다. 이러한 프로비저닝에는 새 노드를 자동으로 시작하여 워크로드를 처리할 수 있는 관리형 노드 그룹([the section called “관리형 노드 그룹”](#)) 또는 [Karpenter](#)와 같은 서비스를 활성화하는 작업이 포함될 수 있습니다.

- 구성 요소를 원하는 상태로 유지(컨트롤러 관리자) - Kubernetes 컨트롤러 관리자는 대몬 프로세스([kube-controller-manager](#))로 실행되어 클러스터의 상태를 감시하고 클러스터를 변경하여 예상 상태를 재설정합니다. 특히 `statefulset-controller`, `endpoint-controller`, `cronjob-controller`, `node-controller` 등 다양한 Kubernetes 객체를 감시하는 컨트롤러가 여러 개 있습니다.
- 클라우드 리소스 관리(Cloud Controller Manager) - Kubernetes와 기본 데이터 센터 리소스에 대한 요청을 수행하는 클라우드 제공업체 간의 상호 작용은 [Cloud Controller Manager](#)([cloud-controller-manager](#))에서 처리합니다. Cloud Controller Manager에서 관리하는 컨트롤러에는 경로 컨트롤러(클라우드 네트워크 경로 설정에 사용), 서비스 컨트롤러(클라우드 로드 밸런싱 서비스에 사용), 노드 수명 주기 컨트롤러(전체 해당 수명 주기에서 노드를 Kubernetes와 동기화하는 데 사용)가 포함될 수 있습니다.

워커 노드(데이터 플레인)

단일 노드 Kubernetes 클러스터의 경우 워크로드는 컨트롤 플레인과 동일한 시스템에서 실행됩니다. 하지만 좀 더 일반적인 구성은 Kubernetes 워크로드 실행 전용으로 사용되는 별도의 컴퓨터 시스템([노드](#))을 하나 이상 두는 것입니다.

Kubernetes 클러스터를 처음 생성할 때 일부 클러스터 생성 도구를 사용하면 기존 컴퓨터 시스템을 식별하거나 제공업체가 새 컴퓨터 시스템을 생성하도록 하여 클러스터에 추가할 특정 수의 노드를 구성할 수 있습니다. 이러한 시스템에 워크로드를 추가하기 전에 각 노드에 서비스를 추가하여 다음 기능을 구현합니다.

- 각 노드 관리([kubelet](#)) - API 서버는 각 노드에서 실행되는 [kubelet](#) 서비스와 통신하여 노드가 제대로 등록되었으며 스케줄러가 요청한 포드가 실행되고 있는지 확인합니다. kubelet은 포드 매니페스트를 읽고 로컬 시스템의 포드에 필요한 스토리지 볼륨 또는 기타 기능을 설정할 수 있습니다. 또한 로컬에서 실행 중인 컨테이너의 상태를 확인할 수 있습니다.
- 노드에서 컨테이너 실행(컨테이너 런타임) — 각 노드의 [컨테이너 런타임](#)은 노드에 할당된 각 포드에 요청된 컨테이너를 관리합니다. 즉, 적절한 레지스트리에서 컨테이너 이미지를 가져와 컨테이너를 실행하고 중지하며 컨테이너에 대한 쿼리에 응답할 수 있습니다. 기본 컨테이너 런타임은 [containerd](#)입니다. Kubernetes 1.24부터 컨테이너 런타임으로 사용할 수 있는 Docker(`dockershim`)의 특수 통합이 Kubernetes에서 삭제되었습니다. 로컬 시스템에서 컨테이너

를 테스트하고 실행하는 데 여전히 Docker를 사용할 수 있지만 Docker와 Kubernetes를 함께 사용하려면 이제 각 노드에 [Docker Engine을 설치](#)하여 Kubernetes와 함께 사용해야 합니다.

- 컨테이너 간 네트워킹 관리(**kube-proxy**) - 포드 간 통신을 지원하기 위해 Kubernetes는 포드 네트워크를 설정하여 해당 포드와 관련된 IP 주소 및 포트를 추적하는 [서비스](#)라는 기능을 사용합니다. [kube-proxy](#) 서비스는 모든 노드에서 실행되어 포드 간 통신이 이루어질 수 있도록 합니다.

확장 클러스터

클러스터를 지원하기 위해 Kubernetes에 추가할 수 있는 일부 서비스가 있지만 이러한 서비스는 컨트롤 플레인에서는 실행되지 않습니다. 이러한 서비스는 종종 kube-system 네임스페이스의 노드나 자체 네임스페이스에서 직접 실행됩니다(타사 서비스 공급자에서 자주 실행됨). 일반적인 예로는 클러스터에 DNS 서비스를 제공하는 CoreDNS 서비스를 들 수 있습니다. 클러스터의 kube-system에서 실행 중인 클러스터 서비스를 확인하는 방법에 대한 자세한 내용은 [기본 제공 서비스 검색](#)을 참조하세요.

클러스터에 추가하는 것을 고려할 수 있는 다양한 유형의 추가 기능이 있습니다. 클러스터를 정상 상태로 유지하기 위해 로깅, 감사, 지표 등의 작업을 수행할 수 있는 관찰성 기능([클러스터 모니터링](#) 참조)을 추가할 수 있습니다. 이 정보를 사용하여 종종 동일한 관찰성 인터페이스를 통해 발생하는 문제를 해결할 수 있습니다. 이러한 유형의 서비스에 대한 예로 [Amazon GuardDuty](#), CloudWatch([the section called “Amazon CloudWatch”](#) 참조), [AWS Distro for OpenTelemetry](#), Kubernetes용 Amazon VPC CNI 플러그인([the section called “Amazon VPC CNI”](#) 참조), [Grafana Kubernetes Monitoring](#)이 있습니다. 스토리지([앱 데이터 스토리지](#) 참조)의 경우 Amazon EKS에 대한 추가 기능으로 Amazon Elastic Block Store CSI 드라이버([the section called “Amazon EBS”](#) 참조), Amazon Elastic File System CSI 드라이버([the section called “Amazon EFS”](#) 참조), Amazon FSx for NetApp ONTAP CSI 드라이버([the section called “Amazon FSx for NetApp ONTAP”](#) 참조)와 같은 여러 서드 파티 스토리지 추가 기능이 있습니다.

Amazon EKS 추가 기능의 전체 목록은 [the section called “Amazon EKS 추가 기능”](#) 섹션을 참조하세요.

워크로드

Kubernetes는 [워크로드](#)를 ‘Kubernetes에서 실행되는 애플리케이션’으로 정의합니다. 해당 애플리케이션은 [포드](#)에서 [컨테이너](#)로 실행되는 마이크로서비스 세트로 구성되거나 배치 작업 또는 다른 유형의 애플리케이션으로 실행될 수 있습니다. Kubernetes의 작업은 해당 객체의 설정 또는 배포에 대한 요청이 수행되도록 하는 것입니다. 애플리케이션을 배포하는 사용자는 컨테이너가 빌드되는 방식, 포드가 정의되는 방식 및 컨테이너를 배포하는 데 사용할 수 있는 방법에 대해 알아야 합니다.

컨테이너

Kubernetes에서 배포하고 관리하는 애플리케이션 워크로드의 가장 기본적인 요소는 [포드](#)입니다. 포드는 애플리케이션의 구성 요소를 유지하는 방법과 포드의 속성을 설명하는 사양을 정의하는 방법을 나타냅니다. 이것은 Linux 시스템용 소프트웨어를 함께 패키징하지만 그 자체가 하나의 엔터티로 실행되지는 않는 RPM 또는 Deb 패키지와 대조를 이룹니다.

포드는 배포 가능한 가장 작은 단위이기 때문에 보통 단일 컨테이너를 유지합니다. 하지만 컨테이너가 긴밀하게 연계된 경우 한 포드에 여러 컨테이너가 있을 수 있다. 예를 들어, 웹 서버 컨테이너는 로깅, 모니터링 또는 웹 서버 컨테이너와 밀접하게 연결된 기타 서비스를 제공할 수 있는 [사이드카](#) 유형의 컨테이너와 함께 포드에 패키징될 수 있습니다. 이 경우 동일한 포드에 있으면 실행 중인 각 포드 인스턴스에 대해 두 컨테이너가 항상 같은 노드에서 실행됩니다. 마찬가지로, 포드의 모든 컨테이너는 동일한 환경을 공유하며, 포드의 컨테이너는 동일한 격리된 호스트에 있는 것처럼 실행됩니다. 결과적으로, 컨테이너는 포드에 대한 액세스를 제공하는 단일 IP 주소를 공유하고 컨테이너는 자체 로컬 호스트에서 실행되는 것처럼 서로 통신할 수 있습니다.

포드 사양([PodSpec](#))은 원하는 포드 상태를 정의합니다. 워크로드 리소스를 사용하여 [포드 템플릿](#)을 관리하는 방식으로 개별 포드 또는 여러 포드를 배포할 수 있습니다. 워크로드 리소스에는 [Deployments](#)(여러 포드 복제본 관리), [StatefulSets](#)(데이터베이스 포드와 같이 고유해야 하는 포드를 배포) 및 [DaemonSets](#)(포드가 모든 노드에서 지속적으로 실행되어야 하는 경우)가 포함됩니다. 이에 대한 자세한 내용은 나중에 다루겠습니다.

포드는 배포하는 가장 작은 단위인 반면, 컨테이너는 빌드하고 관리하는 가장 작은 단위입니다.

컨테이너 빌드

포드는 실제로 하나 이상의 컨테이너를 포함하는 구조이며, 각 컨테이너 자체는 파일 시스템, 실행 파일, 구성 파일, 라이브러리 및 애플리케이션을 실제로 실행하기 위한 기타 구성 요소를 유지합니다. Docker Inc.라는 회사가 처음으로 컨테이너를 대중화했기 때문에 일부 사람들은 컨테이너를 Docker 컨테이너라고 부릅니다. 그러나 이후 [Open Container Initiative](#)가 업계를 위한 컨테이너 런타임, 이미지 및 배포 방법을 정의했습니다. 컨테이너가 기존의 많은 Linux 기능을 기반으로 생성되었다는 사실에 추가로, 컨테이너를 OCI 컨테이너, Linux 컨테이너 또는 그냥 컨테이너라고 부르는 경우가 많습니다.

컨테이너를 빌드할 때는 일반적으로 Dockerfile(문자 그대로 이름이 지정됨)로 시작합니다. 해당 Dockerfile 내에서 다음을 식별합니다.

- 기본 이미지 — 기본 컨테이너 이미지는 일반적으로 최소 버전의 운영 체제 파일 시스템(예: [Red Hat Enterprise Linux](#) 또는 [Ubuntu](#)) 또는 특정 유형의 애플리케이션을 실행하기 위한 소프트웨어를 제공하도록 개선된 최소 시스템(예: [nodejs](#) 또는 [python](#) 앱)에서 빌드되는 컨테이너입니다.

- 애플리케이션 소프트웨어 — Linux 시스템에 추가하는 것과 거의 같은 방식으로 애플리케이션 소프트웨어를 컨테이너에 추가할 수 있습니다. 예를 들어, Dockerfile에서 npm 및 yarn을 실행하여 Java 애플리케이션을 설치하거나 yum 및 dnf를 실행하여 RPM 패키지를 설치할 수 있습니다. 즉, Dockerfile에서 RUN 명령을 사용하면 기본 이미지의 파일 시스템에서 사용 가능한 모든 명령을 실행하여 결과 컨테이너 이미지 내에서 소프트웨어를 설치하거나 소프트웨어를 구성할 수 있습니다.
- 지침 — [Dockerfile 참조](#)는 구성 시 Dockerfile에 추가할 수 있는 지침을 기술합니다. 여기에는 컨테이너 자체의 내용을 빌드하는 데 사용되고(로컬 시스템의 파일 ADD 또는 COPY), 컨테이너가 실행될 때 실행할 명령을 식별하고(CMD 또는 ENTRYPOINT), 컨테이너를 실행되는 시스템에 연결하는 데 사용되는(실행할 USER, 마운트할 로컬 VOLUME 또는 EXPOSE할 포트) 지침이 포함됩니다.

docker 명령 및 서비스는 전통적으로 컨테이너를 빌드하는 데 사용되었지만(docker build), 컨테이너 이미지를 빌드하는 데 사용할 수 있는 다른 도구로는 [podman](#) 및 [nerdctl](#)이 있습니다. 컨테이너 빌드에 대해 알아보려면 [더 나은 컨테이너 이미지 빌드](#) 또는 [Docker 빌드 개요](#)를 참조하세요.

컨테이너 저장

컨테이너 이미지를 빌드한 후에는 워크스테이션의 컨테이너 [배포 레지스트리](#) 또는 퍼블릭 컨테이너 레지스트리에 이미지를 저장할 수 있습니다. 워크스테이션에 있는 프라이빗 컨테이너 레지스트리를 실행하면 컨테이너 이미지를 로컬에 저장하여 필요할 때 즉시 사용할 수 있습니다.

컨테이너 이미지를 보다 공개적인 방식으로 저장하려면 퍼블릭 컨테이너 레지스트리로 푸시할 수 있습니다. 퍼블릭 컨테이너 레지스트리는 컨테이너 이미지를 저장하고 배포할 수 있는 중앙 위치를 제공합니다. 퍼블릭 컨테이너 레지스트리의 예로는 [Amazon Elastic Container Registry](#), [Red Hat Quay](#) 레지스트리 및 [Docker Hub](#) 레지스트리가 있습니다.

Amazon Elastic Kubernetes Service(Amazon EKS)에서 컨테이너화된 워크로드를 실행할 경우 Amazon Elastic Container Registry에 저장된 Docker 공식 이미지의 사본을 가져오는 것이 좋습니다. Amazon ECR은 2021년부터 이러한 이미지를 저장하고 있습니다. [Amazon ECR 퍼블릭 갤러리](#)에서 인기 있는 컨테이너 이미지를 검색할 수 있으며, 특히 Docker Hub 이미지의 경우 [Amazon ECR Docker 갤러리](#)를 검색할 수 있습니다.

실행 중인 컨테이너

컨테이너는 표준 형식으로 빌드되므로 컨테이너 런타임을 실행할 수 있고(예: Docker) 콘텐츠가 로컬 시스템의 아키텍처와 일치하는(예: x86_64 또는 arm) 모든 시스템에서 컨테이너를 실행할 수 있습니다. 컨테이너를 테스트하거나 로컬 데스크톱에서 실행하려면 docker run 또는 podman run 명령을 사용하여 로컬 호스트에서 컨테이너를 시작할 수 있습니다. 하지만 Kubernetes의 경우 각 워커 노드에는 컨테이너 런타임이 배포되어 있으며 Kubernetes가 노드에서 컨테이너를 실행하도록 요청하기만 하면 됩니다.

컨테이너가 노드에서 실행되도록 할당되면 노드는 요청된 버전의 컨테이너 이미지가 노드에 이미 존재하는지 확인합니다. 존재하지 않는 경우 Kubernetes는 컨테이너 런타임에 적절한 컨테이너 레지스트리에서 해당 컨테이너를 가져온 다음 해당 컨테이너를 로컬에서 실행하도록 지시합니다. 컨테이너 이미지는 랩톱, 컨테이너 레지스트리 및 Kubernetes 노드 사이에서 이동하는 소프트웨어 패키지를 나타낸다는 것을 기억하세요. 컨테이너는 해당 이미지의 실행 중인 인스턴스를 나타냅니다.

포드

컨테이너가 준비되면 포드를 사용하는 작업에 포드를 구성하고, 배포하고, 액세스 가능하게 만드는 작업이 포함됩니다.

포드 구성

포드를 정의할 때 포드에 속성 세트를 할당합니다. 이러한 속성에는 최소한 포드 이름과 실행할 컨테이너 이미지가 포함되어야 합니다. 하지만 포드 정의를 사용하여 구성할 수 있는 다른 사항도 많습니다 (포드에 들어갈 수 있는 항목에 대한 자세한 내용은 [PodSpec](#) 페이지 참조). 다음이 포함됩니다.

- 스토리지 - 실행 중인 컨테이너를 중지하고 삭제하면 더 많은 영구 스토리지를 설정하지 않는 한 해당 컨테이너의 데이터 스토리지가 사라집니다. Kubernetes는 다양한 스토리지 유형을 지원하며 이를 **볼륨**이라는 이름으로 추상화합니다. 스토리지 유형에는 [CephFS](#), [NFS](#), [iSCSI](#) 등이 포함됩니다. 로컬 컴퓨터의 [로컬 블록 디바이스](#)를 사용할 수도 있습니다. 클러스터에서 이러한 스토리지 유형 중 하나를 사용할 경우 컨테이너 파일 시스템에서 선택한 탑재 지점에 스토리지 볼륨을 탑재할 수 있습니다. **영구 볼륨**은 포드가 삭제된 후에도 계속 존재하는 볼륨이고, **임시 볼륨**은 포드가 삭제될 때 삭제됩니다. 클러스터 관리자가 클러스터에 대해 서로 다른 [스토리지 클래스](#)를 생성한 경우 사용 후 볼륨을 삭제하거나 회수할지 여부, 추가 공간이 필요할 경우 볼륨을 확장할지 여부, 특정 성능 요구 사항을 충족하는지 여부 등, 사용하는 스토리지의 속성을 선택할 수 있는 옵션이 제공됩니다.
- 보안 암호 — 포드 사양의 컨테이너에 **보안 암호**를 제공할 때 해당 컨테이너에 파일 시스템, 데이터베이스 또는 기타 보호된 자산에 액세스하는 데 필요한 권한을 제공할 수 있습니다. 키, 암호 및 토큰은 비밀로 저장할 수 있는 항목입니다. 보안 암호를 사용하면 이 정보를 컨테이너 이미지에 저장할 필요가 없어지고 실행 중인 컨테이너에 암호만 제공하면 됩니다. 보안 암호와 비슷한 것은 [ConfigMap](#)입니다. ConfigMap에는 서비스 구성을 위한 키-값 쌍과 같은 덜 중요한 정보가 들어 있는 경향이 있습니다.
- 컨테이너 리소스 — 컨테이너를 추가로 구성하기 위한 객체는 리소스 구성의 형태를 취할 수 있습니다. 각 컨테이너에 사용할 수 있는 메모리 및 CPU 양을 요청하고 컨테이너가 사용할 수 있는 리소스의 총량을 제한할 수 있습니다. 예제는 [포드 및 컨테이너 리소스 관리](#)를 참조하세요.
- 중단 — 포드는 모르는 사이에 중단되거나(노드가 다운됨) 자발적으로 중단될 수 있습니다(업그레이드가 필요함). [포드 중단 예산](#)을 구성하면 중단이 발생했을 때 애플리케이션의 가용성을 어느 정도 제어할 수 있습니다. 예제는 애플리케이션에 대한 [중단 예산 지정](#)을 참조하세요.

- 네임스페이스 - Kubernetes는 Kubernetes 구성 요소와 워크로드를 서로 분리하는 다양한 방법을 제공합니다. 동일한 [네임스페이스](#)에서 특정 애플리케이션의 모든 포드를 실행하는 것은 해당 포드를 함께 보호하고 관리하는 일반적인 방법입니다. 사용할 네임스페이스를 직접 생성하거나 네임스페이스를 지정하지 않도록 선택할 수 있습니다(이 경우 Kubernetes가 default 네임스페이스 사용). Kubernetes 컨트롤 플레인 구성 요소는 일반적으로 [kube 시스템](#) 네임스페이스에서 실행됩니다.

앞서 설명한 구성은 일반적으로 Kubernetes 클러스터에 적용할 YAML 파일에 함께 수집됩니다. 개인용 Kubernetes 클러스터의 경우 이러한 YAML 파일을 로컬 시스템에 저장하기만 하면 됩니다. 그러나 더 중요한 클러스터와 워크로드의 경우 [GitOps](#)가 워크로드 및 Kubernetes 인프라 리소스 모두에 대한 스토리지 및 업데이트를 자동화하는 데 널리 사용되는 방법입니다.

포드 정보를 함께 수집하고 배포하는 데 사용되는 객체는 다음 배포 방법 중 하나로 정의됩니다.

포드 배포

포드를 배포하기 위해 선택하는 방법은 해당 포드로 실행하려는 애플리케이션의 유형에 따라 달라집니다. 다음은 몇 가지 옵션입니다.

- 상태 비저장 애플리케이션 — 상태 비저장 애플리케이션은 클라이언트의 세션 데이터를 저장하지 않으므로 다른 세션에서 이전 세션에서 발생한 일을 다시 참조할 필요가 없습니다. 따라서 포드가 비정상 상태가 된 경우 새 포드로 대체하거나 상태를 저장하지 않고 다른 포드로 이동하는 것이 더 쉬워집니다. 상태 비저장 애플리케이션(예: 웹 서버)을 실행 중인 경우, [배포](#)를 사용하여 [포드](#)와 [ReplicaSet](#)을 배포할 수 있습니다. ReplicaSet은 동시에 실행하려는 포드의 인스턴스 수를 정의합니다. ReplicaSet을 직접 실행할 수도 있지만 한 번에 실행해야 하는 포드의 복제본 수를 정의하기 위해 배포 내에서 직접 복제본을 실행하는 것이 일반적입니다.
- 상태 저장 애플리케이션 — 상태 저장 애플리케이션은 포드의 자격 증명과 포드가 시작되는 순서가 중요한 애플리케이션입니다. 이러한 애플리케이션에는 안정적인 영구 스토리지가 필요하며 일관된 방식으로 배포하고 확장해야 합니다. Kubernetes에 상태 저장 애플리케이션을 배포하려면 [StatefulSets](#)를 사용하면 됩니다. 일반적으로 StatefulSet으로 실행되는 애플리케이션의 예로는 데이터베이스가 있습니다. StatefulSet 내에서 복제본, 포드 및 해당 컨테이너, 마운트할 스토리지 볼륨, 컨테이너에서 데이터가 저장되는 위치를 정의할 수 있습니다. ReplicaSet으로 배포되는 데이터베이스의 예는 [Run a Replicated Stateful Application](#)을 참조하세요.
- 노드별 애플리케이션 - Kubernetes 클러스터의 모든 노드에서 애플리케이션을 실행하려는 경우가 있습니다. 예를 들어, 데이터 센터에서 모든 컴퓨터가 모니터링 애플리케이션 또는 특정 원격 액세스 서비스를 실행하도록 요구할 수 있습니다. Kubernetes의 경우 [DaemonSet](#)을 사용하여 선택한 애플리케이션이 클러스터의 모든 노드에서 실행되도록 할 수 있습니다.

- 애플리케이션 실행 완료 — 일부 애플리케이션은 특정 작업을 완료하기 위해 실행되어야 합니다. 여기에는 월별 상태 보고서를 실행하거나 오래된 데이터를 정리하는 애플리케이션이 포함될 수 있습니다. [Job](#) 객체를 사용하여 애플리케이션을 시작 및 실행한 다음 작업이 완료되면 종료하도록 설정할 수 있습니다. [CronJob](#) 객체를 사용하면 Linux [crontab](#) 형식으로 정의된 구조를 사용하여 특정 시간, 분, 날짜 또는 요일에 애플리케이션이 실행되도록 설정할 수 있습니다.

네트워크에서 애플리케이션에 액세스할 수 있게 만들기

애플리케이션이 여러 위치로 이동하는 마이크로서비스 세트로 배포되는 경우가 많기 때문에 Kubernetes에는 이러한 마이크로서비스가 서로를 찾을 수 있는 방법이 필요했습니다. 또한 다른 사용자가 Kubernetes 클러스터 외부에서 애플리케이션에 액세스하려면 Kubernetes에는 해당 애플리케이션을 외부 주소 및 포트에 노출할 방법이 필요했습니다. 이러한 네트워킹 관련 기능은 각각 서비스 객체와 수신 객체를 통해 수행됩니다.

- 서비스 - 포드는 서로 다른 노드와 주소로 이동할 수 있기 때문에 첫 번째 포드와 통신해야 하는 다른 포드가 첫 번째 포드의 위치를 알아내기 어려울 수 있습니다. 이 문제를 해결하기 위해 Kubernetes에서는 애플리케이션을 [서비스](#)로 표현할 수 있습니다. 서비스를 사용하면 특정 이름으로 포드 또는 포드 세트를 식별한 다음, 포드에서 해당 애플리케이션의 서비스를 노출하는 포트와 다른 애플리케이션이 해당 서비스에 연결하기 위해 사용할 수 있는 포트를 지정할 수 있다. 클러스터 내의 다른 포드는 단순히 이름을 사용하여 서비스를 요청할 수 있으며 Kubernetes는 해당 요청을 해당 서비스를 실행하는 포드의 인스턴스에 대한 적절한 포트로 전달합니다.
- 수신 - [수신](#)은 Kubernetes 서비스로 표시되는 애플리케이션을 클러스터 외부의 클라이언트도 사용할 수 있게 해주는 역할을 합니다. 수신의 기본 기능에는 로드 밸런서(수신에서 관리), 수신 컨트롤러, 컨트롤러에서 서비스로 요청을 라우팅하는 규칙이 포함됩니다. Kubernetes에서 선택할 수 있는 [수신 컨트롤러](#)는 여러 개가 있습니다.

다음 단계

기본적인 Kubernetes 개념 및 Amazon EKS와의 관계를 이해하면 [Amazon EKS 설명서](#)와 [쿠버네티스 문서](#)에서 Amazon EKS 클러스터를 관리하고 해당 클러스터에 워크로드를 배포하는 데 필요한 정보를 쉽게 찾을 수 있습니다. Amazon EKS 사용을 시작하려면 다음 중에서 선택하세요.

- [the section called “클러스터 생성\(eksctl\)”](#)
- [the section called “클러스터 생성”](#)
- [the section called “샘플 배포\(Linux\)”](#)
- [클러스터 관리](#)

클라우드 및 온프레미스 환경 전반에 Amazon EKS 클러스터 배포

Amazon EKS 배포 옵션 이해

Amazon Elastic Kubernetes Service(Amazon EKS)는 클라우드 및 온프레미스 환경에서 Kubernetes를 원활하게 실행할 수 있는 완전 관리형 Kubernetes 서비스입니다.

클라우드에서 Amazon EKS는 Kubernetes 컨트롤 플레인 및 노드에 대한 Kubernetes 클러스터 인프라 관리를 자동화합니다. 이는 컨테이너 예약, 애플리케이션 가용성 관리, 리소스 동적 조정, 컴퓨팅 최적화, 클러스터 데이터 저장 및 기타 중요한 함수 수행에 필수적입니다. Amazon EKS를 사용하면 AWS 네트워킹, 보안, 스토리지 및 관찰성 서비스와의 기본 통합과 함께 AWS 인프라의 강력한 성능, 확장성, 안정성, 가용성을 얻을 수 있습니다.

온프레미스 환경에서 Kubernetes 실행을 간소화하기 위해 자체 인프라의 [the section called “노드”](#) 또는 [Amazon EKS Hybrid Nodes](#)에 동일한 Amazon EKS 클러스터, 기능, 도구를 사용하거나 독립형 에어 갭 환경에 [Amazon EKS Anywhere](#)를 사용할 수 있습니다.

클라우드의 Amazon EKS

AWS 리전, AWS 로컬 영역, AWS Wavelength 영역의 컴퓨팅과 함께 Amazon EKS를 사용할 수 있습니다. 클라우드에서 Amazon EKS를 사용하면 Kubernetes 컨트롤 플레인의 보안, 확장성, 가용성이 AWS 리전에서 AWS에 의해 완전히 관리됩니다. AWS 리전에서 컴퓨팅으로 애플리케이션을 실행할 때 클릭 한 번으로 AWS에서 컴퓨팅, 스토리지, 네트워킹을 위한 Kubernetes 클러스터 인프라 관리를 완전히 자동화하는 Amazon EKS Auto Mode를 포함하여 AWS 및 Amazon EKS의 모든 기능을 이용할 수 있습니다. AWS 로컬 영역 및 AWS Wavelength 영역에서 컴퓨팅을 사용하여 애플리케이션을 실행하는 경우 Amazon EKS 자체 관리형 노드를 사용하여 클러스터 컴퓨팅에 Amazon EC2 인스턴스를 연결하고 AWS 로컬 영역 및 AWS Wavelength 영역에서 제공되는 다른 AWS 서비스를 사용할 수 있습니다. 자세한 내용은 [AWS 로컬 영역 기능](#) 및 [AWS Wavelength 영역 기능](#)을 참조하세요.

	AWS 리전의 Amazon EKS	로컬/Wavelength 영역의 Amazon EKS
Kubernetes 컨트롤 플레인 관리	AWS 관리형	AWS 관리형
Kubernetes 제어 영역 위치	AWS 리전	AWS 리전
Kubernetes 데이터 플레인	Amazon EKS Auto Mode	Amazon EKS 관리형 노드 그룹(로컬 영역만 해당)

	AWS 리전의 Amazon EKS	로컬/Wavelength 영역의 Amazon EKS
	- Amazon EKS 관리형 노드 그룹 - Amazon EC2 자체 관리형 노드 - AWS Fargate	Amazon EC2 자체 관리형 노드
Kubernetes 데이터 영역 위치	AWS 리전	AWS 로컬 또는 Wavelength 영역

데이터 센터 또는 엣지 환경의 Amazon EKS

자체 데이터 센터 또는 엣지 환경에서 애플리케이션을 실행해야 하는 경우 [AWS Outposts의 Amazon EKS](#) 또는 [Amazon EKS Hybrid Nodes](#)를 사용할 수 있습니다. 클러스터 컴퓨팅을 위해 AWS Outposts의 Amazon EC2 인스턴스에서 자체 관리형 노드를 사용하거나 클러스터 컴퓨팅을 위해 자체 온프레미스 또는 엣지 인프라에서 Amazon EKS Hybrid Nodes를 사용할 수 있습니다. AWS Outposts는 데이터 센터 또는 코로케이션 시설에서 실행하는 AWS 관리형 인프라인 반면, Amazon EKS Hybrid Nodes는 온프레미스 또는 엣지 환경에서 관리하는 물리적 또는 가상 머신에서 실행됩니다. Amazon EKS on AWS Outposts 및 Amazon EKS Hybrid Nodes는 온프레미스 환경에서 AWS 리전으로 안정적으로 연결해야 하며 클라우드에서 애플리케이션을 실행하는 데 사용하는 것과 동일한 Amazon EKS 클러스터, 기능 및 도구를 사용할 수 있습니다. AWS Outposts에서 실행할 때 AWS Outposts의 Amazon EKS 로컬 클러스터를 사용하여 AWS Outposts의 전체 Kubernetes 클러스터를 배포할 수 있습니다.

	Amazon EKS Hybrid Nodes	AWS Outposts의 Amazon EKS
Kubernetes 컨트롤 플레인 관리	AWS 관리형	AWS 관리형
Kubernetes 제어 영역 위치	AWS 리전	AWS 리전 또는 AWS Outpost
Kubernetes 데이터 플레인	고객 관리형 물리적 또는 가상 머신	Amazon EC2 자체 관리형 노드
Kubernetes 데이터 영역 위치	고객 데이터 센터 또는 엣지 환경	고객 데이터 센터 또는 엣지 환경

에어 갭 환경을 위한 Amazon EKS Anywhere

[Amazon EKS Anywhere](#)는 온프레미스 및 엣지 환경에서 인프라 설정 및 Kubernetes 클러스터 수명 주기 작업과 같은 차별화되지 않은 과중한 작업을 자동화하여 Kubernetes 클러스터 관리를 간소화합니다. Amazon EKS와 달리 Amazon EKS Anywhere는 고객 관리형 제품이며 고객은 Amazon EKS Anywhere 클러스터의 클러스터 수명 주기 작업 및 유지 관리를 담당합니다. Amazon EKS Anywhere는 Kubernetes 하위 프로젝트 클러스터 API(CAPI)를 기반으로 하며 VMware vSphere, 베어 메탈, Nutanix, Apache CloudStack, AWS Snow를 포함한 다양한 인프라를 지원합니다. Amazon EKS Anywhere는 에어 갭 환경에서 실행할 수 있으며 관찰성 및 자격 증명 관리를 위해 리전의 AWS 서비스와 선택적으로 통합할 수 있습니다. Amazon EKS Anywhere에 대한 지원을 받고 AWS에서 제공하는 Kubernetes 추가 기능을 이용하기 위해 [Amazon EKS Anywhere Enterprise 구독](#)을 구매할 수 있습니다.

	Amazon EKS Anywhere
Kubernetes 컨트롤 플레인 관리	고객 관리형
Kubernetes 제어 영역 위치	고객 데이터 센터 또는 엣지 환경
Kubernetes 데이터 플레인	고객 관리형 물리적 또는 가상 머신
Kubernetes 데이터 영역 위치	고객 데이터 센터 또는 엣지 환경

Amazon EKS 도구

[Amazon EKS Connector](#)를 사용하여 적합한 Kubernetes 클러스터를 AWS에 등록 및 연결하고 Amazon EKS 콘솔에서 볼 수 있습니다. 클러스터가 연결되면 Amazon EKS 콘솔에서 해당 클러스터의 상태, 구성 및 워크로드를 확인할 수 있습니다. 이 기능을 사용하여 Amazon EKS 콘솔에서 연결된 클러스터를 볼 수 있지만 Amazon EKS Connector는 Amazon EKS 콘솔을 통해 연결된 클러스터에 대한 관리 또는 변경 작업을 활성화하지 않습니다.

[Amazon EKS Distro](#)는 모든 Amazon EKS 제품을 지원하는 기본 Kubernetes 구성 요소의 AWS 배포입니다. 여기에는 Kubernetes 컨트롤 플레인 구성 요소(etcd, kube-apiserver, kube-scheduler, kube-controller-manager) 및 네트워킹 구성 요소(CoreDNS, kube-proxy, CNI 플러그인)와 같이 작동 중인 Kubernetes 클러스터에 필요한 핵심 구성 요소가 포함됩니다. Amazon EKS Distro를 사용하여 원하는 도구로 Kubernetes 클러스터를 자체 관리할 수 있습니다. Amazon EKS Distro 배포는 AWS 지원 플랜에 포함되지 않습니다.

Amazon EKS를 사용하도록 설정

Amazon EKS 클러스터의 명령줄 관리를 준비하려면 여러 도구를 설치해야 합니다. 다음을 사용하여 자격 증명을 설정하고, 클러스터를 생성 및 수정하고, 클러스터가 실행 중일 때 클러스터로 작업할 수 있습니다.

- [AWS CLI 설정](#) - AWS를 사용하여 Amazon EKS 클러스터와 함께 작업하는 데 필요한 서비스를 설정하고 관리합니다. 특히 자격 증명을 구성하려면 AWS CLI가 필요하지만 다른 AWS 서비스에도 필요합니다.
- [kubectl 및 eksctl 설정](#) - eksctl CLI는 AWS와 상호 작용하여 Amazon EKS 클러스터를 생성, 수정, 삭제합니다. 클러스터가 가동되면 오픈 소스 kubectl 명령을 사용하여 Amazon EKS 클러스터 내에서 Kubernetes 객체를 관리하세요.
- 개발 환경 설정(선택사항) - 다음 도구 추가를 고려해 보세요.
 - 로컬 배포 도구 - Kubernetes를 처음 사용하는 경우 [minikube](#) 또는 [kind](#)와 같은 로컬 배포 도구를 설치해 보세요. 이러한 도구를 사용하면 애플리케이션 테스트를 위해 로컬 시스템에 Amazon EKS 클러스터를 보유할 수 있습니다.
 - 패키지 관리자 - [헬름](#)은 복잡한 패키지의 설치 및 관리를 단순화하는 데 널리 사용되는 Kubernetes용 패키지 관리자입니다. [헬름](#)을 사용하면 Amazon EKS 클러스터에서 AWS Load Balancer Controller와 같은 패키지를 더 쉽게 설치하고 관리할 수 있습니다.

다음 단계

- [AWS CLI 설정](#)
- [kubectl 및 eksctl 설정](#)
- [빠른 시작: 웹 앱 배포 및 데이터 저장](#)

AWS CLI 설정

[AWS CLI](#)는 Amazon EKS를 비롯한 AWS 서비스를 사용한 작업을 위한 명령줄 도구입니다. 로컬 시스템에서 Amazon EKS 클러스터 및 기타 AWS 리소스에 액세스할 수 있도록 IAM 사용자 또는 역할을 인증하는 데에도 사용됩니다. 명령줄에서 AWS의 리소스를 프로비저닝하려면 명령줄에서 사용할 AWS 액세스 키 ID와 비밀 키를 확보해야 합니다. 그런 다음, AWS CLI에서 이러한 보안 인증 정보를 구성해야 합니다. 아직 AWS CLI를 설치하지 않은 경우 AWS 사용 설명서에서 [최신 버전의 AWS CLI 설치 또는 업데이트](#)를 참조하세요.

액세스 키 생성

1. [AWS Management Console](#)에 로그인합니다.
2. 단일 사용자 또는 다중 사용자 계정의 경우:
 - 단일 사용자 계정 -: 오른쪽 상단에서 AWS 사용자 이름을 선택하여 탐색 메뉴를 엽니다. 예를 들어, **webadmin** 를 선택합니다.
 - 다중 사용자 계정 -: 서비스 목록에서 IAM을 선택합니다. IAM 대시보드에서 사용자를 선택하고 사용자 이름을 선택합니다.
3. 보안 자격 증명을 선택합니다.
4. 액세스 키에서 액세스 키 생성을 선택합니다.
5. 명령줄 인터페이스(CLI)를 선택하고 다음을 선택합니다.
6. 액세스 키 생성을 선택합니다.
7. .csv 파일 다운로드를 선택합니다.

AWS CLI를 구성하려면

AWS CLI 다운로드 후 다음 단계에 따라 구성합니다. 자세한 내용은 AWS 명령줄 인터페이스 사용 설명서에서 [AWS CLI 구성](#)을 참조하세요.

1. 터미널 창에 다음 명령을 입력합니다.

```
aws configure
```

선택적으로 명명된 프로파일(예: `--profile cluster-admin`)을 구성할 수 있습니다. AWS CLI에서 명명된 프로파일을 구성하는 경우 후속 명령에서 항상 이 플래그를 전달해야 합니다.

2. AWS 보안 인증을 입력합니다. 예시:

```
Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: region-code
Default output format [None]: json
```

보안 토큰을 받는 방법

필요한 경우 다음 명령을 실행하여 AWS CLI에 대한 새 보안 토큰을 가져옵니다. 자세한 내용은 AWS CLI 명령어 참조에서 [get-session-token](#)을 참조하세요.

기본적으로 토큰은 15분 동안 유효합니다. 기본 세션 제한 시간을 변경하려면 `--duration-seconds` 플래그를 전달합니다. 예시:

```
aws sts get-session-token --duration-seconds 3600
```

이 명령은 AWS CLI 세션의 임시 보안 인증 정보를 반환합니다. 다음과 같은 응답 출력이 표시되어야 합니다.

```
{
  "Credentials": {
    "AccessKeyId": "ASIA5FTRU3LOEXAMPLE",
    "SecretAccessKey": "JnKgwfqUD9mNsPoi9IbxAYEXAMPLE",
    "SessionToken": "VERYLONGSESSIONTOKENSTRING",
    "Expiration": "2023-02-17T03:14:24+00:00"
  }
}
```

사용자 ID를 확인하는 방법

필요한 경우, 다음 명령을 실행하여 터미널 세션에 대한 IAM 사용자 ID(예: *ClusterAdmin*)의 AWS 보안 인증 정보를 확인합니다.

```
aws sts get-caller-identity
```

이 명령은 AWSCLI에 대해 구성된 IAM 엔터티의 Amazon 리소스 이름(ARN)을 반환합니다. 다음 예제 응답 출력이 표시됩니다.

```
{
  "UserId": "AKIAIOSFODNN7EXAMPLE",
  "Account": "01234567890",
  "Arn": "arn:aws:iam::01234567890:user/ClusterAdmin"
}
```

다음 단계

- [kubectl 및 eksctl 설정](#)
- [빠른 시작: 웹 앱 배포 및 데이터 저장](#)

kubectl 및 eksctl 설정

AWS CLI가 설치되면 Kubernetes 클러스터를 생성하고 관리하기 위해 설치해야 하는 두 가지 다른 도구가 있습니다.

- **kubectl:** kubectl 명령줄 도구는 Kubernetes 클러스터 내의 리소스를 관리하는 데 사용할 기본 도구입니다. 이 페이지에서는 Kubernetes 클러스터의 버전과 일치하는 kubectl 바이너리를 다운로드하고 설정하는 방법을 설명합니다. [kubectl 설치 또는 업데이트](#)를 참조하세요.
- **eksctl:** eksctl 명령줄 도구는 AWS 클라우드 또는 온프레미스(EKS Anywhere 사용)에서 EKS 클러스터를 생성하고 해당 클러스터를 수정 및 삭제하기 위해 만들어졌습니다. [eksctl 설치](#)를 참조하세요.

kubectl 설치 또는 업데이트

이 주제는 장치에 kubectl 바이너리를 다운로드하고 설치하거나 업데이트하는 데 도움이 됩니다. 바이너리는 [업스트림 커뮤니티 버전](#)과 동일합니다. 바이너리가 Amazon EKS 또는 AWS에 고유하지 않은 경우, kubectl의 필요한 특정 버전을 구하려면 아래 단계를 사용합니다. 많은 빌더가 단순히 `brew install kubectl`을 실행해서 설치합니다.

Note

Amazon EKS 클러스터 제어 영역과 마이너 버전이 하나 다른 kubectl 버전을 사용해야 합니다. 예를 들어 1.31 kubectl 클라이언트는 Kubernetes 1.30, 1.31 및 1.32 클러스터에서 작업합니다.

1단계: kubectl가 설치되어 있는지 확인

디바이스에 이미 kubectl이 설치되어 있는지 확인합니다.

```
kubectl version --client
```

디바이스의 경로에 `kubectl`이(가) 설치되어 있는 경우, 출력 예제에는 다음과 유사한 정보가 포함되어 있습니다. 현재 설치한 버전을 최신 버전으로 업데이트하려면 다음 단계를 완료하고 현재 버전이 있는 동일한 위치에 새 버전을 설치해야 합니다.

```
Client Version: v1.31.X-eks-1234567
```

출력이 되지 않는다면 `kubectl`이 설치되지 않았거나 디바이스의 경로에 있는 위치에 설치되지 않은 것입니다.

2단계: 설치 또는 **kubectl** 업데이트

다음 운영 체제 중 하나에 `kubectl`을 설치하거나 업데이트합니다.

- [the section called “macOS”](#)
- [the section called “Linux\(AMD64\)”](#)
- [the section called “Linux\(arm64\)”](#)
- [the section called “Windows”](#)

Note

이 섹션에 사용된 AWS 리전에서 사용자의 AWS 리전으로 다운로드 속도가 느려지면 콘텐츠 앞에 CloudFront를 설정하는 것이 좋습니다. 자세한 내용은 [기본 CloudFront 배포 시작하기](#)를 참조하세요.

macOS

아래 단계에 따라 macOS에 `kubectl`을 설치합니다. 이 단계는 다음과 같습니다.

- 원하는 Kubernetes 버전의 바이너리 선택 및 다운로드
- 선택적으로 바이너리의 체크섬 확인
- 바이너리의 권한에 실행 권한을 추가
- 바이너리를 PATH의 폴더에 복사
- 선택적으로 바이너리의 디렉터리를 PATH에 추가

절차:

1. Amazon S3에서 클러스터의 Kubernetes 버전에 대한 바이너리를 다운로드합니다.

- Kubernetes 1.33

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.0/2025-05-01/bin/darwin/amd64/kubectl
```

- Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.3/2025-04-17/bin/darwin/amd64/kubectl
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.7/2025-04-17/bin/darwin/amd64/kubectl
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.11/2025-04-17/bin/darwin/amd64/kubectl
```

- Kubernetes 1.29

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-04-17/bin/darwin/amd64/kubectl
```

- Kubernetes 1.28

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-04-17/bin/darwin/amd64/kubectl
```

- Kubernetes 1.27

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2025-01-10/bin/darwin/amd64/kubectl
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/darwin/amd64/kubectl
```

2. (선택사항) 해당 바이너리의 SHA-256 체크섬을 사용하여 다운로드한 바이너리를 확인합니다.

a. 클러스터의 Kubernetes 버전에 대한 SHA-256 체크섬을 다운로드합니다.

- Kubernetes 1.33

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.0/2025-05-01/bin/darwin/amd64/kubectl.sha256
```

- Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.3/2025-04-17/bin/darwin/amd64/kubectl.sha256
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.7/2025-04-17/bin/darwin/amd64/kubectl.sha256
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.11/2025-04-17/bin/darwin/amd64/kubectl.sha256
```

- Kubernetes 1.29

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-04-17/bin/darwin/amd64/kubectl.sha256
```

- Kubernetes 1.28

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-04-17/bin/darwin/amd64/kubectl.sha256
```

- Kubernetes 1.27

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2025-01-10/bin/darwin/amd64/kubectl.sha256
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/
darwin/amd64/kubectl.sha256
```

b. 다운로드한 바이너리에 대해 SHA-256 체크섬을 확인합니다.

```
openssl sha1 -sha256 kubectl
```

c. 출력에 생성된 체크섬이 다운로드한 kubectl.sha256 파일의 체크섬과 일치하는지 확인합니다.

3. 바이너리에 실행 권한을 적용합니다.

```
chmod +x ./kubectl
```

4. 바이너리를 PATH의 폴더에 복사합니다. kubectl 버전이 이미 설치된 경우 \$HOME/bin/kubectl을 생성하고 \$HOME/bin이 \$PATH로 시작하도록 해야 합니다.

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

5. (선택사항) 셸 초기화 파일에 \$HOME/bin 경로를 추가하면 셸을 열 때 구성됩니다.

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bash_profile
```

Linux(AMD64)

아래 단계에 따라 Linux(amd64)에 kubectl을 설치합니다. 이 단계는 다음과 같습니다.

- 원하는 Kubernetes 버전의 바이너리 선택 및 다운로드
- 선택적으로 바이너리의 체크섬 확인
- 바이너리의 권한에 실행 권한을 추가
- 바이너리를 PATH의 폴더에 복사
- 선택적으로 바이너리의 디렉토리를 PATH에 추가

절차:

1. Amazon S3에서 클러스터의 Kubernetes 버전에 대한 kubectl 바이너리를 다운로드합니다.
 - Kubernetes 1.33

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.0/2025-05-01/bin/linux/amd64/kubectl
```

- Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.3/2025-04-17/bin/linux/amd64/kubectl
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.7/2025-04-17/bin/linux/amd64/kubectl
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.11/2025-04-17/bin/linux/amd64/kubectl
```

- Kubernetes 1.29

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-04-17/bin/linux/amd64/kubectl
```

- Kubernetes 1.28

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-04-17/bin/linux/amd64/kubectl
```

- Kubernetes 1.27

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/linux/amd64/kubectl
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/linux/amd64/kubectl
```

2. (선택사항) 해당 바이너리의 SHA-256 체크섬을 사용하여 다운로드한 바이너리를 확인합니다.

- a. 디바이스의 하드웨어 플랫폼용 명령을 사용하여 Amazon S3에서 클러스터의 Kubernetes 버전에 대한 SHA-256 체크섬을 다운로드합니다.

- Kubernetes 1.33

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.0/2025-05-01/bin/linux/amd64/kubect1.sha256
```

- Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.3/2025-04-17/bin/linux/amd64/kubect1.sha256
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.7/2025-04-17/bin/linux/amd64/kubect1.sha256
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.11/2025-04-17/bin/linux/amd64/kubect1.sha256
```

- Kubernetes 1.29

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-04-17/bin/linux/amd64/kubect1.sha256
```

- Kubernetes 1.28

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-04-17/bin/linux/amd64/kubect1.sha256
```

- Kubernetes 1.27

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/linux/amd64/kubect1.sha256
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/
linux/amd64/kubectl.sha256
```

- b. 다음 명령 중 하나를 사용하여 다운로드한 바이너리의 SHA-256 체크섬을 확인합니다.

```
sha256sum -c kubectl.sha256
```

or

```
openssl sha1 -sha256 kubectl
```

- c. 첫 번째의 경우 kubectl: OK가 표시되고 두 번째의 경우 출력에서 생성된 체크섬이 다운로드한 kubectl.sha256 파일의 체크섬과 일치하는지 확인할 수 있습니다.

3. 바이너리에 실행 권한을 적용합니다.

```
chmod +x ./kubectl
```

4. 바이너리를 PATH의 폴더에 복사합니다. kubectl 버전이 이미 설치된 경우 \$HOME/bin/kubectl을 생성하고 \$HOME/bin이 \$PATH로 시작하도록 해야 합니다.

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

5. (선택사항) 셸 초기화 파일에 \$HOME/bin 경로를 추가하면 셸을 열 때 구성됩니다.

Note

이 단계에는 Bash 셸을 사용한다고 가정합니다. 다른 셸을 사용하는 경우, 특정 셸 초기화 파일을 사용하도록 명령을 변경하세요.

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
```

Linux(arm64)

아래 단계에 따라 Linux(arm64)에 kubectl을 설치합니다. 이 단계는 다음과 같습니다.

- 원하는 Kubernetes 버전의 바이너리 선택 및 다운로드

- 선택적으로 바이너리의 체크섬 확인
- 바이너리의 권한에 실행 권한을 추가
- 바이너리를 PATH의 폴더에 복사
- 선택적으로 바이너리의 디렉터리를 PATH에 추가

절차:

1. Amazon S3에서 클러스터의 Kubernetes 버전에 대한 kubectl 바이너리를 다운로드합니다.
2. Amazon S3에서 클러스터의 Kubernetes 버전에 대한 kubectl 바이너리를 다운로드합니다.

- Kubernetes 1.33

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.0/2025-05-01/bin/linux/arm64/kubectl
```

- Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.3/2025-04-17/bin/linux/arm64/kubectl
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.7/2025-04-17/bin/linux/arm64/kubectl
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.11/2025-04-17/bin/linux/arm64/kubectl
```

- Kubernetes 1.29

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-04-17/bin/linux/arm64/kubectl
```

- Kubernetes 1.28

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-04-17/bin/linux/arm64/kubectl
```

- Kubernetes 1.27

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/linux/arm64/kubectl
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/linux/arm64/kubectl
```

3. (선택사항) 해당 바이너리의 SHA-256 체크섬을 사용하여 다운로드한 바이너리를 확인합니다.

- a. 디바이스의 하드웨어 플랫폼용 명령을 사용하여 Amazon S3에서 클러스터의 Kubernetes 버전에 대한 SHA-256 체크섬을 다운로드합니다.

- Kubernetes 1.33

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.0/2025-05-01/bin/linux/arm64/kubectl.sha256
```

- Kubernetes 1.32

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.3/2025-04-17/bin/linux/arm64/kubectl.sha256
```

- Kubernetes 1.31

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.7/2025-04-17/bin/linux/arm64/kubectl.sha256
```

- Kubernetes 1.30

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.11/2025-04-17/bin/linux/arm64/kubectl.sha256
```

- Kubernetes 1.29

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-04-17/bin/linux/arm64/kubectl.sha256
```

- Kubernetes 1.28


```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-04-17/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes 1.27

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/
linux/arm64/kubectl.sha256
```

- Kubernetes 1.26

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/
linux/arm64/kubectl.sha256
```

b. 다음 명령 중 하나를 사용하여 다운로드한 바이너리의 SHA-256 체크섬을 확인합니다.

```
sha256sum -c kubectl.sha256
```

or

```
openssl sha1 -sha256 kubectl
```

c. 첫 번째의 경우 kubectl: OK가 표시되고 두 번째의 경우 출력에서 생성된 체크섬이 다운로드한 kubectl.sha256 파일의 체크섬과 일치하는지 확인할 수 있습니다.

4. 바이너리에 실행 권한을 적용합니다.

```
chmod +x ./kubectl
```

5. 바이너리를 PATH의 폴더에 복사합니다. kubectl 버전이 이미 설치된 경우 \$HOME/bin/kubectl을 생성하고 \$HOME/bin이 \$PATH로 시작하도록 해야 합니다.

```
mkdir -p $HOME/bin && cp ./kubectl $HOME/bin/kubectl && export PATH=$HOME/bin:$PATH
```

6. (선택사항) 셸 초기화 파일에 \$HOME/bin 경로를 추가하면 셸을 열 때 구성됩니다.

Note

이 단계에는 Bash 셸을 사용한다고 가정합니다. 다른 셸을 사용하는 경우, 특정 셸 초기화 파일을 사용하도록 명령을 변경하세요.

```
echo 'export PATH=$HOME/bin:$PATH' >> ~/.bashrc
```

Windows

아래 단계에 따라 Windows에 kubectl을 설치합니다. 이 단계는 다음과 같습니다.

- 원하는 Kubernetes 버전의 바이너리 선택 및 다운로드
- 선택적으로 바이너리의 체크섬 확인
- 바이너리를 PATH의 폴더에 복사
- 선택적으로 바이너리의 디렉터리를 PATH에 추가

절차:

1. PowerShell 터미널을 엽니다.
2. Amazon S3에서 클러스터의 Kubernetes 버전에 대한 kubectl 바이너리를 다운로드합니다.
 - Kubernetes 1.33

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.0/2025-05-01/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.32

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.3/2025-04-17/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.31

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.7/2025-04-17/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.30

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.11/2025-04-17/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.29

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-04-17/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.28

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-04-17/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.27

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/windows/amd64/kubectl.exe
```

- Kubernetes 1.26

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/windows/amd64/kubectl.exe
```

3. (선택사항) 해당 바이너리의 SHA-256 체크섬을 사용하여 다운로드한 바이너리를 확인합니다.

a. 클러스터의 Windows용 Kubernetes 버전에 대한 SHA-256 체크섬을 다운로드합니다.

- Kubernetes 1.33

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.33.0/2025-05-01/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.32

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.32.3/2025-04-17/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.31

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.31.7/2025-04-17/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.30

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.30.11/2025-04-17/bin/windows/amd64/kubectl.exe.sha256
```

- Kubernetes 1.29

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.29.15/2025-04-17/bin/windows/amd64/kubect1.exe.sha256
```

- Kubernetes 1.28

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.28.15/2025-04-17/bin/windows/amd64/kubect1.exe.sha256
```

- Kubernetes 1.27

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.27.16/2024-12-12/bin/windows/amd64/kubect1.exe.sha256
```

- Kubernetes 1.26

```
curl.exe -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.26.15/2024-12-12/bin/windows/amd64/kubect1.exe.sha256
```

b. 다운로드한 바이너리에 대해 SHA-256 체크섬을 확인합니다.

```
Get-FileHash kubect1.exe
```

c. 출력에 생성된 체크섬이 다운로드한 kubect1.sha256 파일의 체크섬과 일치하는지 확인합니다. PowerShell 출력은 해당하는 대문자 문자열이어야 합니다.

4. 바이너리를 PATH의 폴더에 복사합니다. PATH에 명령줄 유틸리티에 사용하는 기존 디렉터리가 있으면 해당 디렉터리로 바이너리를 복사하세요. 그렇지 않은 경우 다음 단계를 완료합니다.

a. C:\bin과 같이, 명령줄 이진 파일용 새 디렉터를 생성합니다.

b. kubect1.exe 이진 파일을 새 디렉터리로 복사합니다.

c. 사용자 또는 시스템 PATH 환경 변수를 편집하여 PATH에 새 디렉터를 추가합니다.

d. PowerShell 터미널을 닫고 새 PATH 변수를 가져오기 위해 새 터미널을 엽니다.

5. kubect1을 설치한 이후 버전을 확인할 수 있습니다.

```
kubect1 version --client
```

6. kubect1을 처음 설치할 때 아직 어떤 서버와도 통신이 구성되지 않았습니다. 필요에 따라 다른 절차에서 이 구성을 설명합니다. 특정 클러스터와의 통신을 위해 구성을 업데이트해야 하는 경우 다

음 명령을 실행할 수 있습니다. **region-code**를 클러스터가 있는 AWS 리전으로 바꿉니다. **my-cluster**를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

7. 처음 몇 글자를 입력한 후 Tab 키를 사용하여 kubectl 하위 명령을 완료할 수 있는 자동 완성 구성을 고려합니다. 자세한 내용은 Kubernetes Documentation의 [Kubectl autocomplete](#)를 참조하세요.

eksctl 설치

eksctl CLI는 EKS 클러스터 작업에 사용됩니다. 이는 많은 개별 작업을 자동화합니다. eksctl 설치에 대한 지침은 eksctl 설명서의 [설치](#)를 참조하세요. Linux의 경우 UNIX 지침을 사용합니다.

eksctl을 사용하는 경우 사용 중인 IAM 보안 주체에 Amazon EKS IAM 역할, 서비스 연결 역할, AWS CloudFormation, VPC, 관련 리소스를 사용할 수 있는 권한이 있어야 합니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 사용](#)을 참조하세요. 이 가이드의 모든 단계를 동일한 사용자로 완료해야 합니다. 현재 사용자를 확인하려면 다음 명령을 실행합니다.

```
aws sts get-caller-identity
```

다음 단계

- [빠른 시작: 웹 앱 배포 및 데이터 저장](#)

빠른 시작: 웹 앱 배포 및 데이터 저장

게임 애플리케이션을 배포하고 해당 데이터를 Amazon EKS에 유지

이 빠른 시작 자습서에서는 [eksctl](#)을 사용하여 2048 게임 샘플 애플리케이션을 배포하고 Amazon EKS Auto Mode 클러스터에 데이터를 유지하는 단계를 보여줍니다. Amazon EKS Auto Mode는 클러스터 블록 스토리지, 네트워킹, 로드 밸런싱, 컴퓨팅 오토 스케일링을 위한 일상적인 작업을 자동화합니다.

진행하면서 클러스터 설정 프로세스를 안내합니다. Amazon EKS Auto Mode는 EC2 관리형 인스턴스를 사용한 노드 생성, Application Load Balancer 생성, EBS 볼륨 생성 작업을 자동화합니다.

전반적으로 AWS 서비스와 완전히 통합하는 데 필요한 사용자 지정 주석이 포함된 샘플 워크로드를 배포합니다.

이 튜토리얼에서는

다음 eksctl 클러스터 템플릿을 사용하여 자동 노드 프로비저닝을 위해 EKS Auto Mode로 클러스터를 빌드합니다.

VPC 구성 다음 eksctl 클러스터 템플릿을 사용하면 eksctl은 클러스터에 대한 IPv4 가상 프라이빗 클라우드(VPC)를 자동으로 생성합니다. 기본적으로 eksctl은 퍼블릭 엔드포인트와 프라이빗 엔드포인트를 모두 생성할 뿐 아니라 모든 네트워킹 요구 사항에 대응하는 VPC를 구성합니다.

인스턴스 관리 EKS Auto Mode는 Kubernetes 애플리케이션의 요구 사항에 따라 EKS 클러스터의 노드를 동적으로 추가하거나 제거합니다.

데이터 지속성 EKS Auto Mode의 블록 스토리지 기능을 사용하여 포드 재시작 또는 실패와 관련된 시나리오에서도 애플리케이션 데이터의 지속성을 보장합니다.

외부 앱 액세스 EKS Auto Mode의 로드 밸런싱 기능을 사용하여 Application Load Balancer(ALB)를 동적으로 프로비저닝합니다.

사전 조건

시작하기 전에 Amazon EKS를 사용하도록 다음과 같은 사전 조건을 설정해야 합니다.

- AWS CLI 설정 및 자격 증명 구성
- eksctl 설치

- kubectl 설치

자세한 내용은 [설정](#) 섹션을 참조하세요.

클러스터 구성

이 섹션에서는 동적 노드 프로비저닝을 위해 EKS Auto Mode를 사용하여 클러스터를 생성합니다.

cluster-config.yaml 파일을 생성하고 다음 내용을 붙여넣습니다. region-code를 유효한 리전 (예: us-east-1)으로 변경합니다.

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: web-quickstart
  region: <region-code>

autoModeConfig:
  enabled: true
```

이제 클러스터를 생성할 준비가 되었습니다.

Amazon EKS 클러스터를 생성합니다.

```
eksctl create cluster -f cluster-config.yaml
```

Important

eksctl을 사용하여 클러스터를 생성하지 않는 경우 VPC 서브넷에 수동으로 태그를 지정해야 합니다.

IngressClass 생성

EKS Auto Mode용 Kubernetes IngressClass를 생성합니다. IngressClass는 EKS Auto Mode가 수신 리소스를 처리하는 방법을 정의합니다. 이 단계에서는 EKS Auto Mode의 로드 밸런싱 기능을 구성합니다. 애플리케이션에 대한 수신 리소스를 생성할 때 EKS Auto Mode는 이 IngressClass를 사용하여

로드 밸런서를 자동으로 프로비저닝 및 관리하고 Kubernetes 애플리케이션을 AWS 로드 밸런싱 서비스와 통합합니다.

다음 yaml 파일을 ingressclass.yaml로 저장합니다.

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: alb
  annotations:
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  controller: eks.amazonaws.com/alb
```

IngressClass를 클러스터에 적용합니다.

```
kubectl apply -f ingressclass.yaml
```

2048 게임 샘플 애플리케이션 배포

이 섹션에서는 인기 있는 '2048 게임'을 클러스터 내 샘플 애플리케이션으로 배포하는 단계를 설명합니다. 제공된 매니페스트에는 Application Load Balancer(ALB)에 대한 사용자 지정 주석이 포함됩니다. 이러한 주석은 통합되어 들어오는 HTTP 트래픽을 '인터넷 경계'로 처리하고 대상 유형 'ip'를 사용하여 game-2048 네임스페이스의 적절한 서비스로 라우팅하도록 지시합니다.

Note

이 예제의 docker-2048 이미지는 x86_64 컨테이너 이미지이며 다른 아키텍처에서는 실행되지 않습니다.

1. --save-config 플래그를 사용하여 game-2048이라는 Kubernetes 네임스페이스를 생성합니다.

```
kubectl create namespace game-2048 --save-config
```

다음과 같은 응답 출력이 표시되어야 합니다.

```
namespace/game-2048 created
```


2. [2048 게임 샘플 애플리케이션](#) 배포

```
kubectl apply -n game-2048 -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.8.0/docs/examples/2048/2048_full.yaml
```

이 매니페스트는 game-2048 네임스페이스에 대한 Kubernetes 배포, 서비스, 수신을 설정하여 클러스터 내에서 game-2048 애플리케이션을 배포하고 노출하는 데 필요한 리소스를 생성합니다. 여기에는 포트 80의 배포를 노출하는 service-2048이라는 서비스 생성과 수신 HTTP 트래픽에 대한 라우팅 규칙과 인터넷 경계 Application Load Balancer(ALB)에 대한 주석을 정의하는 ingress-2048이라는 수신 리소스 생성이 포함되어 있습니다. 다음과 같은 응답 출력이 표시되어야 합니다.

```
namespace/game-2048 configured
deployment.apps/deployment-2048 created
service/service-2048 created
ingress.networking.k8s.io/ingress-2048 created
```

3. 다음 명령을 실행하여 game-2048 네임스페이스에 수신 리소스를 생성합니다.

```
kubectl get ingress -n game-2048
```

다음과 같은 응답 출력이 표시되어야 합니다.

NAME	CLASS	HOSTS	ADDRESS
		PORTS	AGE
ingress-2048	alb	*	k8s-game2048-ingress2-eb379a0f83-378466616.region-code.elb.amazonaws.com
		80	31s

다음 단계를 시작하기 전에 Application Load Balancer(ALB)가 프로비저닝될 때까지 몇 분 정도 기다려야 합니다.

4. 웹 브라우저를 열고 이전 단계의 ADDRESS를 입력하여 웹 애플리케이션에 액세스합니다. 예시:

```
k8s-game2048-ingress2-eb379a0f83-378466616.region-code.elb.amazonaws.com
```

브라우저에 2048 게임이 표시되어야 합니다. 게임을 하세요!

2048

SCORE

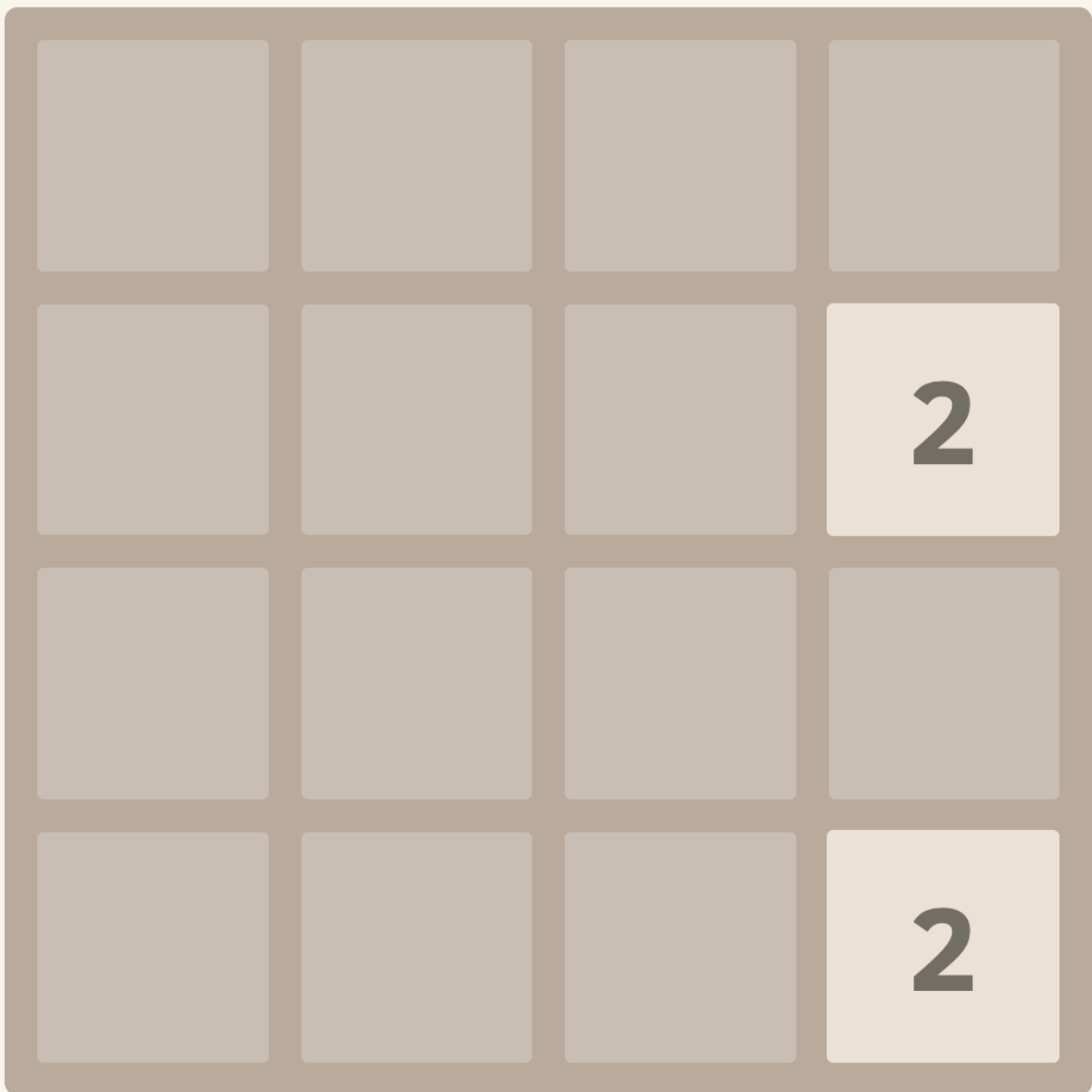
0

BEST

48

Join the numbers and get to the **2048** tile!

New Game



Amazon EKS Auto Mode를 사용하여 데이터 유지

이제 Amazon EKS 클러스터에서 2048 게임이 실행되고 있으므로 Amazon EKS Auto Mode의 블록 스토리지 기능을 사용하여 게임 데이터가 안전하게 유지되는지 확인해야 합니다.

1. storage-class.yaml라는 이름의 파일을 만듭니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: auto-ebs-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
```

2. StorageClass를 적용합니다.

```
kubectl apply -f storage-class.yaml
```

3. 게임 데이터에 대한 스토리지를 요청하려면 영구 볼륨 클레임(PVC)을 생성하세요. ebs-pvc.yaml이라는 파일을 생성하고 다음 내용을 추가합니다.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: game-data-pvc
  namespace: game-2048
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi
  storageClassName: auto-ebs-sc
```

4. 클러스터에 PVC를 적용:

```
kubectl apply -f ebs-pvc.yaml
```

다음과 같은 응답 출력이 표시되어야 합니다.

```
persistentvolumeclaim/game-data-pvc created
```

- 이제 이 PVC를 데이터를 저장하는 데 사용하려면 2048 게임 배포를 업데이트해야 합니다. 다음 배포는 게임 데이터를 저장하는 데 PVC를 사용하도록 구성됩니다. `ebs-deployment.yaml`이라는 파일을 생성하고 다음 내용을 추가합니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: game-2048
  name: deployment-2048
spec:
  replicas: 3 # Adjust the number of replicas as needed
  selector:
    matchLabels:
      app.kubernetes.io/name: app-2048
  template:
    metadata:
      labels:
        app.kubernetes.io/name: app-2048
    spec:
      containers:
        - name: app-2048
          image: public.ecr.aws/l6m2t8p7/docker-2048:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 80
          volumeMounts:
            - name: game-data
              mountPath: /var/lib/2048
      volumes:
        - name: game-data
          persistentVolumeClaim:
            claimName: game-data-pvc
```

- 업데이트된 배포를 적용합니다.

```
kubectl apply -f ebs-deployment.yaml
```

다음과 같은 응답 출력이 표시되어야 합니다.

```
deployment.apps/deployment-2048 configured
```

이러한 단계를 통해 이제 Amazon EKS Auto Mode의 블록 스토리지 기능을 사용하여 데이터를 유지하도록 클러스터의 2048 게임이 설정됩니다. 이렇게 하면 포드나 노드에 장애가 발생하더라도 게임 진행 상황과 데이터를 안전하게 보호할 수 있습니다.

이 자습서가 마음에 드셨다면 피드백을 통해 알려주세요. 이와 같은 사용 사례별 빠른 시작 자습서를 더 많이 제공할 수 있도록 하겠습니다.

클러스터와 노드 삭제

이 자습서용으로 생성한 클러스터의 사용을 끝낸 후에는 다음 명령으로 클러스터를 삭제하여 정리해야 합니다. 정리하기 전에 이 클러스터로 더 많은 작업을 수행하려면 다음 단계를 참조하세요.

```
eksctl delete cluster -f ./cluster-config.yaml
```

EKS는 클러스터가 삭제될 때 프로비저닝된 모든 노드를 자동으로 정리합니다.

예제를 통해 Amazon EKS 알아보기

개요

이 Amazon EKS 사용 설명서에는 [명령줄](#)이나 [AWS Management Console](#) 관리 콘솔에서 첫 번째 EKS 클러스터를 생성하기 위한 범용 절차와 모든 주요 Amazon EKS 구성 요소에 대한 확실한 참조가 포함되어 있습니다. 그러나 Amazon EKS 클러스터 관리자 또는 개발자는 이 사용 설명서의 외부 사이트에 있는 학습 경로를 따라가면 Amazon EKS에 대해 더 깊이 이해할 수 있습니다. 해당 사이트는 다음과 같은 도움을 줄 수 있습니다.

- 특정 유형의 클러스터를 설정합니다. 특정 클러스터 유형은 워크로드 유형이나 보안 요구 사항에 따라 달라질 수 있습니다. 예를 들어, 배치, 기계 학습 또는 컴퓨팅 집약적 워크로드를 실행하도록 클러스터를 튜닝하고 싶을 수 있습니다.
- 클러스터를 강화합니다. 클러스터에 고급 기능을 추가하여 관찰성, 유연한 스토리지, Auto Scaling 또는 특수 클러스터 네트워킹과 같은 기능을 제공할 수 있습니다.
- 업데이트를 자동화합니다. GitOps와 같은 기능을 사용하면 Git 리포지토리의 해당 구성 요소에 발생하는 변경 사항에 따라 클러스터 인프라와 워크로드를 자동으로 프로비저닝하도록 설정할 수 있습니다.
- 고급 클러스터 설정 도구를 사용합니다. eksctl은 클러스터를 빠르게 생성할 수 있는 방법을 제공하지만, 더 복잡한 클러스터를 더 쉽게 구성하고 업그레이드할 수 있는 다른 도구도 있습니다. 여기에는 [Terraform](#) 및 [CloudFormation](#)과 같은 도구가 포함됩니다.

Amazon EKS 학습 경로를 시작하려면 이 페이지에 설명된 일부 사이트를 방문하는 것이 좋습니다. 도중에 문제가 발생하는 경우 해당 문제를 해결하는 데 도움이 되는 리소스도 있습니다. 예를 들어, [Re:post 지식 센터](#)에서는 Amazon EKS 관련 지원 문제에 대한 지원 데이터베이스를 검색할 수 있습니다. 또한 [Amazon EKS 모범 사례 가이드](#)에서는 프로덕션 등급 클러스터를 설정하는 가장 좋은 방법에 대한 팁을 제공합니다.

Amazon EKS 워크샵

[Amazon EKS 워크숍](#)은 Kubernetes와 컨테이너에 대한 기본적인 이해부터 시작하여 클러스터 관리자에게 Amazon EKS의 중요한 기능을 안내하는 학습 플랫폼입니다. Amazon EKS 워크숍에 참여할 수 있는 방법은 다음과 같습니다.

- Amazon EKS 기본 사항: [소개](#) 페이지의 비디오를 시청하여 Amazon EKS가 AWS 클라우드에서 Kubernetes 기능을 구현하는 방법을 알아봅니다. Kubernetes에 대해 더 기본적으로 이해하려면 [Kubernetes란?](#) 비디오를 시청합니다.
- Amazon EKS 설정: AWS 계정이 있는 경우 [설정](#) 섹션을 사용하면 클러스터를 생성할 수 있는 CloudShell 환경을 설정할 수 있습니다. Amazon EKS 클러스터를 생성하기 위해 [eksctl](#)(간단한 클러스터 생성 명령줄) 및 [Terraform](#)(클러스터 생성에 대한 코드형 인프라(IaC) 접근 방식)을 선택할 수 있습니다.
- Amazon EKS 시작하기: [샘플 애플리케이션](#) 섹션에서 간단한 웹 스토어를 사용해 보세요. 다른 연습에서도 이를 사용할 수 있습니다. 이 섹션에서는 [컨테이너 이미지 패키징](#)과 Kubernetes 포드, 배포, 서비스, StatefulSet, 네임스페이스를 사용하여 마이크로서비스를 관리하는 방법에 대해서도 알아볼 수 있습니다. 그런 다음 Kustomize를 사용하여 Kubernetes 매니페스트에 변경 사항을 배포합니다.
- Amazon EKS 기초: 이 워크숍에서는 [AWS 로드 밸런서 컨트롤러](#)와 같은 AWS 기능을 사용하여 애플리케이션을 외부에 노출하는 방법을 보여줍니다. 스토리지의 경우 워크숍에서는 블록 스토리지에 [Amazon EBS](#), 파일 시스템 스토리지에 [Amazon EFS](#), Amazon FSx for NetApp ONTAP을 사용하여 AWS에서 ONTAP 파일 시스템을 관리하는 방법을 보여줍니다. 노드 관리의 경우 워크숍은 [관리형 노드 그룹](#)을 설정하는 데 도움이 됩니다.
- Amazon EKS 고급 기능: Amazon EKS 워크숍을 통해 제공되는 고급 기능에는 다음 설정을 위한 실습이 포함되어 있습니다.
 - Auto Scaling: 노드 Auto Scaling([Cluster Autoscaler](#) 또는 [Karpenter](#) 사용) 및 워크로드 Auto Scaling([Horizontal Pod Autoscaler](#) 및 [Cluster Proportional Autoscaler](#) 사용)을 포함합니다.
 - 관찰성: [관찰성 실습](#) 세트에서 [로깅](#), [OpenSearch](#), [Amazon EKS의 Container Insights](#), [Kubecost를 사용한 비용 가시성](#)에 대해 알아봅니다.
 - 보안: 이 [보안 실습](#) 세트에서는 [비밀 관리](#), [Amazon GuardDuty](#), [포드 보안 표준](#), [Kyverno 정책 관리](#)를 살펴볼 수 있습니다.
 - 네트워킹: [Amazon VPC CNI](#)(네트워크 플러그인 지원) 및 [Amazon VPC Lattice](#)(VC 및 사용자 계정 전반에 걸쳐 클러스터 구성용)가 포함된 [네트워킹](#) 실습에서 Amazon EKS의 네트워킹 기능을 알아봅니다.
 - 자동화: [자동화](#)에 대한 실습에서는 Amazon EKS 컨트롤 플레인 관리를 위한 [Kubernetes용 AWS 컨트롤러](#) 및 [Crossplane](#)과 같은 클러스터와 프로젝트를 관리하는 [GitOps](#) 방법을 단계별로 안내합니다.

Amazon EKS 실습 클러스터 설정 자습서

AWS 커뮤니티 사이트에 있는 일련의 [Amazon EKS 클러스터 설정 자습서](#)를 통해 특수 목적의 Amazon EKS 클러스터를 생성하고 다양한 방법으로 클러스터를 개선하는 데 도움을 받을 수 있습니다. 자습서는 세 가지 유형으로 나누어집니다.

클러스터 구축

이 자습서는 특별한 목적으로 사용할 수 있는 클러스터를 구축하는 데 도움이 됩니다. 특별한 목적에는 다음을 실행하는 기능이 포함됩니다.

- [IPv6 기반 전역적으로 확장 가능한 애플리케이션](#)
- [비동기 배치 작업](#)
- [트래픽이 많은 마이크로서비스](#)
- [Fargate에서 Karpenter를 사용한 Auto Scaling](#)
- [금융 워크로드](#)
- [Windows 관리형 노드 그룹](#)

클러스터 향상

기존 클러스터가 있으면 특수한 워크로드를 실행하거나 클러스터를 향상할 수 있는 방식으로 해당 클러스터를 확장하고 향상할 수 있습니다. 이 자습서에는 다음과 같은 방법이 포함되어 있습니다.

- [EFS CSI를 사용한 스토리지 솔루션 제공](#)
- [EBS CSI를 사용한 동적 데이터베이스 스토리지 제공](#)
- [AWS 로드 밸런서 컨트롤러를 사용한 IPv4 클러스터에 애플리케이션 노출](#)
- [AWS 로드 밸런서 컨트롤러를 사용한 IPv6 클러스터에 애플리케이션 노출](#)

AWS 서비스 최적화

이 자습서를 사용하면 클러스터를 AWS 서비스와 더 잘 통합할 수 있습니다. 이 자습서에는 다음과 같은 도움이 되는 내용이 포함되어 있습니다.

- [ExternalDNS를 사용한 마이크로서비스에 대한 DNS 레코드 관리](#)
- [CloudWatch를 사용한 애플리케이션 모니터링](#)
- [SQS 및 EFS 스토리지를 사용한 비동기 작업 관리](#)

- [워크로드에서 AWS Secrets Manager 비밀 사용](#)
- [Fargate, NGINX, ACM PCA를 사용한 mTLS 설정](#)

Amazon EFS 샘플

[Amazon 샘플](#) 리포지토리에는 Amazon EKS와 함께 사용할 매니페스트가 저장됩니다. 이러한 매니페스트를 통해 Amazon EKS에서 다양한 종류의 애플리케이션을 시험해 보거나 특정 유형의 Amazon EKS 클러스터를 생성할 수 있습니다. 샘플에는 다음을 위한 매니페스트가 포함됩니다.

- [AWS Amazon EKS Fargate 클러스터 생성](#)
- [기존 IAM 역할을 사용한 클러스터 생성](#)
- [클러스터에 Ubuntu 관리형 노드 그룹 추가](#)
- [볼륨 스냅샷을 사용한 포드 스토리지 백업 및 복원](#)
- [여러 계정을 사용하여 PVC로 마운트된 EBS 볼륨 복구](#)
- [Classic Load Balancer를 사용하여 NGINX 수신 컨트롤러에 대한 프록시 프로토콜 활성화](#)
- [Fargate에서 AWS OpenSearch에 대한 로깅 구성](#)
- [웹 페더레이션 ID 공급자를 사용한 Python SDK 실행](#)
- [NFS CSI 컨트롤러에 샘플 앱 배포](#)
- [StatefulSet에 볼륨 스냅샷 사용](#)
- [다양한 가용 영역의 노드 전체에 포드 배포](#)

이러한 샘플은 학습 및 테스트 목적으로만 사용되며 프로덕션에 사용하기 위한 것이 아니라는 점에 유의하세요.

AWS 자습서

[AWS 자습서](#) 사이트는 몇 가지 Amazon EKS 자습서를 게시하지만 AWS 사이트(예: AWS 커뮤니티 사이트)에 게시된 다른 자습서를 찾을 수 있는 검색 도구도 제공합니다. 이 사이트에 직접 게시된 Amazon EKS 자습서에는 다음이 포함되어 있습니다.

- [Amazon EKS에 컨테이너 웹 앱 배포](#)
- [더 적은 비용으로 Kubernetes 클러스터 실행\(Amazon EKS 및 스팟 인스턴스\)](#)
- [Kubernetes에서 Jenkins 작업 비용을 최적화하는 방법](#)

개발자 워크숍

Amazon EKS에서 실행할 애플리케이션을 생성하거나 리팩터링하려는 소프트웨어 개발자라면 [Amazon EKS 개발자 워크숍](#)을 시작하는 것이 좋습니다. 이 워크숍은 컨테이너화된 애플리케이션을 구축하는 데 도움이 될 뿐만 아니라 해당 컨테이너를 컨테이너 레지스트리([ECR](#))에 배포하고 컨테이너 레지스트리에서 Amazon EKS 클러스터로 배포하는 데도 도움이 됩니다.

[Amazon EKS Python 워크숍](#)으로 시작하여 Python 애플리케이션 리팩터링 프로세스를 거친 다음 개발 환경을 설정하여 애플리케이션 배포를 준비합니다. 컨테이너, Kubernetes, Amazon EKS에 대한 섹션을 단계별로 진행하여 해당 환경에서 컨테이너화된 애플리케이션을 실행할 준비를 하세요.

Terraform 워크숍

eksctl은 클러스터를 생성하기 위한 간단한 도구이지만, 보다 복잡한 유형의 Amazon EKS 배포의 경우에는 [Terraform](#)이 널리 사용되는 Amazon EKS 클러스터 생성 및 관리 도구입니다. [Terraform Amazon EKS 워크숍](#)에서는 Terraform을 사용하여 AWS VPC를 구축하고, Amazon EKS 클러스터를 생성하고, 클러스터에 선택적 개선 사항을 추가하는 방법을 설명합니다. 특히 [프라이빗 Amazon EKS 클러스터](#)를 생성하는 섹션이 있습니다.

AWS Amazon EKS 교육

AWS는 Amazon EKS에 대해 알아보기 위한 공식 교육을 제공합니다. [Running Containers on Amazon Elastic Kubernetes Service](#)라는 제목의 3일 교육 과정에서는 다음을 설명합니다.

- Kubernetes 및 Amazon EKS 기초
- Amazon EKS 클러스터를 구축하는 방법
- AWS IAM 및 Kubernetes RBAC 인증을 사용한 Amazon EKS 보안
- GitOps 자동화 도구
- 모니터링 도구
- 비용, 효율성, 복원력을 개선하는 기술

Amazon EKS 시작하기

시작 안내서를 검토하기 전에 Amazon EKS를 사용하도록 설정되었는지 확인합니다. 자세한 내용은 [설정](#) 단원을 참조하십시오.

Amazon EKS에서 노드로 새 Kubernetes 클러스터를 생성할 때 사용할 수 있는 시작 가이드는 다음 두 가지입니다.

- [Amazon EKS 시작하기 - eksctl](#) - 이 시작 가이드는 Amazon EKS에서 Kubernetes 클러스터를 생성하고 관리하기 위한 간단한 명령줄 유틸리티인 eksctl을 사용하여 Amazon EKS를 시작할 때 필요한 모든 리소스를 설치하는 데 도움이 됩니다. 자습서의 마지막에서는 애플리케이션을 배포할 수 있는 Amazon EKS에 클러스터를 실행하게 됩니다. 이것이 Amazon EKS를 시작하는 가장 빠르고 쉬운 방법입니다.
- [Amazon EKS - AWS Management Console 및 AWS CLI 시작하기](#) - 이 시작 가이드는 AWS Management Console 및 AWS를 사용하여 Amazon EKS를 시작할 때 필요한 모든 리소스를 만드는 데 도움이 됩니다. 자습서의 마지막에서는 애플리케이션을 배포할 수 있는 Amazon EKS에 클러스터를 실행하게 됩니다. 이 가이드에서는 Amazon EKS 클러스터에 필요한 리소스를 수동으로 생성합니다. 이 절차를 통해 각 리소스가 만들어지는 방식과 리소스가 상호 작용하는 방식을 파악할 수 있습니다.

또한 다음과 같은 참고 자료를 제공합니다.

- 실습 자습서 모음은 AWS Community의 [EKS Cluster Setup](#)을 참조하세요.
- 코드 예제는 [AWS SDK를 사용하는 Amazon EKS의 코드 예제](#)를 참조하세요.

Amazon EKS 시작하기-EKS Auto Mode

다른 EKS 시작하기 경험과 마찬가지로 EKS Auto Mode를 사용하여 첫 번째 클러스터를 생성하면 클러스터 자체의 관리가 AWS로 위임됩니다. 그러나 EKS Auto Mode는 워크로드 인프라(노드, 네트워크, 다양한 서비스)를 설정하는 데 필요한 많은 필수 서비스 관련 책임을 처리하여 EKS 자동화를 확장하므로 노드를 더 쉽게 관리하고 워크로드 수요에 맞춰 스케일 업할 수 있습니다.

다음 방법 중 하나를 선택하고 EKS Auto Mode를 사용하여 클러스터를 생성합니다.

- [the section called “AWS CLI”](#): aws 명령줄 인터페이스를 사용하여 클러스터를 생성합니다.
- [the section called “관리 콘솔”](#): AWS Management Console을 사용하여 클러스터를 생성합니다.

- [the section called “eksctl CLI”](#): eksctl 명령줄 인터페이스를 사용하여 클러스터를 생성합니다.

첫 번째 EKS 클러스터를 생성하는 다양한 접근 방식을 비교하는 경우 EKS Auto Mode에서 구성 요소를 다음과 같이 설정하는 등의 추가 클러스터 관리 책임을 AWS가 담당하도록 설정한다는 점을 알아야 합니다.

- 워크로드 수요가 증가 및 감소함에 따라 노드를 시작하고 확장합니다.
- 클러스터 자체(컨트롤 플레인), 노드 운영 체제, 노드에서 실행되는 서비스를 정기적으로 업그레이드합니다.
- 노드 스토리지의 크기 및 속도, 포드 네트워크 구성 등을 결정하는 기본 설정을 선택합니다.

EKS Auto Mode 클러스터에서 얻는 것에 대한 자세한 내용은 [EKS Auto Mode](#) 섹션을 참조하세요.

Amazon EKS 시작하기 – eksctl

Note

이 주제에서는 EKS Auto Mode 없이 시작하는 방법을 설명합니다.
EKS Auto Mode는 클러스터 컴퓨팅, 스토리지, 네트워킹의 일상적인 작업을 자동화합니다.
[Amazon EKS Auto Mode를 시작하는 방법을 자세히 알아봅니다.](#)

이 가이드는 Amazon EKS에서 Kubernetes 클러스터를 생성 및 관리하기 위한 간단한 명령줄 유틸리티인 eksctl을 사용하여 Amazon Elastic Kubernetes Service(Amazon EKS)를 시작할 때 필요한 모든 리소스를 설치하는 데 도움이 됩니다. 이 자습서의 마지막에서는 애플리케이션을 배포할 수 있는 Amazon EKS에 클러스터를 실행하게 됩니다.

이 가이드의 절차에서는 AWS Management Console을 사용하여 클러스터를 생성할 경우 수동으로 생성해야 하는 여러 리소스를 자동으로 생성합니다. 대부분의 리소스를 수동으로 만들고 리소스가 상호 작용하는 방식을 더 잘 이해하려면 AWS Management Console을 사용하여 클러스터 및 컴퓨팅을 생성합니다. 자세한 내용은 [the section called “클러스터 생성\(콘솔 및 CLI\)”](#) 섹션을 참조하세요.

사전 조건

이 자습서를 시작하기 전에 [Amazon EKS 사용을 위한 설정](#)에 설명된 대로 AWS CLI, kubectl 및 eksctl 도구를 설치하고 구성해야 합니다.

1단계: Amazon EKS 클러스터 및 노드 생성

Important

이 주제에는 가능한 한 간단하고 빠르게 시작하기 위해 기본 설정으로 클러스터와 노드를 생성하는 단계가 포함되어 있습니다. 프로덕션 용도로 클러스터 및 노드를 생성하기 전에 모든 설정을 숙지하고 요구 사항을 충족하는 설정으로 클러스터와 노드를 배포하는 것이 좋습니다. 자세한 내용은 [the section called “클러스터 생성”](#) 및 [컴퓨팅 관리](#) 섹션을 참조하세요. 일부 설정은 클러스터와 노드를 생성할 때만 사용 설정할 수 있습니다.

다음 노드 유형 중 하나를 사용하여 클러스터를 생성할 수 있습니다. 각 유형에 대한 자세한 내용은 [컴퓨팅 관리](#) 부분을 참조하세요. 클러스터를 배포한 후에 다른 노드 유형을 추가할 수 있습니다.

- Fargate – Linux - [the section called “AWS Fargate”](#)에서 Linux 애플리케이션을 실행하려는 경우 이 노드 유형을 선택합니다. Fargate는 Amazon EC2 인스턴스를 관리하지 않고도 Kubernetes 포드를 배포할 수 있는 서버리스 컴퓨팅 엔진입니다.
- 관리형 노드 — Linux - Amazon EC2 인스턴스에서 Amazon Linux 애플리케이션을 실행하려면 이 노드 유형을 선택합니다. 이 가이드에서는 다루지 않지만 [Windows 자체 관리형](#) 및 [Bottlerocket](#) 노드를 클러스터에 추가할 수도 있습니다.

다음 명령을 사용하여 Amazon EKS 클러스터를 생성합니다. *my-cluster*를 원하는 값으로 바꿀 수 있습니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다. *region-code*를 Amazon EKS에서 지원하는 AWS 리전으로 바꿉니다. AWS 리전 목록은 AWS General Reference 가이드의 [Amazon EKS endpoints and quotas](#)를 참조하세요.

Example

Fargate - Linux

```
eksctl create cluster --name my-cluster --region region-code --fargate
```

Managed nodes - Linux

```
eksctl create cluster --name my-cluster --region region-code
```

클러스터 생성에 몇 분 정도 걸립니다. 생성하는 동안 여러 줄의 출력이 표시됩니다. 출력의 마지막 줄은 다음 예제와 유사합니다.

```
[...]
[#] EKS cluster "my-cluster" in "region-code" region is ready
```

eksctl이 ~/.kube/config에 kubectl 구성 파일을 생성했거나 컴퓨터의 ~/.kube/config 기존 파일에 새 클러스터의 구성을 추가했습니다.

클러스터 생성이 완료된 후, AWS CloudFormation [콘솔](#)에서 eksctl-*my-cluster*-cluster라는 이름의 AWS CloudFormation 스택을 확인하여 생성된 모든 리소스를 확인합니다.

2단계: Kubernetes 리소스 보기

1. 클러스터 노드를 확인합니다.

```
kubectl get nodes -o wide
```

예제 출력은 다음과 같습니다.

Example

Fargate - Linux

NAME	STATUS	ROLES	AGE
VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE
CONTAINER-RUNTIME	KERNEL-VERSION		
fargate-ip-192-0-2-0.region-code.compute.internal	Ready	<none>	
8m3s v1.2.3-eks-1234567 192.0.2.0 <none>	Amazon Linux 2		
1.23.456-789.012.amzn2.x86_64 containerd://1.2.3			
fargate-ip-192-0-2-1.region-code.compute.internal	Ready	<none>	
7m30s v1.2.3-eks-1234567 192-0-2-1 <none>	Amazon Linux 2		
1.23.456-789.012.amzn2.x86_64 containerd://1.2.3			

Managed nodes - Linux

NAME	STATUS	ROLES	AGE	VERSION
INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	
CONTAINER-RUNTIME				

```
ip-192-0-2-0.region-code.compute.internal    Ready    <none>    6m7s
v1.2.3-eks-1234567    192.0.2.0    192.0.2.2    Amazon Linux 2
1.23.456-789.012.amzn2.x86_64    containerd://1.2.3
ip-192-0-2-1.region-code.compute.internal    Ready    <none>    6m4s
v1.2.3-eks-1234567    192.0.2.1    192.0.2.3    Amazon Linux 2
1.23.456-789.012.amzn2.x86_64    containerd://1.2.3
```

출력에 표시되는 항목에 대한 자세한 내용은 [the section called “클러스터 리소스 액세스”](#) 섹션을 참조하세요.

2. 클러스터에서 실행 중인 워크로드를 확인합니다.

```
kubectl get pods -A -o wide
```

예제 출력은 다음과 같습니다.

Example

Fargate - Linux

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP
	NODE					
kube-system	coredns-1234567890-abcde	1/1	Running	0	18m	
192.0.2.0	fargate-ip-192-0-2-0.region-code.compute.internal					<none>
kube-system	coredns-1234567890-12345	1/1	Running	0	18m	
192.0.2.1	fargate-ip-192-0-2-1.region-code.compute.internal					<none>

Managed nodes - Linux

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP
	NODE					
kube-system	aws-node-12345	1/1	Running	0	7m43s	
192.0.2.1	ip-192-0-2-1.region-code.compute.internal					<none>
kube-system	aws-node-67890	1/1	Running	0	7m46s	
192.0.2.0	ip-192-0-2-0.region-code.compute.internal					<none>
kube-system	coredns-1234567890-abcde	1/1	Running	0	14m	
192.0.2.3	ip-192-0-2-3.region-code.compute.internal					<none>

kube-system	coredns-1234567890-12345	1/1	Running	0	14m
192.0.2.4	ip-192-0-2-4.region-code.compute.internal		<none>		<none>
kube-system	kube-proxy-12345	1/1	Running	0	7m46s
192.0.2.0	ip-192-0-2-0.region-code.compute.internal		<none>		<none>
kube-system	kube-proxy-67890	1/1	Running	0	7m43s
192.0.2.1	ip-192-0-2-1.region-code.compute.internal		<none>		<none>

출력에 표시되는 항목에 대한 자세한 내용은 [the section called “클러스터 리소스 액세스”](#) 섹션을 참조하세요.

3단계: 클러스터 및 노드 삭제

이 자습서용으로 생성한 클러스터와 노드의 사용을 끝낸 후에는 다음 명령으로 클러스터와 노드를 삭제하여 정리해야 합니다. 정리하기 전에 이 클러스터로 추가 작업을 수행하려면 [the section called “다음 단계”](#) 섹션을 참조하세요.

```
eksctl delete cluster --name my-cluster --region region-code
```

다음 단계

다음은 클러스터의 기능을 확장하는 데 도움이 되는 설명서 주제입니다.

- 클러스터에 [샘플 애플리케이션](#)을 배포합니다.
- 클러스터를 생성한 [IAM 위탁자](#)는 kubectl 또는 AWS Management Console을 사용하여 Kubernetes API 서버를 직접적으로 호출할 수 있는 유일한 위탁자입니다. 다른 IAM 보안 주체가 클러스터에 액세스할 수 있게 하려면 해당 보안 주체를 추가해야 합니다. 자세한 내용은 [the section called “Kubernetes API 액세스”](#) 및 [the section called “필수 권한”](#) 섹션을 참조하세요.
- 프로덕션 용도로 클러스터를 배포하기 전에 [클러스터](#)와 [노드](#)에 대한 모든 설정을 숙지하는 것이 좋습니다. Amazon EC2 노드에 대한 SSH 액세스 사용 설정과 같은 일부 설정은 클러스터를 생성할 때 지정해야 합니다.
- 클러스터의 보안을 강화하려면 [서비스 계정에 IAM 역할을 사용하도록 Amazon VPC 컨테이너 네트워킹 인터페이스 플러그 인을 구성합니다](#).

Amazon EKS 시작하기 - AWS Management Console 및 AWS CLI

Note

이 주제에서는 EKS Auto Mode 없이 시작하는 방법을 설명합니다. 관리형 노드 그룹을 사용하여 노드를 배포합니다.

EKS Auto Mode는 클러스터 컴퓨팅, 스토리지, 네트워킹의 일상적인 작업을 자동화합니다.

[Amazon EKS Auto Mode를 시작하는 방법을 자세히 알아봅니다.](#) EKS 자율 모드는 노드를 배포하는 기본적인 방법입니다.

이 가이드는 AWS Management Console 및 AWS CLI를 사용하여 Amazon Elastic Kubernetes Service(Amazon EKS)를 시작할 때 필요한 모든 리소스를 생성하는 데 도움이 됩니다. 이 가이드에서는 수동으로 각 리소스를 만듭니다. 이 튜토리얼의 마지막에서는 애플리케이션을 배포할 수 있는 Amazon EKS에 클러스터를 실행하게 됩니다.

이 안내서의 절차를 통해 각 리소스가 만들어지는 방식과 리소스가 상호 작용하는 방식을 완벽하게 파악할 수 있습니다. 대부분의 리소스를 자동으로 생성하려는 경우 eksctl CLI를 사용하여 클러스터 및 노드를 생성합니다. 자세한 내용은 [the section called “클러스터 생성\(eksctl\)”](#) 섹션을 참조하세요.

사전 조건

이 튜토리얼에서는 Amazon EKS 클러스터를 생성하고 관리할 때 필요한 다음 도구 및 리소스를 설치하고 구성해야 합니다.

- **AWS CLI** - Amazon EKS를 비롯한 AWS 서비스를 사용한 작업을 위한 명령줄 도구입니다. 자세한 내용은 AWS 명령줄 인터페이스 사용 설명서에서 [설치하기](#)를 참조하세요. AWS CLI 설치 후, 구성 작업도 수행하는 것이 좋습니다. 자세한 내용은 AWS 명령줄 인터페이스 사용 설명서에서 [aws config를 사용한 빠른 구성](#)을 참조하세요. 이 페이지에 나와 있는 update-kubeconfig 옵션을 사용하려면 AWS CLI v2가 필요합니다.
- **kubectl** - Kubernetes 클러스터 작업을 위한 명령줄 도구입니다. 자세한 내용은 [the section called “kubectl 및 eksctl 설정”](#) 섹션을 참조하세요.
- **필요한 IAM 권한** - 사용하는 IAM 보안 주체에 Amazon EKS IAM 역할, 서비스 연결 역할, AWS CloudFormation, VPC 및 관련 리소스를 사용할 수 있는 권한이 있어야 합니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 사용](#)을 참조하세요. 이 가이드의 모든 단계를 동일한 사용자로 완료해야 합니다. 현재 사용자를 확인하려면 다음 명령을 실행합니다.

```
aws sts get-caller-identity
```

Bash 셸에서 이 주제의 단계를 완료하는 것이 좋습니다. Bash 셸을 사용하지 않는 경우 줄 연속 문자 및 변수 설정 및 사용 방식과 같은 일부 스크립트 명령을 통해 셸이 조정되어야 합니다. 또한 셸의 인용 및 이스케이프 규칙이 다를 수 있습니다. 자세한 내용은 AWS 명령줄 인터페이스 사용 설명서의 [AWS CLI에서 문자열에 따옴표 사용하기](#)를 참조하세요.

1단계: Amazon EKS 클러스터 생성

Important

이 주제에는 가능한 한 간단하고 빠르게 시작하기 위해 기본 설정으로 클러스터를 생성하는 단계가 포함되어 있습니다. 프로덕션 용도로 클러스터를 생성하기 전에 모든 설정을 숙지하고 요구 사항을 충족하는 설정으로 클러스터를 배포하는 것이 좋습니다. 자세한 내용은 [the section called “클러스터 생성”](#) 섹션을 참조하세요. 일부 설정은 클러스터를 생성할 때만 사용 설정할 수 있습니다.

1. Amazon EKS 요구 사항을 충족하는 퍼블릭 및 프라이빗 서브넷이 있는 Amazon VPC PC를 생성합니다. *region-code*를 Amazon EKS에서 지원하는 AWS 리전으로 바꿉니다. AWS 리전 목록은 AWS General Reference 가이드의 [Amazon EKS endpoints and quotas](#)를 참조하세요. 선택하는 이름으로 *my-eks-vpc-stack*을 바꿀 수 있습니다.

```
aws cloudformation create-stack \  
  --region region-code \  
  --stack-name my-eks-vpc-stack \  
  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/  
cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml
```

Tip

이전 명령이 생성하는 모든 리소스 목록을 보려면 AWS CloudFormation 콘솔(<https://console.aws.amazon.com/cloudformation/>)을 엽니다. *my-eks-vpc-stack* 스택을 선택한 다음 리소스(Resources) 탭을 선택합니다.

2. 클러스터 IAM 역할을 생성하고 필요한 Amazon EKS IAM 관리형 정책을 연결합니다. Amazon EKS에서 관리하는 Kubernetes 클러스터는 사용자 대신 다른 AWS 서비스를 직접 호출하여 서비스에 사용하는 리소스를 관리합니다.

- a. 다음 콘텐츠를 *eks-cluster-role-trust-policy.json*이라는 파일에 복사합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- b. 역할을 생성합니다.

```
aws iam create-role \
  --role-name myAmazonEKSClusterRole \
  --assume-role-policy-document file:///eks-cluster-role-trust-policy.json
```

- c. 필요한 Amazon EKS 관리형 IAM 정책을 역할에 연결합니다.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy \
  --role-name myAmazonEKSClusterRole
```

3. <https://console.aws.amazon.com/eks/home#/clusters>에서 Amazon EKS 콘솔을 엽니다.

콘솔의 오른쪽 상단에 표시된 AWS 리전이 클러스터를 생성하려는 AWS 리전인지 확인합니다. 그렇지 않은 경우 AWS 리전 이름 옆에 있는 드롭다운을 선택하고 사용할 AWS 리전을 선택합니다.

4. 클러스터 생성을 선택합니다. 이 옵션이 표시되지 않으면 먼저 왼쪽 검색 창에서 클러스터를 선택합니다.
5. 클러스터 구성 페이지에서 다음을 수행합니다.
 - a. 사용자 지정 구성을 선택하고 EKS 자율 모드 사용을 비활성화하세요. (EKS 자율 모드 클러스터를 선호하는 경우 대신 [the section called “관리 콘솔”](#) 섹션을 참조하세요.)

- b. 클러스터 이름을 입력하세요(예: *my-cluster*). 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다.
 - c. 클러스터 서비스 역할에서 *myAmazonEKSClusterRole*을 선택합니다.
 - d. 나머지 설정을 기본값으로 두고 다음을 선택합니다.
6. 네트워킹 지정 페이지에서 다음을 수행합니다.
- a. VPC 드롭다운 목록에서 이전 단계에서 생성한 VPC의 ID를 선택합니다. VPC는 ** / my-eks-vpc-stack-VPC*와 같이 표시됩니다.
 - b. 서브넷 드롭다운 목록에서 이전 단계에서 생성된 서브넷을 선택합니다. 서브넷은 ** / my-eks-vpc-stack-**와 같이 표시됩니다.
 - c. 추가 보안 그룹 드롭다운 목록에서 이전 단계에서 생성한 보안 그룹을 선택합니다. 보안 그룹은 ** / my-eks-vpc-stack-ControlPlaneSecurityGroup-**와 같이 표시됩니다.
 - d. 나머지 설정을 기본값으로 두고 다음을 선택합니다.
7. 관찰성 구성 페이지에서 다음을 선택합니다.
8. 추가 기능 선택 페이지에서 다음을 선택합니다.

추가 기능에 대한 자세한 내용은 [the section called “Amazon EKS 추가 기능”](#) 섹션을 참조하세요.

- 9. 선택한 추가 기능 설정 구성 페이지에서 다음을 선택합니다.
- 10. 검토 및 생성 페이지에서 생성을 선택합니다.

클러스터 이름 오른쪽에 있는 클러스터 상태는 클러스터 프로비저닝 프로세스가 완료될 때까지 몇 분 동안 생성 중(Creating)으로 표시됩니다. 상태가 활성이 되면 다음 단계를 진행합니다.

Note

요청의 가용 영역 중 하나에 Amazon EKS 클러스터를 생성하는 데 충분한 용량이 없다는 오류가 표시될 수 있습니다. 이 경우 오류 출력에는 새 클러스터를 지원할 수 있는 가용 영역이 포함됩니다. 사용자 계정의 지원 가용 영역에 있는 2개 이상의 서브넷을 사용하여 클러스터를 다시 생성합니다. 자세한 내용은 [the section called “용량 부족”](#) 섹션을 참조하세요.

2단계: 클러스터와 통신하도록 컴퓨터 구성

이 부분에서는 클러스터에 대해 kubeconfig 파일을 생성합니다. 이 파일의 설정을 사용하면 kubectl CLI를 사용하여 클러스터와 통신할 수 있습니다.

진행하기 전에 1단계에서 클러스터 생성이 성공적으로 완료되었는지 확인합니다.

1. 클러스터에 대해 kubeconfig 파일을 생성 또는 업데이트합니다. *region-code*를 클러스터를 생성한 AWS 리전으로 바꿉니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

기본적으로 config 파일이 ~/.kube에 생성되거나 새 클러스터의 구성이 ~/.kube의 기존 config 파일에 추가됩니다.

2. 구성을 테스트합니다.

```
kubectl get svc
```

Note

권한 부여 또는 리소스 유형 오류가 표시되는 경우 문제 해결 주제의 [the section called “권한이 없거나 액세스가 거부됨\(kubectl\)”](#) 부분을 참조하세요.

예제 출력은 다음과 같습니다.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
svc/kubernetes	ClusterIP	10.100.0.1	<none>	443/TCP	1m

3단계: 노드 생성

Important

가능한 한 간단하고 빠르게 시작하기 위해 이 주제에는 대부분 기본 설정을 사용하여 노드를 생성하는 단계가 포함되어 있습니다. 프로덕션 용도로 노드를 생성하기 전에 모든 설정을 숙지하고 요구 사항을 충족하는 설정으로 노드를 배포하는 것이 좋습니다. 자세한 내용은 [컴퓨팅 관리](#) 섹션을 참조하세요. 일부 설정은 노드를 생성할 때만 사용 설정할 수 있습니다.

이 절차에서는 관리형 노드 그룹을 사용하여 이전 단계에서 생성한 서브넷 및 노드 IAM 역할을 지정하여 노드를 생성하도록 클러스터를 구성합니다. 이를 통해 Amazon EC2 인스턴스에서 Amazon Linux 애플리케이션을 실행할 수 있습니다.

EKS에서 노드를 구성하는 다양한 방법에 대한 자세한 내용은 [컴퓨팅 관리](#) 섹션을 참조하세요. 클러스터를 배포한 후에 다른 노드 유형을 추가할 수 있습니다. 이 가이드에서는 다루지 않지만 [Windows 자체 관리형](#) 및 [Bottlerocket](#) 노드를 클러스터에 추가할 수도 있습니다.

Amazon EC2 Linux 관리형 노드 그룹 생성

1. 노드 IAM 역할을 생성하고 필요한 Amazon EKS IAM 관리형 정책을 연결합니다. Amazon EKS 노드 kubelet 대몬은 사용자를 대신하여 AWS API를 직접 호출합니다. 노드는 IAM 인스턴스 프로파일 및 연결 정책을 통해 이 API 직접 호출에 대한 권한을 수신합니다.

a. 다음 콘텐츠를 node-role-trust-policy.json라는 파일에 복사합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

b. 노드 IAM 역할을 생성합니다.

```
aws iam create-role \
  --role-name myAmazonEKSNodeRole \
  --assume-role-policy-document file://node-role-trust-policy.json
```

c. 필요한 관리형 IAM 정책을 역할에 연결합니다.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name myAmazonEKSNodeRole
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly \
  --role-name myAmazonEKSNodeRole
```

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
  --role-name myAmazonEKSNodeRole
```

- d. <https://console.aws.amazon.com/eks/home#/clusters>에서 Amazon EKS 콘솔을 엽니다.
 - e. [1단계: Amazon EKS 클러스터 만들기](#)에서 생성한 클러스터의 이름(예: *my-cluster*)을 선택합니다.
 - f. *my-cluster* 페이지에서 다음을 수행합니다.
 - g. 컴퓨팅 탭을 선택합니다.
 - h. 노드 그룹 추가를 선택합니다.
2. 노드 그룹 구성 페이지에서 다음을 수행합니다.
 - a. 이름에 관리형 노드 그룹(예: *my-nodegroup*)의 고유한 이름을 입력합니다. 노드 그룹 이름은 63자를 초과할 수 없습니다. 문자나 숫자로 시작하되, 나머지 문자의 경우 하이픈과 밑줄을 포함할 수 있습니다.
 - b. 노드 IAM 역할 이름에서 이전 단계에서 생성한 *myAmazonEKSNodeRole* 역할을 선택합니다. 각 노드 그룹은 고유한 IAM 역할을 사용하는 것이 좋습니다.
 - c. 다음을 선택합니다.
 3. 컴퓨팅 및 크기 조정 구성 설정 페이지에서 기본값으로 두고 다음을 선택합니다.
 4. 네트워킹 지정 페이지에서 기본값을 수락하고 다음을 선택합니다.
 5. 검토 및 생성 페이지에서 관리형 노드 그룹 구성을 검토하고 생성을 선택합니다.
 6. 몇 분 후 노드 그룹 구성 섹션의 상태가 생성 중에서 활성으로 바뀝니다. 상태가 활성이 되면 다음 단계를 진행합니다.

4단계: 리소스 보기

노드 및 Kubernetes 워크로드를 볼 수 있습니다.

1. 좌측 탐색 창에서 클러스터를 선택합니다. 클러스터 목록에서 생성한 클러스터의 이름을 선택합니다(예: *my-cluster*).
2. *my-cluster* 페이지에서 다음을 선택합니다.
 - a. 컴퓨팅 탭 - 클러스터에 대해 배포된 노드 목록이 표시됩니다. 노드 이름을 선택하면 노드에 대한 자세한 정보를 볼 수 있습니다.
 - b. 리소스 탭 - Amazon EKS 클러스터에 기본적으로 배포되는 Kubernetes 리소스가 모두 표시됩니다. 자세한 내용을 알아보려면 콘솔에서 모든 리소스 유형을 선택하세요.

5단계: 리소스 삭제

이 튜토리얼용으로 생성한 클러스터와 노드의 사용을 끝낸 후에는 생성한 리소스를 삭제해야 합니다. 리소스를 삭제하기 전에 이 클러스터로 추가 작업을 수행하려면 [the section called “다음 단계”](#) 부분을 참조하세요.

1. 생성한 노드 그룹 프로파일을 삭제하세요.
 - a. <https://console.aws.amazon.com/eks/home#/clusters>에서 Amazon EKS 콘솔을 엽니다.
 - b. 좌측 탐색 창에서 클러스터를 선택합니다. 클러스터 목록에서 **my-cluster**를 선택합니다.
 - c. 컴퓨팅 탭을 선택합니다.
 - d. 노드 그룹을 생성한 경우 **my-nodegroup** 노드 그룹을 선택한 다음 삭제를 선택합니다. **my-nodegroup**을 입력하고 삭제를 선택합니다.
 - e. 노드 그룹 프로파일이 삭제될 때까지 계속 진행하지 마세요.
2. 클러스터를 삭제합니다.
 - a. 좌측 탐색 창에서 클러스터를 선택합니다. 클러스터 목록에서 **my-cluster**를 선택합니다.
 - b. 클러스터 삭제를 선택합니다.
 - c. **my-cluster**를 입력하고 삭제를 선택합니다. 클러스터가 삭제될 때까지 계속하지 마세요.
3. 생성한 VPC AWS CloudFormation 스택을 삭제합니다.
 - a. <https://console.aws.amazon.com/cloudformation/>에서 CloudFormation 콘솔을 엽니다.
 - b. **my-eks-vpc-stack** 스택을 선택하고 삭제를 선택합니다.
 - c. **my-eks-vpc-stack** 삭제 확인 대화 상자에서 스택 삭제를 선택합니다.
4. 생성한 IAM 역할을 삭제합니다.
 - a. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
 - b. 왼쪽 탐색 창에서 역할을 선택합니다.
 - c. 목록에서 생성한 각 역할(**myAmazonEKSClusterRole** 및 **myAmazonEKSNodeRole**)을 선택하세요. 삭제를 선택하고 요청한 확인 텍스트를 입력한 다음 삭제를 선택합니다.

다음 단계

다음은 클러스터의 기능을 확장하는 데 도움이 되는 설명서 주제입니다.

- 클러스터를 생성한 [IAM 위탁자](#)는 kubectl 또는 AWS Management Console을 사용하여 Kubernetes API 서버를 직접적으로 직접 호출할 수 있는 유일한 위탁자입니다. 다른 IAM 보안 주

체가 클러스터에 액세스할 수 있게 하려면 해당 보안 주체를 추가해야 합니다. 자세한 내용은 [the section called “Kubernetes API 액세스”](#) 및 [the section called “필수 권한”](#) 섹션을 참조하세요.

- 클러스터에 [샘플 애플리케이션](#)을 배포합니다.
- 프로덕션 용도로 클러스터를 배포하기 전에 [클러스터](#)와 [노드](#)에 대한 모든 설정을 숙지하는 것이 좋습니다. Amazon EC2 노드에 대한 SSH 액세스 사용 설정과 같은 일부 설정은 클러스터를 생성할 때 지정해야 합니다.
- 클러스터의 보안을 강화하려면 [서비스 계정에 IAM 역할을 사용하도록 Amazon VPC 컨테이너 네트워크 인터페이스 플러그 인을 구성합니다](#).

EKS Auto Mode를 사용하여 클러스터 인프라 자동화

EKS Auto Mode는 Kubernetes 클러스터의 AWS 관리를 클러스터 자체 이상으로 확장하여 AWS가 워크로드를 원활하게 운영할 수 있는 인프라를 설정하고 관리할 수 있게 합니다. 주요 인프라 결정을 위임하고 일상적인 작업에 AWS의 전문 지식을 활용할 수 있습니다. AWS에서 관리하는 클러스터 인프라에는 컴퓨팅 오토 스케일링, 포드 및 서비스 네트워킹, 애플리케이션 로드 밸런싱, 클러스터 DNS, 블록 스토리지, GPU 지원과 같은 추가 기능과 달리 많은 Kubernetes 기능이 핵심 구성 요소로 포함되어 있습니다.

시작하려면 새 EKS Auto Mode 클러스터를 배포하거나 기존 클러스터에서 EKS Auto Mode를 활성화할 수 있습니다. eksctl, AWS CLI, AWS Management Console, EKS API 또는 선호하는 코드형 인프라 도구를 사용하여 EKS 자동 모드 클러스터를 배포, 업그레이드 또는 수정할 수 있습니다.

EKS Auto Mode를 사용하면 원하는 Kubernetes 호환 도구를 계속 사용할 수 있습니다. EKS Auto Mode는 모범 사례를 따르는 AWS 클라우드 리소스를 활용하여 Amazon EC2, Amazon EBS 및 ELB와 같은 AWS 서비스와 통합됩니다. 이러한 리소스는 자동으로 조정되고, 비용이 최적화되고, 정기적으로 업데이트되어 운영 비용 및 오버헤드를 최소화하는 데 도움이 됩니다.

Features

EKS Auto Mode은 다음의 대략적인 기능을 제공합니다.

Kubernetes 클러스터 관리 간소화: EKS Auto Mode는 운영 오버헤드를 최소화하면서 프로덕션 지원 클러스터를 제공하여 EKS 관리를 간소화합니다. EKS Auto Mode를 사용하면 심층적인 EKS 전문 지식 없이도 까다롭고 동적인 워크로드를 자신 있게 실행할 수 있습니다.

애플리케이션 가용성: EKS Auto Mode는 Kubernetes 애플리케이션의 요구 사항에 따라 EKS 클러스터의 노드를 동적으로 추가하거나 제거합니다. 이렇게 하면 수동 용량 계획의 필요성을 최소화하고 애플리케이션 가용성을 보장할 수 있습니다.

효율성: EKS 자동 모드는 NodePool 및 워크로드 요구 사항에 정의된 유연성을 준수하면서 비용을 최적화하도록 설계되었습니다. 또한 미사용 인스턴스를 종료하고 워크로드를 다른 노드로 통합하여 비용 효율성을 개선합니다.

보안: EKS Auto Mode는 노드에 대해 변경할 수 없는 것으로 취급되는 AMI를 사용합니다. 이러한 AMI는 잠긴 소프트웨어를 적용하고 SELinux 필수 액세스 제어를 활성화하며 읽기 전용 루트 파일 시스템을 제공합니다. 또한 EKS Auto Mode에서 시작된 노드의 최대 수명은 21일(단축할 수 있음)이며, 그 후에는 새 노드로 자동 대체됩니다. 이 접근 방식은 많은 고객이 이미 채택한 모범 사례에 따라 노드를 정기적으로 사이클링하여 보안 태세를 강화합니다.

자동 업그레이드: EKS Auto Mode는 구성된 포드 중단 예산(PDB) 및 NodePool 중단 예산(NDB)을 준수하면서 최신 패치를 사용하여 Kubernetes 클러스터, 노드, 관련 구성 요소를 최신 상태로 유지합니다. 최대 수명 21일까지 PDB 또는 기타 구성을 차단하여 업데이트가 불가능한 경우 중재가 필요할 수 있습니다.

관리형 구성 요소: EKS Auto Mode에는 Kubernetes 및 AWS 클라우드 기능이 추가 기능으로 관리해야 하는 핵심 구성 요소로 포함되어 있습니다. 여기에는 포드 IP 주소 할당, 포드 네트워크 정책, 로컬 DNS 서비스, GPU 플러그인, 상태 확인기, EBS CSI 스토리지에 대한 기본 지원이 포함됩니다.

사용자 지정 가능한 NodePool 및 NodeClass: 워크로드에서 스토리지, 컴퓨팅 또는 네트워킹 구성을 변경해야 하는 경우 EKS Auto Mode를 사용하여 사용자 지정 NodePool 및 NodeClass를 생성할 수 있습니다. 기본 NodePool 및 NodeClass는 편집하면 안 되지만 특정 요구 사항을 충족하기 위해 기본 구성과 함께 새 사용자 지정 NodePool 또는 NodeClass를 추가할 수 있습니다.

자동 구성 요소

EKS Auto Mode는 주요 인프라 구성 요소를 자동화하여 Amazon EKS 클러스터의 작업을 간소화합니다. EKS Auto Mode를 활성화하면 EKS 클러스터를 관리하는 작업이 더욱 줄어듭니다.

다음은 자동화된 데이터 플레인 구성 요소의 목록입니다.

- 컴퓨팅: 많은 워크로드에서 EKS Auto Mode를 사용하면 EKS 클러스터에 대한 컴퓨팅의 여러 측면을 잊어버릴 수 있습니다. 다음이 포함됩니다.
 - 노드: EKS Auto Mode 노드는 어플라이언스처럼 처리되도록 설계되었습니다. EKS Auto Mode는 다음을 수행합니다.
 - 개입 없이 워크로드를 실행하는 데 필요한 많은 서비스로 구성된 적절한 AMI를 선택합니다.
 - SELinux 적용 모드와 읽기 전용 루트 파일 시스템을 사용하여 AMI에서 파일 액세스를 잠급니다.
 - SSH 또는 SSM 액세스를 허용하지 않음으로써 노드에 대한 직접 액세스를 방지합니다.
 - NVIDIA 및 Neuron GPU에 대한 별도의 커널 드라이버와 플러그인을 통해 고성능 워크로드를 지원하는 GPU 지원이 포함되어 있습니다.
 - 오토 스케일링: [Karpenter](#) 오토 스케일링을 사용하는 EKS Auto Mode는 예약 불가능한 포드를 모니터링하고 새 노드를 배포하여 해당 포드를 실행할 수 있습니다. 워크로드가 종료되면 EKS Auto Mode는 노드가 더 이상 필요하지 않을 때 노드를 동적으로 중단 및 종료하여 리소스 사용을 최적화합니다.
- 업그레이드: 노드를 제어하면 필요에 따라 보안 패치와 운영 체제 및 구성 요소 업그레이드를 제공하는 EKS Auto Mode의 기능이 간소화됩니다. 이러한 업그레이드는 워크로드 중단을 최소화하도

록 설계되었습니다. EKS Auto Mode는 최신 소프트웨어 및 API를 보장하기 위해 최대 21일의 노드 수명을 적용합니다.

- 로드 밸런싱: EKS Auto Mode는 Amazon의 Elastic Load Balancing 서비스와 통합하여 Kubernetes 서비스 및 수신 리소스용 로드 밸런서의 프로비저닝 및 구성을 자동화함으로써 로드 밸런싱을 간소화합니다. 애플리케이션과 Network Load Balancer 모두에 고급 기능을 지원하고, 수명 주기를 관리하며, 클러스터 수요에 맞게 확장합니다. 이러한 통합은 AWS 모범 사례를 준수하는 프로덕션 지원 로드 밸런싱 솔루션을 제공하므로 인프라 관리가 아닌 애플리케이션에 집중할 수 있습니다.
- 스토리지: EKS Auto Mode는 노드 종료 시 볼륨 유형, 볼륨 크기, 암호화 정책, 삭제 정책을 설정하여 임시 스토리지를 구성합니다.
- 네트워킹: EKS Auto Mode는 포드 및 서비스 연결을 위한 중요한 네트워킹 작업을 자동화합니다. 여기에는 IPv4/IPv6 지원과 IP 주소 공간을 확장하기 위한 보조 CIDR 블록 사용이 포함됩니다.
- ID 및 액세스 관리: EKS Auto Mode 클러스터에 EKS Pod Identity 에이전트를 설치할 필요가 없습니다.

이러한 구성 요소에 대한 자세한 내용은 [the section called “작동 방법”](#) 섹션을 참조하세요.

구성

EKS Auto Mode는 개입 없이 대부분의 데이터 플레인 서비스를 효과적으로 관리하지만 일부 서비스의 동작을 변경해야 하는 경우가 있을 수 있습니다. 다음과 같은 방법으로 EKS Auto Mode 클러스터의 구성을 수정할 수 있습니다.

- Kubernetes DaemonSets: 노드에 설치된 서비스를 수정하는 대신 Kubernetes DaemonSets를 사용할 수 있습니다. DaemonSets는 Kubernetes에서 관리하도록 설계되었지만 클러스터의 모든 노드에서 실행됩니다. 이렇게 하면 노드를 모니터링하거나 다른 방식으로 감시하기 위한 특수 서비스를 추가할 수 있습니다.
- 사용자 지정 NodePool 및 NodeClass: 기본 NodePool 및 NodeClass는 EKS Auto Mode에 의해 구성되며 편집하면 안 됩니다. 노드 동작을 사용자 지정하려면 다음과 같은 사용 사례에 대해 추가 NodePool 또는 NodeClass를 생성할 수 있습니다.
 - 특정 인스턴스 유형(예: 가속 프로세서 또는 EC2 스팟 인스턴스)을 선택합니다.
 - 보안 또는 비용 추적을 위해 워크로드를 격리합니다.
 - IOPS, 크기, 처리량과 같은 임시 스토리지 설정을 구성합니다.
- 로드 밸런싱: EKS Auto Mode가 Kubernetes 객체로 실행되는 로드 밸런싱과 같은 일부 서비스를 EKS Auto Mode 클러스터에서 직접 구성할 수 있습니다.

EKS Auto Mode 구성 옵션에 대한 자세한 내용은 [the section called “구성”](#) 섹션을 참조하세요.

Amazon EKS 자율 모드에서 클러스터 생성

이 장에서는 다양한 도구와 인터페이스를 사용하여 Auto Mode가 활성화된 Amazon EKS 클러스터를 생성하는 방법을 설명합니다. Auto Mode는 클러스터의 컴퓨팅, 네트워킹, 스토리지 인프라를 자동으로 구성하고 관리하여 클러스터 생성을 간소화합니다. AWS CLI, AWS Management Console 또는 eksctl 명령줄 도구를 사용하여 자동 모드 클러스터를 생성하는 방법을 알아봅니다.

Note

EKS Auto Mode에는 Kubernetes 버전 1.29 이상이 필요합니다. EKS 자율 모드는 ap-southeast-7 또는 mx-central-1 AWS 리전에서 사용할 수 없습니다.

필요에 따라 원하는 도구를 선택합니다. AWS Management Console은 EKS 자동 모드 기능을 알아보고 개별 클러스터를 생성하는 데 이상적인 시각적 인터페이스를 제공합니다. AWS CLI는 스크립팅 및 자동화 작업, 특히 클러스터 생성을 기존 워크플로 또는 CI/CD 파이프라인에 통합할 때 가장 적합합니다. eksctl CLI는 Kubernetes 네이티브 환경을 제공하며, 합리적인 기본값으로 간소화된 명령줄 작업을 원하는 Kubernetes 도구에 익숙한 사용자에게 권장됩니다.

시작하기 전에 EKS 클러스터를 생성하는 데 필요한 IAM 권한을 포함하여 필요한 사전 요구 사항을 설치하고 구성했는지 확인합니다. kubectl, aws, eksctl과 같은 CLI 도구를 설치하는 방법은 [설정](#) 섹션을 참조하세요.

AWS CLI, AWS Management Console 또는 eksctl CLI를 사용하여 Amazon EKS 자동 모드로 클러스터를 생성할 수 있습니다.

주제

- [eksctl CLI를 사용하여 EKS Auto Mode 클러스터 생성](#)
- [AWS CLI를 사용하여 EKS Auto Mode 클러스터 생성](#)
- [AWS Management Console을 사용하여 EKS 자동 모드 클러스터 생성](#)

eksctl CLI를 사용하여 EKS Auto Mode 클러스터 생성

이 주제에서는 eksctl 명령줄 인터페이스(CLI)를 사용하여 Amazon EKS Auto Mode 클러스터를 생성하는 방법을 보여줍니다. 단일 CLI 명령을 실행하거나 YAML 구성 파일을 적용하여 Auto Mode 클러스

터를 생성할 수 있습니다. 두 방법 모두 동일한 기능을 제공하며, YAML 접근 방식은 클러스터 설정을 더 세밀하게 제어할 수 있습니다.

eksctl CLI는 기본 AWS 리소스 생성 및 구성을 처리하여 EKS Auto Mode 클러스터를 생성하고 관리하는 프로세스를 간소화합니다. 계속하기 전에 로컬 시스템에 필요한 AWS 자격 증명과 권한이 구성되어 있는지 확인합니다. 이 안내서에서는 사용자가 기본 Amazon EKS 개념을 잘 알고 필요한 CLI 도구를 이미 설치했다고 가정합니다.

Note

버전 0.195.0 이상의 eksctl을 설치해야 합니다. 자세한 내용은 GitHub의 [eksctl 릴리스](#)를 참조하세요.

CLI 명령을 사용하여 EKS Auto Mode 클러스터 생성

aws 및 eksctl 도구가 설치되어 있어야 합니다. EC2 인스턴스, EC2 네트워킹, EKS 클러스터, IAM 역할을 포함한 AWS 리소스를 관리하려면 충분한 권한을 가지고 AWS CLI에 로그인해야 합니다. 자세한 내용은 [설정](#) 단원을 참조하십시오.

다음 명령을 실행하여 새 EKS Auto Mode 클러스터를 생성합니다.

```
eksctl create cluster --name=<cluster-name> --enable-auto-mode
```

YAML 파일을 사용하여 EKS Auto Mode 클러스터 생성

aws 및 eksctl 도구가 설치되어 있어야 합니다. EC2 인스턴스, EC2 네트워킹, EKS 클러스터, IAM 역할을 포함한 AWS 리소스를 관리하려면 충분한 권한을 가지고 AWS CLI에 로그인해야 합니다. 자세한 내용은 [설정](#) 단원을 참조하십시오.

아래 샘플 ClusterConfig 리소스의 EKS Auto Mode 구성 옵션을 검토합니다. 전체 ClusterConfig 사양은 [eksctl documentation](#)를 참조하세요.

AWS에서는 EKS Auto Mode를 활성화할 것을 제안합니다. EKS Auto Mode 클러스터를 처음 생성하는 경우 nodeRoleARN을 지정되지 않은 상태로 두고 EKS Auto Mode에 대한 노드 IAM 역할을 생성합니다. AWS 계정에 노드 IAM 역할이 이미 있는 경우 AWS는 이를 재사용할 것을 제안합니다.

AWS는 nodePools에 대한 값을 지정하지 않을 것을 제안합니다. EKS Auto Mode는 기본 노드 풀을 생성합니다. Kubernetes API를 사용하여 추가 노드 풀을 생성할 수 있습니다.

```
# cluster.yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: <cluster-name>
  region: <aws-region>

iam:
  # ARN of the Cluster IAM Role
  # optional, eksctl creates a new role if not supplied
  # suggested to use one Cluster IAM Role per account
  serviceRoleARN: <arn-cluster-iam-role>

autoModeConfig:
  # defaults to false
  enabled: boolean
  # optional, defaults to [general-purpose, system].
  # suggested to leave unspecified
  # To disable creation of nodePools, set it to the empty array ([]).
  nodePools: []string
  # optional, eksctl creates a new role if this is not supplied
  # and nodePools are present.
  nodeRoleARN: string
```

ClusterConfig 파일을 `cluster.yaml`로 저장하고 다음 명령을 사용하여 클러스터를 생성합니다.

```
eksctl create cluster -f cluster.yaml
```

AWS CLI를 사용하여 EKS Auto Mode 클러스터 생성

EKS Auto Mode 클러스터는 컴퓨팅, 스토리지, 네트워킹을 위한 일상적인 클러스터 관리 작업을 자동화합니다. 예를 들어 EKS Auto Mode 클러스터는 추가 노드가 필요한 시기를 자동으로 감지하고 워크로드 수요에 맞춰 새 EC2 인스턴스를 프로비저닝합니다.

이 주제에서는 AWS CLI를 사용하여 새 EKS Auto Mode 클러스터를 생성하고 선택적으로 샘플 워크로드를 배포하는 방법을 안내합니다.

사전 조건

- 장치에 최신 버전의 AWS Command Line Interface(AWS CLI)가 설치 및 구성되어 있습니다. 현재 버전을 확인하려면 `aws --version`을 사용합니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요.
- 충분한 IAM 권한을 사용하여 CLI에 로그인하고 IAM 정책, IAM 역할, EKS 클러스터를 포함한 AWS 리소스를 생성합니다.
- 장치에 설치된 kubectl 명령줄 도구입니다. AWS에서는 EKS 클러스터의 Kubernetes 버전과 동일한 kubectl 버전을 사용할 것을 권장합니다. kubectl을 설치하거나 업그레이드하려면 [the section called “kubectl 및 eksctl 설정”](#) 섹션을 참조하세요.

VPC 서브넷 지정

Amazon EKS Auto Mode는 노드를 VPC 서브넷에 배포합니다. EKS 클러스터를 생성할 때 노드가 배포될 VPC 서브넷을 지정해야 합니다. AWS 계정에서 기본 VPC 서브넷을 사용하거나 중요한 워크로드를 위한 전용 VPC를 생성할 수 있습니다.

- AWS는 클러스터에 대한 전용 VPC를 생성할 것을 제안합니다. [the section called “VPC 생성”](#) 방법에 대해 알아봅니다.
- EKS 콘솔은 새 VPC 생성을 지원합니다. [the section called “관리 콘솔”](#) 방법에 대해 알아봅니다.
- 또는 AWS 계정의 기본 VPC를 사용할 수 있습니다. 다음 지침에 따라 서브넷 ID를 찾습니다.

기본 VPC의 서브넷 ID를 찾으려면

AWS CLI 사용:

1. 다음 명령을 실행하여 기본 VPC와 해당 서브넷을 나열합니다.

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$(aws ec2 describe-vpcs --query 'Vpcs[?IsDefault==`true`].VpcId' --output text)" --query 'Subnets[*].{ID:SubnetId,AZ:AvailabilityZone}' --output table
```

2. 출력을 저장하고 서브넷 ID를 기록합니다.

샘플 출력:

```
-----
| DescribeSubnets |
```


SubnetId	AvailabilityZone
subnet-012345678	us-west-2a
subnet-234567890	us-west-2b
subnet-345678901	us-west-2c

EKS Auto Mode 클러스터에 대한 IAM 역할

클러스터 IAM 역할

EKS Auto Mode에서 새 EC2 인스턴스 프로비저닝과 같은 작업을 AWS 계정에서 수행하려면 클러스터 IAM 역할이 필요합니다. EKS에 필요한 권한을 부여하려면 이 역할을 생성해야 합니다. AWS는 클러스터 IAM 역할에 다음 AWS 관리형 정책을 연결할 것을 권장합니다.

- [AmazonEKSClusterPolicy](#)
- [AmazonEKSBlockStoragePolicy](#)
- [AmazonEKSLoadBalancingPolicy](#)
- [AmazonEKSNetworkingPolicy](#)
- [AmazonEKSClusterPolicy](#)

노드 IAM 역할

EKS Auto Mode 클러스터를 생성할 때 노드 IAM 역할을 지정합니다. EKS Auto Mode가 보류 중인 워크로드를 처리할 노드를 생성하면 각 새 EC2 인스턴스 노드에 노드 IAM 역할이 할당됩니다. 이 역할은 노드가 EKS와 통신하도록 허용하지만 일반적으로 노드에서 실행되는 워크로드에서는 액세스하지 않습니다.

노드에서 실행되는 워크로드에 권한을 부여하려면 EKS Pod Identity를 사용합니다. 자세한 내용은 [the section called “Pod Identity”](#) 단원을 참조하십시오.

이 역할을 생성하고 다음 AWS 관리형 정책을 연결해야 합니다.

- [AmazonEKSECSWorkerNodeMinimalPolicy](#)
- [AmazonEC2ContainerRegistryPullOnly](#)

서비스 연결 역할

또한 EKS Auto Mode에는 AWS에서 자동으로 생성 및 구성하는 서비스 연결 역할이 필요합니다. 자세한 내용은 [AWSServiceRoleForAmazonEKS](#)를 참조하세요.

EKS Auto Mode 클러스터 IAM 역할 생성

1단계: 신뢰 정책 생성

Amazon EKS 서비스가 이 역할을 수입하도록 허용하는 신뢰 정책을 생성합니다. 정책을 `trust-policy.json`으로 저장합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

2단계: IAM 역할 생성

신뢰 정책을 사용하여 클러스터 IAM 역할을 생성합니다.

```
aws iam create-role \
  --role-name AmazonEKSAutoClusterRole \
  --assume-role-policy-document file://trust-policy.json
```

3단계: 역할 ARN 기록

다음 단계에서 사용할 새 역할의 ARN을 검색하고 저장합니다.

```
aws iam get-role --role-name AmazonEKSAutoClusterRole --query "Role.Arn" --output text
```

4단계: 필수 정책 연결

다음 AWS 관리형 정책을 클러스터 IAM 역할에 연결하여 필요한 권한을 부여합니다.

AmazonEKSClusterPolicy:

```
aws iam attach-role-policy \  
  --role-name AmazonEKSAutoClusterRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
```

AmazonEKSComputePolicy:

```
aws iam attach-role-policy \  
  --role-name AmazonEKSAutoClusterRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSComputePolicy
```

AmazonEKSBlockStoragePolicy:

```
aws iam attach-role-policy \  
  --role-name AmazonEKSAutoClusterRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSBlockStoragePolicy
```

AmazonEKSLoadBalancingPolicy:

```
aws iam attach-role-policy \  
  --role-name AmazonEKSAutoClusterRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSLoadBalancingPolicy
```

AmazonEKSNetworkingPolicy:

```
aws iam attach-role-policy \  
  --role-name AmazonEKSAutoClusterRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSNetworkingPolicy
```

EKS Auto Mode 노드 IAM 역할 생성

1단계: 신뢰 정책 생성

Amazon EKS 서비스가 이 역할을 수임하도록 허용하는 신뢰 정책을 생성합니다. 정책을 `node-trust-policy.json`으로 저장합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2단계: 노드 IAM 역할 생성

이전 단계의 node-trust-policy.json 파일을 사용하여 역할을 수입할 수 있는 엔터티를 정의합니다. 다음 명령을 실행하여 노드 IAM 역할을 만듭니다.

```
aws iam create-role \
  --role-name AmazonEKSAutoNodeRole \
  --assume-role-policy-document file:///node-trust-policy.json
```

3단계: 역할 ARN 기록

역할을 생성한 후 노드 IAM 역할의 ARN을 검색하고 저장합니다. 이후 단계에서 이 ARN이 필요합니다. 다음 명령을 사용하여 ARN을 가져옵니다.

```
aws iam get-role --role-name AmazonEKSAutoNodeRole --query "Role.Arn" --output text
```

4단계: 필수 정책 연결

다음 AWS 관리형 정책을 노드 IAM 역할에 연결하여 필요한 권한을 제공합니다.

AmazonEKSWorkerNodeMinimalPolicy:

```
aws iam attach-role-policy \
  --role-name AmazonEKSAutoNodeRole \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodeMinimalPolicy
```

AmazonEC2ContainerRegistryPullOnly:

```
aws iam attach-role-policy \
```

```
--role-name AmazonEKSAutoNodeRole \  
--policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

EKS Auto Mode 클러스터 생성

개요

AWS CLI를 사용하여 EKS Auto Mode 클러스터를 생성하려면 다음 파라미터가 필요합니다.

- `cluster-name`: 클러스터의 이름입니다.
- `k8s-version`: Kubernetes 버전(예: 1.31)입니다.
- `subnet-ids`: 이전 단계에서 식별된 서브넷 ID입니다.
- `cluster-role-arn`: 클러스터 IAM 역할의 ARN입니다.
- `node-role-arn`: 노드 IAM 역할의 ARN입니다.

기본 클러스터 구성

클러스터를 생성하기 전에 다음 기본값과 기능을 검토합니다.

- `nodePools`: EKS Auto Mode에는 범용 및 시스템 기본 노드 풀이 포함됩니다. [노드 풀](#)에 대해 자세히 알아봅니다.

참고: EKS Auto Mode의 노드 풀은 Amazon EKS 관리형 노드 그룹과 다르지만 동일한 클러스터에 공존할 수 있습니다.

- `computeConfig.enabled`: EC2 인스턴스 생성 및 삭제와 같은 일상적인 컴퓨팅 작업을 자동화합니다.
- `kubernetesNetworkConfig.elasticLoadBalancing.enabled`: Elastic Load Balancer 생성 및 삭제와 같은 로드 밸런싱 작업을 자동화합니다.
- `storageConfig.blockStorage.enabled`: Amazon EBS 볼륨 생성 및 삭제와 같은 스토리지 작업을 자동화합니다.
- `accessConfig.authenticationMode`: EKS 액세스 항목이 필요합니다. [EKS 인증 모드](#)에 대해 자세히 알아봅니다.

명령 실행

다음 명령을 사용하여 클러스터를 생성합니다.

```
aws eks create-cluster \
  --region ${AWS_REGION} \
  --cli-input-json \
  "{
    \"name\": \"${CLUSTER_NAME}\",
    \"version\": \"${K8S_VERSION}\",
    \"roleArn\": \"${CLUSTER_ROLE_ARN}\",
    \"resourcesVpcConfig\": {
      \"subnetIds\": ${SUBNETS_JSON},
      \"endpointPublicAccess\": true,
      \"endpointPrivateAccess\": true
    },
    \"computeConfig\": {
      \"enabled\": true,
      \"nodeRoleArn\": \"${NODE_ROLE_ARN}\",
      \"nodePools\": [\"general-purpose\", \"system\"]
    },
    \"kubernetesNetworkConfig\": {
      \"elasticLoadBalancing\": {
        \"enabled\": true
      }
    },
    \"storageConfig\": {
      \"blockStorage\": {
        \"enabled\": true
      }
    },
    \"accessConfig\": {
      \"authenticationMode\": \"API\"
    }
  }
```

클러스터 상태 확인

1단계: 클러스터 생성 확인

다음 명령을 실행하여 클러스터의 상태를 확인합니다. 클러스터 생성에는 일반적으로 15분 정도 걸립니다.

```
aws eks describe-cluster --name "${CLUSTER_NAME}" --output json
```

2단계: kubeconfig 업데이트

클러스터가 준비되면 `kubectl`이 클러스터와 통신할 수 있도록 로컬 kubeconfig 파일을 업데이트합니다. 이 구성은 인증에 AWS CLI를 사용합니다.

```
aws eks update-kubeconfig --name "${CLUSTER_NAME}"
```

3단계: 노드 풀 확인

다음 명령을 사용하여 클러스터의 노드 풀을 나열합니다.

```
kubectl get nodepools
```

다음 단계

- 새 EKS Auto Mode 클러스터에 [샘플 워크로드를 배포](#)하는 방법을 알아봅니다.

AWS Management Console을 사용하여 EKS 자동 모드 클러스터 생성

AWS Management Console에서 EKS 자동 모드 클러스터를 생성하려면 다른 옵션보다 구성이 적게 필요합니다. EKS는 AWS IAM 및 VPC 네트워킹과 통합되어 EKS 클러스터와 연결된 리소스를 생성하는 데 도움이 됩니다.

콘솔에서 클러스터를 생성하는 두 가지 옵션이 있습니다.

- 빠른 구성(EKS Auto Mode 사용)
- 사용자 지정 구성

이 주제에서는 빠른 구성 옵션을 사용하여 EKS Auto Mode 클러스터를 생성하는 방법을 알아봅니다.

빠른 구성 옵션을 사용하여 EKS Auto Mode 생성

EC2 인스턴스, EC2 네트워킹, EKS 클러스터, IAM 역할을 포함한 AWS 리소스를 관리하려면 충분한 권한을 가지고 AWS Management Console에 로그인해야 합니다.

1. EKS 콘솔로 이동합니다.
2. 클러스터 생성을 클릭합니다.
3. 빠른 구성 옵션이 선택되었는지 확인합니다.
4. 다음 값을 결정하거나 테스트 클러스터의 기본값을 사용합니다.

- 클러스터 이름
 - Kubernetes 버전
5. 클러스터 IAM 역할을 선택합니다. EKS Auto Mode 클러스터를 처음 생성하는 경우 권장 역할 생성 옵션을 사용합니다.
 - 선택적으로 모든 EKS Auto Mode 클러스터에 대해 AWS 계정의 단일 클러스터 IAM 역할을 재사용할 수 있습니다.
 - 클러스터 IAM 역할에는 EKS Auto Mode가 EC2 인스턴스, EBS 볼륨, EC2 로드 밸런서 등의 리소스를 관리하는 데 필요한 권한이 포함됩니다.
 - 권장 역할 생성 옵션은 모든 필드를 권장 값으로 미리 채웁니다. 다음을 선택하고 생성을 선택합니다. 역할은 제안된 AmazonEKSAutoClusterRole 이름을 사용합니다.
 - 최근에 새 역할을 생성한 경우 새로 고침 아이콘을 사용하여 역할 선택 드롭다운을 다시 로드합니다.
 6. 노드 IAM 역할을 선택합니다. EKS Auto Mode 클러스터를 처음 생성하는 경우 권장 역할 생성 옵션을 사용합니다.
 - 선택적으로 모든 EKS Auto Mode 클러스터에 대해 AWS 계정의 단일 노드 IAM 역할을 재사용할 수 있습니다.
 - 노드 IAM 역할에는 Auto Mode 노드가 클러스터에 연결하는 데 필요한 권한이 포함되어 있습니다. 노드 IAM 역할에는 컨테이너의 ECR 이미지를 검색할 수 있는 권한이 포함되어야 합니다.
 - 권장 역할 생성 옵션은 모든 필드를 권장 값으로 미리 채웁니다. 다음을 선택하고 생성을 선택합니다. 역할은 제안된 AmazonEKSAutoNodeRole 이름을 사용합니다.
 - 최근에 새 역할을 생성한 경우 새로 고침 아이콘을 사용하여 역할 선택 드롭다운을 다시 로드합니다.
 7. EKS Auto Mode 클러스터의 VPC를 선택합니다. VPC 생성을 선택하여 EKS용 새 VPC를 생성하거나 이전에 EKS용으로 생성한 VPC를 선택합니다.
 - VPC 콘솔을 사용하여 새 VPC를 생성하는 경우 AWS는 가용 영역당 하나 이상의 NAT 게이트웨이를 생성할 것을 제안합니다. 또는 다른 모든 기본값을 사용할 수 있습니다.
 - IPv6 클러스터 요구 사항에 대한 자세한 내용은 [the section called “VPC 생성”](#) 섹션을 참조하세요.
 8. (선택 사항) EKS Auto Mode는 선택한 VPC의 프라이빗 서브넷을 자동으로 채웁니다. 원치 않는 서브넷을 제거할 수 있습니다.
 - EKS는 모범 사례에 따라 VPC에서 프라이빗 서브넷을 자동으로 선택합니다. 필요에 따라 VPC에서 퍼블릭 서브넷과 같은 추가 서브넷을 선택할 수 있습니다.
 9. (선택 사항) 빠른 구성 기본값 보기를 선택하고 새 클러스터의 모든 구성 값을 검토합니다. 테이블은 클러스터가 생성된 후 일부 값을 편집할 수 없음을 나타냅니다.

10[클러스터 생성(Create cluster)]을 선택합니다. 클러스터 생성을 완료하는 데 15분이 걸릴 수 있습니다.

다음 단계

- [EKS Auto Mode 클러스터에 샘플 워크로드를 배포](#)하는 방법을 알아봅니다.

기존 EKS 클러스터에서 EKS Auto Mode 활성화

기존 EKS 클러스터에서 EKS Auto Mode를 활성화할 수 있습니다.

Note

EKS Auto Mode에는 Kubernetes 버전 1.29 이상이 필요합니다. EKS 자율 모드는 ap-southeast-7 또는 mx-central-1 AWS 리전에서 사용할 수 없습니다.

AWS는 다음과 같은 마이그레이션을 지원합니다.

- Karpenter에서 EKS 자동 모드 노드로 마이그레이션. 자세한 내용은 [the section called “Karpenter에서 마이그레이션”](#) 섹션을 참조하세요.
- EKS 관리형 노드 그룹에서 EKS 자동 모드 노드로 마이그레이션. 자세한 내용은 [the section called “MNG에서 마이그레이션”](#) 섹션을 참조하세요.
- EKS Fargate에서 EKS 자동 모드로 마이그레이션. 자세한 내용은 [the section called “Fargate에서 마이그레이션”](#) 섹션을 참조하세요.

AWS는 다음과 같은 마이그레이션을 지원하지 않습니다.

- EBS CSI 컨트롤러(Amazon EKS 추가 기능 사용)에서 EKS 자동 모드 EBS CSI 컨트롤러(EKS 자동 모드에서 관리)로 볼륨 마이그레이션. 둘 중 하나로 만든 PVC는 다른 하나로 마운트할 수 없습니다. 두 개의 서로 다른 Kubernetes 볼륨 프로비저너를 사용하기 때문입니다.
- [eks-auto-mode-ebs-migration-tool](#)(AWS Labs 프로젝트)은 표준 EBS CSI StorageClass(`ebs.csi.aws.com`)와 EKS 자동 EBS CSI StorageClass(`ebs.csi.eks.amazonaws.com`) 간의 마이그레이션을 가능하게 합니다. 마이그레이션하려면 기존 PersistentVolumeClaim/PersistentVolume 리소스를 삭제했다가 다시 생성해야 하므로 구현하기 전에 비 프로덕션 환경에서 검증해야 합니다.

- AWS 로드 밸런서 컨트롤러에서 EKS Auto Mode로 로드 밸런서 마이그레이션

Amazon EKS Auto Mode 클러스터에 AWS 로드 밸런서 컨트롤러를 설치할 수 있습니다.

IngressClass 또는 loadBalancerClass 옵션을 사용하여 서비스 및 수신 리소스를 로드 밸런서 컨트롤러 또는 EKS Auto Mode와 연결합니다.

- 대체 CNI 또는 기타 지원되지 않는 네트워킹 구성을 사용하여 EKS 클러스터 마이그레이션

마이그레이션 참조

다음 마이그레이션 참조를 사용하여 자체 관리형 컨트롤러 또는 EKS 자동 모드가 소유하도록 Kubernetes 리소스를 구성합니다.

기능	리소스	필드	자체 관리형	EKS Auto Mode
블록 스토리지	StorageClass	provisioner	ebs.csi.aws.com	ebs.csi.eks.amazonaws.com
로드 밸런싱	Service	loadBalancerClass	service.k8s.aws/nlb	eks.amazonaws.com/nlb
로드 밸런싱	IngressClass	controller	ingress.k8s.aws/alb	eks.amazonaws.com/alb
로드 밸런싱	IngressClassParams	apiversion	elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1
로드 밸런싱	TargetGroupBinding	apiversion	elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1
컴퓨팅	NodeClass	apiVersion	karpenter.sh/v1alpha5	eks.amazonaws.com/v1

EBS 볼륨 마이그레이션

워크로드를 EKS 자동 모드로 마이그레이션할 때는 다음과 같이 다양한 CSI 드라이버 프로비저너로 인해 EBS 볼륨 마이그레이션을 처리해야 합니다.

- EKS 자동 모드 프로비저너: `ebs.csi.eks.amazonaws.com`
- 오픈 소스 EBS CSI 프로비저너: `ebs.csi.aws.com`

영구 볼륨을 마이그레이션하려면 다음 단계를 따르세요.

1. 볼륨 보존 정책 수정: 기본 EBS 볼륨이 삭제되지 않도록 기존 플랫폼 버전(PV)의 `persistentVolumeReclaimPolicy`를 `Retain`으로 변경합니다.
2. Kubernetes에서 PV 제거: 실제 EBS 볼륨을 그대로 유지하면서 이전 PV 리소스를 삭제합니다.
3. 정적 프로비저닝을 사용하여 새 PV 생성: 동일한 EBS 볼륨을 참조하지만 대상 CSI 드라이버에서 작동하는 새 PV를 생성합니다.
4. 새 PVC에 바인딩: `volumeName` 필드를 사용하여 PV를 구체적으로 참조하는 새 PVC를 생성합니다.

고려 사항

- 이 마이그레이션을 시작하기 전에 애플리케이션이 중지되었는지 확인합니다.
- 마이그레이션 프로세스를 시작하기 전에 데이터를 백업합니다.
- 각 영구 볼륨에 대해 이 프로세스를 수행해야 합니다.
- 새 PVC를 사용하도록 워크로드를 업데이트해야 합니다.

로드 밸런서 마이그레이션

자체 관리형 AWS 로드 밸런서 컨트롤러에서 EKS Auto Mode로 기존 로드 밸런서를 직접 이전할 수 없습니다. 대신 블루/그린 배포 전략을 구현해야 합니다. 관리형 컨트롤러에서 새 로드 밸런서를 생성하는 동안 기존 로드 밸런서 구성을 유지하는 작업이 포함됩니다.

서비스 중단을 최소화하려면 DNS 기반 트래픽 이동 접근 방식을 사용하는 것이 좋습니다. 먼저 기존 구성의 작동을 유지하면서 EKS 자동 모드를 사용하여 새 로드 밸런서를 생성합니다. 그런 다음 DNS 라우팅(예: Route 53)을 사용하여 트래픽을 이전 로드 밸런서에서 새 로드 밸런서로 점진적으로 이동합니다. 트래픽이 성공적으로 마이그레이션되고 새 구성을 확인한 후에는 이전 로드 밸런서와 자체 관리형 컨트롤러를 폐기할 수 있습니다.

기존 클러스터에서 EKS Auto Mode 활성화

이 주제에서는 기존 Amazon EKS 클러스터에서 Amazon EKS Auto Mode를 활성화하는 방법을 설명합니다. 기존 클러스터에서 Auto Mode를 활성화하려면 IAM 권한을 업데이트하고 코어 EKS Auto Mode 설정을 구성해야 합니다. 활성화되면 기존 컴퓨팅 워크로드의 마이그레이션을 시작하여 Auto Mode의 간소화된 운영과 자동화된 인프라 관리를 활용할 수 있습니다.

Important

EKS Auto Mode를 활성화하기 전에 특정 Amazon EKS 추가 기능의 최소 필수 버전이 설치되어 있는지 확인합니다. 자세한 내용은 [the section called “필요한 추가 기능 버전”](#) 섹션을 참조하세요.

시작하기 전에 Amazon EKS 클러스터에 대한 관리자 액세스 권한과 IAM 역할을 수정할 권한이 있는지 확인합니다. 이 주제의 단계는 AWS Management Console 또는 AWS CLI를 사용하여 자동 모드를 활성화하는 방법을 안내합니다.

AWS Management Console

IAM, EKS, EC2 리소스를 관리할 수 있는 권한을 가지고 AWS 콘솔에 로그인해야 합니다.

Note

클러스터가 생성된 후에는 EKS 클러스터의 클러스터 IAM 역할을 변경할 수 없습니다. EKS Auto Mode에는 이 역할에 대한 추가 권한이 필요합니다. 현재 역할에 추가 정책을 연결해야 합니다.

클러스터 IAM 역할 업데이트

1. AWS Management Console에서 클러스터 개요 페이지를 엽니다.
2. 클러스터 IAM 역할 ARN에서 IAM에서 보기를 선택합니다.
3. 권한 추가 드롭다운에서 정책 연결을 선택합니다.
4. 검색 상자를 사용하여 다음 정책을 찾아 선택합니다.
 - AmazonEKSCoordinatePolicy
 - AmazonEKSBLOCKStoragePolicy

- AmazonEKSLoadBalancingPolicy
- AmazonEKSNetworkingPolicy
- AmazonEKSClusterPolicy

5. 권한 추가를 선택합니다.

6. 신뢰 관계 탭에서 신뢰 정책 편집을 선택합니다.

7. 다음 클러스터 IAM 역할 신뢰 정책을 삽입하고 정책 업데이트를 선택합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

EKS Auto Mode 활성화

1. AWS Management Console에서 클러스터 개요 페이지를 엽니다.

2. EKS Auto Mode에서 관리를 선택합니다.

3. EKS Auto Mode를 켜기로 전환합니다.

4. EKS 노드 풀 드롭다운에서 생성할 기본 노드 풀을 선택합니다.

- EKS Auto Mode의 노드 풀에 대해 자세히 알아봅니다. 자세한 내용은 [the section called “노드 풀 생성”](#) 섹션을 참조하세요.

5. 이전에 이 AWS 계정의 EKS Auto Mode 노드 IAM 역할을 생성한 경우 노드 IAM 역할 드롭다운에서 해당 역할을 선택합니다. 이전에 이 역할을 생성하지 않은 경우 권장 역할 생성을 선택하고 단계를 따릅니다.

AWS CLI

사전 조건

- 기존 EKS 클러스터의 클러스터 IAM 역할에는 다음 정책과 같은 EKS Auto Mode에 대한 충분한 권한이 포함되어야 합니다.
 - AmazonEKSCoordinatePolicy
 - AmazonEKSBlockStoragePolicy
 - AmazonEKSLoadBalancingPolicy
 - AmazonEKSNetworkPolicy
 - AmazonEKSClusterPolicy
- 클러스터 IAM 역할에는 sts:TagSession 작업을 포함하여 업데이트된 신뢰 정책이 있어야 합니다. 클러스터 IAM 역할 생성에 대한 자세한 내용은 [the section called “AWS CLI”](#) 섹션을 참조하세요.
- aws CLI가 설치되고, 로그인되고, 충분한 버전이 있습니다. IAM, EKS 및 EC2 리소스를 관리할 권한이 있어야 합니다. 자세한 내용은 [설정](#) 섹션을 참조하세요.

절차

다음 명령을 사용하여 기존 클러스터에서 EKS Auto Mode를 활성화합니다.

Note

동일한 요청에서 컴퓨팅, 블록 스토리지, 로드 밸런싱 기능을 모두 활성화 또는 비활성화해야 합니다.

```
aws eks update-cluster-config \
  --name $CLUSTER_NAME \
  --compute-config enabled=true \
  --kubernetes-network-config '{"elasticLoadBalancing":{"enabled": true}}' \
  --storage-config '{"blockStorage":{"enabled": true}}'
```

필요한 추가 기능 버전

기존 클러스터에서 EKS Auto Mode를 활성화하려는 경우 특정 추가 기능을 업데이트해야 할 수 있습니다. 다음을 참고하세요.

- 이는 EKS Auto Mode로 전환하는 기존 클러스터에만 적용됩니다.
- EKS Auto Mode가 활성화된 상태로 생성된 새 클러스터에는 이러한 업데이트가 필요하지 않습니다.

다음 추가 기능 중 하나라도 설치되어 있는 경우 지정된 최소 버전 이상이어야 합니다.

추가 기능 이름	필요한 최소 버전
Kubernetes용 Amazon VPC CNI 플러그인	v1.19.0-eksbuild.1
Kube-proxy	• v1.26.15-eksbuild.19 • v1.27.16-eksbuild.14 • v1.28.15-eksbuild.4 • v1.29.10-eksbuild.3 • v1.30.6-eksbuild.3 • v1.31.2-eksbuild.3
Amazon EBS CSI 드라이버	v1.37.0-eksbuild.1
CSI 스냅샷 컨트롤러	v8.1.0-eksbuild.2
EKS Pod Identity 에이전트	v1.3.4-eksbuild.1

자세한 내용은 [the section called “추가 기능 업데이트”](#) 섹션을 참조하세요.

다음 단계

- 노드 그룹 워크로드 관리를 마이그레이션하려면 [the section called “MNG에서 마이그레이션”](#) 섹션을 참조하세요.
- 자체 관리형 Karpenter에서 마이그레이션하려면 [the section called “Karpenter에서 마이그레이션”](#) 섹션을 참조하세요.

kubectl을 사용하여 Karpenter에서 EKS Auto Mode로 마이그레이션

이 주제에서는 kubectl을 사용하여 Karpenter에서 Amazon EKS Auto Mode로 워크로드를 마이그레이션하는 프로세스를 안내합니다. 마이그레이션을 점진적으로 수행할 수 있으므로 전환 기간 동안 클러스터 안정성과 애플리케이션 가용성을 유지하면서 원하는 속도로 워크로드를 이동할 수 있습니다.

아래에 설명된 단계별 접근 방식을 사용하면 마이그레이션 기간에 Karpenter와 EKS Auto Mode를 나란히 실행할 수 있습니다. 이러한 이중 작업 전략은 Karpenter를 완전히 폐기하기 전에 EKS Auto Mode에서 워크로드 동작을 검증할 수 있게 하여 원활한 전환을 보장하는 데 도움이 됩니다. 애플리케이션을 개별적으로 또는 그룹으로 마이그레이션하여 특정 운영 요구 사항과 위험 허용 범위를 유연하게 수용할 수 있습니다.

사전 조건

마이그레이션을 시작하기 전에 다음이 있는지 확인합니다.

- 클러스터에 Karpenter v1.1 이상이 설치되어 있습니다. 자세한 내용은 Karpenter 설명서의 [Upgrading to 1.1.0+](#)를 참조하세요.
- kubectl이 설치되고 클러스터에 연결되었습니다. 자세한 내용은 [설정](#) 섹션을 참조하세요.

이 주제에서는 사용자가 Karpenter 및 NodePools를 잘 알고 있다고 가정합니다. 자세한 내용은 [Karpenter Documentation](#)를 참조하세요.

1단계: 클러스터에서 EKS Auto Mode 활성화

AWS CLI 또는 관리 콘솔을 사용하여 기존 클러스터에서 EKS Auto Mode를 활성화합니다. 자세한 내용은 [the section called “클러스터에서 활성화”](#) 섹션을 참조하세요.

Note

EKS Auto Mode를 활성화하는 동안 전환 중에 이 단계에서 general purpose 노드 풀을 활성화하지 마세요. 이 노드 풀은 선택 사항이 아닙니다.
자세한 내용은 [the section called “내장 노드 풀 검토”](#) 섹션을 참조하세요.

2단계: 테인트된 EKS Auto Mode NodePool 생성

테인트가 있는 EKS Auto Mode용 새 NodePool을 생성합니다. 이렇게 하면 기존 포드가 새 EKS Auto Mode 노드에서 자동으로 예약되지 않습니다. 이 노드 풀은 EKS Auto Mode에 내장된 default NodeClass를 사용합니다. 자세한 내용은 [the section called “노드 클래스 생성”](#) 섹션을 참조하세요.

테인트가 있는 노드 풀 예제:

```
apiVersion: karpenter.sh/v1
```



```
kind: NodePool
metadata:
  name: eks-auto-mode
spec:
  template:
    spec:
      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m", "r"]
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      taints:
        - key: "eks-auto-mode"
          effect: "NoSchedule"
```

마이그레이션의 소스에 해당하는 Karpenter 구성과 일치하도록 노드 풀의 요구 사항을 업데이트합니다. 하나 이상의 요구 사항이 필요합니다.

3단계: 마이그레이션할 워크로드 업데이트

EKS Auto Mode로 마이그레이션하려는 워크로드를 식별하고 업데이트합니다. 이러한 워크로드에 허용 값과 노드 선택기를 모두 추가합니다.

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      tolerations:
        - key: "eks-auto-mode"
          effect: "NoSchedule"
      nodeSelector:
        eks.amazonaws.com/compute-type: auto
```

이 변경 사항을 통해 새 EKS Auto Mode 노드에서 워크로드를 예약할 수 있습니다.

EKS Auto Mode는 Karpenter와 다른 레이블을 사용합니다. EC2 관리형 인스턴스와 관련된 레이블은 `eks.amazonaws.com`으로 시작합니다. 자세한 내용은 [the section called “노드 풀 생성”](#) 섹션을 참조하세요.

4단계: 점진적으로 워크로드 마이그레이션

마이그레이션하려는 각 워크로드에 대해 3단계를 반복합니다. 이를 통해 요구 사항 및 위험 허용 범위에 따라 워크로드를 개별적으로 또는 그룹으로 이동할 수 있습니다.

5단계: 원래 Karpenter NodePool 제거

모든 워크로드가 마이그레이션되면 원래 Karpenter NodePool을 제거할 수 있습니다.

```
kubectl delete nodepool <original-nodepool-name>
```

6단계: EKS Auto Mode NodePool에서 테인트 제거(선택 사항)

EKS Auto Mode가 새 워크로드의 기본값이 되게 하려면 EKS Auto Mode NodePool에서 테인트를 제거할 수 있습니다.

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: eks-auto-mode
spec:
  template:
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      # Remove the taints section
```

7단계: 워크로드에서 노드 선택기 제거(선택 사항)

EKS Auto Mode NodePool에서 테인트를 제거한 경우 이제 EKS Auto Mode가 기본값이므로 선택적으로 워크로드에서 노드 선택기를 제거할 수 있습니다.

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    spec:
      # Remove the nodeSelector section
```

```
tolerations:
- key: "eks-auto-mode"
  effect: "NoSchedule"
```

8단계: 클러스터에서 Karpenter 제거

Karpenter를 제거하는 단계는 설치 방법에 따라 다릅니다. 자세한 내용은 [Karpenter 설치 지침](#)을 참조하세요.

EKS 관리형 노드 그룹에서 EKS Auto Mode로 마이그레이션

Amazon EKS 클러스터를 EKS 자동 모드를 사용하도록 전환하면 eksctl CLI 도구를 사용하여 관리형 노드 그룹(MNG)에서 기존 워크로드를 원활하게 마이그레이션할 수 있습니다. 이 프로세스는 EKS Auto Mode가 컴퓨팅 리소스를 최적화하는 동안 지속적인 애플리케이션 가용성을 보장합니다. 실행 중인 애플리케이션의 중단을 최소화하면서 마이그레이션을 수행할 수 있습니다.

이 주제에서는 기존 관리형 노드 그룹에서 포드를 안전하게 드레이닝하고 EKS Auto Mode가 새로 프로비저닝된 인스턴스에서 포드를 다시 예약하도록 허용하는 단계를 안내합니다. 이 절차에 따라 마이그레이션 전반에 걸쳐 애플리케이션의 가용성을 유지하면서 EKS Auto Mode의 지능형 워크로드 통합을 활용할 수 있습니다.

사전 조건

- EKS Auto Mode가 활성화된 클러스터
- eksctl CLI가 설치되고 클러스터에 연결되었습니다. 자세한 내용은 [설정](#) 단원을 참조하십시오.
- Karpenter가 클러스터에 설치되지 않았습니다.

절차

다음 eksctl CLI 명령을 사용하여 기존 관리형 노드 그룹 인스턴스에서 포드 드레이닝을 시작합니다. EKS Auto Mode는 대체된 포드를 지원하는 새 노드를 생성합니다.

```
eksctl delete nodegroup --cluster=<clusterName> --name=<nodegroupName>
```

클러스터의 각 관리형 노드 그룹에 이 명령을 실행해야 합니다.

이 명령에 대한 자세한 내용은 eksctl 문서의 [노드 그룹 삭제 및 드레이닝](#)을 참조하세요.

EKS Fargate에서 EKS 자동 모드로 마이그레이션

이 주제에서는 `kubectl`을 사용하여 EKS Fargate에서 Amazon EKS 자동 모드로 워크로드를 마이그레이션하는 프로세스를 안내합니다. 마이그레이션을 점진적으로 수행할 수 있으므로 전환 기간 동안 클러스터 안정성과 애플리케이션 가용성을 유지하면서 원하는 속도로 워크로드를 이동할 수 있습니다.

아래에 설명된 단계별 접근 방식을 사용하면 마이그레이션 기간에 EKS Fargate와 EKS 자동 모드를 나란히 실행할 수 있습니다. 이러한 이중 작업 전략은 EKS Fargate를 완전히 폐기하기 전에 EKS 자동 모드에서 워크로드 동작을 검증할 수 있게 하여 원활한 전환을 보장하는 데 도움이 됩니다. 애플리케이션을 개별적으로 또는 그룹으로 마이그레이션하여 특정 운영 요구 사항과 위험 허용 범위를 유연하게 수용할 수 있습니다.

Amazon EKS 자동 모드와 AWS Fargate가 포함된 EKS 비교

EKS를 실행하려는 고객에게는 AWS Fargate가 포함된 Amazon EKS가 여전히 옵션으로 제공되지만, 앞으로는 Amazon EKS 자동 모드를 사용하는 것이 권장됩니다. EKS 자동 모드는 Kubernetes를 완벽하게 준수하며 Fargate가 지원할 수 없는 Istio와 같은 모든 업스트림 Kubernetes 기본 요소와 플랫폼 도구를 지원합니다. 또한 EKS 자동 모드는 GPU 및 스팟 인스턴스를 포함한 모든 EC2 런타임 구매 옵션을 완벽하게 지원하므로 고객은 협상된 EC2 할인 및 기타 절감 메커니즘을 활용할 수 있습니다. Fargate가 포함된 EKS를 사용할 때는 이러한 기능을 이용할 수 없습니다.

또한 EKS 자동 모드를 통해 고객은 각 EC2 인스턴스가 단일 애플리케이션 컨테이너를 실행하도록 표준 Kubernetes 예약 기능을 사용하여 Fargate와 동일한 격리 모델을 구현할 수 있습니다. Amazon EKS 자동 모드를 채택함으로써 고객은 Fargate가 제공하는 인프라 관리의 사용 편의성과 추상화를 유지하면서 모든 범위의 EC2 및 구매 옵션을 활용할 수 있는 유연성을 제공하는 완전한 Kubernetes 호환 플랫폼인 AWS에서 Kubernetes를 실행하는 모든 이점을 누릴 수 있습니다.

사전 조건

마이그레이션을 시작하기 전에 다음을 수행했는지 확인합니다.

- Fargate가 포함된 클러스터를 설정합니다. 자세한 내용은 [the section called “시작”](#) 섹션을 참조하세요.
- `kubectl`을 설치하고 클러스터에 연결합니다. 자세한 내용은 [설정](#) 섹션을 참조하세요.

1단계: Fargate 클러스터 확인

1. Fargate가 포함된 EKS 클러스터가 실행 중인지 확인합니다.

```
kubectl get node
```

```
NAME STATUS ROLES AGE VERSION
fargate-ip-192-168-92-52.ec2.internal Ready <none> 25m v1.30.8-eks-2d5f260
fargate-ip-192-168-98-196.ec2.internal Ready <none> 24m v1.30.8-eks-2d5f260
```

2. 실행 중인 포드를 확인합니다.

```
kubectl get pod -A
```

```
NAMESPACE NAME READY STATUS RESTARTS AGE
kube-system coredns-6659cb98f6-gxpjz 1/1 Running 0 26m
kube-system coredns-6659cb98f6-gzzsx 1/1 Running 0 26m
```

3. deployment_fargate.yaml이라는 파일에 배포를 생성합니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    annotations:
      eks.amazonaws.com/compute-type: fargate
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
        - containerPort: 80
```

4. 배포를 적용합니다.

```
kubectl apply -f deployment_fargate.yaml
```

```
deployment.apps/nginx-deployment created
```

5. 포트와 배포를 확인합니다.

```
kubectl get pod,deploy
```

NAME	READY	STATUS	RESTARTS	AGE
pod/nginx-deployment-5c7479459b-6trtm	1/1	Running	0	61s
pod/nginx-deployment-5c7479459b-g8ssb	1/1	Running	0	61s
pod/nginx-deployment-5c7479459b-mq4mf	1/1	Running	0	61s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/nginx-deployment	3/3	3	3	61s

6. 노드를 확인합니다.

```
kubectl get node -owide
```

NAME	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	STATUS	ROLES	AGE	VERSION
fargate-ip-192-168-111-43	192.168.111.43	<none>	Amazon Linux 2	Ready	<none>	31s	v1.30.8-eks-2d5f260
containerd://1.7.25							
fargate-ip-192-168-117-130	192.168.117.130	<none>	Amazon Linux 2	Ready	<none>	36s	v1.30.8-eks-2d5f260
containerd://1.7.25							
fargate-ip-192-168-74-140	192.168.74.140	<none>	Amazon Linux 2	Ready	<none>	36s	v1.30.8-eks-2d5f260
containerd://1.7.25							

2단계: 클러스터에서 EKS 자동 모드 활성화

1. AWS CLI 또는 관리 콘솔을 사용하여 기존 클러스터에서 EKS Auto Mode를 활성화합니다. 자세한 내용은 [the section called “클러스터에서 활성화”](#) 섹션을 참조하세요.
2. NodePool을 확인합니다.

```
kubectl get nodepool
```

NAME	NODECLASS	NODES	READY	AGE
general-purpose	default	1	True	6m58s
system	default	0	True	3d14h

3단계: 마이그레이션할 워크로드 업데이트

EKS Auto Mode로 마이그레이션하려는 워크로드를 식별하고 업데이트합니다.

Fargate에서 EKS 자동 모드로 워크로드를 마이그레이션하려면 주석 `eks.amazonaws.com/compute-type: ec2`를 적용합니다. 이렇게 하면 Fargate 프로파일에도 불구하고 Fargate에서 워크로드를 예약하지 않고 EKS 자동 모드 NodePool에서 워크로드를 포착할 수 있습니다. 자세한 내용은 [the section called “노드 풀 생성”](#) 섹션을 참조하세요.

1. 배포(예: `deployment_fargate.yaml` 파일)를 수정하여 컴퓨팅 유형을 `ec2`로 변경합니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
      annotations:
        eks.amazonaws.com/compute-type: ec2
    spec:
      containers:
      - name: nginx
        image: nginx
        ports:
```

```
- containerPort: 80
```

2. 배포를 적용합니다. 이 변경 사항을 통해 새 EKS 자동 모드 노드에서 워크로드를 예약할 수 있습니다.

```
kubectl apply -f deployment_fargate.yaml
```

3. 배포가 EKS 자동 모드 클러스터에서 실행 중인지 확인합니다.

```
kubectl get pod -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP
NODE	NOMINATED	READINESS	GATES		
nginx-deployment-97967b68d-ffxxh	1/1	Running	0	3m31s	
192.168.43.240	i-0845aafcb51630ffb	<none>	<none>		
nginx-deployment-97967b68d-mbcgj	1/1	Running	0	2m37s	
192.168.43.241	i-0845aafcb51630ffb	<none>	<none>		
nginx-deployment-97967b68d-qpd8x	1/1	Running	0	2m35s	
192.168.43.242	i-0845aafcb51630ffb	<none>	<none>		

4. EKS 자동 모드 관리형 노드에서 실행 중인 Fargate 노드와 배포가 없는지 확인합니다.

```
kubectl get node -owide
```

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP
IP OS-IMAGE					KERNEL-VERSION	CONTAINER-RUNTIME
i-0845aafcb51630ffb	Ready	<none>	3m30s	v1.30.8-eks-3c20087	192.168.41.125	
3.81.118.95	Bottlerocket	(EKS Auto)	2025.3.14	(aws-k8s-1.30)	6.1.129	
containerd://1.7.25+bottlerocket						

4단계: 점진적으로 워크로드 마이그레이션

마이그레이션하려는 각 워크로드에 대해 3단계를 반복합니다. 이를 통해 요구 사항 및 위험 허용 범위에 따라 워크로드를 개별적으로 또는 그룹으로 이동할 수 있습니다.

5단계: 원래 fargate 프로파일 제거

모든 워크로드가 마이그레이션되면 원래 fargate 프로파일을 제거할 수 있습니다. *<fargate profile name>*을 Fargate 프로파일의 이름으로 바꿉니다.


```
aws eks delete-fargate-profile --cluster-name eks-fargate-demo-cluster --fargate-profile-name <fargate profile name>
```

6단계: CoreDNS 스케일 다운

EKS 자동 모드는 CoreDNS를 처리하기 때문에 coredns 배포를 0으로 스케일 다운합니다.

```
kubectl scale deployment coredns -n kube-system --replicas=0
```

EKS 자율 모드 클러스터에서 샘플 워크로드 실행

이 장에서는 Auto Mode에서 실행되는 Amazon EKS 클러스터에 다양한 유형의 워크로드를 배포하는 방법의 예를 제공합니다. 이 예제에서는 샘플 애플리케이션, 로드 밸런싱된 웹 애플리케이션, 영구 스토리지를 사용하는 상태 저장 워크로드, 특정 노드 배치 요구 사항이 있는 워크로드를 포함한 주요 워크로드 패턴을 보여줍니다. 각 예제에는 자체 애플리케이션의 템플릿으로 사용할 수 있는 전체 매니페스트와 단계별 배포 지침이 포함되어 있습니다.

예제를 진행하기 전에 Auto Mode에서 실행 중인 EKS 클러스터가 있고 AWS CLI 및 kubectl을 설치했는지 확인합니다. 자세한 내용은 [설정](#) 단원을 참조하십시오. 이 예제에서는 Kubernetes 개념 및 kubectl 명령을 기본적으로 잘 알고 있다고 가정합니다.

이러한 사용 사례 기반 샘플을 사용하여 EKS Auto Mode 클러스터에서 워크로드를 실행할 수 있습니다.

[the section called “워크로드 팅창 배포”](#)

kubectl 명령을 사용하여 EKS Auto Mode 클러스터에 샘플 워크로드를 배포하는 방법을 보여줍니다.

[the section called “로드 밸런서 배포”](#)

Amazon EKS에서 2048 게임의 컨테이너화된 버전을 배포하는 방법을 보여줍니다.

[the section called “상태 저장 워크로드 배포”](#)

EKS Auto Mode 클러스터에 샘플 상태 저장 애플리케이션을 배포하는 방법을 보여줍니다.

[the section called “배포 제어”](#)

주석을 사용하여 EKS Auto Mode에서 관리하는 노드에 워크로드가 배포되는지 여부를 제어하는 방법을 보여줍니다.

Amazon EKS Auto Mode 클러스터에 샘플 팅창 워크로드 배포

이 자습서에서는 EKS Auto Mode 클러스터에 샘플 워크로드를 배포하고 필요한 컴퓨팅 리소스를 자동으로 프로비저닝하는 방법을 알아봅니다. `kubectl` 명령을 사용하여 클러스터의 동작을 관찰하고 Auto Mode가 AWS에서 Kubernetes 작업을 간소화하는 방법을 직접 확인할 수 있습니다. 이 자습서를 마치면 수동 노드 그룹 구성 없이 기본 컴퓨팅 리소스를 자동으로 관리하여 EKS Auto Mode가 워크로드 배포에 대응하는 방식을 이해할 수 있습니다.

사전 조건

- Amazon EKS Auto Mode 클러스터의 컴퓨팅 기능이 활성화되어 있습니다. 클러스터의 이름과 AWS 리전을 기록해 둡니다.
- IAM 보안 주체가 네트워킹, 컴퓨팅, EKS 리소스를 관리할 수 있는 충분한 권한이 있는 사용자 또는 역할입니다.
 - 자세한 내용은 IAM 사용 설명서의 [Creating roles and attaching policies in the IAM User Guide](#)을 참조하세요.
- aws CLI가 IAM 자격 증명으로 설치 및 구성되었습니다.
- `kubectl` CLI가 설치되고 클러스터에 연결되었습니다.
 - 자세한 내용은 [설정](#) 단원을 참조하십시오.

1단계: 기존 컴퓨팅 리소스 검토(선택 사항)

먼저 `kubectl`을 사용하여 클러스터의 노드 풀을 나열합니다.

```
kubectl get nodepools
```

샘플 출력:

```
general-purpose
```

이 자습서에서는 `general-purpose` 노드 풀을 사용하도록 구성된 워크로드를 배포합니다. 이 노드 풀은 EKS Auto Mode에 내장되어 있으며 마이크로서비스 및 웹 앱과 같은 일반 워크로드에 대한 합리적인 기본값을 포함합니다. 자체 노드 풀을 생성할 수 있습니다. 자세한 내용은 [the section called “노드 풀 생성”](#) 단원을 참조하십시오.

둘째, `kubectl`을 사용하여 클러스터에 연결된 노드를 나열합니다.

```
kubectl get nodes
```

방금 EKS Auto Mode 클러스터를 생성한 경우 노드가 없습니다.

이 자습서에서는 샘플 워크로드를 배포합니다. 노드가 없거나 워크로드가 기존 노드에 맞지 않는 경우 EKS Auto Mode가 새 노드를 프로비저닝합니다.

2단계: 클러스터에 샘플 애플리케이션 배포

다음 Kubernetes 배포를 검토하고 `inflate.yaml`로 저장합니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: inflate
spec:
  replicas: 1
  selector:
    matchLabels:
      app: inflate
  template:
    metadata:
      labels:
        app: inflate
    spec:
      terminationGracePeriodSeconds: 0
      nodeSelector:
        eks.amazonaws.com/compute-type: auto
      securityContext:
        runAsUser: 1000
        runAsGroup: 3000
        fsGroup: 2000
      containers:
        - name: inflate
          image: public.ecr.aws/eks-distro/kubernetes/pause:3.7
          resources:
            requests:
              cpu: 1
          securityContext:
            allowPrivilegeEscalation: false
```

`eks.amazonaws.com/compute-type: auto` 선택기를 사용하려면 Amazon EKS Auto Mode 노드에 워크로드를 배포해야 합니다.

배포를 클러스터에 적용합니다.

```
kubectl apply -f inflate.yaml
```

3단계: Kubernetes 이벤트 보기

다음 명령을 사용하여 새 노드 생성을 포함한 Kubernetes 이벤트를 관찰합니다. `ctrl+c`를 사용하여 이벤트 시청을 중지합니다.

```
kubectl get events -w --sort-by '.lastTimestamp'
```

`kubectl`을 사용하여 클러스터에 연결된 노드를 다시 나열합니다. 새로 생성된 노드를 기록해 둡니다.

```
kubectl get nodes
```

4단계: AWS 콘솔에서 노드 및 인스턴스 보기

EKS 콘솔에서 EKS Auto Mode 노드를 보고 EC2 콘솔에서 연결된 EC2 인스턴스를 볼 수 있습니다.

EKS Auto Mode에서 배포한 EC2 인스턴스는 제한됩니다. EKS Auto Mode 노드에서는 임의의 명령을 실행할 수 없습니다.

5단계: 배포 삭제

`kubectl`을 사용하여 샘플 배포를 삭제합니다.

```
kubectl delete -f inflate.yaml
```

클러스터에 배포된 다른 워크로드가 없는 경우 EKS Auto Mode에서 생성한 노드는 비어 있습니다.

기본 구성에서 EKS Auto Mode는 30초 동안 비어 있는 노드를 감지하고 종료합니다.

`kubectl` 또는 EC2 콘솔을 사용하여 연결된 인스턴스가 삭제되었는지 확인합니다.

EKS 자율 모드에 샘플 로드 밸런서 워크로드 배포

이 가이드에서는 로드 밸런싱 및 인터넷 접근성이 포함된 Amazon EKS에 2048 게임의 컨테이너화된 버전을 배포하는 방법을 안내합니다.

사전 조건

- EKS Auto Mode 클러스터
- kubectl이 클러스터와 상호 작용하도록 구성됨
- ALB 리소스 생성을 위한 적절한 IAM 권한

1단계: 네임스페이스 생성

먼저 2048 게임 애플리케이션을 위한 전용 네임스페이스를 생성합니다.

01-namespace.yaml라는 이름의 파일을 만듭니다.

```
apiVersion: v1
kind: Namespace
metadata:
  name: game-2048
```

네임스페이스 구성을 적용합니다.

```
kubectl apply -f 01-namespace.yaml
```

2단계: 애플리케이션 배포

애플리케이션은 2048 게임 컨테이너의 여러 복제본을 실행합니다.

02-deployment.yaml라는 이름의 파일을 만듭니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: game-2048
  name: deployment-2048
spec:
  selector:
    matchLabels:
      app.kubernetes.io/name: app-2048
  replicas: 5
  template:
    metadata:
      labels:
        app.kubernetes.io/name: app-2048
```

```
spec:
  containers:
    - image: public.ecr.aws/l6m2t8p7/docker-2048:latest
      imagePullPolicy: Always
      name: app-2048
      ports:
        - containerPort: 80
      resources:
        requests:
          cpu: "0.5"
```

Note

이미지 `public.ecr.aws/l6m2t8p7/docker-2048:latest`를 로드하는 동안 오류가 발생하면 노드 IAM 역할에 ECR에서 이미지를 가져올 수 있는 충분한 권한이 있는지 확인합니다. 자세한 내용은 [the section called “노드 IAM 역할”](#) 섹션을 참조하세요. 또한 예제의 `docker-2048` 이미지는 `x86_64` 이미지이며 다른 아키텍처에서는 실행되지 않습니다.

핵심 구성 요소:

- 애플리케이션의 복제본 5개 배포
- 퍼블릭 ECR 이미지 사용
- 포드당 0.5 CPU 코어 요청
- HTTP 트래픽에 포트 80 노출

배포를 적용합니다.

```
kubectl apply -f 02-deployment.yaml
```

3단계: 서비스 생성

서비스는 배포를 클러스터 네트워크에 노출합니다.

`03-service.yaml`라는 이름의 파일을 만듭니다.

```
apiVersion: v1
kind: Service
metadata:
```

```
namespace: game-2048
name: service-2048
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: NodePort
  selector:
    app.kubernetes.io/name: app-2048
```

핵심 구성 요소:

- NodePort 서비스 생성
- 포트 80을 컨테이너의 포트 80에 매핑
- 레이블 선택기를 사용하여 포드 찾기

서비스를 적용합니다.

```
kubectl apply -f 03-service.yaml
```

4단계: 로드 밸런싱 구성

애플리케이션을 인터넷에 노출하도록 수신을 설정합니다.

먼저 IngressClass를 생성합니다. 04-ingressclass.yaml라는 이름의 파일을 만듭니다.

```
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  labels:
    app.kubernetes.io/name: LoadBalancerController
  name: alb
spec:
  controller: eks.amazonaws.com/alb
```

Note

EKS 자동 모드에서는 퍼블릭 및 프라이빗 서브넷을 식별하기 위해 서브넷 태그가 필요합니다. eksctl로 클러스터를 생성한 경우 이미 이러한 태그가 있는 것입니다.

[the section called “서브�트에 태그 지정”](#) 방법에 대해 알아봅니다.

그런 다음 수신 리소스를 생성합니다. 05-ingress.yaml라는 이름의 파일을 만듭니다.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  namespace: game-2048
  name: ingress-2048
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: service-2048
                port:
                  number: 80
```

핵심 구성 요소:

- 인터넷 연결 ALB 생성
- 직접 포트 라우팅에 IP 대상 유형 사용
- 모든 트래픽(/)을 게임 서비스로 라우팅

수신 구성을 적용합니다.

```
kubectl apply -f 04-ingressclass.yaml
kubectl apply -f 05-ingress.yaml
```

5단계: 배포 확인

1. 모든 포드가 실행 중인지 확인합니다.


```
kubectl get pods -n game-2048
```

2. 서비스가 생성되었는지 확인합니다.

```
kubectl get svc -n game-2048
```

3. ALB 엔드포인트를 가져옵니다.

```
kubectl get ingress -n game-2048
```

수신 출력의 ADDRESS 필드에 ALB 엔드포인트가 표시됩니다. ALB가 모든 대상을 프로비저닝하고 등록할 때까지 2~3분 기다립니다.

6단계: 게임에 액세스

웹 브라우저를 열고 이전 단계에서 ALB 엔드포인트 URL로 이동합니다. 2048 게임 인터페이스가 표시됩니다.

7단계: 정리

이 자습서에서 만든 리소스를 모두 제거합니다.

```
kubectl delete namespace game-2048
```

그러면 배포, 서비스 및 수신 리소스를 포함하여 네임스페이스의 모든 리소스가 삭제됩니다.

비하인드 스토리

1. 배포를 통해 2048 게임을 실행하는 포드 5개 생성
2. 해당 서비스는 이러한 포드에 대한 안정적인 네트워크 액세스 제공
3. EKS Auto Mode:
 - AWS에서 Application Load Balancer 생성
 - 포드의 대상 그룹 구성
 - 트래픽이 서비스로 전달되도록 라우팅 규칙 설정

문제 해결

게임이 로드되지 않는 경우 다음을 수행합니다.

- 모든 포드가 실행 중인지 확인: `kubectl get pods -n game-2048`
- 수신 상태 확인: `kubectl describe ingress -n game-2048`
- ALB 상태 확인: AWS 콘솔에서 대상 그룹 상태 확인

EKS Auto Mode에 샘플 상태 저장 워크로드 배포

이 자습서에서는 EKS Auto Mode 클러스터에 샘플 상태 저장 애플리케이션을 배포하는 방법을 안내합니다. 애플리케이션은 영구 볼륨에 타임스탬프를 기록하여 EKS Auto Mode의 자동 EBS 볼륨 프로비저닝 및 지속성 기능을 보여줍니다.

사전 조건

- EKS Auto Mode 클러스터
- 적절한 권한으로 구성된 AWS CLI
- `kubectl` 설치 및 구성
 - 자세한 내용은 [설정](#) 단원을 참조하십시오.

1단계: 환경 구성

1. 환경 변수를 설정합니다.

```
export CLUSTER_NAME=my-auto-cluster
export AWS_REGION="us-west-2"
```

2. `kubeconfig`를 업데이트합니다.

```
aws eks update-kubeconfig --name "${CLUSTER_NAME}"
```

2단계: 스토리지 클래스 생성

`StorageClass`는 EKS 자동 모드가 EBS 볼륨을 프로비저닝하는 방법을 정의합니다.

EKS 자동 모드는 StorageClass를 생성하지 않습니다. EKS 자동 모드의 스토리지 기능을 사용하려면 `ebs.csi.eks.amazonaws.com`을 참조하는 StorageClass를 생성해야 합니다.

1. `storage-class.yaml`라는 이름의 파일을 만듭니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: auto-ebs-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
```

2. StorageClass를 적용합니다.

```
kubectl apply -f storage-class.yaml
```

핵심 구성 요소:

- `provisioner: ebs.csi.eks.amazonaws.com`-EKS Auto Mode를 사용합니다.
- `volumeBindingMode: WaitForFirstConsumer`-포드에 필요할 때까지 볼륨 생성을 지연합니다.
- `type: gp3`-EBS 볼륨 유형을 지정합니다.
- `encrypted: "true"`-EBS는 기본 `aws/ebs` 키를 사용하여 이 클래스로 생성된 볼륨을 암호화합니다. 이는 선택 사항이며, 권장 사항은 아닙니다.
- `storageclass.kubernetes.io/is-default-class: "true"`-영구 볼륨 클레임에서 다른 볼륨 클래스를 지정하지 않는 한 Kubernetes는 기본적으로 이 스토리지 클래스를 사용합니다. 다른 스토리지 컨트롤러에서 마이그레이션하는 경우 이 값을 설정할 때 주의하세요. (선택 사항)

3단계: 영구 볼륨 클레임 생성

PVC는 StorageClass에서 스토리지를 요청합니다.

1. `pvc.yaml`라는 이름의 파일을 만듭니다.

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: auto-ebs-claim
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: auto-ebs-sc
  resources:
    requests:
      storage: 8Gi

```

2. PVC를 적용합니다.

```
kubectl apply -f pvc.yaml
```

핵심 구성 요소:

- `accessModes: ReadWriteOnce`-한 번에 한 노드씩 볼륨 탑재 가능
- `storage: 8Gi`-8GiB 볼륨 요청
- `storageClassName: auto-ebs-sc` - 생성한 `StorageClass`를 참조합니다.

4단계: 애플리케이션 배포

배포는 영구 볼륨에 타임스탬프를 기록하는 컨테이너를 실행합니다.

1. `deployment.yaml`라는 이름의 파일을 만듭니다.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: inflate-stateful
spec:
  replicas: 1
  selector:
    matchLabels:
      app: inflate-stateful
  template:
    metadata:

```

```
labels:
  app: inflate-stateful
spec:
  terminationGracePeriodSeconds: 0
  nodeSelector:
    eks.amazonaws.com/compute-type: auto
  containers:
    - name: bash
      image: public.ecr.aws/docker/library/bash:4.4
      command: ["/usr/local/bin/bash"]
      args: ["-c", "while true; do echo $(date -u) >> /data/out.txt; sleep 60;
done"]
  resources:
    requests:
      cpu: "1"
    volumeMounts:
      - name: persistent-storage
        mountPath: /data
  volumes:
    - name: persistent-storage
      persistentVolumeClaim:
        claimName: auto-ebs-claim
```

2. 배포를 적용합니다.

```
kubectl apply -f deployment.yaml
```

핵심 구성 요소:

- 파일에 타임스탬프를 기록하는 단순 Bash 컨테이너
- /data에 PVC 탑재
- CPU 코어 1개 요청
- EKS 관리형 노드에 노드 선택기 사용

5단계: 설정 확인

1. 포드가 실행 중인지 확인합니다.

```
kubectl get pods -l app=inflate-stateful
```

2. PVC가 바인딩되었는지 확인합니다.

```
kubectl get pvc auto-ebs-claim
```

3. EBS 볼륨을 확인합니다.

```
# Get the PV name
PV_NAME=$(kubectl get pvc auto-ebs-claim -o jsonpath='{.spec.volumeName}')
# Describe the EBS volume
aws ec2 describe-volumes \
  --filters Name=tag:CSIVolumeName,Values=${PV_NAME}
```

4. 데이터가 작성되고 있는지 확인합니다.

```
kubectl exec "$(kubectl get pods -l app=inflate-stateful \
  -o=jsonpath='{.items[0].metadata.name}')" -- \
  cat /data/out.txt
```

6단계: 정리

다음 명령을 실행하여 이 자습서에서 만든 리소스를 모두 제거합니다.

```
# Delete all resources in one command
kubectl delete deployment/inflate-stateful pvc/auto-ebs-claim storageclass/auto-ebs-sc
```

비하인드 스토리

1. PVC는 StorageClass에서 스토리지를 요청합니다.
2. 포드가 예약된 경우:
 - a. EKS Auto Mode는 EBS 볼륨을 프로비저닝합니다.
 - b. PersistentVolume을 생성합니다.
 - c. 노드에 볼륨을 연결합니다.
3. 포드가 볼륨을 탑재하고 타임스탬프 작성을 시작합니다.

스냅샷 컨트롤러

EKS Auto Mode는 스냅샷 컨트롤러라고도 하는 Kubernetes CSI Snapshotter와 호환됩니다. 그러나 EKS Auto Mode에는 스냅샷 컨트롤러가 포함되지 않습니다. 스냅샷 컨트롤러의 설치와 구성은 사용자의 책임입니다. 자세한 내용은 [the section called “CSI 스냅샷 컨트롤러”](#) 단원을 참조하십시오.

EKS Auto Mode의 스토리지 기능을 참조하는 다음 VolumeSnapshotClass를 검토합니다.

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: auto-ebs-vsclass
driver: ebs.csi.eks.amazonaws.com
deletionPolicy: Delete
```

[Kubernetes CSI Snapshotter에 대해 자세히 알아보세요.](#)

EKS 자율 모드 설정 구성

이 장에서는 Amazon Elastic Kubernetes Service(EKS) Auto Mode 클러스터의 특정 측면을 구성하는 방법을 설명합니다. EKS Auto Mode는 대부분의 인프라 구성 요소를 자동으로 관리하지만 워크로드 요구 사항에 맞게 특정 기능을 사용자 지정할 수 있습니다.

이 주제에 설명된 구성 옵션을 사용하여 자동화된 인프라 관리의 이점을 유지하면서 네트워킹 설정, 컴퓨팅 리소스, 로드 밸런싱 동작을 수정할 수 있습니다. 구성을 변경하기 전에 다음 섹션에서 사용할 수 있는 옵션을 검토하여 필요에 가장 적합한 접근 방식을 결정합니다.

구성하려는 기능	구성 옵션
노드 네트워킹 및 스토리지 - 퍼블릭 및 프라이빗 서브넷에서 노드 배치 구성 - 노드 액세스 제어를 위한 사용자 지정 보안 그룹 정의 - 네트워크 주소 변환(SNAT) 정책 사용자 지정 - Kubernetes 네트워크 정책, 세부 네트워크 정책 로깅 및 모니터링 활성화	the section called “노드 클래스 생성”

구성하려는 기능	구성 옵션
- 임시 스토리지 파라미터 설정(크기, IOPS, 처리량) - 사용자 지정 KMS 키를 사용하여 암호화된 임시 스토리지 구성 - 노드와 별도의 서브넷에서 포트 트래픽 격리	
노드 컴퓨팅 리소스 - 특정 EC2 인스턴스 유형 및 패밀리 선택 - CPU 아키텍처 정의(x86_64, ARM64) - 용량 유형 구성(온디맨드, 스팟) - 가용 영역 지정 - 노드 테인트 및 레이블 구성 - 최소 및 최대 노드 수 설정	the section called “노드 풀 생성”
Application Load Balancer 설정 - 내부 또는 인터넷 연결 로드 밸런서 배포 - 교차 영역 로드 밸런싱 구성 - 유휴 제한 시간 설정 - HTTP/2 및 WebSocket 지원 활성화 - 상태 확인 파라미터 구성 - TLS 인증서 설정 지정 - 대상 그룹 속성 정의 - IP 주소 유형 설정(IPv4, 듀얼 스택)	the section called “수신 클래스 생성”

구성하려는 기능	구성 옵션
Network Load Balancer 설정 - 직접 포트 IP 라우팅 구성 - 교차 영역 로드 밸런싱 활성화 - 연결 유효 제한 시간 설정 - 상태 확인 파라미터 구성 - 서브넷 배치 지정 - IP 주소 유형 설정(IPv4, 듀얼 스택) - 클라이언트 소스 IP 보존 구성 - 대상 그룹 속성 정의	the section called “서비스 생성”
스토리지 클래스 설정 - EBS 볼륨 유형 정의(gp3, io1, io2 등) - 볼륨 암호화 및 KMS 키 사용 구성 - IOPS 및 처리량 파라미터 설정 - 기본 스토리지 클래스로 설정 - 프로비저닝된 볼륨에 대한 사용자 지정 태그 정의	the section called “StorageClass 생성”

Amazon EKS용 노드 클래스 생성

Amazon EKS 노드 클래스는 EKS 자동 모드 관리형 노드의 구성을 세부적으로 제어할 수 있는 템플릿입니다. 노드 클래스는 네트워크 구성, 스토리지 설정, 리소스 태그 지정을 포함하여 EKS 클러스터의 노드 그룹에 적용되는 인프라 수준 설정을 정의합니다. 이 주제에서는 특정 운영 요구 사항에 맞게 노드 클래스를 생성하고 구성하는 방법을 설명합니다.

EKS Auto Mode가 기본 설정 이상으로 EC2 인스턴스를 프로비저닝하고 구성하는 방법을 사용자 지정해야 하는 경우, 노드 클래스를 생성하면 중요한 인프라 파라미터를 정확하게 제어할 수 있습니다. 예를 들어 보안 강화를 위해 프라이빗 서브넷 배치를 지정하거나, 성능에 민감한 워크로드에 대해 인스턴스 임시 스토리지를 구성하거나, 비용 할당을 위해 사용자 지정 태그 지정을 적용할 수 있습니다.

노드 클래스 생성

NodeClass를 생성하려면 다음 단계를 따르세요.

1. 노드 클래스 구성으로 YAML 파일(예: nodeclass.yaml) 생성
2. kubectl을 사용하여 클러스터에 해당 구성 적용
3. 노드 풀 구성에서 노드 클래스를 참조하세요. 자세한 내용은 [the section called “노드 풀 생성”](#) 섹션을 참조하세요.

kubectl을 설치하고 구성해야 합니다. 자세한 내용은 [설정](#) 섹션을 참조하세요.

기본 노드 클래스 예제

다음은 노드 클래스 예제입니다.

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: private-compute
spec:
  ephemeralStorage:
    size: "160Gi"
```

이 NodeClass는 노드의 임시 스토리지 양을 늘립니다.

다음을 사용하여 이 구성을 적용합니다.

```
kubectl apply -f nodeclass.yaml
```

그런 다음 노드 풀 구성에서 노드 클래스를 참조합니다. 자세한 내용은 [the section called “노드 풀 생성”](#) 섹션을 참조하세요.

노드 클래스 액세스 항목 생성

사용자 지정 노드 클래스를 생성하는 경우 노드가 클러스터에 조인할 수 있도록 EKS 액세스 항목을 생성해야 합니다. EKS는 내장 노드 클래스와 노드 풀을 사용할 때 액세스 항목을 자동으로 생성합니다.

Access Entries 작동 방법에 대한 자세한 내용은 [the section called “권한 부여”](#) 섹션을 참조하세요.

EKS 자동 모드 노드 클래스에 대한 액세스 항목을 생성할 때 EC2 액세스 항목 유형을 사용해야 합니다.

CLI를 사용하여 액세스 항목 생성

EC2 노드에 대한 액세스 항목 생성 및 EKS 자동 모드 정책 연결

클러스터 이름과 노드 역할 ARN으로 다음 CLI 명령을 업데이트합니다. 노드 역할 ARN은 노드 클래스 YAML에 지정됩니다.

```
# Create the access entry for EC2 nodes
aws eks create-access-entry \
  --cluster-name <cluster-name> \
  --principal-arn <node-role-arn> \
  --type EC2

# Associate the auto node policy
aws eks associate-access-policy \
  --cluster-name <cluster-name> \
  --principal-arn <node-role-arn> \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy \
  --access-scope type=cluster
```

CloudFormation을 사용하여 액세스 항목 생성

EC2 노드에 대한 액세스 항목 생성 및 EKS 자동 모드 정책 연결

클러스터 이름과 노드 역할 ARN으로 다음 CLI 명령을 업데이트합니다. 노드 역할 ARN은 노드 클래스 YAML에 지정됩니다.

```
EKSAutoNodeRoleAccessEntry:
  Type: AWS::EKS::AccessEntry
  Properties:
    ClusterName: <cluster-name>
    PrincipalArn: <node-role-arn>
    Type: "EC2"
    AccessPolicies:
      - AccessScope:
          Type: cluster
          PolicyArn: arn:aws:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy
    DependsOn: [ <cluster-name> ] # previously defined in CloudFormation
```

CloudFormation 스택 배포에 대한 자세한 내용은 [CloudFormation 시작하기](#)를 참조하세요.

노드 클래스 사양

```

apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: my-node-class
spec:
  # Required fields
  role: MyNodeRole # IAM role for EC2 instances

  subnetSelectorTerms:
    - tags:
        Name: "private-subnet"
        kubernetes.io/role/internal-elb: "1"
      # Alternative using direct subnet ID
      # - id: "subnet-0123456789abcdef0"

  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"
      # Alternative approaches:
      # - id: "sg-0123456789abcdef0"
      # - name: "eks-cluster-security-group"

  # Optional: Pod subnet selector for advanced networking
  podSubnetSelectorTerms:
    - tags:
        Name: "pod-subnet"
        kubernetes.io/role/pod: "1"
      # Alternative using direct subnet ID
      # - id: "subnet-0987654321fedcba0"

  # Optional fields
  snatPolicy: Random # or Disabled

  networkPolicy: DefaultAllow # or DefaultDeny
  networkPolicyEventLogs: Disabled # or Enabled

  ephemeralStorage:
    size: "80Gi" # Range: 1-59000Gi or 1-64000G or 1-58Ti or 1-64T
    iops: 3000 # Range: 3000-16000
    throughput: 125 # Range: 125-1000
    # Optional KMS key for encryption

```

```

kmsKeyID: "arn:aws:kms:region:account:key/key-id"
# Accepted formats:
# KMS Key ID
# KMS Key ARN
# Key Alias Name
# Key Alias ARN

# Optional: Forward proxy, commonly requires certificateBundles as well
#for EC2, see https://repost.aws/knowledge-center/eks-http-proxy-containerd-
automation
advancedNetworking:
  httpsProxy: http://192.0.2.4:3128 #commonly port 3128 (Squid) or 8080 (NGINX) #Max
255 characters
  #httpsProxy: http://[2001:db8::4]:3128 # IPv6 address with port, use []
  noProxy: #Max 50 entries
    - localhost #Max 255 characters each
    - 127.0.0.1
    #- ::1 # IPv6 localhost
    #- 0:0:0:0:0:0:0:1 # IPv6 localhost
    - 169.254.169.254 # EC2 Instance Metadata Service
    #- [fd00:ec2::254] # IPv6 EC2 Instance Metadata Service
    # Domains to exclude, put all VPC endpoints here
    - .internal
    - .eks.amazonaws.com

# Optional: Custom certificate bundles.
certificateBundles:
  - name: "custom-cert"
    data: "base64-encoded-cert-data"

# Optional: Additional EC2 tags (with restrictions)
tags:
  Environment: "production"
  Team: "platform"
# Note: Cannot use restricted tags like:
# - kubernetes.io/cluster/*
# - karpenter.sh/provisioner-name
# - karpenter.sh/nodepool
# - karpenter.sh/nodeclaim
# - karpenter.sh/managed-by
# - eks.amazonaws.com/nodeclass

```

고려 사항

- **노드 IAM 역할 바꾸기** - NodeClass와 연결된 노드 IAM 역할을 변경하는 경우 새 액세스 항목을 생성해야 합니다. EKS는 클러스터 생성 중 노드 IAM 역할에 대한 액세스 항목을 자동으로 생성합니다. 노드 IAM 역할에는 AmazonEKSAutoNodePolicy EKS 액세스 정책이 필요합니다. 자세한 내용은 [the section called “권한 부여”](#) 섹션을 참조하세요.
- **최대 포드 밀도** - EKS는 노드의 최대 포드 수를 110개로 제한합니다. 이 제한은 기존의 최대 포드 계산 후에 적용됩니다. 자세한 내용은 [the section called “Amazon EC2 인스턴스 유형”](#) 섹션을 참조하세요.
- **태그** - Kubernetes에서 EC2로 태그를 전파하려면 추가 IAM 권한을 구성해야 합니다. 자세한 내용은 [the section called “자격 증명 및 액세스”](#) 섹션을 참조하세요.
- **기본 노드 클래스** - 사용자 지정 노드 클래스의 이름을 default로 지정하지 마세요. 이는 EKS 자동 모드에 내장 NodePool을 하나 이상 활성화하면 자동으로 프로비저닝되는 default라는 NodeClass가 포함되어 있기 때문입니다. 내장 NodePools 활성화에 대한 자세한 내용은 [the section called “내장 노드 풀 검토”](#) 섹션을 참조하세요.
- **여러 서브넷이 있는 subnetSelectorTerms 동작** - subnetSelectorTerms 조건과 일치하거나 ID로 제공하는 서브넷이 여러 개 있는 경우 EKS 자동 모드는 서브넷 전체에 분산된 노드를 생성합니다.
 - 서브넷이 서로 다른 가용 영역(AZ)에 있는 경우 [포드 토폴로지 확산 제약 조건](#) 및 [토폴로지 인식 라우팅](#)과 같은 Kubernetes 기능을 사용하여 각각 영역에 걸쳐 포드와 트래픽을 확산할 수 있습니다.
 - 동일한 AZ에 subnetSelectorTerms와 일치하는 서브넷이 여러 개 있는 경우 EKS 자동 모드는 해당 AZ의 서브넷에 분산된 각 노드에 Pod를 생성합니다. EKS 자동 모드는 동일한 AZ의 다른 서브넷에 있는 각 노드에 보조 네트워크 인터페이스를 생성합니다. 각 서브넷에서 사용 가능한 IP 주소 수를 기준으로 선택하여 서브넷을 보다 효율적으로 사용합니다. 하지만 EKS 자동 모드가 각 포드에 대해 어떤 서브넷을 사용할지는 지정할 수 없습니다. 포드가 특정 서브넷에서 실행되도록 하려면 대신 [the section called “포드의 서브넷 선택”](#)를 사용하세요.

포드의 서브넷 선택

podSubnetSelectorTerms 및 podSecurityGroupSelectorTerms 필드를 사용하면 포드가 노드와 다른 서브넷에서 실행되도록 하여 고급 네트워킹 구성을 사용할 수 있습니다. 이 분리는 네트워크 트래픽 라우팅 및 보안 정책에 대한 향상된 제어를 제공합니다. podSecurityGroupSelectorTerms는 podSubnetSelectorTerms와 함께 필요합니다.

사용 사례

다음 작업 시 podSubnetSelectorTerms를 사용합니다.

- 애플리케이션 트래픽(포드 간 통신)에서 인프라 트래픽(노드 간 통신)을 분리합니다.
- 포드 서브넷과 다른 네트워크 구성을 노드 서브넷에 적용합니다.
- 노드 및 포드에 대해 서로 다른 보안 정책 또는 라우팅 규칙을 구현합니다.
- 포드 트래픽에 영향을 주지 않고 노드 트래픽에 대해 특히 역방향 프록시 또는 네트워크 필터링을 구성합니다. advancedNetworking 및 certificateBundles를 사용하여 역방향 프록시와 프록시에 대한 자체 서명 또는 프라이빗 인증서를 정의합니다.

구성의 예

```
apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: advanced-networking
spec:
  role: MyNodeRole

  # Subnets for EC2 instances (nodes)
  subnetSelectorTerms:
    - tags:
        Name: "node-subnet"
        kubernetes.io/role/internal-elb: "1"

  securityGroupSelectorTerms:
    - tags:
        Name: "eks-cluster-sg"

  # Separate subnets for pods
  podSubnetSelectorTerms:
    - tags:
        Name: "pod-subnet"
        kubernetes.io/role/pod: "1"

  podSecurityGroupSelectorTerms:
    - tags:
        Name: "eks-pod-sg"
```

포드의 서브넷 선택기에 대한 고려 사항

- 포드 밀도 감소: 노드의 기본 네트워크 인터페이스가 노드 서브넷에 있고 포드 서브넷의 포드에 사용할 수 없으므로 podSubnetSelectorTerms 사용 시 각 노드에서 더 적은 포드를 실행할 수 있습니다.
- 서브넷 선택기 제한 사항: 포드 서브넷 선택에는 표준 subnetSelectorTerms 및 securityGroupSelectorTerms 구성이 적용되지 않습니다.
- 네트워크 계획: 워크로드 요구 사항을 지원하기 위해 노드와 포드 서브넷 모두에서 적절한 IP 주소 공간을 확보합니다.
- 라우팅 구성: 포드 서브넷의 라우팅 테이블과 네트워크 액세스 제어 목록(ACL)이 노드와 포드 서브넷 간의 통신을 위해 올바르게 구성되어 있는지 확인합니다.

EKS Auto Mode용 노드 풀 생성

Amazon EKS 노드 풀은 Kubernetes 클러스터에서 컴퓨팅 리소스를 관리하는 유연한 방법을 제공합니다. 이 주제에서는 클러스터 조정 및 리소스 사용률을 최적화하는 데 도움이 되는 노드 프로비저닝 도구인 Karpenter를 사용하여 노드 풀을 생성하고 구성하는 방법을 보여줍니다. Karpenter의 NodePool 리소스를 사용하면 인스턴스 유형, 가용 영역, 아키텍처, 용량 유형을 포함한 컴퓨팅 리소스에 대한 특정 요구 사항을 정의할 수 있습니다.

내장 system 및 general-purpose 노드 풀은 수정할 수 없습니다. 활성화 또는 비활성화만 가능합니다. 자세한 내용은 [the section called “내장 노드 풀 검토”](#) 섹션을 참조하세요.

NodePool 사양을 사용하면 지원되는 다양한 레이블 및 요구 사항을 통해 EKS 클러스터의 컴퓨팅 리소스를 세밀하게 제어할 수 있습니다. 여기에는 EC2 인스턴스 범주, CPU 구성, 가용 영역, 아키텍처 (ARM64/AMD64), 용량 유형(스팟 또는 온디맨드)을 지정하는 옵션이 포함됩니다. 또한 CPU 및 메모리 사용량에 대한 리소스 제한을 설정하여 클러스터가 필요한 운영 경계 내에 있게 할 수 있습니다.

EKS Auto Mode는 잘 알려진 Kubernetes 레이블을 활용하여 일관되고 표준화된 노드 특성 식별 방법을 제공합니다. 가용 영역의 topology.kubernetes.io/zone 및 CPU 아키텍처의 kubernetes.io/arch와 같은 이러한 레이블은 설정된 Kubernetes 규칙을 따릅니다. 또한 EKS별 레이블(접두어 eks.amazonaws.com/)은 인스턴스 유형, CPU 제조업체, GPU 기능, 네트워킹 사양 등의 AWS 특정 속성으로 이 기능을 확장합니다. 이 표준화된 레이블 지정 시스템을 사용하면 기존 Kubernetes 도구와 원활하게 통합하는 동시에 심층적인 AWS 인프라 통합을 제공할 수 있습니다.

NodePool 생성

다음 단계에 따라 Amazon EKS 클러스터용 NodePool을 생성합니다.

1. 필요한 NodePool 구성으로 `nodepool1.yaml`이라는 YAML 파일을 생성합니다. 아래 샘플 구성을 사용할 수 있습니다.
2. 클러스터에 NodePool을 적용합니다.

```
kubectl apply -f nodepool1.yaml
```

3. NodePool이 성공적으로 생성되었는지 확인합니다.

```
kubectl get nodepools
```

4. (선택 사항) NodePool 상태를 모니터링합니다.

```
kubectl describe nodepool default
```

NodePool이 클러스터에 있는 유효한 NodeClass를 참조하는지 확인합니다. NodeClass는 컴퓨팅 리소스에 대한 AWS 특정 구성을 정의합니다. 자세한 내용은 [the section called “노드 클래스 생성”](#) 섹션을 참조하세요.

샘플 NodePool

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: my-node-pool
spec:
  template:
    metadata:
      labels:
        billing-team: my-team
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "eks.amazonaws.com/instance-category"
          operator: In
          values: ["c", "m", "r"]
        - key: "eks.amazonaws.com/instance-cpu"
          operator: In
```

```

    values: ["4", "8", "16", "32"]
  - key: "topology.kubernetes.io/zone"
    operator: In
    values: ["us-west-2a", "us-west-2b"]
  - key: "kubernetes.io/arch"
    operator: In
    values: ["arm64", "amd64"]

limits:
  cpu: "1000"
  memory: 1000Gi

```

EKS 자동 모드 지원 레이블

EKS Auto Mode는 다음과 같이 잘 알려진 레이블을 지원합니다.

레이블	예제	설명
topology.kubernetes.io/zone	us-east-2a	AWS 리전
node.kubernetes.io/instance-type	g4dn.8xlarge	AWS 인스턴스 유형
kubernetes.io/arch	amd64	아키텍처는 인스턴스의 GOARCH 값 으로 정의됨
karpenter.sh/capacity-type	spot	용량 유형에 spot, on-demand 포함
eks.amazonaws.com/instance-hypervisor	nitro	특정 하이퍼바이저를 사용하는 인스턴스 유형
eks.amazonaws.com/compute-type	auto	EKS Auto Mode 관리형 노드 식별
eks.amazonaws.com/instance-encryption-in-transit-supported	true	전송 중 암호화를 지원하는(또는 지원하지 않는) 인스턴스 유형
eks.amazonaws.com/instance-category	g	동일한 범주의 인스턴스 유형, 일반적으로 생성 번호 앞의 문자열

레이블	예제	설명
eks.amazonaws.com/instance-generation	4	인스턴스 범주 내의 인스턴스 유형 생성 번호
eks.amazonaws.com/instance-family	g4dn	속성이 유사하지만 리소스 수량이 다른 인스턴스 유형
eks.amazonaws.com/instance-size	8xlarge	리소스 수량은 비슷하지만 속성이 다른 인스턴스 유형
eks.amazonaws.com/instance-cpu	32	인스턴스의 CPU 수
eks.amazonaws.com/instance-cpu-manufacturer	aws	CPU 제조업체 이름
eks.amazonaws.com/instance-memory	131072	인스턴스의 메모리 메비바이트 수
eks.amazonaws.com/instance-ebs-bandwidth	9500	인스턴스에서 사용 가능한 EBS의 최대 메가비트 수
eks.amazonaws.com/instance-network-bandwidth	131072	인스턴스에서 사용 가능한 기준 메가비트 수
eks.amazonaws.com/instance-gpu-name	t4	사용 가능한 경우 인스턴스의 GPU 이름
eks.amazonaws.com/instance-gpu-manufacturer	nvidia	GPU 제조업체 이름
eks.amazonaws.com/instance-gpu-count	1	인스턴스의 GPU 수
eks.amazonaws.com/instance-gpu-memory	16384	GPU의 메모리 메비바이트 수
eks.amazonaws.com/instance-local-nvme	900	인스턴스의 로컬 nvme 스토리지 기비바이트 수

Note

EKS 자동 모드는 특정 인스턴스만 지원하며 최소 크기 요구 사항이 있습니다. 자세한 내용은 [the section called “EKS 자동 모드 지원 인스턴스 참조”](#) 섹션을 참조하세요.

EKS 자동 모드 미지원 레이블

EKS Auto Mode는 다음 레이블을 지원하지 않습니다.

- EKS Auto Mode는 Linux만 지원
 - `node.kubernetes.io/windows-build`
 - `kubernetes.io/os`

내장 노드 풀 비활성화

사용자 지정 노드 풀을 생성하는 경우 내장 노드 풀을 비활성화할 수 있습니다. 자세한 내용은 [the section called “내장 노드 풀 검토”](#) 섹션을 참조하세요.

내장 노드 풀이 없는 클러스터

내장 노드 풀 없이 클러스터를 생성할 수 있습니다. 이는 조직에서 사용자 지정 노드 풀을 생성한 경우에 유용합니다.

Note

내장 노드 풀 없이 클러스터를 생성하면 default NodeClass가 자동으로 프로비저닝되지 않습니다. 사용자 지정 NodeClass를 생성해야 합니다. 자세한 내용은 [the section called “노드 클래스 생성”](#) 섹션을 참조하세요.

개요:

1. nodePools 및 nodeRoleArn 값을 모두 비워두고 EKS 클러스터를 생성합니다.

- 샘플 `eksctl autoModeConfig`:

```
autoModeConfig:
  enabled: true
  nodePools: []
```

```
# Do not set a nodeRoleARN
```

자세한 내용은 [the section called “eksctl CLI”](#) 섹션을 참조하세요.

2. 노드 역할 ARN을 사용하여 사용자 지정 노드 클래스 생성
 - 자세한 내용은 [the section called “노드 클래스 생성”](#) 섹션을 참조하세요.
3. 사용자 지정 노드 클래스에 대한 액세스 항목 생성
 - 자세한 내용은 [the section called “노드 클래스 액세스 항목 생성”](#) 섹션을 참조하세요.
4. 위에서 설명한 대로 사용자 지정 노드 풀을 생성합니다.

중단

여러 가지 방법으로 NodePool을 통해 노드를 중단하도록 EKS 자동 모드를 구성할 수 있습니다. `spec.disruption.consolidationPolicy`, `spec.disruption.consolidateAfter` 또는 `spec.template.spec.expireAfter`를 사용할 수 있습니다. 또한 NodePool의 `spec.disruption.budgets`를 통해 EKS 자동 모드의 중단 비율을 제한할 수 있습니다. 시간 범위와 중단된 동시 노드 수를 제어할 수도 있습니다. 이 동작을 구성하는 방법은 Karpenter 설명서의 [Disruption](#)을 참조하세요.

노드 풀에 대한 중단을 구성하여 다음을 수행할 수 있습니다.

- 인스턴스 활용도가 낮은 시기를 식별하고 워크로드를 통합합니다.
- 노드 풀 중단 예산을 생성하여 드리프트, 비우기 및 통합으로 인한 노드 종료 비율을 제한합니다.

기본적으로 EKS 자동 모드는 다음을 수행합니다.

- 활용도가 낮은 인스턴스를 통합합니다.
- 336시간 후에 인스턴스를 종료합니다.
- 단일 중단 예산을 노드의 10%로 설정합니다.
- 약 일주일에 한 번씩 발생하는 새로운 자동 모드 AMI가 릴리스될 때 드리프트로 인한 노드 교체가 가능합니다.

종료 유예 기간

`terminationGracePeriod`가 EKS Auto NodePool에 명시적으로 정의되지 않은 경우 시스템은 연결된 NodeClaim에 기본 24시간 종료 유예 기간을 자동으로 적용합니다. EKS 자동 고객은 사용자 지

정 NodePool 구성에서 `terminationGracePeriod`가 기본값으로 설정되어 있는 것을 볼 수 없지만 NodeClaim에서는 이 기본값이 표시됩니다. 이 기능은 유예 기간이 NodePool에 명시적으로 설정되어 있든, NodeClaim에 기본값으로 설정되어 있든 일관되게 유지되므로 클러스터 전체에서 예측 가능한 노드 종료 동작을 보장합니다.

IngressClass를 생성하여 Application Load Balancer 구성

EKS Auto Mode는 클러스터 앱을 인터넷에 노출하는 등 로드 밸런싱을 위한 일상적인 작업을 자동화합니다.

AWS에서는 Application Load Balancer(ALB)를 사용하여 HTTP 및 HTTPS 트래픽을 제공할 것을 제안합니다. Application Load Balancer는 요청의 내용을 기반으로 요청을 라우팅할 수 있습니다. Application Load Balancer에 대한 자세한 내용은 [What is Elastic Load Balancing?](#)을 참조하세요.

EKS Auto Mode는 Application Load Balancer(ALB)를 생성하고 구성합니다. 예를 들어 EKS Auto Mode는 Ingress Kubernetes 객체를 생성할 때 로드 밸런서를 생성하고 클러스터 워크로드로 트래픽을 라우팅하도록 구성합니다.

개요

1. 인터넷에 노출하려는 워크로드를 생성합니다.
2. SSL/TLS 및 VPC 서브넷에 사용할 인증서와 같은 AWS 특정 구성 값을 지정하여 IngressClassParams 리소스를 생성합니다.
3. EKS Auto Mode가 IngressClass 리소스의 컨트롤러가 되도록 지정하는 리소스를 생성합니다.
4. HTTP 경로 및 포트를 클러스터 워크로드와 연결하는 Ingress 리소스를 생성합니다.

EKS Auto Mode는 Ingress 리소스에 지정된 로드 밸런서 구성을 사용하여 IngressClassParams 리소스에 지정된 워크로드를 가리키는 Application Load Balancer를 생성합니다.

사전 조건

- Amazon EKS 클러스터에서 활성화된 EKS Auto Mode
- 클러스터에 연결하도록 구성된 Kubectl
 - `kubectl apply -f <filename>`를 사용하여 아래의 샘플 구성 YAML 파일을 클러스터에 적용할 수 있습니다.

Note

EKS 자동 모드에서는 퍼블릭 및 프라이빗 서브넷을 식별하기 위해 서브넷 태그가 필요합니다. `eksctl`로 클러스터를 생성한 경우 이미 이러한 태그가 있는 것입니다. [the section called “서브넷에 태그 지정”](#) 방법에 대해 알아봅니다.

1단계: 워크로드 생성

인터넷에 노출하려는 워크로드를 생성합니다. 배포 또는 서비스와 같이 HTTP 트래픽을 제공하는 모든 Kubernetes 리소스일 수 있습니다. 이 예제에서는 포트 80에서 HTTP 트래픽을 제공하는 `service-2048`이라는 간단한 HTTP 서비스를 사용합니다.

2단계: IngressClassParams 생성

IngressClassParams 객체를 생성하여 Application Load Balancer에 대한 AWS 특정 구성 옵션을 지정합니다. 이 예제에서는 다음 단계에서 사용할 `alb`라는 IngressClassParams 리소스를 생성합니다. 이 리소스는 `alb-ingressclassparams.yaml`이라는 파일에서 로드 밸런서 체계를 `internet-facing`으로 지정합니다.

```
apiVersion: eks.amazonaws.com/v1
kind: IngressClassParams
metadata:
  name: alb
spec:
  scheme: internet-facing
```

클러스터에 구성을 적용합니다.

```
kubectl apply -f alb-ingressclassparams.yaml
```

3단계: IngressClass 생성

`alb-ingressclass.yaml`이라는 파일에 IngressClassParams 리소스에 설정된 AWS 특정 구성 값을 참조하는 IngressClass를 생성합니다. IngressClass의 이름을 기록해 둡니다. 이 예제에서는 IngressClass 및 IngressClassParams의 이름이 모두 `alb`입니다.

`is-default-class` 주석을 사용하여 Ingress 리소스가 기본적으로 이 클래스를 사용해야 하는지 여부를 제어합니다.

```

apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: alb
  annotations:
    # Use this annotation to set an IngressClass as Default
    # If an Ingress doesn't specify a class, it will use the Default
    ingressclass.kubernetes.io/is-default-class: "true"
spec:
  # Configures the IngressClass to use EKS Auto Mode
  controller: eks.amazonaws.com/alb
  parameters:
    apiGroup: eks.amazonaws.com
    kind: IngressClassParams
    # Use the name of the IngressClassParams set in the previous step
    name: alb

```

구성 옵션에 대한 자세한 내용은 [the section called “IngressClassParams 참조”](#) 섹션을 참조하세요.

클러스터에 구성을 적용합니다.

```
kubectl apply -f alb-ingressclass.yaml
```

4단계: 수신 생성

alb-ingress.yaml이라는 파일에 Ingress 리소스를 생성합니다. 이 리소스의 목적은 Application Load Balancer의 경로 및 포트를 클러스터의 워크로드와 연결하는 것입니다. 이 예제에서는 포트 80에서 service-2048이라는 서비스로 트래픽을 라우팅하는 2048-ingress라는 Ingress 리소스를 생성합니다.

이 리소스 구성에 대한 자세한 내용은 Kubernetes 설명서의 [Ingress](#) 섹션을 참조하세요.

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: 2048-ingress
spec:
  # this matches the name of IngressClass.
  # this can be omitted if you have a default ingressClass in cluster: the one with
  ingressclass.kubernetes.io/is-default-class: "true"  annotation
  ingressClassName: alb

```



```
rules:
  - http:
      paths:
        - path: /*
          pathType: ImplementationSpecific
          backend:
            service:
              name: service-2048
              port:
                number: 80
```

클러스터에 구성을 적용합니다.

```
kubectl apply -f alb-ingress.yaml
```

5단계: 상태 확인

kubectl을 사용하여 Ingress의 상태를 찾습니다. 로드 밸런서를 사용할 수 있기까지 몇 분 정도 걸릴 수 있습니다.

이전 단계에서 설정한 Ingress 리소스의 이름을 사용합니다. 예시:

```
kubectl get ingress 2048-ingress
```

리소스가 준비되면 로드 밸런서의 도메인 이름을 검색합니다.

```
kubectl get ingress 2048-ingress -o
jsonpath='{.status.loadBalancer.ingress[0].hostname}'
```

웹 브라우저에서 서비스를 보려면 Ingress 복구에 지정된 포트와 경로를 검토합니다.

6단계: 정리

로드 밸런서를 정리하려면 다음 명령을 사용합니다.

```
kubectl delete ingress 2048-ingress
kubectl delete ingressclass alb
kubectl delete ingressclassparams alb
```

EKS Auto Mode는 AWS 계정에서 연결된 로드 밸런서를 자동으로 삭제합니다.

IngressClassParams 참조

아래 표는 일반적으로 사용되는 구성 옵션에 대한 빠른 참조입니다.

필드	설명	예시 값
scheme	ALB가 내부인지 인터넷 연결 인지를 정의	internet-facing
namespaceSelector	이 IngressClass를 사용할 수 있는 네임스페이스 제한	environment: prod
group.name	여러 수신을 그룹화하여 단일 ALB 공유	retail-apps
ipAddressType	ALB의 IP 주소 유형 설정	dualstack
subnets.ids	ALB 배포를 위한 서브넷 ID 목록	subnet-xxxx, subnet-yyyy
subnets.tags	필터에 태그를 지정하여 ALB의 서브넷 선택	Environment: prod
certificateARNs	사용할 SSL 인증서의 ARN	arn:aws:acm:region:account:certificate/id
tags	AWS 리소스에 대한 사용자 지정 태그	Environment: prod, Team: platform
loadBalancerAttributes	로드 밸런서별 속성	idle_timeout.timeout_seconds: 60

고려 사항

- IngressClass에서 주석을 사용하여 EKS Auto Mode로 로드 밸런서를 구성할 수 없습니다.
- Kubernetes에서 AWS 로드 밸런서 리소스로 태그 전파를 활성화하려면 클러스터 IAM 역할을 업데이트해야 합니다. 자세한 내용은 [the section called “EKS Auto 리소스에 대한 사용자 지정 AWS 태그”](#) 섹션을 참조하세요.

- EKS Auto Mode 또는 자체 관리형 AWS 로드 밸런서 컨트롤러와 리소스를 연결하는 방법에 대한 자세한 내용은 [the section called “마이그레이션 참조”](#) 섹션을 참조하세요.
- 로드 밸런서 관련 문제 해결에 대한 자세한 내용은 [the section called “문제 해결”](#) 섹션을 참조하세요.
- EKS Auto Mode의 로드 밸런싱 기능 사용에 대한 자세한 고려 사항은 [the section called “로드 밸런싱”](#) 섹션을 참조하세요.

다음 표에서는 EKS Auto Mode에 대한 IngressClassParams, Ingress 주석, TargetGroupBinding 구성의 변경 사항을 자세히 비교합니다. 이 표에서는 API 버전 변경, 사용 중단된 기능, 업데이트된 파라미터 이름을 포함하여 EKS Auto Mode의 로드 밸런싱 기능과 오픈 소스 로드 밸런서 컨트롤러 간의 주요 차이점을 강조합니다.

IngressClassParams

이전	New	설명
elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1	API 버전 변경
spec.certificateArn	spec.certificateARNs	여러 인증서 ARN 지원
spec.subnets.tags	spec.subnets.matchTags	서브넷 일치 스키마가 변경됨
spec.listeners.listenerAttributes	spec.listeners.attributes	간소화된 속성 이름 지정

수신 주석

이전	New	설명
kubernetes.io/ingress.class	지원되지 않음	수신 객체에서 spec.ingressClassName 사용
alb.ingress.kubernetes.io/group.name	지원되지 않음	IngressClass에서만 그룹 지정

이전	New	설명
<code>alb.ingress.kubernetes.io/waf-acl-id</code>	지원되지 않음	대신 WAF v2 사용
<code>alb.ingress.kubernetes.io/web-acl-id</code>	지원되지 않음	대신 WAF v2 사용
<code>alb.ingress.kubernetes.io/shield-advanced-protection</code>	지원되지 않음	Shield 통합 비활성화됨
<code>alb.ingress.kubernetes.io/auth-type: oidc</code>	지원되지 않음	OIDC 인증 유형은 현재 지원되지 않습니다.

TargetGroupBinding

이전	New	설명
<code>elbv2.k8s.aws/v1beta1</code>	<code>eks.amazonaws.com/v1</code>	API 버전 변경
<code>spec.targetType</code> 선택 사항	<code>spec.targetType</code> 필수	명시적 대상 유형 사양
<code>spec.networking.ingress.from</code>	지원되지 않음	보안 그룹 없이 더 이상 NLB를 지원하지 않음

서비스 주석을 사용하여 Network Load Balancer 구성

Kubernetes 서비스 주석을 사용하여 Amazon EKS에서 Network Load Balancer(NLB)를 구성하는 방법을 알아봅니다. 이 주제에서는 인터넷 접근성, 상태 확인, SSL/TLS 종료, IP 대상 지정 모드를 포함하여 NLB 동작을 사용자 지정하기 위해 EKS Auto Mode에서 지원하는 주석에 대해 설명합니다.

EKS Auto Mode에서 LoadBalancer 유형의 Kubernetes 서비스를 생성하면 EKS는 사용자가 지정한 주석을 기반으로 AWS Network Load Balancer를 자동으로 프로비저닝하고 구성합니다. 이 선언적 접

근 방식을 사용하면 Kubernetes 매니페스트를 통해 직접 로드 밸런서 구성을 관리하여 코드형 인프라 관행을 유지할 수 있습니다.

EKS Auto Mode는 기본적으로 LoadBalancer 유형의 모든 서비스에 대해 Network Load Balancer 프로비저닝을 처리합니다. 추가 컨트롤러 설치 또는 구성은 필요하지 않습니다. `loadBalancerClass: eks.amazonaws.com/nlb` 사양은 클러스터 기본값으로 자동 설정되므로 기존 Kubernetes 워크로드와의 호환성을 유지하면서 배포 프로세스를 간소화합니다.

Note

EKS 자동 모드에서는 퍼블릭 및 프라이빗 서브넷을 식별하기 위해 서브넷 태그가 필요합니다. `eksctl`로 클러스터를 생성한 경우 이미 이러한 태그가 있는 것입니다. [the section called “서브넷에 태그 지정”](#) 방법에 대해 알아보십시오.

샘플 서비스

Kubernetes Service 리소스에 대한 자세한 내용은 [the Kubernetes Documentation](#)를 참조하세요.

아래의 샘플 Service 리소스를 검토합니다.

```
apiVersion: v1
kind: Service
metadata:
  name: echoserver
  annotations:
    # Specify the load balancer scheme as internet-facing to create a public-facing
    Network Load Balancer (NLB)
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  selector:
    app: echoserver
  ports:
    - port: 80
      targetPort: 8080
      protocol: TCP
  type: LoadBalancer
  # Specify the new load balancer class for NLB as part of EKS Auto Mode feature
  # For clusters with Auto Mode enabled, this field can be omitted as it's the default
  loadBalancerClass: eks.amazonaws.com/nlb
```

일반적으로 사용되는 주석

다음 표에는 EKS Auto Mode에서 지원하는 일반적으로 사용되는 주석이 나열되어 있습니다. EKS Auto Mode가 일부 주석을 지원하지 않을 수 있습니다.

Tip

다음 주석은 모두 `service.beta.kubernetes.io/` 접두사로 지정해야 합니다.

필드	설명	예제
<code>aws-load-balancer-type</code>	로드 밸런서 유형을 지정합니다. 새 배포에 <code>external</code> 을 사용합니다.	<code>external</code>
<code>aws-load-balancer-nlb-target-type</code>	트래픽을 노드 인스턴스로 라우팅할지 아니면 포트 IP로 직접 라우팅할지 지정합니다. 표준 배포에 <code>instance</code> 또는 직접 포트 라우팅에 <code>ip</code> 를 사용합니다.	<code>instance</code>
<code>aws-load-balancer-scheme</code>	로드 밸런서가 내부인지 인터넷 연결인지를 제어합니다.	<code>internet-facing</code>
<code>aws-load-balancer-healthcheck-protocol</code>	대상 그룹에 대한 상태 확인 프로토콜입니다. 일반적인 옵션은 TCP(기본값) 또는 HTTP입니다.	<code>HTTP</code>
<code>aws-load-balancer-healthcheck-path</code>	HTTP/HTTPS 프로토콜을 사용할 때 상태 확인을 위한 HTTP 경로입니다.	<code>/healthz</code>
<code>aws-load-balancer-healthcheck-port</code>	상태 확인에 사용되는 포트입니다. 특정 포트 번호 또는	<code>traffic-port</code>

필드	설명	예제
	traffic-port 일 수 있습니다.	
aws-load-balancer-subnets	로드 밸런서를 생성할 서브넷을 지정합니다. 서브넷 ID 또는 이름일 수 있습니다.	subnet-xxxx, subnet-yyyy
aws-load-balancer-ssl-cert	HTTPS/TLS용 AWS Certificate Manager의 SSL 인증서 ARN입니다.	arn:aws:acm:region:account:certificate/cert-id
aws-load-balancer-ssl-ports	SSL/TLS를 사용해야 하는 포트를 지정합니다.	443, 8443
load-balancer-source-ranges	로드 밸런서에 액세스할 수 있는 CIDR 범위입니다.	10.0.0.0/24, 192.168.1.0/24
aws-load-balancer-additional-resource-tags	로드 밸런서 및 관련 리소스에 적용할 추가 AWS 태그입니다.	Environment=prod, Team=platform
aws-load-balancer-ip-address-type	로드 밸런서가 IPv4 또는 듀얼 스택(IPv4 + IPv6)을 사용할지 여부를 지정합니다.	ipv4 또는 dualstack

고려 사항

- Kubernetes에서 AWS 로드 밸런서 리소스로 태그 전파를 활성화하려면 클러스터 IAM 역할을 업데이트해야 합니다. 자세한 내용은 [the section called “EKS Auto 리소스에 대한 사용자 지정 AWS 태그”](#) 단원을 참조하십시오.
- EKS Auto Mode 또는 자체 관리형 AWS 로드 밸런서 컨트롤러와 리소스를 연결하는 방법에 대한 자세한 내용은 [the section called “마이그레이션 참조”](#) 섹션을 참조하세요.
- 로드 밸런서 관련 문제 해결에 대한 자세한 내용은 [the section called “문제 해결”](#) 섹션을 참조하세요.
- EKS Auto Mode의 로드 밸런싱 기능 사용에 대한 자세한 고려 사항은 [the section called “로드 밸런싱”](#) 섹션을 참조하세요.

로드 밸런싱을 위해 EKS Auto Mode로 마이그레이션할 때 서비스 주석 및 리소스 구성을 여러 번 변경해야 합니다. 다음 표에는 지원되지 않는 옵션과 권장 대안을 포함하여 이전 구현과 새 구현 간의 주요 차이점이 요약되어 있습니다.

서비스 주석

이전	New	설명
<code>service.beta.kubernetes.io/load-balancer-source-ranges</code>	지원되지 않음	서비스에서 <code>spec.loadBalancerSourceRanges</code> 사용
<code>service.beta.kubernetes.io/aws-load-balancer-type</code>	지원되지 않음	서비스에서 <code>spec.loadBalancerClass</code> 사용
<code>service.beta.kubernetes.io/aws-load-balancer-internal</code>	지원되지 않음	<code>service.beta.kubernetes.io/aws-load-balancer-scheme</code> 사용
다양한 로드 밸런서 속성	지원되지 않음	<code>service.beta.kubernetes.io/aws-load-balancer-attributes</code> 사용
<code>service.beta.kubernetes.io/aws-load-balancer-proxy-protocol</code>	지원되지 않음	대신 <code>service.beta.kubernetes.io/aws-load-balancer-attributes</code> 사용
<code>service.beta.kubernetes.io/aws-load-balancer-access-log-enabled</code>	지원되지 않음	대신 <code>service.beta.kubernetes.io/aws-load-balancer-attributes</code> 사용
<code>service.beta.kubernetes.io/aws-load-</code>	지원되지 않음	대신 <code>service.beta.kubernetes.io/aws-load-balancer-attributes</code> 사용

이전	New	설명
balancer-access-log-s3-bucket-name		
service.beta.kubernetes.io/aws-load-balancer-access-log-s3-bucket-prefix	지원되지 않음	대신 service.beta.kubernetes.io/aws-load-balancer-attributes 사용
service.beta.kubernetes.io/aws-load-balancer-cross-zone-load-balancing-enabled	지원되지 않음	대신 service.beta.kubernetes.io/aws-load-balancer-attributes 사용

더 이상 사용되지 않는 로드 밸런서 속성 주석에서 마이그레이션하려면 이러한 설정을 service.beta.kubernetes.io/aws-load-balancer-attributes 주석으로 통합합니다. 이 주석은 다양한 로드 밸런서 속성에 대해 쉼표로 구분된 키-값 페어 목록을 허용합니다. 예를 들어 프록시 프로토콜, 액세스 로깅, 교차 영역 로드 밸런싱을 지정하려면 다음 형식을 사용합니다.

```
service.beta.kubernetes.io/aws-load-balancer-attributes: |
  access_logs.s3.enabled=true
  access_logs.s3.bucket=my-bucket
  access_logs.s3.prefix=my-prefix
  load_balancing.cross_zone.enabled=true
```

```
service.beta.kubernetes.io/aws-load-balancer-target-group-attributes: |
  proxy_protocol_v2.enabled=true
```

이 통합 형식은 필요한 개별 주석 수를 줄이면서 로드 밸런서 속성을 구성하는 더욱 일관되고 유연한 방법을 제공합니다. 기존 서비스 구성을 검토하고 이 통합 형식을 사용하도록 업데이트합니다.

TargetGroupBinding

이전	New	설명
elbv2.k8s.aws/v1beta1	eks.amazonaws.com/v1	API 버전 변경
spec.targetType 선택 사항	spec.targetType 필수	명시적 대상 유형 사양
spec.networking ingress.from	지원되지 않음	보안 그룹 없이 더 이상 NLB를 지원하지 않음

스토리지 클래스 생성

Amazon EKS 자동 모드의 StorageClass는 애플리케이션이 영구 스토리지를 요청할 때 Amazon EBS 볼륨이 자동으로 프로비저닝되는 방식을 정의합니다. 이 페이지에서는 Amazon EKS 자동 모드와 함께 작동하여 EBS 볼륨을 프로비저닝하는 StorageClass를 생성하고 구성하는 방법을 설명합니다.

StorageClass를 구성하여 볼륨 유형, 암호화, IOPS 및 기타 스토리지 파라미터를 포함하여 EBS 볼륨에 대한 기본 설정을 지정할 수 있습니다. 암호화 관리에 AWS KMS 키를 사용하도록 StorageClass를 구성할 수도 있습니다.

EKS 자동 모드는 StorageClass를 생성하지 않습니다. EKS 자동 모드의 스토리지 기능을 사용하려면 `ebs.csi.eks.amazonaws.com`을 참조하는 StorageClass를 생성해야 합니다.

첫째, `storage-class.yaml`라는 이름의 파일을 만듭니다.

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: auto-ebs-sc
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
provisioner: ebs.csi.eks.amazonaws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
  type: gp3
  encrypted: "true"
```

둘째, 클러스터에 스토리지 클래스를 적용합니다.

```
kubectl apply -f storage-class.yaml
```

핵심 구성 요소:

- `provisioner`: `ebs.csi.eks.amazonaws.com`-EKS Auto Mode를 사용합니다.
- `volumeBindingMode`: `WaitForFirstConsumer`-포드에 필요할 때까지 볼륨 생성을 지연합니다.
- `type`: `gp3`-EBS 볼륨 유형을 지정합니다.
- `encrypted`: `"true"` - EBS는 `StorageClass`를 사용하여 생성된 모든 볼륨을 암호화합니다. EBS는 기본 `aws/ebs` 키 별칭을 사용합니다. 자세한 내용은 Amazon EBS 사용 설명서의 [How Amazon EBS encryption works](#)을 참조하세요. 이 값은 선택 사항이지만 권장됩니다.
- `storageclass.kubernetes.io/is-default-class`: `"true"`-영구 볼륨 클레임에서 다른 볼륨 클래스를 지정하지 않는 한 Kubernetes는 기본적으로 이 스토리지 클래스를 사용합니다. 이 값은 선택 사항입니다. 다른 스토리지 컨트롤러에서 마이그레이션하는 경우 이 값을 설정할 때 주의하세요.

자체 관리형 KMS 키를 사용하여 EBS 볼륨 암호화

자체 관리형 KMS 키를 사용하여 EKS Auto Mode로 자동화된 EBS 볼륨을 암호화하려면 다음을 수행해야 합니다.

1. 자체 관리형 KMS 키를 생성합니다.
 - 자세한 내용은 KMS 사용 설명서의 [Create a symmetric encryption KMS key](#) 또는 [How Amazon Elastic Block Store \(Amazon EBS\) uses KMS](#)을 참조하세요.
2. KMS 키에 대한 액세스를 허용하는 새 정책을 생성합니다.
 - 아래의 샘플 IAM 정책을 사용하여 정책을 생성합니다. 새 자체 관리형 KMS 키의 ARN을 삽입합니다. 자세한 내용은 AWS IAM 사용 설명서의 [Creating roles and attaching policies \(console\)](#)을 참조하세요.
3. 정책을 EKS 클러스터 역할에 연결합니다.
 - AWS 콘솔을 사용하여 EKS 클러스터 역할의 ARN을 찾습니다. 역할 정보는 개요 섹션에 표시됩니다. 자세한 내용은 [the section called “클러스터 IAM 역할”](#) 단원을 참조하십시오.
4. `parameters.kmsKeyId` 필드에서 KMS 키 ID를 참조하도록 `StorageClass`를 업데이트합니다.

샘플 자체 관리형 KMS IAM 정책

아래 정책에서 다음 값을 업데이트합니다.

- <account-id> - AWS 계정 ID(예: 111122223333)
- <aws-region> - 클러스터의 AWS 리전(예: us-west-2)

```
{
  "Version": "2012-10-17",
  "Id": "key-auto-policy-3",
  "Statement": [
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::<account-id>:root"
      },
      "Action": "kms:*",
      "Resource": "*"
    },
    {
      "Sid": "Allow access through EBS for all principals in the account that are authorized to use EBS",
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:CreateGrant",
        "kms:DescribeKey"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "kms:CallerAccount": "<account-id>",
          "kms:ViaService": "ec2.<aws-region>.amazonaws.com"
        }
      }
    }
  ]
}
```

```

    }
  ]
}
```

샘플 자체 관리형 KMS **StorageClass**

```

parameters:
  type: gp3
  encrypted: "true"
  kmsKeyId: <custom-key-arn>
```

StorageClass 파라미터 참조

Kubernetes StorageClass 리소스에 대한 일반적인 내용은 Kubernetes 설명서의 [Storage Classes](#)를 참조하세요.

StorageClass 리소스의 parameters 섹션은 AWS와 관련이 있습니다. 다음 표를 이용하여 사용 가능한 옵션을 검토하세요.

파라미터	값	Default	설명
"csi.storage.k8s.io/fstype"	xfs, ext2, ext3, ext4	ext4	볼륨 생성 중에 형식을 지정할 파일 시스템 유형입니다. 이 파라미터는 대소문자를 구분합니다.
"type"	io1, io2, gp2, gp3, sc1, st1, standard, sbp1, sbg1	gp3	EBS 볼륨 유형.
"iopsPerGB"			GiB당 초당 I/O 작업. IO1, IO2, GP3 볼륨에 대해 지정할 수 있습니다.
"allowAutoIOPSPerGBIncrease"	true, false	false	"true"인 경우, CSI 드라이버는 iopsPerGB *

파라미터	값	Default	설명
			<volume size>가 너무 낮아 AWS에서 지원하는 IOPS 범위에 맞지 않는 경우 볼륨에 대해 IOPS를 늘립니다. 이렇게 하면 사용자가 너무 작은 PVC 용량 또는 iopsPerGB 값을 지정하는 경우에도 동적 프로비저닝이 항상 성공할 수 있습니다. 반면, 이러한 볼륨은 iopsPerGB에서 요청된 것보다 IOPS가 높기 때문에 추가 비용이 발생할 수 있습니다.
"iops"			초당 I/O 작업. IO1, IO2, GP3 볼륨에 대해 지정할 수 있습니다.
"throughput"		125	처리량 단위는 MiB/s입니다. gp3 볼륨 유형이 지정된 경우에만 유효합니다.
"encrypted"	true, false	false	볼륨이 암호화되어야 하는지 여부입니다. 유효한 값은 "true" 또는 "false"입니다.
"blockExpress"	true, false	false	io2 Block Express 볼륨 생성을 활성화합니다.

파라미터	값	Default	설명
"kmsKeyId"			볼륨을 암호화할 때 사용할 키의 전체 ARN입니다. 지정하지 않으면 AWS는 볼륨이 있는 리전에 기본 KMS 키를 사용합니다. 변경하지 않으면 /aws/ebs라는 자동 생성된 키가 됩니다.
"blockSize"			기본 파일 시스템을 포맷할 때 사용할 블록 크기입니다. Linux 노드와 fstype ext2, ext3, ext4 또는 xfs에서만 지원됩니다.
"inodeSize"			기본 파일 시스템을 포맷할 때 사용할 inode 크기입니다. Linux 노드와 fstype ext2, ext3, ext4 또는 xfs에서만 지원됩니다.
"bytesPerInode"			기본 파일 시스템을 포맷할 때 사용할 bytes-per-inode입니다. linux 노드와 fstype ext2, ext3, ext4에서만 지원됩니다.

파라미터	값	Default	설명
"numberOfInodes"			기본 파일 시스템을 포맷할 때 사용할 number-of-inodes입니다. linux 노드와 fstype ext2, ext3, ext4에서만 지원됩니다.
"ext4BigAlloc"	true, false	false	bigalloc 형식 지정 옵션을 활성화하여 클러스터링된 블록 할당을 사용하도록 ext4 파일 시스템을 변경합니다. 경고: bigalloc가 노드의 Linux 커널에서 완전히 지원되지 않을 수 있습니다.
"ext4ClusterSize"			bigalloc 기능이 활성화된 경우 ext4 파일 시스템을 포맷할 때 사용할 클러스터 크기입니다. 참고: ext4BigAlloc 파라미터를 true로 설정해야 합니다.

자세한 내용은 GitHub의 [AWS EFS CSI 드라이버](#)를 참조하세요.

고려 사항

EKS Auto Mode의 블록 스토리지 기능은 EBS CSI 드라이버와 다릅니다.

- 정적 프로비저닝

- EKS Auto Mode에서 외부에서 생성된 EBS 볼륨을 사용하려면 키 `eks:eks-cluster-name`와 클러스터 이름의 값이 포함된 AWS 태그를 수동으로 추가해야 합니다.
- 노드 시작 테인트
 - 노드 시작 테인트 기능을 사용하여 스토리지 기능 준비 전에 포드 예약을 방지할 수 없습니다.
- 동적으로 프로비저닝된 볼륨의 사용자 지정 태그
 - 추가 태그 CLI 플래그를 사용하여 동적으로 프로비저닝된 EBS 볼륨에 사용자 지정 태그를 구성할 수 없습니다.
 - StorageClass 태깅을 사용하여 사용자 지정 태그를 추가할 수 있습니다. EKS Auto Mode는 연결된 AWS 리소스에 태그를 추가합니다. 사용자 지정 태그에 대한 클러스터 IAM 역할을 업데이트해야 합니다. 자세한 내용은 [the section called “EKS Auto 리소스에 대한 사용자 지정 AWS 태그”](#) 단원을 참조하십시오.
- EBS 세부 성능 지표
 - EBS 세부 성능에 대한 Prometheus 지표에 액세스할 수 없습니다.

CSI 스냅샷 컨트롤러 추가 기능 설치

EKS Auto Mode는 CSI 스냅샷 컨트롤러 Amazon EKS 추가 기능과 호환됩니다.

AWS에서는 이 추가 기능을 내장 system 노드 풀에서 실행하도록 구성할 것을 제안합니다.

자세한 내용은 다음을 참조하세요.

- [the section called “핵심 추가 기능 실행”](#)
- [the section called “내장 노드 풀 검토”](#)
- [the section called “CSI 스냅샷 컨트롤러”](#)

시스템 노드 풀에 스냅샷 컨트롤러를 설치하려면

1. AWS 콘솔에서 EKS 클러스터 열기
2. 추가 기능 탭에서 추가 기능 더 가져오기를 선택합니다.
3. CSI 스냅샷 컨트롤러와 다음을 차례로 선택합니다.
4. 선택한 추가 기능 설정 구성 페이지에서 선택적 구성 설정을 선택하여 추가 기능 구성 스키마를 확인합니다.
 - a. 스냅샷 컨트롤러를 system 노드 풀과 연결하려면 다음 yamI을 삽입합니다. 스냅샷 컨트롤러에는 CriticalAddonsOnly 테인트에 대한 허용치가 포함됩니다.

```
{
  "nodeSelector": {
    "karpenter.sh/nodepool": "system"
  }
}
```

b. 다음을 선택합니다.

5. 추가 기능 구성을 검토한 다음 생성을 선택합니다.

EKS Auto Mode 비활성화

기존 EKS 클러스터에서 EKS Auto Mode를 비활성화할 수 있습니다. 이는 파괴적인 작업입니다.

- EKS는 EKS Auto Mode에서 운영하는 모든 EC2 인스턴스를 종료합니다.
- EKS는 EKS Auto Mode에서 운영하는 모든 로드 밸런서를 삭제합니다.
- EKS는 EKS Auto Mode에서 프로비저닝한 EBS 볼륨을 삭제하지 않습니다.

EKS 자율 모드는 생성한 리소스를 완전히 관리하도록 설계되었습니다. 수동 개입으로 인해 EKS 자율 모드가 비활성화되면 해당 리소스를 완전히 정리하지 못할 수 있습니다. 예를 들어 외부 보안 그룹 규칙에서 관리형 보안 그룹을 참조하고 클러스터에 대해 EKS 자율 모드를 비활성화하기 전에 해당 참조를 제거하는 것을 잊어버린 경우 관리형 보안 그룹이 유출됩니다(삭제되지 않음). 아래 단계에서는 누출된 보안 그룹을 제거하는 경우 해당 방법을 설명합니다.

EKS Auto Mode 비활성화(AWS 콘솔)

1. AWS Management Console에서 클러스터 개요 페이지를 엽니다.
2. EKS Auto Mode에서 관리를 선택합니다.
3. EKS Auto Mode를 off로 전환합니다.

이 프로세스가 끝나는 시점에서 관리형 보안 그룹이 삭제되지 않은 경우 [보안 그룹 삭제](#)의 설명을 사용하여 수동으로 삭제할 수 있습니다.

EKS Auto Mode 비활성화(AWS CLI)

다음 명령을 사용하여 기존 클러스터에서 EKS Auto Mode를 비활성화합니다.

aws CLI가 설치되어 있으며 EKS 클러스터를 관리할 수 있는 충분한 권한으로 로그인해야 합니다. 자세한 내용은 [설정](#) 단원을 참조하십시오.

Note

동일한 요청에서 컴퓨팅, 블록 스토리지, 로드 밸런싱 기능을 모두 활성화 또는 비활성화해야 합니다.

```
aws eks update-cluster-config \
  --name $CLUSTER_NAME \
  --compute-config enabled=false \
  --kubernetes-network-config '{"elasticLoadBalancing":{"enabled": false}}' \
  --storage-config '{"blockStorage":{"enabled": false}}'
```

다음과 같이 EKS 자율 모드를 비활성화한 후 누출된 EKS 자율 모드 보안 그룹을 삭제하지 못했는지 여부를 확인할 수 있습니다.

```
aws ec2 describe-security-groups \
  --filters Name=tag:eks:eks-cluster-name,Values=<cluster-name> Name=tag-key,Values=ingress.eks.amazonaws.com/resource,service.eks.amazonaws.com/resource --query "SecurityGroups[*].[GroupName]"
```

보안 그룹을 삭제하는 방법:

```
aws ec2 delete-security-group --group-name=<sg-name>
```

EKS Auto Mode 클러스터의 Kubernetes 버전 업데이트

이 주제에서는 Auto Mode 클러스터의 Kubernetes 버전을 업데이트하는 방법을 설명합니다. Auto Mode는 노드 교체와 컨트롤 플레인 업데이트의 조정을 처리하는 동시에 포드 중단 예산을 통해 워크로드 가용성을 유지하여 버전 업데이트 프로세스를 간소화합니다.

Auto Mode 클러스터를 업그레이드할 때 기존에 수동 업데이트가 필요했던 많은 구성 요소가 이제 서비스의 일부로 관리됩니다. 업그레이드 프로세스의 자동화된 측면과 사용자의 책임을 이해하면 클러스터의 원활한 버전 전환을 보장하는 데 도움이 됩니다.

EKS Auto Mode를 사용한 업데이트 알아보기

컨트롤 플레인 업그레이드가 시작된 후에 EKS Auto Mode가 클러스터의 노드를 교체하기 시작합니다. 새 노드에는 해당하는 새 Kubernetes 버전이 있습니다. EKS Auto Mode는 노드 업그레이드 시 포드 중단 예산을 관찰합니다.

또한 더 이상 다음과 같은 구성 요소를 업데이트할 필요가 없습니다.

- Amazon VPC CNI
- AWS 로드 밸런서 컨트롤러
- CoreDNS
- kube-proxy
- Karpenter
- AWS EBS CSI 드라이버

EKS Auto Mode는 이러한 구성 요소를 서비스 기능으로 대체합니다.

여전히 다음을 업데이트할 책임이 있습니다.

- 클러스터에 배포된 앱 및 워크로드
- 자체 관리형 추가 기능 및 컨트롤러
- Amazon EKS 추가 기능
 - [the section called “추가 기능 업데이트”](#) 방법 알아보기

[클러스터 업그레이드 모범 사례](#) 알아보기

클러스터 업데이트 시작

클러스터 업데이트를 시작하려면 [the section called “Kubernetes 버전 업데이트”](#) 섹션을 참조하세요.

내장 NodePools 활성화 또는 비활성화

EKS Auto Mode에는 내장 NodePools 2개가 있습니다. AWS 콘솔, CLI 또는 API를 사용하여 이러한 NodePools를 활성화 또는 비활성화할 수 있습니다.

내장 NodePool 참조

- system

- 이 NodePool에는 CriticalAddonsOnly 테인트가 있습니다. CoreDNS와 같은 많은 EKS 추가 기능은 이 테인트를 허용합니다. 이 시스템 노드 풀을 사용하여 클러스터의 핵심 애플리케이션을 분리합니다.
- amd64 및 arm64 아키텍처를 모두 지원합니다.
- general-purpose
 - 이 NodePool은 클러스터에서 범용 워크로드를 위한 노드 시작을 지원합니다.
 - amd64 아키텍처만 사용합니다.

두 내장 NodePools의 특성은 다음과 같습니다.

- 기본 EKS NodeClass 사용
- 온디맨드 EC2 용량만 사용
- C, M 및 R EC2 인스턴스 패밀리 사용
- 5세대 이상의 EC2 인스턴스 필요

Note

EKS가 '기본' NodeClass를 프로비저닝하려면 하나 이상의 내장 NodePool을 활성화해야 합니다. 모든 내장 NodePool을 비활성화하는 경우 사용자 지정 NodeClass를 생성하고 이를 사용하도록 NodePool을 구성해야 합니다. NodeClasses에 대한 자세한 내용은 [the section called “노드 클래스 생성”](#) 섹션을 참조하세요.

절차

사전 조건

- 장치에 최신 버전의 AWS Command Line Interface(AWS CLI)가 설치 및 구성되어 있습니다. 현재 버전을 확인하려면 `aws --version`을 사용합니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요.
- 충분한 IAM 권한을 사용하여 CLI에 로그인하고 IAM 정책, IAM 역할, EKS 클러스터를 포함한 AWS 리소스를 생성합니다.

AWS CLI로 활성화

다음 명령을 사용하여 두 내장 NodePools를 모두 활성화합니다.

```
aws eks update-cluster-config \
  --name <cluster-name> \
  --compute-config '{
    "nodeRoleArn": "<node-role-arn>",
    "nodePools": ["general-purpose", "system"],
    "enabled": true
  }' \
  --kubernetes-network-config '{
    "elasticLoadBalancing":{"enabled": true}
  }' \
  --storage-config '{
    "blockStorage":{"enabled": true}
  }'
```

명령을 수정하여 NodePools를 선택적으로 활성화할 수 있습니다.

AWS CLI로 비활성화

다음 명령을 사용하여 두 내장 NodePools를 모두 비활성화합니다.

```
aws eks update-cluster-config \
  --name <cluster-name> \
  --compute-config '{
    "enabled": true,
    "nodePools": []
  }' \
  --kubernetes-network-config '{
    "elasticLoadBalancing":{"enabled": true}}' \
  --storage-config '{
    "blockStorage":{"enabled": true}
  }'
```

EKS Auto Mode 노드에 워크로드 배포 여부 제어

EKS Auto Mode를 사용하여 EKS 클러스터에서 워크로드를 실행하는 경우 특정 워크로드가 EKS Auto Mode 노드에서 실행되는지 또는 기타 컴퓨팅 유형에서 실행되는지 여부를 제어해야 할 수 있습니다. 이 주제에서는 노드 선택기 및 선호도 규칙을 사용하여 워크로드가 의도한 컴퓨팅 인프라에 예약되게 하는 방법을 설명합니다.

이 주제의 예제에서는 `eks.amazonaws.com/compute-type` 레이블을 사용하여 EKS Auto Mode 노드에서 워크로드 배포를 요구하거나 방지하는 방법을 보여줍니다. 이는 EKS Auto Mode와 자체 관리형 Karpenter 프로비저너 또는 EKS 관리형 노드 그룹과 같은 기타 컴퓨팅 유형을 모두 실행하는 혼합 모드 클러스터에서 특히 유용합니다.

EKS Auto Mode 노드가 `eks.amazonaws.com/compute-type` 레이블 값을 `auto`로 설정했습니다. 이 레이블을 사용하여 EKS Auto Mode에서 관리하는 노드에 워크로드가 배포되는지 여부를 제어할 수 있습니다.

워크로드가 EKS Auto Mode 노드에 배포되어야 함

Note

이 `nodeSelector` 값은 EKS Auto Mode에 필요하지 않습니다. 이 `nodeSelector` 값은 EKS Auto Mode에서 관리하지 않는 노드 유형인 혼합 모드에서 클러스터를 실행하는 경우에만 관련이 있습니다. 예를 들어 EKS 관리형 노드 그룹을 사용하여 클러스터에 정적 컴퓨팅 용량을 배포하고 EKS Auto Mode에서 동적 컴퓨팅 용량을 관리할 수 있습니다.

이 `nodeSelector`를 배포 또는 기타 워크로드에 추가하여 Kubernetes가 EKS Auto Mode 노드에서 예약하도록 요구할 수 있습니다.

```
apiVersion: apps/v1
kind: Deployment
spec:
  template:
    nodeSelector:
      eks.amazonaws.com/compute-type: auto
```

워크로드가 EKS Auto Mode 노드에 배포되지 않아야 함

이 `nodeAffinity`를 배포 또는 기타 워크로드에 추가하여 Kubernetes가 EKS Auto Mode 노드에서 예약하지 않도록 요구할 수 있습니다.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
```

```
- key: eks.amazonaws.com/compute-type
  operator: NotIn
  values:
    - auto
```

전용 인스턴스에서 핵심 추가 기능 실행

이 주제에서는 EKS Auto Mode가 system 노드 풀에 워크로드를 예약하도록 CriticalAddonsOnly 허용 오차가 있는 워크로드를 배포하는 방법을 알아봅니다.

EKS Auto Mode의 내장 system 노드 풀은 전용 인스턴스에서 핵심 추가 기능을 실행하도록 설계되었습니다. 이러한 분리를 통해 필수 구성 요소에 전용 리소스가 있고 일반 워크로드에서 격리되어 전반적인 클러스터 안정성과 성능이 향상됩니다.

이 가이드는 CriticalAddonsOnly 허용 및 적절한 노드 선택기를 활용하여 system 노드 풀에 추가 기능을 배포하는 방법을 보여줍니다. 이러한 단계를 따르면 EKS Auto Mode의 특수 노드 풀 구조에서 제공하는 격리 및 리소스 할당 이점을 활용하여 핵심 애플리케이션이 전용 system 노드로 예약되게 할 수 있습니다.

EKS Auto Mode에는 general-purpose 및 system이라는 기본 제공 노드 풀 2개가 있습니다. 자세한 내용은 [the section called “내장 노드 풀 검토”](#) 단원을 참조하십시오.

system 노드 풀의 목적은 핵심 추가 기능을 서로 다른 노드로 분리하는 것입니다. system 노드 풀에서 프로비저닝한 노드에는 CriticalAddonsOnly Kubernetes 테인트가 있습니다. Kubernetes는 해당 허용이 있는 경우에만 이러한 노드에 포드를 예약합니다. 자세한 내용은 Kubernetes 설명서의 [테인트와 허용 오차](#)를 참조하세요.

사전 조건

- EKS Auto Mode 클러스터는 내장 system 노드 풀이 활성화되어 있습니다. 자세한 내용은 [the section called “내장 노드 풀 검토”](#) 단원을 참조하세요.
- kubectl을 설치하고 구성했습니다. 자세한 내용은 [설정](#) 단원을 참조하십시오.

절차

아래 예제 yaml을 검토합니다. 다음 구성에 유의하세요.

- nodeSelector-워크로드를 내장 system 노드 풀과 연결합니다. 이 노드 풀은 AWS API로 활성화해야 합니다. 자세한 내용은 [the section called “내장 노드 풀 검토”](#) 단원을 참조하십시오.

- tolerations-이 허용 오차는 system 노드 풀의 노드에 있는 CriticalAddonsOnly 테인트를 극복합니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sample-app
  template:
    metadata:
      labels:
        app: sample-app
    spec:
      nodeSelector:
        karpenter.sh/nodepool: system
      tolerations:
        - key: "CriticalAddonsOnly"
          operator: "Exists"
      containers:
        - name: app
          image: nginx:latest
          resources:
            requests:
              cpu: "500m"
              memory: "512Mi"
```

system 노드 풀에서 실행할 워크로드를 업데이트하려면 다음을 수행해야 합니다.

1. 기존 워크로드를 업데이트하여 위에 설명된 다음 구성을 추가합니다.

- nodeSelector
- tolerations

2. `kubectl apply`를 사용하여 업데이트된 워크로드를 클러스터에 배포합니다.

워크로드를 업데이트하면 전용 노드에서 실행됩니다.

EKS Auto Mode에서 네트워크 정책 사용

네트워크 정책을 사용하면 Amazon EKS 클러스터 내의 IP 주소 또는 포트 수준에서 트래픽 흐름을 제어할 수 있습니다. 이 주제에서는 EKS Auto Mode로 네트워크 정책을 활성화하고 사용하는 방법을 설명합니다.

사전 조건

- EKS Auto Mode가 활성화된 Amazon EKS 클러스터
- 클러스터에 연결하도록 구성된 kubectl

1단계: 네트워크 정책 컨트롤러 활성화

EKS Auto Mode로 네트워크 정책을 사용하려면 먼저 클러스터에 ConfigMap을 적용하여 네트워크 정책 컨트롤러를 활성화해야 합니다.

1. 다음 콘텐츠가 포함된 `enable-network-policy.yaml`이라는 파일을 생성합니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: amazon-vpc-cni
  namespace: kube-system
data:
  enable-network-policy-controller: "true"
```

2. 클러스터에 ConfigMap 적용:

```
kubectl apply -f enable-network-policy.yaml
```

2단계: 노드 클래스에서 네트워크 정책 활성화

네트워크 정책을 사용하려면 먼저 노드 클래스가 이를 지원하도록 구성되어 있는지 확인해야 합니다. 다음 단계를 따릅니다.

1. 다음 내용을 사용하여 노드 클래스 YAML 파일(예: `nodeclass-network-policy.yaml`)을 생성하거나 편집합니다.

```
apiVersion: eks.amazonaws.com/v1
```

```
kind: NodeClass
metadata:
  name: network-policy-enabled
spec:
  # Enables network policy support
  networkPolicy: DefaultAllow
  # Optional: Enables logging for network policy events
  networkPolicyEventLogs: Enabled
  # Include other Node Class configurations as needed
```

2. 클러스터에 노드 클래스 구성을 적용합니다.

```
kubectl apply -f nodeclass-network-policy.yaml
```

3. 노드 클래스가 생성되었는지 확인합니다.

```
kubectl get nodeclass network-policy-enabled
```

4. 이 노드 클래스를 사용하도록 노드 풀을 업데이트합니다. 자세한 내용은 [the section called “노드 풀 생성”](#) 단원을 참조하십시오.

노드가 이 노드 클래스를 사용하고 나면 네트워크 정책을 적용할 수 있습니다. 이제 네트워크 정책을 생성하고 적용하여 클러스터 내의 트래픽을 제어할 수 있습니다. 모든 노드 클래스 구성 옵션은 [the section called “노드 클래스 생성”](#) 섹션을 참조하세요.

3단계: 네트워크 정책 생성 및 테스트

이제 Kubernetes 네트워크 정책을 지원하도록 EKS Auto Mode 클러스터가 구성되었습니다. [the section called “Stars Policy 데모”](#)를 사용하여 테스트할 수 있습니다.

EKS 자동 모드의 태그 서브넷

EKS 자동 모드의 로드 밸런싱 기능을 사용하는 경우 VPC 서브넷에 AWS 태그를 추가해야 합니다.

배경

이 태그는 클러스터와 연결된 서브넷을 식별하며, 더 중요한 것은 서브넷이 퍼블릭인지 프라이빗인지 식별합니다.

퍼블릭 서브넷은 인터넷 게이트웨이를 통해 인터넷에 직접 액세스할 수 있으며, 로드 밸런서와 같이 공개적으로 액세스해야 하는 리소스에 사용됩니다.

프라이빗 서브넷은 인터넷에 직접 액세스할 수 없고 아웃바운드 트래픽에 NAT 게이트웨이를 사용하며, 퍼블릭 IP가 필요 없는 EKS 노드와 같은 내부 리소스에 사용됩니다.

NAT 게이트웨이 및 인터넷 게이트웨이에 대한 자세한 내용은 Amazon Virtual Private Cloud(VPC) 사용 설명서의 [다른 네트워크에 VPC 연결](#)을 참조하세요.

요구 사항

현재 EKS 자동 모드에서 로드 밸런싱에 사용되는 서브넷에는 다음 태그 중 하나가 있어야 합니다.

퍼블릭 서브넷

퍼블릭 서브넷은 인터넷에 연결된 로드 밸런서에 사용되며, 다음 태그가 있어야 합니다.

키	값
kubernetes.io/role/elb	1 또는 ``

프라이빗 서브넷

프라이빗 서브넷은 내부 로드 밸런서에 사용되며, 다음 태그가 있어야 합니다.

키	값
kubernetes.io/role/internal-elb	1 또는 ``

절차

시작하기 전에 퍼블릭 서브넷(인터넷 게이트웨이 액세스 가능)과 프라이빗 서브넷(NAT 게이트웨이 사용)을 식별합니다. VPC 리소스를 수정하려면 권한이 필요합니다.

AWS Management Console

1. Amazon VPC 콘솔을 열고 서브넷으로 이동합니다.
2. 태그를 지정할 서브넷 선택
3. 태그 탭을 선택하고 태그 추가를 선택합니다.
4. 적절한 태그를 추가합니다.

- 퍼블릭 서브넷의 경우: Key=kubernetes.io/role/elb
- 프라이빗 서브넷의 경우: Key=kubernetes.io/role/internal-elb

5. 값을 1로 설정하거나 비워 둡니다.

6. 저장하고 나머지 서브넷에 대해 반복합니다.

AWS CLI

퍼블릭 서브넷의 경우

```
aws ec2 create-tags \  
  --resources subnet-ID \  
  --tags Key=kubernetes.io/role/elb,Value=1
```

프라이빗 서브넷의 경우

```
aws ec2 create-tags \  
  --resources subnet-ID \  
  --tags Key=kubernetes.io/role/internal-elb,Value=1
```

subnet-ID를 실제 서브넷 ID로 바꿉니다.

가속화된 워크로드 배포

이 자습서에서는 Amazon EKS Auto Mode가 가속화된 워크로드 실행을 어떻게 간소화하는지 보여줍니다. Amazon EKS Auto Mode는 컴퓨팅, 네트워킹, 로드 밸런싱, 스토리지, Identity Access and Management 기능을 즉시 제공하는 주요 인프라 구성 요소를 자동화하여 클러스터 자체를 넘어 운영을 간소화합니다.

Amazon EKS Auto Mode에는 NVIDIA 및 AWS Neuron 드라이버와 같은 특정 인스턴스 유형에 필요한 드라이버와 디바이스 플러그인이 포함되어 있습니다. 이러한 구성 요소를 설치 또는 업데이트할 필요가 없습니다.

EKS Auto Mode는 이러한 액셀러레이터의 드라이버를 자동으로 관리합니다.

- [AWS Trainium](#)
- [AWS Inferentia](#)
- [Amazon EC2 가속 인스턴스의 NVIDIA GPU](#)

Note

EKS Auto Mode에는 Kubernetes용 NVIDIA 디바이스 플러그인이 포함되어 있습니다. 이 플러그인은 자동으로 실행되고 클러스터에서 데몬 세트로 표시되지 않습니다.

추가 네트워킹 지원:

- [Elastic Fabric Adapter\(EFA\)](#)

Amazon EKS Auto Mode를 사용하면 액셀러레이터 드라이버 및 디바이스 플러그인 관리할 필요가 없습니다.

클러스터를 0으로 조정하면 비용을 절감할 수도 있습니다. 실행 중인 워크로드가 없을 때 인스턴스를 종료하도록 EKS Auto Mode를 구성할 수 있습니다. 이는 배치 기반 추론 워크로드에 유용합니다.

다음은 Amazon EKS Auto Mode를 사용하여 가속화된 워크로드를 시작하는 방법의 예제입니다.

사전 조건

- Amazon EKS Auto Mode가 구성된 Kubernetes 클러스터.
- general-purpose 또는 system 관리형 노드 풀이 활성화될 때 생성된 default EKS 노드 클래스.

1단계: GPU 워크로드 배포

이 예제에서는 45GB GPU 메모리가 필요한 NVIDIA 기반 워크로드용 NodePool을 생성합니다. EKS Auto Mode에서는 Kubernetes 예약 제약 조건을 사용하여 인스턴스 요구 사항을 정의합니다.

Amazon EKS Auto Mode NodePool 및 샘플 workload를 배포하려면 다음 NodePool 및 포드 정의를 검토하고 nodepool-gpu.yaml 및 pod.yaml로 저장합니다.

nodepool-gpu.yaml

```
apiVersion: karpenter.sh/v1
kind: NodePool
metadata:
  name: gpu
spec:
```

```

disruption:
  budgets:
    - nodes: 10%
  consolidateAfter: 1h
  consolidationPolicy: WhenEmpty
template:
  metadata: {}
  spec:
    nodeClassRef:
      group: eks.amazonaws.com
      kind: NodeClass
      name: default
    requirements:
      - key: "karpenter.sh/capacity-type"
        operator: In
        values: ["on-demand"]
      - key: "kubernetes.io/arch"
        operator: In
        values: ["amd64"]
      - key: "eks.amazonaws.com/instance-family"
        operator: In
        values:
          - g6e
          - g6
    taints:
      - key: nvidia.com/gpu
        effect: NoSchedule
    terminationGracePeriod: 24h0m0s

```

pod.yaml

```

apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
spec:
  nodeSelector:
    eks.amazonaws.com/instance-gpu-name: l40s
    eks.amazonaws.com/compute-type: auto
  restartPolicy: OnFailure
  containers:
    - name: nvidia-smi
      image: public.ecr.aws/amazonlinux/amazonlinux:2023-minimal

```

```

args:
- "nvidia-smi"
resources:
  requests:
    memory: "30Gi"
    cpu: "3500m"
    nvidia.com/gpu: 1
  limits:
    memory: "30Gi"
    nvidia.com/gpu: 1
tolerations:
- key: nvidia.com/gpu
  effect: NoSchedule
  operator: Exists

```

eks.amazonaws.com/compute-type: auto 선택기를 사용하려면 Amazon EKS Auto Mode 노드에 워크로드를 배포해야 합니다. 또한 NodePool은 Nvidia GPU가 예약되도록 허용 오차가 있는 포드만 허용하는 테인트를 설정합니다.

클러스터에 NodePool과 워크로드를 적용합니다.

```

kubectl apply -f nodepool-gpu.yaml
kubectl apply -f pod.yaml

```

다음 결과가 표시됩니다.

```

nodepool.karpenter.sh/gpu configured created
pod/nvidia-smi created

```

몇 초 정도 기다리고, 클러스터의 노드를 확인합니다. 이제 Amazon EKS Auto Mode 클러스터에 프로비저닝된 새 노드가 표시됩니다.

```
> kubectl get nodes
```

NAME	TYPE	CAPACITY	ZONE	NODE	READY	AGE
gpu-dnknr	g6e.2xlarge	on-demand	us-west-2b	i-02315c7d7643cdee6	True	76s

2단계: 검증

다음 Kubernetes 예약 제약 조건에 따라 워크로드에 I40s GPU가 있는 인스턴스가 필요하므로 Amazon EKS Auto Mode가 g6.2xlarge 대신 g6e.2xlarge를 시작한 것을 확인할 수 있습니다.


```
...
nodeSelector:
  eks.amazonaws.com/instance-gpu-name: l40s
...
requests:
  memory: "30Gi"
  cpu: "3500m"
  nvidia.com/gpu: 1
limits:
  memory: "30Gi"
  nvidia.com/gpu: 1
```

이제 다음 명령을 실행하여 컨테이너 로그를 살펴봅니다.

```
kubectl logs nvidia-smi
```

샘플 출력:

```
+-----+
+
| NVIDIA-SMI 535.230.02                Driver Version: 535.230.02   CUDA Version: 12.2
|
|-----+-----+
+-----+
| GPU   Name                               Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC
|
| Fan   Temp   Perf              Pwr:Usage/Cap |      Memory-Usage | GPU-Util  Compute M.
|
|                               |                      |              MIG M.
|
|=====+=====+
+=====+
|   0   NVIDIA L40S                     On   | 00000000:30:00.0 Off |
|
| N/A    27C    P8                     23W / 350W |      0MiB / 46068MiB |      0%      Default
|
|                               |                      |              N/A
|
+-----+
+-----+
```

```

+-----+
+
+ | Processes:
+ |
+ | GPU    GI    CI          PID    Type    Process name                      GPU Memory
+ |
+ |          ID    ID
+ |
+ |
+ |=====|
+ | No running processes found
+ |
+-----+
+

```

컨테이너가 NVIDIA GPU가 있는 인스턴스에서 실행 중임을 감지했으며 Amazon EKS Auto Mode에서 관리하므로 디바이스 드라이버를 설치할 필요가 없음을 알 수 있습니다.

3단계: 정리

생성된 모든 객체를 제거하려면 `kubectl`을 사용하여 노드가 종료되도록 샘플 배포와 NodePool을 삭제합니다.

```

kubectl delete -f nodepool-gpu.yaml
kubectl delete -f pod.yaml

```

NodePools 참조 예제

NVIDIA NodePool 생성

다음 NodePool에서 정의하는 사항:

- g6e 및 g6 패밀리의 인스턴스만 시작
- 1시간 동안 비어 있는 경우 노드 통합
 - `consolidateAfter`의 1시간 값은 급증하는 워크로드를 지원하고 노드 이탈을 줄입니다. 워크로드 요구 사항에 따라 `consolidateAfter`를 조정할 수 있습니다.

GPU 인스턴스 패밀리 및 통합이 포함된 NodePool 예제

```

apiVersion: karpenter.sh/v1
kind: NodePool

```

```

metadata:
  name: gpu
spec:
  disruption:
    budgets:
      - nodes: 10%
    consolidateAfter: 1h
    consolidationPolicy: WhenEmpty
  template:
    metadata: {}
    spec:
      nodeClassRef:
        group: eks.amazonaws.com
        kind: NodeClass
        name: default
      requirements:
        - key: "karpenter.sh/capacity-type"
          operator: In
          values: ["on-demand"]
        - key: "kubernetes.io/arch"
          operator: In
          values: ["amd64"]
        - key: "eks.amazonaws.com/instance-family"
          operator: In
          values:
            - g6e
            - g6
      terminationGracePeriod: 24h0m0s

```

eks.amazonaws.com/instance-gpu-name을 설정하는 대신 eks.amazonaws.com/instance-family를 사용하여 인스턴스 패밀리를 지정할 수 있습니다. 예약 검토에 영향을 미치는 다른 잘 알려진 레이블은 [the section called “EKS 자동 모드 지원 레이블”](#) 섹션을 참조하세요.

특정 스토리지 요구 사항이 있는 경우 NodePool의 참조에 대해 [NodeClass](#)를 생성하여 노드 임시 스토리지 iops, size 및 throughput를 조정할 수 있습니다. [구성 가능한 NodeClass 옵션](#)에 대해 자세히 알아봅니다.

NodeClass의 스토리지 구성 예제

```

apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: gpu

```

```
spec:
  ephemeralStorage:
    iops: 3000
    size: 80Gi
    throughput: 125
```

AWS Trainium 및 AWS Inferentia NodePool 정의

다음 NodePool에는 Inferentia 및 Trainium 패밀리의 인스턴스만 시작하도록 하는 `eks.amazonaws.com/instance-category` 세트가 있습니다.

```
- key: "eks.amazonaws.com/instance-category"
  operator: In
  values:
    - inf
    - trn
```

kubectl 디버그를 사용하여 Kubernetes 노드에서 CIS 규정 준수 보고서 생성

이 주제에서는 `kubectl debug` 명령을 사용하여 Amazon EKS 노드에 대한 인터넷 보안 센터(CIS) 규정 준수 보고서를 생성하는 방법을 설명합니다. 이 명령을 사용하면 Kubernetes 노드에 디버깅 컨테이너를 임시로 생성하고 `apiclient` 도구를 사용하여 CIS 규정 준수 검사를 실행할 수 있습니다. `apiclient` 도구는 EKS 자동 모드 노드에서 사용하는 OS인 Bottlerocket OS의 일부입니다.

사전 조건

시작하기 전에 다음을 갖추었는지 확인하세요.

- `kubectl`이 구성된 Amazon EKS 클러스터에 대한 액세스 권한(버전이 v1.32.0 이상이어야 함, 확인하려면 `kubectl version` 입력)
- 노드를 디버깅할 수 있는 적절한 IAM 권한
- 디버그 작업을 허용하는 유효한 프로파일(예: `sysadmin`)

`kubectl`을 사용하여 디버깅 프로파일을 사용하는 방법에 대한 자세한 내용은 Kubernetes Documentation의 [Debugging a Pod or Node while applying a profile](#)을 참조하세요.

절차

1. 보고서를 실행할 노드의 AWS 인스턴스 ID를 결정합니다. 다음 명령을 사용하여 클러스터의 노드를 나열합니다. 인스턴스 ID는 이름 열에서 찾을 수 있으며 `i-`로 시작합니다.

```
kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
i-0ea0ba0f8ef9ad609	Ready	<none>	62s	v1.30.10-eks-1a9dacd

2. <instance-id>를 쿼리하려는 노드의 인스턴스 ID로 바꿔서 다음 명령을 실행합니다.

```
kubectl debug node/<instance-id> -it --profile=sysadmin --image=public.ecr.aws/amazonlinux/amazonlinux:2023 -- bash -c "yum install -q -y util-linux-core; nsenter -t 1 -m apiclient report cis --level 1 --format text"
```

이 명령의 구성 요소는 다음과 같습니다.

- `kubectl debug node/<instance-id>` - 지정된 EC2 인스턴스 ID에 디버깅 세션을 생성합니다.
- `-it` - TTY(명령줄 셸)를 할당하고 대화형 사용을 위해 stdin을 열어 둡니다.
- `--profile=sysadmin` - 적절한 권한으로 지정된 `kubectl` 프로파일을 사용합니다.
- `--image=public.ecr.aws/amazonlinux/amazonlinux:2023` - 디버깅을 위한 컨테이너 이미지로 `amazonlinux:2023`을 사용합니다.
- `bash -c "..."` - bash 셸에서 다음 명령을 실행합니다.
 - `yum install -q -y util-linux-core` - 필요한 유틸리티 패키지를 자동으로 설치합니다.
 - `nsenter -t 1 -m nsenter`를 실행하여 호스트 프로세스의 네임스페이스(PID 1)로 들어갑니다.
 - `apiclient report cis --level 1 --format text` - 텍스트 출력을 사용하여 레벨 1에서 CIS 규정 준수 보고서를 실행합니다.

3. 보고서 텍스트 출력을 검토합니다.

출력 해석

명령은 다양한 CIS 제어의 규정 준수 상태를 보여주는 텍스트 기반 보고서를 생성합니다. 출력에는 다음이 포함됩니다.

- 개별 CIS 제어 ID
- 각 제어에 대한 설명

- 각 검사의 통과, 실패 또는 건너뛰기 상태
- 규정 준수 문제를 설명하는 세부 정보

다음은 Bottlerocket 인스턴스에서 실행되는 보고서의 출력 예입니다.

```
Benchmark name:  CIS Bottlerocket Benchmark
Version:         v1.0.0
Reference:       https://www.cisecurity.org/benchmark/bottlerocket
Benchmark level: 1
Start time:      2025-04-11T01:40:39.055623436Z

[SKIP] 1.2.1      Ensure software update repositories are configured (Manual)
[PASS] 1.3.1      Ensure dm-verity is configured (Automatic)[PASS] 1.4.1      Ensure
setuid programs do not create core dumps (Automatic)
[PASS] 1.4.2      Ensure address space layout randomization (ASLR) is enabled
(Automatic)
[PASS] 1.4.3      Ensure unprivileged eBPF is disabled (Automatic)
[PASS] 1.5.1      Ensure SELinux is configured (Automatic)
[SKIP] 1.6        Ensure updates, patches, and additional security software are
installed (Manual)
[PASS] 2.1.1.1    Ensure chrony is configured (Automatic)
[PASS] 3.2.5      Ensure broadcast ICMP requests are ignored (Automatic)
[PASS] 3.2.6      Ensure bogus ICMP responses are ignored (Automatic)
[PASS] 3.2.7      Ensure TCP SYN Cookies is enabled (Automatic)
[SKIP] 3.4.1.3    Ensure IPv4 outbound and established connections are configured
(Manual)
[SKIP] 3.4.2.3    Ensure IPv6 outbound and established connections are configured
(Manual)
[PASS] 4.1.1.1    Ensure journald is configured to write logs to persistent disk
(Automatic)
[PASS] 4.1.2      Ensure permissions on journal files are configured (Automatic)

Passed:         11
Failed:         0
Skipped:        4
Total checks:   15
```

벤치마크에 대한 자세한 내용은 인터넷 보안 센터(CIS)의 [Kubernetes Benchmark](#)를 참조하세요.

관련 리소스

- Bottlerocket OS Documentation의 [Bottlerocket CIS Benchmark](#)

- 쿠버네티스 문서의 [동작 중인 파드 디버그](#)
- CIS(Center for Internet Security)의 [Kubernetes Benchmark](#)

EKS Auto Mode용 고객 관리형 KMS 키를 사용하여 EBS 볼륨 암호화 활성화

고객 관리형 KMS 키를 사용하여 EKS Auto Mode 인스턴스의 임시 루트 볼륨을 암호화할 수 있습니다.

Amazon EKS Auto Mode는 Kubernetes 클러스터의 암호화된 EBS 볼륨을 관리할 때 서비스 연결 역할을 사용하여 다른 AWS 서비스에 권한을 위임합니다. 이 주제에서는 EKS Auto Mode에서 Amazon EBS 암호화용 고객 관리형 키를 지정할 때 필요한 키 정책을 설정하는 방법에 대해 설명합니다.

고려 사항:

- EKS Auto Mode는 기본 AWS 관리형 키를 사용하여 계정의 암호화된 볼륨을 보호하는 데 추가 승인이 필요하지 않습니다.
- 이 주제에서는 EC2 인스턴스의 루트 볼륨인 임시 볼륨의 암호화를 다룹니다. 워크로드에 사용되는 데이터 볼륨 암호화에 대한 자세한 내용은 [the section called “StorageClass 생성”](#) 섹션을 참조하세요.

개요

다음 AWS KMS 키는 EKS Auto Mode에서 인스턴스를 시작할 때 Amazon EBS 루트 볼륨 암호화에 사용할 수 있습니다.

- AWS 관리형 키 - Amazon EBS에서 생성, 소유 및 관리하는 계정의 암호화 키입니다. 이 키는 새 계정을 위한 기본 암호화 키입니다.
- 고객 관리형 키 - 사용자가 생성, 소유 및 관리하는 사용자 지정 암호화 키입니다.

Note

키는 대칭이어야 합니다. Amazon EBS에서는 비대칭 고객 관리형 키를 지원하지 않습니다.

1단계: 키 정책 구성

KMS 키에는 EKS Auto Mode가 고객 관리형 키로 암호화된 Amazon EBS 볼륨으로 인스턴스를 시작할 수 있도록 허용하는 키 정책이 있어야 합니다.

다음 구조로 키 정책을 구성합니다.

Note

이 정책에는 EKS Auto Mode에 대한 권한만 포함되어 있습니다. 다른 자격 증명이 키를 사용하거나 권한 부여를 관리해야 하는 경우 키 정책에 추가 권한이 필요할 수 있습니다.

```
{
  "Version": "2012-10-17",
  "Id": "MyKeyPolicy",
  "Statement": [
    {
      "Sid": "Allow use of the key",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::<account-id>:role/ClusterServiceRole"
        ]
      },
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow attachment of persistent resources",
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "arn:aws:iam::<account-id>:role/ClusterServiceRole"
        ]
      }
    }
  ]
}
```



```

    "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
    ],
    "Resource": "*",
    "Condition": {
        "Bool": {
            "kms:GrantIsForAWSResource": "true"
        }
    }
}
]
}

```

<account-id>를 실제 AWS 계정 ID로 바꿔야 합니다.

키 정책을 구성할 때:

- ClusterServiceRole에는 암호화 작업에 KMS 키를 사용하는 데 필요한 IAM 권한이 있어야 합니다.
- kms:GrantIsForAWSResource 조건은 AWS 서비스에 대해서만 권한 부여를 생성할 수 있도록 합니다.

2단계: 고객 관리형 키를 사용하여 NodeClass 구성

키 정책을 구성한 후 EKS Auto Mode NodeClass 구성에서 KMS 키를 참조합니다.

```

apiVersion: eks.amazonaws.com/v1
kind: NodeClass
metadata:
  name: my-node-class
spec:
  # Insert existing configuration

  ephemeralStorage:
    size: "80Gi" # Range: 1-59000Gi or 1-64000G or 1-58Ti or 1-64T
    iops: 3000 # Range: 3000-16000
    throughput: 125 # Range: 125-1000

    # KMS key for encryption

```

```
kmsKeyID: "arn:aws:kms:<region>:<account-id>:key/<key-id>"
```

자리 표시자 값을 실제 값으로 바꿉니다.

- <region>을 실제 AWS 리전으로 바꿈
- <account-id>를 실제 AWS 계정 ID로 바꿈
- <key-id>를 실제 KMS 키 ID로 바꿈

다음 형식 중 하나를 사용하여 KMS 키를 지정할 수 있습니다.

- KMS 키 ID: 1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d
- KMS 키 ARN: arn:aws:kms:us-west-2:111122223333:key/1a2b3c4d-5e6f-1a2b-3c4d-5e6f1a2b3c4d
- 키 별칭 이름: alias/eks-auto-mode-key
- 키 별칭 ARN: arn:aws:kms:us-west-2:111122223333:alias/eks-auto-mode-key

kubectl을 사용하여 NodeClass 구성을 적용합니다.

```
kubectl apply -f nodeclass.yaml
```

관련 리소스

- [Amazon EKS용 NodeClass 생성](#)
- AWS Key Management Service 개발자 안내서에서 자세한 내용을 참조하세요.
 - [키 정책의 AWS 서비스에 대한 권한](#)
 - [키 정책 변경](#)
 - [AWS KMS의 권한 부여](#)

EKS Auto Mode 작동 방식 알아보기

이 장에서는 Amazon EKS Auto Mode 클러스터의 구성 요소가 작동하는 방법을 알아봅니다.

주제

- [Amazon EKS 자율 모드 관리형 인스턴스에 대해 알아보기](#)

- [EKS Auto Mode의 자격 증명 및 액세스에 대해 알아보기](#)
- [EKS 자율 모드에서 VPC 네트워킹 및 로드 밸런싱에 대해 알아보기](#)

Amazon EKS 자율 모드 관리형 인스턴스에 대해 알아보기

이 주제에서는 Amazon EKS Auto Mode가 EKS 클러스터에서 Amazon EC2 인스턴스를 관리하는 방법을 설명합니다. EKS Auto Mode를 활성화하면 클러스터의 컴퓨팅 리소스가 EKS에 의해 자동으로 프로비저닝되고 관리되므로 클러스터의 노드 역할을 하는 EC2 인스턴스와 상호 작용하는 방식이 변경됩니다.

Amazon EKS Auto Mode가 인스턴스를 관리하는 방법을 이해하는 것은 워크로드 배포 전략 및 운영 절차를 계획하는 데 필수적입니다. 기존 EC2 인스턴스 또는 관리형 노드 그룹과 달리 이러한 인스턴스는 EKS가 특정 유형의 액세스 및 사용자 지정을 제한하면서 여러 운영 측면에 대한 책임을 지는 다른 수명 주기 모델을 따릅니다.

Amazon EKS Auto Mode는 새 EC2 인스턴스를 생성하기 위한 일상적인 작업을 자동화하고 이를 EKS 클러스터에 노드로 연결합니다. EKS Auto Mode는 워크로드가 기존 노드에 맞지 않는 시기를 감지하고 새 EC2 인스턴스를 생성합니다.

Amazon EKS Auto Mode는 EC2 인스턴스를 생성 및 삭제하고 패치를 적용하는 역할을 합니다. 인스턴스에 배포된 컨테이너와 포드는 사용자의 책임입니다.

EKS Auto Mode에서 생성한 EC2 인스턴스는 다른 EC2 인스턴스와 차이가 있으며 관리형 인스턴스입니다. 이러한 관리형 인스턴스는 EKS에서 소유하며 더 제한적입니다. EKS Auto Mode에서 관리하는 인스턴스에는 직접 액세스하거나 소프트웨어를 설치할 수 없습니다.

AWS에서는 EKS Auto Mode 또는 자체 관리형 Karpenter를 실행할 것을 제안합니다. 마이그레이션 중에 또는 고급 구성으로 둘 다 설치할 수 있습니다. 둘 다 설치한 경우 워크로드가 Karpenter 또는 EKS Auto Mode와 연결되도록 노드 풀을 구성합니다.

자세한 내용은 Amazon EC2 사용 설명서의 [Amazon EC2 managed instances](#)를 참조하세요.

비교 테이블

표준 EC2 인스턴스	EKS Auto Mode 관리형 인스턴스
인스턴스의 패치 적용 및 업데이트는 사용자의 책임입니다.	AWS는 인스턴스를 자동으로 패치하고 업데이트합니다.

표준 EC2 인스턴스	EKS Auto Mode 관리형 인스턴스
EKS는 인스턴스의 소프트웨어에 대해 책임을 지지 않습니다.	EKS는 kubelet, 컨테이너 런타임, 운영 체제와 같은 인스턴스의 특정 소프트웨어를 담당합니다.
EC2 API를 사용하여 EC2 인스턴스를 삭제할 수 있습니다.	EKS는 계정에 배포된 인스턴스 수를 결정합니다. 워크로드를 삭제하면 EKS가 계정의 인스턴스 수를 줄입니다.
SSH를 사용하여 EC2 인스턴스에 액세스할 수 있습니다.	포드와 컨테이너를 관리형 인스턴스에 배포할 수 있습니다.
운영 체제 및 이미지(AMI)를 결정합니다.	AWS가 운영 체제와 이미지를 결정합니다.
Windows 또는 Ubuntu 기능에 의존하는 워크로드를 배포할 수 있습니다.	Linux 기반의 컨테이너를 배포할 수 있지만 특정 OS 종속성은 사용할 수 없습니다.
시작할 인스턴스 유형과 패밀리를 결정합니다.	AWS가 시작할 인스턴스 유형과 패밀리를 결정합니다. 노드 풀을 사용하여 EKS Auto Mode가 선택하는 인스턴스 유형을 제한할 수 있습니다.

다음 기능은 관리형 인스턴스와 표준 EC2 인스턴스 모두에서 작동합니다.

- AWS 콘솔에서 인스턴스를 볼 수 있습니다.
- 인스턴스 스토리지를 워크로드의 임시 스토리지로 사용할 수 있습니다.

AMI 지원

EKS 자동 모드를 사용하면 AWS가 컴퓨팅 노드에 사용되는 이미지(AMI)를 결정합니다. AWS는 새 EKS 자동 모드 AMI 버전의 롤아웃을 모니터링합니다. AMI 버전과 관련된 워크로드 문제가 발생하는 경우 지원 사례를 생성합니다. 자세한 내용은 AWS Support User Guide의 [Creating support cases and case management](#)를 참조하세요.

일반적으로 EKS는 매주 CVE 및 보안 수정 사항이 포함된 새 AMI를 릴리스합니다.

EKS 자동 모드 지원 인스턴스 참조

EKS 자동 모드는 최소 크기 요구 사항을 충족하는 지원되는 유형의 인스턴스만 생성합니다.

EKS Auto Mode는 다음 인스턴스 유형을 지원합니다.

Family	인스턴스 유형
컴퓨팅 최적화(C)	c8g, c7a, c7g, c7gn, c7gd, c7i, c7i-flex, c6a, c6g, c6i, c6gn, c6id, c6in, c6gd, c5, c5a, c5d, c5ad, c5n, c4
범용(M)	m8g, m7i, m7a, m7g, m7gd, m7i-flex, m6a, m6i, m6in, m6g, m6idn, m6id, m6gd, m5, m5a, m5ad, m5n, m5dn, m5d, m5zn, m4
메모리 최적화(R)	r8g, r7a, r7iz, r7gd, r7i, r7g, r6a, r6i, r6id, r6in, r6idn, r6g, r6gd, r5, r5n, r5a, r5dn, r5b, r5ad, r5d, r4
버스트 가능(T)	t4g, t3, t3a, t2
고용량 메모리(Z/X)	z1d, x8g, x2gd
스토리지 최적화(I/D)	i4g, i4i, i3, i3en, is4gen, d3, d3en, im4gn
가속 컴퓨팅(P/G/Inf/Trn)	p5, p4d, p3, p3dn, gr6, g6, g6e, g5g, g5, g4dn, inf2, inf1, trn1, trn1n
고성능 컴퓨팅(X2)	x2iezn, x2iedn, x2idn

또한 EKS 자동 모드는 다음 요구 사항을 충족하는 EC2 인스턴스만 생성합니다.

- 1개 이상의 vCPU
- 인스턴스 크기가 나노, 마이크로 또는 스몰이 아님

자세한 내용은 [Amazon EC2 인스턴스 유형 명명 규칙](#)을 참조하세요.

인스턴스 메타데이터 서비스

- EKS 자동 모드는 AWS 보안 모범 사례를 준수하여 기본적으로 홉 제한 1로 IMDSv2를 적용합니다.
- 자동 모드에서는 이 기본 구성을 수정할 수 없습니다.

- 일반적으로 IMDS 액세스가 필요한 추가 기능의 경우
 - IMDS 조회를 방지하기 위해 설치 중 AWS 리전과 같은 파라미터 제공
 - 자세한 내용은 [the section called “사용자 지정할 수 있는 필드”](#) 섹션을 참조하세요.
- 포드가 자동 모드에서 실행될 때 IMDS 액세스가 절대적으로 필요한 경우
 - `hostNetwork: true`로 실행되도록 포드를 구성해야 합니다.
 - 이렇게 하면 포드가 인스턴스 메타데이터 서비스에 직접 액세스할 수 있습니다.
- 포드에 인스턴스 메타데이터에 대한 액세스 권한을 부여할 때 보안 영향을 고려합니다.

EC2 인스턴스 메타데이터 서비스(IMDS)에 대한 자세한 내용은 EC2 사용 설명서의 [인스턴스 메타데이터 서비스 옵션 구성](#)을 참조하세요.

고려 사항

- EKS Auto Mode는 지원되는 인스턴스 유형에 NVMe 로컬 스토리지의 형식을 자동으로 지정하고 구성합니다. NVMe 드라이브가 여러 개 있는 노드의 경우 EKS는 RAID 0 배열을 설정합니다. 이 자동화를 사용하면 EKS 클러스터에서 로컬 NVMe 스토리지의 수동 형식 지정 및 RAID 구성이 필요하지 않습니다.
- Amazon EKS Auto Mode는 AWS Fault Injection Service를 지원하지 않습니다. 자세한 내용은 AWS Resilience Hub 사용 설명서의 [Managing Fault Injection Service experiments](#)를 참조하세요.
- EKS Auto Mode 노드에 Neuron Device Plugin을 설치할 필요가 없습니다.
 - 클러스터에 다른 유형의 노드가 있는 경우 자동 모드 노드에서 실행되지 않도록 Neuron Device 플러그인을 구성해야 합니다. 자세한 내용은 [the section called “배포 제어”](#) 섹션을 참조하세요.

EKS Auto Mode의 자격 증명 및 액세스에 대해 알아보기

이 주제에서는 EKS Auto Mode를 사용하는 데 필요한 Identity and Access Management(IAM) 역할 및 권한을 설명합니다. EKS Auto Mode는 클러스터 IAM 역할과 노드 IAM 역할이라는 두 가지 기본 IAM 역할을 사용합니다. 이러한 역할은 EKS Pod Identity 및 EKS 액세스 항목과 함께 작동하여 EKS 클러스터에 대한 포괄적인 액세스 관리를 제공합니다.

EKS Auto Mode를 구성할 때 AWS 서비스가 클러스터 리소스와 상호 작용하도록 허용하는 특정 권한을 사용하여 이러한 IAM 역할을 설정해야 합니다. 여기에는 컴퓨팅 리소스, 스토리지 볼륨, 로드 밸런서, 네트워킹 구성 요소를 관리하는 권한이 포함됩니다. 이러한 역할 구성을 이해하는 것은 적절한 클러스터 작업 및 보안에 필수적입니다.

EKS Auto Mode에서 AWS IAM 역할은 EKS 액세스 항목을 통해 Kubernetes 권한에 자동으로 매핑되므로 aws-auth ConfigMaps 또는 사용자 지정 바인딩을 수동으로 구성할 필요가 없습니다. 새 자동 모드 클러스터를 생성할 경우 EKS는 액세스 항목을 사용하여 해당 Kubernetes 권한을 자동으로 생성하여 AWS 서비스 및 클러스터 구성 요소가 AWS 및 Kubernetes 권한 부여 시스템 모두에서 적절한 액세스 수준을 갖도록 합니다. 이 자동 통합은 구성의 복잡성을 줄이고 EKS 클러스터를 관리할 때 일반적으로 발생하는 권한 관련 문제를 방지하는 데 도움이 됩니다.

클러스터 IAM 역할

클러스터 IAM 역할은 Amazon EKS에서 Kubernetes 클러스터에 대한 권한을 관리하는 데 사용하는 AWS Identity and Access Management(IAM) 역할입니다. 이 역할은 Amazon EKS에 클러스터를 대신하여 다른 AWS 서비스와 상호 작용하는 데 필요한 권한을 부여하며, EKS 액세스 항목을 사용하여 Kubernetes 권한으로 자동 구성됩니다.

- AWS IAM 정책을 이 역할에 연결해야 합니다.
- EKS Auto Mode는 EKS 액세스 항목을 사용하여 이 역할에 Kubernetes 권한을 자동으로 연결합니다.
- EKS Auto Mode를 사용할 경우 AWS에서는 AWS 계정당 단일 클러스터 IAM 역할을 생성할 것을 제안합니다.
- AWS는 이 역할의 이름을 AmazonEKSAutoClusterRole로 지정할 것을 제안합니다.
- 이 역할에는 EBS 볼륨, Elastic Load Balancer, EC2 인스턴스를 포함한 리소스를 관리할 수 있는 여러 AWS 서비스에 대한 권한이 필요합니다.
- 이 역할에 대해 제안된 구성에는 EKS 자동 모드의 다양한 기능과 관련된 여러 AWS 관리형 IAM 정책이 포함되어 있습니다.
 - AmazonEKSCoordinatePolicy
 - AmazonEKSBlockStoragePolicy
 - AmazonEKSLoadBalancingPolicy
 - AmazonEKSNetworkingPolicy
 - AmazonEKSClusterPolicy

클러스터 IAM 역할 및 AWS 관리형 IAM 정책에 대한 자세한 내용은 다음을 참조하세요.

- [the section called “AWS 관리형 정책”](#)
- [the section called “클러스터 IAM 역할”](#)

Kubernetes 액세스에 대한 자세한 내용은 다음을 참조하세요.

- [the section called “정책 액세스 검토”](#)

노드 IAM 역할

노드 IAM 역할은 Amazon EKS에서 Kubernetes 클러스터의 워커 노드에 대한 권한을 관리하는 데 사용하는 AWS Identity and Access Management(IAM) 역할입니다. 이 역할은 Kubernetes 노드로 실행되는 EC2 인스턴스에 AWS 서비스 및 리소스와 상호 작용하는 데 필요한 권한을 부여하며, EKS 액세스 항목을 사용하여 Kubernetes RBAC 권한으로 자동 구성됩니다.

- AWS IAM 정책을 이 역할에 연결해야 합니다.
- EKS Auto Mode는 EKS 액세스 항목을 사용하여 이 역할에 Kubernetes RBAC 권한을 자동으로 연결합니다.
- AWS는 이 역할의 이름을 AmazonEKSAutoNodeRole로 지정할 것을 제안합니다.
- EKS Auto Mode를 사용할 경우 AWS에서는 AWS 계정당 단일 노드 IAM 역할을 생성할 것을 제안합니다.
- 이 역할에는 제한된 권한이 있습니다. 주요 권한에는 Pod Identity 역할 수입, ECR에서 이미지 가져오기가 포함됩니다.
- AWS는 다음과 같은 AWS 관리형 IAM 정책을 제안합니다.
 - AmazonEKSWorkerNodeMinimalPolicy
 - AmazonEC2ContainerRegistryPullOnly

클러스터 IAM 역할 및 AWS 관리형 IAM 정책에 대한 자세한 내용은 다음을 참조하세요.

- [the section called “AWS 관리형 정책”](#)
- [the section called “노드 IAM 역할”](#)

Kubernetes 액세스에 대한 자세한 내용은 다음을 참조하세요.

- [the section called “정책 액세스 검토”](#)

서비스 연결 역할

Amazon EKS는 특정 작업에 서비스 연결 역할(SLR)을 사용합니다. 서비스 연결 역할은 Amazon EKS에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Amazon EKS에서 사전 정의하며 서비스에서 다른 AWS 서비스를 자동으로 호출하기 위해 필요한 모든 권한을 포함합니다.

AWS는 SLR을 자동으로 생성하고 구성합니다. 먼저 관련 리소스를 삭제해야만 SLR을 삭제할 수 있습니다. 이렇게 하면 리소스에 대한 액세스 권한을 실수로 삭제할 수 없기 때문에 Amazon EKS 리소스가 보호됩니다.

SLR 정책은 EC2 리소스(인스턴스, 네트워크 인터페이스, 보안 그룹), ELB 리소스(로드 밸런서, 대상 그룹), CloudWatch 기능(로깅 및 지표), 접두사가 'eks'인 IAM 역할 등 핵심 인프라 구성 요소를 관찰하고 삭제할 수 있는 권한을 Amazon EKS에 부여합니다. 또한 VPC/호스팅 영역 연결을 통해 프라이빗 엔드포인트 네트워킹을 활성화하고 EKS 태그가 지정된 리소스의 EventBridge 모니터링 및 정리에 대한 권한을 포함합니다.

자세한 내용은 다음을 참조하세요.

- [the section called “AWS 관리형 정책: AmazonEKSServiceRolePolicy”](#)
- [the section called “Amazon EKS에 대한 서비스 연결 역할 권한”](#)

EKS Auto 리소스에 대한 사용자 지정 AWS 태그

기본적으로 EKS Auto Mode와 관련된 관리형 정책은 Auto Mode 프로비저닝 AWS 리소스에 사용자 정의 태그를 적용하는 것을 허용하지 않습니다. AWS 리소스에 사용자 정의 태그를 적용하려면 AWS 리소스에 태그를 생성하고 수정할 수 있는 충분한 권한이 있는 클러스터 IAM 역할에 추가 권한을 연결해야 합니다. 다음은 무제한 태그 지정 액세스를 허용하는 정책의 예입니다.

사용자 지정 태그 정책 예제 보기

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Compute",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateFleet",
        "ec2:RunInstances",
        "ec2:CreateLaunchTemplate"
      ],
    }
  ],
}
```

```

        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
            },
            "StringLike": {
                "aws:RequestTag/eks:kubernetes-node-class-name": "*",
                "aws:RequestTag/eks:kubernetes-node-pool-name": "*"
            }
        }
    },
    {
        "Sid": "Storage",
        "Effect": "Allow",
        "Action": [
            "ec2:CreateVolume",
            "ec2:CreateSnapshot"
        ],
        "Resource": [
            "arn:aws:ec2:*:*:volume/*",
            "arn:aws:ec2:*:*:snapshot/*"
        ],
        "Condition": {
            "StringEquals": {
                "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
            }
        }
    },
    {
        "Sid": "Networking",
        "Effect": "Allow",
        "Action": "ec2:CreateNetworkInterface",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-
cluster-name}"
            },
            "StringLike": {
                "aws:RequestTag/eks:kubernetes-cni-node-name": "*"
            }
        }
    }
}

```

```

    },
    {
      "Sid": "LoadBalancer",
      "Effect": "Allow",
      "Action": [
        "elasticloadbalancing:CreateLoadBalancer",
        "elasticloadbalancing:CreateTargetGroup",
        "elasticloadbalancing:CreateListener",
        "elasticloadbalancing:CreateRule",
        "ec2:CreateSecurityGroup"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-cluster-name}"
        }
      }
    },
    {
      "Sid": "ShieldProtection",
      "Effect": "Allow",
      "Action": [
        "shield:CreateProtection"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-cluster-name}"
        }
      }
    },
    {
      "Sid": "ShieldTagResource",
      "Effect": "Allow",
      "Action": [
        "shield:TagResource"
      ],
      "Resource": "arn:aws:shield::*:protection/*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/eks:eks-cluster-name": "${aws:PrincipalTag/eks:eks-cluster-name}"
        }
      }
    }
  ]
}

```

```

    }
  }
}
]
}

```

액세스 정책 참조

EKS Auto Mode에서 사용하는 Kubernetes 권한에 대한 자세한 내용은 [the section called “정책 액세스 검토”](#) 섹션을 참조하세요.

EKS 자율 모드에서 VPC 네트워킹 및 로드 밸런싱에 대해 알아보기

이 주제에서는 EKS Auto Mode에서 가상 프라이빗 클라우드(VPC) 네트워킹 및 로드 밸런싱 기능을 구성하는 방법을 설명합니다. EKS 자동 모드는 대부분의 네트워킹 구성 요소를 자동으로 관리하지만 NodeClass 리소스 및 로드 밸런서 주석을 통해 클러스터 네트워킹 구성의 특정 측면을 사용자 지정할 수 있습니다.

EKS Auto Mode를 사용하는 경우 AWS는 클러스터에 대한 VPC 컨테이너 네트워크 인터페이스(CNI) 구성 및 로드 밸런서 프로비저닝을 관리합니다. EKS 자동 모드가 제공하는 자동화된 운영 모델을 유지하면서 NodeClass 객체를 정의하고 서비스 및 수신 리소스에 특정 주석을 적용하여 네트워킹 동작에 영향을 미칠 수 있습니다.

네트워킹 기능

EKS 자율 모드는 노드 및 포드 네트워킹을 처리하는 새로운 네트워킹 기능을 제공합니다. NodeClass Kubernetes 객체를 생성하여 구성할 수 있습니다.

이전 AWS VPC CNI의 구성 옵션은 EKS 자율 모드에 적용되지 않습니다.

NodeClass를 사용하여 네트워킹 구성

EKS 자율 모드의 NodeClass 리소스를 사용하면 네트워킹 기능의 특정 측면을 사용자 지정할 수 있습니다. NodeClass를 통해 보안 그룹 선택 지정, VPC 서브넷 간 노드 배치 제어, SNAT 정책 설정, 네트워크 정책 구성, 네트워크 이벤트 로깅 활성화가 가능합니다. 이 접근 방식은 EKS Auto Mode의 자동화된 운영 모델을 유지하면서 네트워크 사용자 지정에 유연성을 제공합니다.

NodeClass를 사용하여 다음을 수행할 수 있습니다.

- 노드에 대한 보안 그룹 선택

- VPC 서브넷에 노드를 배치하는 방법 제어
- 노드 SNAT 정책을 random 또는 disabled로 설정
- 다음을 포함한 Kubernetes 네트워크 정책을 활성화합니다.
 - 네트워크 정책을 기본 거부 또는 기본 허용으로 설정
 - 파일에 네트워크 이벤트 로깅 활성화
- 다른 서브넷에 포드를 연결하여 노드 트래픽에서 포드 트래픽을 격리합니다.

[Amazon EKS NodeClass 생성](#) 방법을 알아봅니다.

고려 사항

EKS Auto Mode는 다음을 지원합니다.

- EKS 네트워크 정책
- Kubernetes 포드의 HostPort 및 HostNetwork 옵션
- 퍼블릭 또는 프라이빗 서브넷의 노드 및 포드
- 노드에서 DNS 쿼리 캐싱.

EKS Auto Mode는 다음을 지원하지 않습니다.

- 포드당 보안 그룹(SGPP)
- ENIConfig의 사용자 지정 네트워킹. 여러 서브넷에 포드를 배치하거나 [the section called “포드의 서브넷 선택”](#)을 사용하여 노드 트래픽에서 포드를 독점적으로 분리할 수 있습니다.
- 워م IP, 워م 접두사, 워م ENI 구성
- 최소 IP 대상 구성
- 접두사 위임 활성화 또는 비활성화
- 오픈 소스 AWS VPC CNI에서 지원하는 기타 구성
- conntrack 타이머 사용자 지정과 같은 네트워크 정책 구성(기본값은 300초)
- CloudWatch로 네트워크 이벤트 로그 내보내기

네트워크 리소스 관리

EKS 자동 모드는 네트워킹 구성에 대한 NodeClass 리소스를 모니터링하여 접두사, IP 주소 지정 및 네트워크 인터페이스 관리를 처리합니다. 서비스는 여러 키 작업을 자동으로 수행합니다.

접두사 위임

EKS 자동 모드는 노드의 프라이머리 네트워크 인터페이스에 /28 IPv4 접두사를 프로비저닝하고 예약된 포트 수에 따라 규모가 조정되는 사전 정의된 웹 리소스 풀을 유지합니다. 필요한 경우 EKS 자동 모드는 노드 서브넷의 프라이머리 인터페이스와 동일한 보안 그룹을 사용하여 보조 네트워크 인터페이스를 프로비저닝합니다. 서브넷에서 접두사를 더 이상 사용할 수 없는 경우 서비스는 보조 IPv4 주소로 풀백합니다.

휴지 기간 관리

이 서비스는 더 이상 사용되지 않는 접두사 또는 보조 IPv4 주소에 대한 휴지 기간 풀을 구현합니다. 휴지 기간이 만료되면 이러한 리소스가 VPC로 다시 릴리스됩니다. 그러나 포트가 휴지 기간 동안 이러한 리소스를 재사용하는 경우, 해당 리소스는 휴지 기간 풀에서 복원됩니다.

IPv6 지원

IPv6 클러스터의 경우 EKS 자동 모드는 프라이머리 네트워크 인터페이스에서 노드당 /80 IPv6 접두사를 프로비저닝합니다.

또한 서비스는 모든 네트워크 인터페이스의 적절한 관리 및 폐영역 회수를 보장합니다.

로드 밸런싱

서비스 및 수신 리소스에 대한 주석을 사용하여 EKS Auto Mode에서 프로비저닝한 AWS Elastic Load Balancer를 구성합니다.

자세한 내용은 [the section called “수신 클래스 생성”](#) 또는 [the section called “서비스 생성”](#)을 참조하세요.

EKS Auto Mode를 사용한 로드 밸런싱 고려 사항

- 기본 대상 지정 모드는 인스턴스 모드가 아닌 IP 모드입니다.
- EKS Auto Mode는 Network Load Balancer에 대한 보안 그룹 모드만 지원합니다.
- AWS는 자체 관리형 AWS 로드 밸런서 컨트롤러에서 EKS Auto Mode의 관리로 로드 밸런서 마이그레이션을 지원하지 않습니다.
- TargetGroupBinding 사양의 `networking.ingress.ipBlock` 필드는 지원되지 않습니다.
- 워커 노드가 사용자 지정 보안 그룹(eks-cluster-sg- `명명 패턴 아님`)을 사용하는 경우 클러스터 역할에 추가 IAM 권한이 필요합니다. 기본 EKS 관리형 정책은 EKS가 eks-cluster-sg-라는 보안 그룹만 수정하도록 허용합니다. 사용자 지정 보안 그룹을 수정할 권한이 없는 경우 EKS는 ALB/NLB 트래픽이 포트에 도달하도록 허용하는 필수 수신 규칙을 추가할 수 없습니다.

EKS Auto Mode 문제 해결

EKS Auto Mode를 사용하면 AWS가 AWS 계정의 EC2 인스턴스에 대해 더 많은 책임을 집니다. EKS는 노드의 컨테이너 런타임, 노드의 운영 체제, 특정 컨트롤러에 대한 책임을 집니다. 여기에는 블록 스토리지 컨트롤러, 로드 밸런싱 컨트롤러, 컴퓨팅 컨트롤러가 포함됩니다.

노드 문제를 해결하려면 AWS 및 Kubernetes API를 사용해야 합니다. 다음을 할 수 있습니다.

- [the section called “노드 모니터링 에이전트”](#)을 참조해 Kubernetes NodeDiagnostic 리소스를 사용하여 노드 로그를 검색합니다. 자세한 단계는 [the section called “노드 로그 가져오기”](#) 섹션을 참조하세요.
- AWS EC2 CLI 명령 `get-console-output`을 사용하여 노드에서 콘솔 출력을 검색합니다. 자세한 단계는 [the section called “AWS EC2 CLI를 사용하여 EC2 관리형 인스턴스에서 콘솔 출력 가져오기”](#) 섹션을 참조하세요.
- Kubernetes 디버깅 컨테이너를 사용하여 노드 로그를 검색합니다. 자세한 단계는 [the section called “디버그 컨테이너와 kubectl CLI를 사용하여 노드 로그 가져오기”](#) 섹션을 참조하세요.

Note

EKS Auto Mode는 EC2 관리형 인스턴스를 사용합니다. SSH를 포함하여 EC2 관리형 인스턴스에 직접 액세스할 수 없습니다.

EKS 자동 모드 구성 요소와 관련된 해결 방법이 있는 다음과 같은 문제가 있을 수 있습니다.

- 자동 모드 노드에 예약되지 않은 Pending 상태로 멈춘 포드. 해결 방법은 [the section called “포드가 자율 모드 노드로 예약되지 않음 문제 해결”](#) 섹션을 참조하세요.
- 클러스터에 Kubernetes 노드로 조인하지 않는 EC2 관리형 인스턴스. 해결 방법은 [the section called “클러스터에 조인하지 않는 노드 문제 해결”](#) 섹션을 참조하세요.
- EKS 자동 모드에 포함된 컨트롤러를 사용하는 NodePools, PersistentVolumes 및 Services의 오류 및 문제. 해결 방법은 [the section called “자동 모드에서 포함된 컨트롤러 문제 해결”](#) 섹션을 참조하세요.
- 향상된 포드 보안으로 인해 포드 간에 볼륨을 공유할 수 없음. 해결 방법은 [the section called “포드 간 볼륨 공유”](#) 섹션을 참조하세요.

다음 방법을 사용하여 EKS 자동 모드 구성 요소의 문제를 해결할 수 있습니다.

- [the section called “AWS EC2 CLI를 사용하여 EC2 관리형 인스턴스에서 콘솔 출력 가져오기”](#)
- [the section called “디버그 컨테이너와 kubectl CLI를 사용하여 노드 로그 가져오기”](#)
- [the section called “AWS 콘솔에서 EKS Auto Mode와 연결된 리소스 보기”](#)
- [the section called “AWS 계정에서 IAM 오류 보기”](#)
- [the section called “VPC Reachability Analyzer로 노드 연결 문제 탐지”](#)

노드 모니터링 에이전트

EKS Auto Mode에는 Amazon EKS 노드 모니터링 에이전트가 포함되어 있습니다. 이 에이전트를 사용하여 노드에 관한 문제 해결 및 디버깅 정보를 볼 수 있습니다. 노드 모니터링 에이전트는 Kubernetes events 및 노드 conditions를 게시합니다. 자세한 내용은 [the section called “노드 상태”](#) 단원을 참조하십시오.

AWS EC2 CLI를 사용하여 EC2 관리형 인스턴스에서 콘솔 출력 가져오기

이 절차는 부팅 시간 또는 커널 수준 문제를 해결하는 데 도움이 됩니다.

먼저 워크로드와 연결된 인스턴스의 EC2 인스턴스 ID를 확인해야 합니다. 둘째, AWS CLI를 사용하여 콘솔 출력을 검색합니다.

1. kubectl을 설치하고 클러스터에 연결했는지 확인합니다.
2. (선택 사항) Kubernetes 배포의 이름을 사용하여 연결된 포드를 나열합니다.

```
kubectl get pods -l app=<deployment-name>
```

3. Kubernetes 포드의 이름을 사용하여 연결된 노드의 EC2 인스턴스 ID를 확인합니다.

```
kubectl get pod <pod-name> -o wide
```

4. EC2 인스턴스 ID를 사용하여 콘솔 출력을 검색합니다.

```
aws ec2 get-console-output --instance-id <instance id> --latest --output text
```

디버그 컨테이너와 kubectl CLI를 사용하여 노드 로그 가져오기

EKS 자동 모드 노드에서 로그를 검색하는 데 권장되는 방법은 NodeDiagnostic 리소스를 사용하는 것입니다. 자세한 단계는 [the section called “노드 로그 가져오기”](#) 섹션을 참조하세요.

그러나 `kubectl debug node` 명령을 사용하여 인스턴스에서 라이브로 로그를 스트리밍할 수 있습니다. 이 명령은 디버깅하려는 노드에서 대화형으로 사용할 수 있는 새 포드를 시작합니다.

1. 디버그 컨테이너를 시작합니다. 다음 명령은 노드의 인스턴스 ID로 `i-01234567890123456`를 사용하고, `-it`는 대화형 사용을 위해 `tty`를 할당하고 `stdin`을 연결하고, `kubeconfig` 파일의 `sysadmin` 프로필을 사용합니다.

```
kubectl debug node/i-01234567890123456 -it --profile=sysadmin --image=public.ecr.aws/amazonlinux/amazonlinux:2023
```

예제 출력은 다음과 같습니다.

```
Creating debugging pod node-debugger-i-01234567890123456-nxb9c with container
debugger on node i-01234567890123456.
If you don't see a command prompt, try pressing enter.
bash-5.2#
```

2. 이제 셸에서 `nsenter` 명령을 제공하는 `util-linux-core`를 설치할 수 있습니다. `nsenter`를 사용하여 호스트의 PID 1(`init`)의 탑재 네임스페이스를 입력하고 `journalctl` 명령을 실행하여 `kubelet`에서 로그를 스트리밍합니다.

```
yum install -y util-linux-core
nsenter -t 1 -m journalctl -f -u kubelet
```

보안을 위해 Amazon Linux 컨테이너 이미지는 기본적으로 많은 바이너리를 설치하지 않습니다. `yum whatprovides` 명령을 사용하여 지정된 바이너리를 제공하기 위해 설치해야 하는 패키지를 식별할 수 있습니다.

```
yum whatprovides ps
```

```
Last metadata expiration check: 0:03:36 ago on Thu Jan 16 14:49:17 2025.
procps-ng-3.3.17-1.amzn2023.0.2.x86_64 : System and process monitoring utilities
Repo          : @System
Matched from:
Filename      : /usr/bin/ps
Provide       : /bin/ps

procps-ng-3.3.17-1.amzn2023.0.2.x86_64 : System and process monitoring utilities
Repo          : amazonlinux
```

```
Matched from:
Filename      : /usr/bin/ps
Provide       : /bin/ps
```

AWS 콘솔에서 EKS Auto Mode와 연결된 리소스 보기

AWS 콘솔을 사용하여 EKS Auto Mode 클러스터와 연결된 리소스의 상태를 볼 수 있습니다.

- [EBS 볼륨](#)
 - 태그 키(eks:eks-cluster-name)를 검색하여 EKS Auto Mode 볼륨 보기
- [로드밸런서](#)
 - 태그 키(eks:eks-cluster-name)를 검색하여 EKS Auto Mode 로드 밸런서 보기
- [EC2 인스턴스](#)
 - 태그 키(eks:eks-cluster-name)를 검색하여 EKS Auto Mode 인스턴스 보기

AWS 계정에서 IAM 오류 보기

1. CloudTrail 콘솔로 이동
2. 왼쪽 탐색 창에서 '이벤트 기록' 선택
3. 오류 코드 필터 적용:
 - AccessDenied
 - UnauthorizedOperation
 - InvalidClientTokenId

EKS 클러스터와 관련된 오류를 찾습니다. 오류 메시지를 사용하여 EKS 액세스 항목, 클러스터 IAM 역할 또는 노드 IAM 역할을 업데이트합니다. EKS 자율 모드에 대한 권한이 있는 이러한 역할에 새 정책을 연결해야 할 수 있습니다.

포드가 자율 모드 노드로 예약되지 않음 문제 해결

포드가 자율 모드 노드로 예약되지 않은 경우 포드/배포 매니페스트에 PendingnodeSelectornodeSelector가 있는지 확인합니다. nodeSelector가 있는 경우 EKS 자동 모드에서 만든 노드에서 예약할 eks.amazonaws.com/compute-type: auto를 사용하고 있는지 확인합니다. EKS 자동 모드에서 사용하는 노드 레이블에 대한 자세한 내용은 [the section called “배포 제어”](#) 섹션을 참조하세요.

클러스터에 조인하지 않는 노드 문제 해결

EKS 자동 모드는 클러스터 엔드포인트 클러스터 인증 기관(CA)을 포함하여 클러스터에 조인하는 데 필요한 올바른 정보로 새 EC2 인스턴스를 자동 구성합니다. 그러나 이러한 인스턴스는 여전히 EKS 클러스터에 노드로 조인하지 못할 수 있습니다. 클러스터에 조인하지 않은 인스턴스를 식별하려면 다음 명령을 실행하세요.

1. `kubectl get nodeclaim`를 실행하여 `Ready = False`인 `NodeClaims`를 확인합니다.

```
kubectl get nodeclaim
```

2. `kubectl describe nodeclaim <node_claim>`을 실행하고 상태 아래에서 노드가 클러스터에 조인하는 데 방해가 되는 문제를 찾습니다.

```
kubectl describe nodeclaim <node_claim>
```

일반적인 오류 메시지:

Error getting launch template configs

기본 클러스터 IAM 역할 권한으로 `NodeClass`에서 사용자 지정 태그를 설정하는 경우 이 오류가 수신될 수 있습니다. [the section called “자격 증명 및 액세스”](#) 섹션을 참조하세요.

Error creating fleet

EC2 API에서 `RunInstances`를 직접적으로 호출하는 데 권한 부여 문제가 있을 수 있습니다. AWS CloudTrail에서 오류를 확인하고 [the section called “자율 모드 클러스터 IAM 역할”](#)에서 필요한 IAM 권한을 확인합니다.

VPC Reachability Analyzer로 노드 연결 문제 탐지

Note

VPC Reachability Analyzer에서 실행하는 각 분석에 대해 요금이 부과됩니다. 요금 세부 정보는 [Amazon VPC 요금](#)을 참조하세요.

인스턴스가 클러스터에 조인하지 못한 한 가지 이유는 인스턴스가 API 서버에 도달하지 못하게 하는 네트워크 연결 문제입니다. 이 문제를 진단하려면 [VPC Reachability Analyzer](#)를 사용하여 클러스터에 조인하지 못한 노드와 API 서버 간의 연결을 분석할 수 있습니다. 두 가지 정보가 필요합니다.

- 클러스터에 조인할 수 없는 노드의 인스턴스 ID
- Kubernetes API 서버 엔드포인트의 IP 주소

인스턴스 ID를 가져오려면 클러스터에서 워크로드를 생성하여 EKS 자동 모드가 EC2 인스턴스를 시작하도록 해야 합니다. 또한 클러스터에 인스턴스 ID가 있는 NodeClaim 객체가 생성됩니다. `kubectl get nodeclaim -o yaml`을 실행하여 클러스터의 모든 NodeClaims를 인쇄합니다. 각 NodeClaim에 인스턴스 ID가 필드로 포함되고 providerID에 다시 포함됩니다.

```
kubectl get nodeclaim -o yaml
```

예제 출력은 다음과 같습니다.

```
nodeName: i-01234567890123456
providerID: aws:///us-west-2a/i-01234567890123456
```

`kubectl get endpoint kubernetes -o yaml`을 실행하여 Kubernetes API 서버 엔드포인트를 확인할 수 있습니다. 주소는 주소 필드에 있습니다.

```
kubectl get endpoints kubernetes -o yaml
```

예제 출력은 다음과 같습니다.

```
apiVersion: v1
kind: Endpoints
metadata:
  name: kubernetes
  namespace: default
subsets:
- addresses:
  - ip: 10.0.143.233
  - ip: 10.0.152.17
  ports:
  - name: https
    port: 443
    protocol: TCP
```

이 두 가지 정보를 사용하여 분석을 수행할 수 있습니다. 먼저 AWS Management Console에서 VPC Reachability Analyzer로 이동합니다.

1. '경로 생성 및 분석'을 클릭합니다.
2. 분석 이름을 입력합니다(예: '노드 조인 실패').
3. '소스 유형'에서 '인스턴스'를 선택합니다.
4. 실패한 노드의 인스턴스 ID를 '소스'로 입력합니다.
5. '경로 대상'에서 'IP 주소'를 선택합니다.
6. API 서버의 IP 주소 중 하나를 '대상 주소'로 입력합니다.
7. '추가 패킷 헤더 구성 섹션'을 확장합니다.
8. '대상 포트'로 443을 입력합니다.
9. 아직 선택하지 않은 경우 '프로토콜'을 TCP로 선택합니다.
10. '경로 생성 및 분석'을 클릭합니다.
11. 분석을 완료하는 데 몇 분 정도 걸릴 수 있습니다. 분석 결과에 연결 실패가 표시되는 경우 네트워크 경로에 실패 위치가 표시되므로 문제를 해결할 수 있습니다.

포드 간 볼륨 공유

EKS 자동 모드 노드는 동일한 노드에서 실행 중인 포드 간에 더 많은 격리를 제공하는 적용 모드의 SELinux로 구성됩니다. SELinux가 활성화되면 권한이 없는 대부분의 포드에 자동으로 자체 다중 범주 보안(MCS) 레이블이 적용됩니다. 이 MCS 레이블은 포드마다 고유하며 한 포드의 프로세스가 다른 포드나 호스트의 프로세스를 조작할 수 없도록 설계되었습니다. 레이블이 지정된 포드가 루트로 실행되고 호스트 파일 시스템에 액세스할 수 있더라도 파일을 조작하거나, 호스트에서 민감한 시스템 직접 호출을 수행하거나, 컨테이너 런타임에 액세스하거나, kubelet의 보안 암호 키 자료를 가져올 수 없습니다.

이로 인해 포드 간에 데이터를 공유하려고 할 때 문제가 발생할 수 있습니다. 예를 들어 액세스 모드가 ReadWriteOnce인 PersistentVolumeClaim은 여전히 여러 포드가 볼륨에 동시에 액세스하는 것을 허용하지 않습니다.

포드 간에 이러한 공유를 활성화하려면 포드의 `seLinuxOptions`를 사용하여 해당 포드에 동일한 MCS 레이블을 구성하면 됩니다. 이 예제에서는 포드에 세 가지 범주인 `c123`, `c456`, `c789`를 할당합니다. 이는 노드의 포드에 자동으로 할당된 범주와 충돌하지 않습니다. 두 가지 범주만 할당되기 때문입니다.

```
securityContext:
```

```
seLinuxOptions:  
  level: "s0:c123,c456,c789"
```

자동 모드에서 포함된 컨트롤러 문제 해결

컨트롤러에 문제가 있는 경우 다음을 조사해야 합니다.

- 해당 컨트롤러와 연결된 리소스의 형식이 적절하고 유효한지 여부.
- AWS IAM 및 Kubernetes RBAC 리소스가 클러스터에 대해 올바르게 구성되었는지 여부. 자세한 내용은 [the section called “자격 증명 및 액세스”](#) 단원을 참조하십시오.

EKS 자동 모드 릴리스 정보 검토

이 페이지에서는 Amazon EKS 자동 모드에 대한 업데이트를 설명합니다. 이 페이지에서 기능, 버그 수정, 알려진 문제, 더 이상 지원되지 않는 기능에 대한 공지 사항을 주기적으로 확인할 수 있습니다.

이 특정 설명서 페이지의 모든 소스 파일 변경 사항에 대한 알림을 받으려면 RSS 리더를 사용하여 다음 URL을 구독하세요.

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/automode/  
auto-change.adoc.atom
```

2025년 6월 13일

기능: NodeClass에서 별도의 포드 서브넷 지원. 포드의 서브넷과 보안 그룹을 설정하기 위해 선택적 키 `podSubnetSelectorTerms`와 `podSecurityGroupSelectorTerms`가 추가되었습니다. 자세한 내용은 [the section called “포드의 서브넷 선택”](#) 섹션을 참조하세요.

2025년 4월 30일

기능: NodeClass에서 순방향 네트워크 프록시를 지원합니다. 이로 인해 HTTPS 프록시를 설정하는 선택적 키 `advancedNetworking`이 추가됩니다. 자세한 내용은 [the section called “노드 클래스 사양”](#) 섹션을 참조하세요.

2025년 4월 18일

기능: 유니캐스트 DNS를 통해 .local 도메인(일반적으로 멀티캐스트 DNS용으로 예약됨)을 확인할 수 있도록 지원합니다.

2025년 4월 11일

기능: NodeClass에 `certificateBundles`와 `ephemeralStorage.kmsKeyID`가 추가되었습니다. 자세한 내용은 [the section called “노드 클래스 사양”](#) 섹션을 참조하세요.

기능: 이미지 풀 속도가 개선되었습니다. 더 빠른 이미지 압축 해제를 활용할 수 있는 로컬 인스턴스 스토리지가 있는 인스턴스 유형의 경우 특히 그렇습니다.

버그 수정: 시작 즉시 노드로 예약하는 포드에 대해 `FailedCreatePodSandBox`(‘다이얼링 중 오류 발생: dial tcp 127.0.0.1:50051: connect: connection refused’)가 가끔 발생하는 경합 상태가 해결되었습니다.

2025년 4월 4일

기능: `registryPullQPS`를 5에서 25로 늘리고 `registryBurst`를 10에서 50으로 늘려 클라이언트가 강제로 적용하는 이미지 풀 스로틀링을 줄입니다(`Failed to pull image xyz: pull QPS exceeded`).

2025년 3월 31일

버그 수정: 코어 DNS 포드가 자동 모드 노드에서 실행 중일 때 노드의 포드에서 보내는 DNS 쿼리가 노드 로컬 DNS 서버가 아닌 해당 코어 DNS 포드에 도달하는 문제를 해결합니다. 자동 모드 노드의 포드에서 보내는 DNS 쿼리는 항상 노드 로컬 DNS로 갑니다.

2025년 3월 21일

버그 수정: 이제 클러스터에 `kube-dns` 서비스가 설치되어 있지 않아도 자동 모드 노드가 `kube-dns.kube-system.svc.cluster.local`을 올바르게 확인합니다. GitHub 문제 [#2546](#)을 해결합니다.

2025년 3월 14일

기능: IPv6 클러스터에서 IPv4 송신이 활성화되었습니다. IPv6 자동 모드 클러스터에서 송신되는 IPv4 트래픽은 이제 노드 프라이머리 ENI의 v4 주소로 자동 변환됩니다.

Amazon EKS 클러스터를 사용한 워크로드 구성

Amazon EKS 클러스터에는 두 가지 기본 구성 요소가 있습니다.

- Amazon EKS 제어 영역
- 제어 영역에 등록된 Amazon EKS 노드

Amazon EKS 제어 영역에는 etcd 및 Kubernetes API 서버와 같은 Kubernetes 소프트웨어를 실행하는 제어 영역 노드가 포함되어 있습니다. 제어 영역은 AWS에서 관리하는 계정에서 실행되며, Kubernetes API는 클러스터와 연결된 Amazon EKS 엔드포인트를 통해 노출됩니다. 각 Amazon EKS 클러스터 제어 영역은 단일 테넌트이고 고유하며 자체 Amazon EC2 인스턴스 집합에서 실행합니다.

etcd 노드에 의해 저장된 모든 데이터 및 연결된 Amazon EBS 볼륨은 AWS KMS를 사용하여 암호화됩니다. 클러스터 제어 플레인에는 여러 가용 영역에 프로비저닝되고 Elastic Load Balancing Network Load Balancer와 마주합니다. Amazon EKS는 또한 VPC 서브넷의 탄력적 네트워크 인터페이스를 프로비저닝하여 제어 영역 인스턴스에서 노드(예: kubectl exec logs proxy 데이터 흐름)로의 연결을 제공합니다.

Important

Amazon EKS 환경에서 etcd 스토리지는 [업스트림](#) 지침에 따라 8GiB로 제한됩니다. 다음 명령을 실행하여 현재 데이터베이스 크기에 대한 지표를 모니터링할 수 있습니다. 클러스터에서 Kubernetes 버전이 1.28 아래인 경우 `apiserver_storage_size_bytes`를 `apiserver_storage_db_total_size_in_bytes`로 바꿉니다.

```
kubectl get --raw=/metrics | grep "apiserver_storage_size_bytes"
```

Amazon EKS 노드는 AWS 계정에서 실행되고, 클러스터에 대해 생성된 인증서 파일 및 API 서버 엔드포인트를 통해 클러스터의 제어 영역에 연결됩니다.

Note

- [Amazon EKS 클러스터용 네트워킹 구성](#)에서 Amazon EKS의 다양한 구성 요소가 작동하는 방법을 확인할 수 있습니다.

- 연결된 클러스터는 [Amazon EKS Connector](#) 섹션을 참조하세요.

주제

- [Amazon EKS Auto Mode 클러스터 생성](#)
- [Amazon EKS 클러스터 생성](#)
- [클러스터 인사이트를 사용한 Kubernetes 버전 업그레이드 준비 및 잘못된 구성 문제 해결](#)
- [기존 클러스터를 새 Kubernetes 버전으로 업데이트](#)
- [클러스터 삭제](#)
- [클러스터 API 서버 엔드포인트](#)
- [EKS 클러스터에 Windows 노드 배포](#)
- [Windows 지원 사용 중지](#)
- [인터넷 액세스가 제한된 프라이빗 클러스터 배포](#)
- [EKS의 Kubernetes 버전 수명 주기 이해](#)
- [각 Kubernetes 버전에 대한 Amazon EKS 플랫폼 버전 보기](#)
- [Karpenter 및 Cluster Autoscaler를 사용한 클러스터 컴퓨팅 규모 조정](#)
- [Amazon EKS에서 Amazon 애플리케이션 복구 컨트롤러\(ARC\)의 구역 이동에 대해 알아보기](#)
- [손상된 가용 영역을 방지하려면 EKS 영역 전환을 활성화합니다.](#)

Amazon EKS Auto Mode 클러스터 생성

이 주제에서는 고급 구성 옵션을 사용하여 Amazon EKS Auto Mode 클러스터를 생성하는 방법에 대한 자세한 지침을 제공합니다. 사전 조건, 네트워킹 옵션, 추가 기능 구성을 다룹니다. 이 프로세스에는 IAM 역할 설정, 클러스터 설정 구성, 네트워킹 파라미터 지정, 추가 기능 선택이 포함됩니다. 사용자는 AWS Management Console 또는 AWS CLI를 사용하여 클러스터를 생성할 수 있으며, 두 방법 모두에 대한 단계별 지침이 제공됩니다.

덜 복잡한 설정 프로세스를 원하는 사용자의 경우 간소화된 클러스터 생성 단계는 다음을 참조하세요.

- [the section called “eksctl CLI”](#)
- [the section called “AWS CLI”](#)
- [the section called “관리 콘솔”](#)

이 고급 구성 가이드는 EKS Auto Mode 클러스터 설정을 보다 세밀하게 제어해야 하고 Amazon EKS 개념 및 요구 사항을 잘 알고 있는 사용자를 대상으로 합니다. 고급 구성을 진행하기 전에 모든 사전 요구 사항을 충족하고 EKS Auto Mode 클러스터의 네트워킹 및 IAM 요구 사항을 철저히 이해해야 합니다.

EKS Auto Mode에는 추가 IAM 권한이 필요합니다. 자세한 내용은 다음을 참조하세요.

- [the section called “EKS Auto Mode 클러스터에 대한 IAM 역할”](#)
- [the section called “자격 증명 및 액세스”](#)

Note

EKS Auto Mode 없이 클러스터를 생성하려면 [the section called “클러스터 생성”](#) 섹션을 참조하세요.

이 주제에서는 고급 구성을 다룹니다. EKS Auto Mode를 시작하려면 [the section called “클러스터 생성”](#) 섹션을 참조하세요.

사전 조건

- [Amazon EKS 요구 사항](#)을 충족하는 기존 VPC 및 서브넷 보유. 프로덕션 용도로 클러스터를 배포하기 전에 VPC 및 서브넷 요구 사항을 충분히 이해하는 것이 좋습니다. VPC 및 서브넷이 없는 경우 [Amazon EKS 제공 AWS CloudFormation 템플릿](#)을 사용하여 생성할 수 있습니다.
- kubectl 명령줄 도구는 장치 또는 AWS CloudShell에 설치됩니다. 버전은 클러스터의 Kubernetes 버전과 동일하거나 최대 하나 이전 또는 이후의 마이너 버전일 수 있습니다. 예를 들어, 클러스터 버전이 1.29인 경우 kubectl 버전 1.28, 1.29 또는 1.30를 함께 사용할 수 있습니다. kubectl을 설치하거나 업그레이드하려면 [the section called “kubectl 및 eksctl 설정”](#) 부분을 참조하세요.
- 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version`을 사용합니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요.
- EKS 및 IAM 리소스를 생성하고 수정할 권한이 있는 [IAM 보안 주체](#)입니다.

클러스터 생성 - AWS 콘솔

1. [Amazon EKS 콘솔](#)을 엽니다.

2. 클러스터 추가를 선택하고 생성을 선택합니다.
3. 구성 옵션에서 사용자 지정 구성을 선택합니다.
 - 이 주제에서는 사용자 지정 구성을 다룹니다. 빠른 구성에 대한 자세한 내용은 [the section called “관리 콘솔”](#) 섹션을 참조하세요.
4. EKS Auto Mode 사용이 활성화되어 있는지 확인합니다.
 - 이 주제에서는 EKS Auto Mode를 사용하여 클러스터를 생성하는 방법을 다룹니다. EKS Auto Mode 없이 클러스터를 생성하는 방법에 관한 자세한 내용은 [the section called “클러스터 생성”](#) 섹션을 참조하세요.
5. 클러스터 구성 페이지에서 다음 필드를 입력합니다.
 - 이름 - 클러스터의 이름 이름에는 영숫자(대소문자 구분)와 하이픈, 밑줄만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다.
 - 클러스터 IAM 역할-생성한 Amazon EKS 클러스터 IAM 역할을 선택하여 Kubernetes 컨트롤 플레인 사용자가 대신 AWS 리소스를 관리하게 할 수 있습니다. 이전에 EKS Auto Mode에 대한 클러스터 IAM 역할을 생성하지 않은 경우 권장 역할 생성 버튼을 선택하여 IAM 콘솔에서 필요한 권한을 가진 역할을 생성합니다.
 - Kubernetes 버전(Kubernetes version) - 클러스터에 대해 사용할 Kubernetes 버전 이전 버전이 필요한 경우가 아니면 최신 버전을 선택하는 것이 좋습니다.
 - 업그레이드 정책 - 클러스터에 설정하려는 Kubernetes 버전 정책입니다. 클러스터가 표준 지원 버전에서만 실행되게 하려면 표준을 선택할 수 있습니다. 버전에 대한 표준 지원이 종료될 때 클러스터가 확장 지원을 시작하게 하려면 확장을 선택할 수 있습니다. 현재 확장 지원 중인 Kubernetes 버전을 선택한 경우에는 표준 지원을 옵션으로 선택할 수 없습니다.
6. 클러스터 구성 페이지의 Auto Mode Compute 섹션에 다음 필드를 입력합니다.
 - 노드 풀-노드 풀에서 빌드를 사용할지 여부를 결정합니다. 자세한 내용은 [the section called “내장 노드 풀 검토”](#) 단원을 참조하십시오.
 - 노드 IAM 역할-내장 노드 풀을 활성화하는 경우 노드 IAM 역할을 선택해야 합니다. EKS Auto Mode는 이 역할을 새 노드에 할당합니다. 클러스터가 생성된 후에는 이 값을 변경할 수 없습니다. 이전에 EKS Auto Mode에 대한 노드 IAM 역할을 생성하지 않은 경우 권장 역할 생성 버튼을 선택하여 필요한 권한을 가진 역할을 생성합니다. 이에 대한 자세한 내용은 [the section called “자격 증명 및 액세스”](#) 섹션을 참조하세요.
7. 클러스터 구성 페이지의 클러스터 액세스 섹션에 다음 필드를 입력합니다.
 - 부트스트랩 클러스터 관리자 액세스-클러스터 생성자는 자동으로 Kubernetes 관리자입니다. 이를 비활성화하려면 클러스터 관리자 액세스 허용 안 함을 선택합니다.

- 클러스터 인증 모드-EKS Auto Mode에는 EKS API 인증 모드인 EKS 액세스 항목이 필요합니다. 필요에 따라 EKS API 및 ConfigMap을 선택하여 ConfigMap 인증 모드를 활성화할 수 있습니다.

8. 클러스터 구성 페이지에서 나머지 필드를 입력합니다.

- 비밀 암호화(Secrets encryption) - (선택 사항) KMS 키를 사용하여 Kubernetes 비밀의 비밀 암호화를 사용 설정하도록 선택합니다. 클러스터를 생성한 후에도 이 기능을 사용 설정할 수 있습니다. 이 기능을 사용 설정하기 전에 [the section called “비밀 암호화 활성화”](#)의 정보를 숙지해야 합니다.
- ARC 영역 전환-EKS Auto Mode는 Arc 영역 전환을 지원하지 않습니다.
- 태그-(선택사항) 클러스터에 태그를 추가합니다. 자세한 내용은 [the section called “리소스에 태그”](#) 섹션을 참조하세요.

이 페이지를 모두 완료하면 다음을 선택합니다.

9. 네트워킹 지정 페이지에서 다음 필드의 값을 선택합니다.

- VPC-클러스터를 생성하려면 [Amazon EKS VPC 요구 사항](#)을 충족하는 기존 VPC를 선택합니다. VPC를 선택하기 전에 [the section called “VPC 및 서브넷 요구 사항”](#)의 요구 사항 및 고려 사항 모두를 숙지하는 것이 좋습니다. 클러스터 생성 후에는 사용하려는 VPC를 변경할 수 없습니다. 목록에 기존 VPC가 없는 경우 먼저 VPC를 생성해야 합니다. 자세한 내용은 [the section called “VPC 생성”](#) 섹션을 참조하세요.
- 서브넷-기본적으로 이전 필드에서 지정한 VPC에서 사용 가능한 모든 서브넷이 사전 선택됩니다. 두 개 이상을 선택해야 합니다.

선택한 서브넷은 [Amazon EKS 서브넷 요구 사항](#)을 충족해야 합니다. 서브넷을 선택하기 전에 [Amazon EKS VPC 및 서브넷 요구 사항 및 고려 사항](#) 모두를 숙지하는 것이 좋습니다.

보안 그룹 - (선택사항) 생성하는 네트워크 인터페이스에 Amazon EKS가 연결하려는 보안 그룹을 하나 이상 지정합니다.

보안 그룹 선택 여부에 관계없이 Amazon EKS는 클러스터와 VPC 간에 통신을 사용 설정할 수 있는 보안 그룹을 생성합니다. Amazon EKS는 이 보안 그룹과 사용자가 선택한 보안 그룹을 생성하는 네트워크 인터페이스에 연결합니다. Amazon EKS에서 생성하는 클러스터 보안 그룹에 대한 자세한 내용을 알아보려면 [the section called “보안 그룹 요구 사항”](#) 부분을 참조하세요. Amazon EKS가 생성하는 클러스터 보안 그룹에서 규칙을 수정할 수 있습니다.

- 클러스터 IP 주소 패밀리 선택 - IPv4와 IPv6 중 하나를 선택할 수 있습니다.

Kubernetes는 기본적으로 포드와 서비스에 IPv4 주소를 할당합니다. IPv6 제품군 사용을 결정하기 전에 [VPC 요구 사항 및 고려 사항](#), [the section called “서브넷 요구 사항 및 고려 사항”](#), [the](#)

[section called “보안 그룹 요구 사항”](#) 및 [the section called “IPv6”](#) 항목의 모든 고려 사항과 요구 사항을 숙지하고 있는지 확인하세요. IPv6 패밀리를 선택하면 IPv4 패밀리처럼 Kubernetes가 IPv6 서비스 주소를 할당할 주소 범위를 지정할 수 없습니다. Kubernetes는 고유한 로컬 주소 범위(fc00::/7)에서 서비스 주소를 할당합니다.

- (선택 사항) Kubernetes 서비스 IP 주소 범위 구성(Configure Kubernetes Service IP address range)을 선택하고 서비스 **IPv4** 범위(Service range)를 지정합니다.

자체 범위를 지정하면 VPC에 피어링되거나 연결된 Kubernetes 서비스와 기타 네트워크 사이의 충돌을 방지할 수 있습니다. CIDR 표기법으로 범위를 입력합니다. 예: 10.2.0.0/16.

CIDR 블록은 다음 요구 사항을 충족해야 합니다.

- 다음 범위 10.0.0.0/8, 172.16.0.0/12 또는 192.168.0.0/16 중 하나 내에 있어야 합니다..
- 최소 크기는 /24고 최대 크기는 /12입니다.
- Amazon EKS 리소스의 VPC 범위와 겹치지 않습니다.

이 옵션은 IPv4 주소 패밀리를 사용할 때 및 클러스터 생성 시에만 지정할 수 있습니다. 이 옵션을 지정하지 않은 경우 Kubernetes는 10.100.0.0/16 또는 172.20.0.0/16 CIDR 블록 중 하나에서 서비스 IP 주소를 할당합니다.

- 클러스터 엔드포인트 액세스에서 옵션을 선택합니다. 클러스터를 생성한 후에는 이 옵션을 변경할 수 있습니다. 기본값이 아닌 옵션을 선택하기 전에 옵션과 그 의미를 숙지해야 합니다. 자세한 내용은 [the section called “클러스터 엔드포인트 액세스”](#) 섹션을 참조하세요.

이 페이지를 모두 완료하면 다음을 선택합니다.

10(선택사항) 관찰성 구성 페이지에서 설정할 지표 및 컨트롤 플레인 로깅 옵션을 선택합니다. 기본적으로 각 로그 유형은 해제되어 있습니다.

- Prometheus 지표 옵션에 자세한 내용은 [the section called “1단계: Prometheus 지표 켜기”](#) 섹션을 참조하세요.
- 컨트롤 플레인 로깅 옵션에 대한 자세한 내용은 [the section called “컨트롤 플레인 로그”](#) 섹션을 참조하세요.
- 이 페이지를 모두 완료하면 다음을 선택합니다.

11.추가 기능 선택 페이지에서 클러스터에 추가할 추가 기능을 선택합니다. 필요한 만큼 Amazon EKS 애드온 및 AWS 마켓플레이스 애드온을 선택할 수 있습니다. 설치하려는 AWS 마켓플레이스 추가 기능이 목록에 없는 경우 페이지 번호를 클릭하여 추가 페이지 결과를 확인하거나 검색창에 텍스트를 입력하여 사용 가능한 AWS 마켓플레이스 추가 기능을 검색할 수 있습니다. 카테고리, 공급업체 또는 가격 모델별로 필터링한 다음 검색 결과 중에서 추가 기능을 선택할 수도 있습니다. 클러스터

를 생성할 때 [the section called “Pod Identity”](#)에 설명된 대로 EKS Pod Identity를 지원하는 추가 기능을 보고, 선택하고, 설치할 수 있습니다.

- EKS Auto Mode는 특정 추가 기능을 자동화합니다. EKS 관리형 노드 그룹을 EKS Auto Mode 클러스터에 배포하려는 경우 추가 Amazon EKS 추가 기능을 선택하고 옵션을 검토합니다. CoreDNS 및 kube-proxy와 같은 추가 기능을 설치해야 할 수 있습니다. EKS는 이 섹션의 추가 기능만 자체 관리형 노드 및 노드 그룹에 설치합니다.
- 이 페이지를 모두 완료하면 다음을 선택합니다.

12.선택한 추가 기능 설정 구성 페이지에서 설치할 버전을 선택합니다. 클러스터 생성 후 언제든지 최신 버전으로 업데이트할 수 있습니다.

EKS Pod Identity를 지원하는 추가 기능의 경우 콘솔을 사용하여 추가 기능에 대해 미리 채워진 이름, AWS 관리형 정책, 신뢰 정책에 따라 역할을 자동으로 생성할 수 있습니다. 지원되는 추가 기능에 기존 역할을 재사용하거나 새 역할을 생성할 수 있습니다. 콘솔을 사용하여 EKS Pod Identity를 지원하는 추가 기능에 대한 역할을 생성하는 단계는 [the section called “추가 기능 생성\(AWS 콘솔\)”](#) 섹션을 참조하세요. 추가 기능이 EKS Pod Identity를 지원하지 않는 경우 클러스터가 생성된 후 마법사를 사용하여 서비스 계정에 대한 IAM 역할(IRSA)을 생성하는 지침이 포함된 메시지가 표시됩니다.

클러스터 생성 후 각 추가 기능의 구성을 업데이트할 수 있습니다. 추가 기능 구성에 대한 자세한 내용은 [the section called “추가 기능 업데이트”](#) 섹션을 참조하세요. 이 페이지를 모두 완료하면 다음을 선택합니다.

13.검토 및 생성 페이지에서 이전 페이지에서 입력하거나 선택한 정보를 검토합니다. 변경해야 하는 경우 편집을 선택합니다 만족하는 경우 생성을 선택합니다. 클러스터가 프로비저닝되는 동안 상태 필드에는 생성 중이 표시됩니다.

Note

요청의 가용 영역 중 하나에 Amazon EKS 클러스터를 생성하는 데 충분한 용량이 없다는 오류가 표시될 수 있습니다. 이 경우 오류 출력에는 새 클러스터를 지원할 수 있는 가용 영역이 포함됩니다. 사용자 계정의 지원 가용 영역에 있는 2개 이상의 서브넷을 사용하여 클러스터를 다시 생성합니다. 자세한 내용은 [the section called “용량 부족”](#) 섹션을 참조하세요.

클러스터 프로비저닝에는 몇 분 정도 걸립니다.

클러스터 생성-AWS CLI

다음 CLI 지침에서는 IAM 리소스 생성 및 클러스터 생성을 다룹니다.

EKS Auto Mode 클러스터 IAM 역할 생성

1단계: 신뢰 정책 생성

Amazon EKS 서비스가 이 역할을 수임하도록 허용하는 신뢰 정책을 생성합니다. 정책을 `trust-policy.json`으로 저장합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

2단계: IAM 역할 생성

신뢰 정책을 사용하여 클러스터 IAM 역할을 생성합니다.

```
aws iam create-role \
  --role-name AmazonEKSAutoClusterRole \
  --assume-role-policy-document file://trust-policy.json
```

3단계: 역할 ARN 기록

다음 단계에서 사용할 새 역할의 ARN을 검색하고 저장합니다.

```
aws iam get-role --role-name AmazonEKSAutoClusterRole --query "Role.Arn" --output text
```

4단계: 필수 정책 연결

다음 AWS 관리형 정책을 클러스터 IAM 역할에 연결하여 필요한 권한을 부여합니다.

AmazonEKSClusterPolicy:

```
aws iam attach-role-policy \  
  --role-name AmazonEKSAutoClusterRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
```

AmazonEKSComputePolicy:

```
aws iam attach-role-policy \  
  --role-name AmazonEKSAutoClusterRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSComputePolicy
```

AmazonEKSBlockStoragePolicy:

```
aws iam attach-role-policy \  
  --role-name AmazonEKSAutoClusterRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSBlockStoragePolicy
```

AmazonEKSLoadBalancingPolicy:

```
aws iam attach-role-policy \  
  --role-name AmazonEKSAutoClusterRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSLoadBalancingPolicy
```

AmazonEKSNetworkingPolicy:

```
aws iam attach-role-policy \  
  --role-name AmazonEKSAutoClusterRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSNetworkingPolicy
```

EKS Auto Mode 노드 IAM 역할 생성

1단계: 신뢰 정책 생성

Amazon EKS 서비스가 이 역할을 수임하도록 허용하는 신뢰 정책을 생성합니다. 정책을 `node-trust-policy.json`으로 저장합니다.


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2단계: 노드 IAM 역할 생성

이전 단계의 node-trust-policy.json 파일을 사용하여 역할을 수임할 수 있는 엔터티를 정의합니다. 다음 명령을 실행하여 노드 IAM 역할을 만듭니다.

```
aws iam create-role \
  --role-name AmazonEKSAutoNodeRole \
  --assume-role-policy-document file://node-trust-policy.json
```

3단계: 역할 ARN 기록

역할을 생성한 후 노드 IAM 역할의 ARN을 검색하고 저장합니다. 이후 단계에서 이 ARN이 필요합니다. 다음 명령을 사용하여 ARN을 가져옵니다.

```
aws iam get-role --role-name AmazonEKSAutoNodeRole --query "Role.Arn" --output text
```

4단계: 필수 정책 연결

다음 AWS 관리형 정책을 노드 IAM 역할에 연결하여 필요한 권한을 제공합니다.

AmazonEKSWorkerNodeMinimalPolicy:

```
aws iam attach-role-policy \
  --role-name AmazonEKSAutoNodeRole \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodeMinimalPolicy
```

AmazonEC2ContainerRegistryPullOnly:

```
aws iam attach-role-policy \
  --role-name AmazonEKSAutoNodeRole \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

클러스터 생성

1. 다음 명령을 사용하여 클러스터를 생성합니다. 명령을 실행하기 전에 다음과 같은 바꾸기를 합니다.

- **region-code**를 클러스터를 생성할 AWS 리전으로 바꿉니다.
- **my-cluster**를 클러스터의 이름으로 바꿉니다. 이름에는 영숫자(대소문자 구분)와 하이픈, 밑줄만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다.
- **1.30**를 [Amazon EKS 지원 버전](#)으로 바꿉니다.
- **111122223333**을 계정 ID로 바꿉니다.
- 클러스터 및 노드 역할에 대해 이름이 다른 IAM 역할을 생성한 경우 ARN을 교체합니다.
- subnetIds의 값을 고유한 값으로 바꿉니다. 또한 추가 ID를 추가할 수 있습니다. 2개 이상의 서브넷 ID를 지정해야 합니다.

선택한 서브넷은 [Amazon EKS 서브넷 요구 사항](#)을 충족해야 합니다. 서브넷을 선택하기 전에 [Amazon EKS VPC 및 서브넷 요구 사항 및 고려 사항](#) 모두를 숙지하는 것이 좋습니다.

- 보안 그룹 ID를 지정하지 않으려면 명령에서 ,securityGroupIds=sg-<ExampleID1>을 제거합니다. 하나 이상의 보안 그룹 ID를 지정하려는 경우 securityGroupIds의 값을 고유한 값으로 바꿉니다. 또한 추가 ID를 추가할 수 있습니다.

보안 그룹 선택 여부에 관계없이 Amazon EKS는 클러스터와 VPC 간에 통신을 사용 설정할 수 있는 보안 그룹을 생성합니다. Amazon EKS는 이 보안 그룹과 사용자가 선택한 보안 그룹을 생성하는 네트워크 인터페이스에 연결합니다. Amazon EKS에서 생성하는 클러스터 보안 그룹에 대한 자세한 내용을 알아보려면 [the section called “보안 그룹 요구 사항”](#) 부분을 참조하세요. Amazon EKS가 생성하는 클러스터 보안 그룹에서 규칙을 수정할 수 있습니다.

```
aws eks create-cluster \
  --region region-code \
  --name my-cluster \
  --kubernetes-version 1.30 \
  --role-arn arn:aws:iam::111122223333:role/AmazonEKSAutoClusterRole \
  --resources-vpc-config '{"subnetIds": ["subnet-ExampleID1", "subnet-ExampleID2"],
  "securityGroupIds": ["sg-ExampleID1"], "endpointPublicAccess": true,
  "endpointPrivateAccess": true}' \
```

```
--compute-config '{"enabled": true, "nodeRoleArn":
"arn:aws:iam::111122223333:role/AmazonEKSAutoNodeRole", "nodePools": ["general-
purpose", "system"]}' \
--kubernetes-network-config '{"elasticLoadBalancing": {"enabled": true}}' \
--storage-config '{"blockStorage": {"enabled": true}}' \
--access-config '{"authenticationMode": "API"}'
```

Note

요청의 가용 영역 중 하나에 Amazon EKS 클러스터를 생성하는 데 충분한 용량이 없다는 오류가 표시될 수 있습니다. 이 경우 오류 출력에는 새 클러스터를 지원할 수 있는 가용 영역이 포함됩니다. 사용자 계정의 지원 가용 영역에 있는 2개 이상의 서브넷을 사용하여 클러스터를 다시 생성합니다. 자세한 내용은 [the section called “용량 부족”](#) 섹션을 참조하세요.

다음은 필요한 경우 이전 명령에 추가해야 하는 선택 가능한 설정입니다. 클러스터 생성 후가 아닌 생성할 때만 이러한 옵션을 사용 설정할 수 없습니다.

- Kubernetes가 서비스 IP 주소를 할당하는 IPv4 Classless Inter-Domain Routing(CIDR) 블록을 지정하려는 경우 다음 명령에 `--kubernetes-network-config serviceIpv4Cidr=<cidr-block>`을 추가하여 해당 블록을 지정해야 합니다.

자체 범위를 지정하면 VPC에 피어링되거나 연결된 Kubernetes 서비스와 기타 네트워크 사이의 충돌을 방지할 수 있습니다. CIDR 표기법으로 범위를 입력합니다. 예: `10.2.0.0/16`.

CIDR 블록은 다음 요구 사항을 충족해야 합니다.

- 다음 범위 `10.0.0.0/8`, `172.16.0.0/12` 또는 `192.168.0.0/16` 중 하나 내에 있어야 합니다..
- 최소 크기는 `/24`고 최대 크기는 `/12`입니다.
- Amazon EKS 리소스의 VPC 범위와 겹치지 않습니다.

이 옵션은 IPv4 주소 패밀리를 사용할 때 및 클러스터 생성 시에만 지정할 수 있습니다. 이 옵션을 지정하지 않은 경우 Kubernetes는 `10.100.0.0/16` 또는 `172.20.0.0/16` CIDR 블록 중 하나에서 서비스 IP 주소를 할당합니다.

- 클러스터를 생성하고 클러스터가 IPv4 주소 대신 포드와 서비스에 IPv6 주소를 할당하도록 하려면 다음 명령에 `--kubernetes-network-config ipFamily=ipv6`를 추가합니다.

Kubernetes는 기본적으로 포트와 서비스에 IPv4 주소를 할당합니다. IPv6 제품군 사용을 결정하기 전에 [VPC 요구 사항 및 고려 사항](#), [the section called “서브넷 요구 사항 및 고려 사항”](#), [the section called “보안 그룹 요구 사항”](#) 및 [the section called “IPv6”](#) 항목의 모든 고려 사항과 요구 사항을 숙지하고 있는지 확인하세요. IPv6 패밀리를 선택하면 IPv4 패밀리처럼 Kubernetes가 IPv6 서비스 주소를 할당할 주소 범위를 지정할 수 없습니다. Kubernetes는 고유한 로컬 주소 범위(fc00::/7)에서 서비스 주소를 할당합니다.

2. 클러스터를 프로비저닝하는 데 몇 분 정도 걸립니다. 다음 명령을 사용하여 클러스터의 상태를 쿼리할 수 있습니다.

```
aws eks describe-cluster --region region-code --name my-cluster --query
"cluster.status"
```

다음 단계

- [the section called “kubectl을 사용하여 클러스터에 액세스”](#)
- [the section called “권한 부여”](#)
- [클러스터에서 보안 암호 암호화 활성화](#)
- [클러스터에서 로깅 구성](#)
- [클러스터에 노드 추가](#)

Amazon EKS 클러스터 생성

Note

이 주제에서는 EKS Auto Mode 없이 EKS 클러스터를 생성하는 방법을 다룹니다.

EKS Auto Mode 클러스터 생성 관련 지침은 [the section called “자동 클러스터 생성”](#) 섹션을 참조하세요.

EKS Auto Mode를 시작하려면 [the section called “클러스터 생성\(EKS 자동 모드\)”](#) 섹션을 참조하세요.

이 주제에서는 사용 가능한 옵션에 대한 개요와 Amazon EKS 클러스터를 생성할 때 고려해야 할 사항을 설명합니다. 온프레미스 인프라를 노드용 컴퓨팅으로 사용하여 클러스터를 생성해야 하는 경우 [the section called “클러스터 생성”](#) 섹션을 참조하세요. Amazon EKS 클러스터를 처음 생성하는 경우 [시](#)

[작](#)의 가이드 중 한 가지를 따르는 것이 좋습니다. 이 가이드를 사용하면 사용 가능한 모든 옵션으로 확장하지 않고 간단한 기본 클러스터를 생성할 수 있습니다.

사전 조건

- [Amazon EKS 요구 사항](#)을 충족하는 기존 VPC 및 서브넷 보유. 프로덕션 용도로 클러스터를 배포하기 전에 VPC 및 서브넷 요구 사항을 충분히 이해하는 것이 좋습니다. VPC 및 서브넷이 없는 경우 [Amazon EKS 제공 AWS CloudFormation 템플릿](#)을 사용하여 생성할 수 있습니다.
- kubectl 명령줄 도구는 장치 또는 AWS CloudShell에 설치됩니다. 버전은 클러스터의 Kubernetes 버전과 동일하거나 최대 하나 이전 또는 이후의 마이너 버전일 수 있습니다. kubectl을 설치하거나 업그레이드하려면 [the section called “kubectl 및 eksctl 설정”](#) 부분을 참조하세요.
- 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.
- Amazon EKS 클러스터를 create 및 describe 할 수 있는 권한이 있는 [IAM 보안 주체](#). 자세한 내용은 [the section called “Outpost에서 로컬 Kubernetes 클러스터 생성”](#) 및 [the section called “모든 클러스터 나열 또는 설명”](#) 섹션을 참조하세요.

1단계: 클러스터 IAM 역할 만들기

1. 이미 클러스터 IAM 역할이 있거나 eksctl을 사용하여 클러스터를 생성하려는 경우 이 단계를 건너뛸 수 있습니다. 기본적으로 eksctl은 사용자를 위한 역할을 생성합니다.
2. 다음 명령을 실행하여 IAM 신뢰 정책 JSON 파일을 생성합니다.

```
cat >eks-cluster-role-trust-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
```

```
    },
    "Action": "sts:AssumeRole"
  }
]
}
EOF
```

3. Amazon EKS 클러스터 IAM 역할을 생성합니다. 필요한 경우 *eks-cluster-role-trust-policy.json* 앞에 이전 단계에서 파일을 작성한 컴퓨터의 경로를 붙입니다. 명령은 사용자가 이 전 단계에서 생성한 신뢰 정책을 역할에 연결합니다. [IAM 역할](#)을 생성하려면 역할을 생성하는 IAM 보안 주체에 iam:CreateRole 작업(권한)을 할당해야 합니다.

```
aws iam create-role --role-name myAmazonEKSClusterRole --assume-role-policy-document
file://"eks-cluster-role-trust-policy.json"
```

4. Amazon EKS 관리형 정책을 할당하거나 사용자 지정 정책을 생성할 수 있습니다. 사용자 지정 정책에서 사용해야 하는 최소 권한은 [the section called “클러스터 IAM 역할”](#)(를) 참조하세요.

역할에 [AmazonEKSClusterPolicy](#)라는 Amazon EKS 관리 정책을 연결합니다. IAM 정책을 [IAM 보안 주체](#)에 연결하려면 정책을 연결하는 보안 주체에 iam:AttachUserPolicy 또는 iam:AttachRolePolicy의 IAM 작업(권한) 중 하나를 할당해야 합니다.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSClusterPolicy --role-name myAmazonEKSClusterRole
```

서비스 연결 역할

Amazon EMR은 AWSServiceRoleForAmazonEKS라고 하는 서비스 연결 역할을 자동으로 생성합니다.

이 역할은 클러스터 IAM 역할에 추가되었습니다. 서비스 연결 역할은 Amazon EKS에 직접 연결된 고유한 유형의 IAM 역할입니다. 이 역할을 통해 Amazon EKS는 계정의 클러스터를 관리할 수 있습니다. 자세한 내용은 [the section called “클러스터 역할”](#) 섹션을 참조하세요.

EKS 클러스터를 생성하는 데 사용하는 IAM 자격 증명에는 서비스 연결 역할을 생성할 수 있는 권한이 있어야 합니다. 여기에는 iam:CreateServiceLinkedRole 권한이 포함되어 있습니다.

서비스 연결 역할이 아직 존재하지 않고 현재 IAM 역할에 클러스터를 생성할 수 있는 충분한 권한이 없는 경우 클러스터 생성 작업이 실패합니다.

2단계: 클러스터 생성

다음을 사용하여 클러스터를 만들 수 있습니다.

- [eksctl](#)
- [AWS Management Console](#)
- [AWS CLI](#)

클러스터 생성 - eksctl

1. 장치에 설치된 eksctl 명령줄 도구의 버전 0.210.0 이상 또는 AWS CloudShell이 필요합니다. eksctl을 설치 또는 업그레이드하려면 eksctl 설명서에서 [설치](#)를 참조하세요.
2. 기본 AWS 리전에서 Amazon EKS 기본 Kubernetes 버전으로 Amazon EKS IPv4 클러스터를 생성합니다. 명령을 실행하기 전에 다음과 같은 바꾸기를 합니다.
3. **region-code**를 클러스터를 생성할 AWS 리전으로 바꿉니다.
4. **my-cluster**를 클러스터의 이름으로 바꿉니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다.
5. **1.32**를 [Amazon EKS 지원 버전](#)으로 바꿉니다.
6. 요구 사항을 충족하는 vpc-private-subnets의 값을 변경합니다. 또한 추가 ID를 추가할 수 있습니다. 2개 이상의 서브넷 ID를 지정해야 합니다. 퍼블릭 서브넷을 지정하려는 경우 --vpc-private-subnets을 --vpc-public-subnets으로 변경할 수 있습니다. 퍼블릭 서브넷에는 인터넷 게이트웨이에 대한 경로가 포함된 연결된 라우팅 테이블이 있지만 프라이빗 서브넷에는 연결된 라우팅 테이블이 없습니다. 가능하면 프라이빗 서브넷을 사용하는 것이 좋습니다.

선택한 서브넷은 [Amazon EKS 서브넷 요구 사항](#)을 충족해야 합니다. 서브넷을 선택하기 전에 [Amazon EKS VPC 및 서브넷 요구 사항 및 고려 사항](#) 모두를 숙지하는 것이 좋습니다.

7. 다음 명령 실행:

```
eksctl create cluster --name my-cluster --region region-code --version 1.32 --vpc-private-subnets subnet-ExampleID1,subnet-ExampleID2 --without-nodegroup
```

클러스터 프로비저닝에는 몇 분 정도 걸립니다. 클러스터가 생성되는 동안 여러 줄의 출력이 나타납니다. 출력의 마지막 줄은 다음 예제와 유사합니다.

```
[#] EKS cluster "my-cluster" in "region-code" region is ready
```

8. [the section called “3단계: kubeconfig 업데이트”](#) 계속 진행

선택적 설정

eksctl을 사용하여 클러스터를 생성할 때 지정할 수 있는 대부분의 옵션을 보려면 `eksctl create cluster --help` 명령을 사용합니다. 사용 가능한 옵션을 모두 확인하려면 `config` 파일을 사용할 수 있습니다. 자세한 내용은 eksctl 설명서에서 [config 파일 사용](#) 및 [config 파일 스키마](#) 부분을 참조하세요. GitHub에서 [구성 파일 예제](#)를 찾을 수 있습니다.

다음은 필요한 경우 이전 명령에 추가해야 하는 선택 가능한 설정입니다. 클러스터 생성 후가 아닌 생성할 때만 이러한 옵션을 사용 설정할 수 없습니다. 이러한 옵션을 지정해야 하는 경우 [eksctl 구성 파일](#)을 사용하여 클러스터를 생성한 다음 이전 명령을 사용하지 않고 설정을 지정해야 합니다.

- 생성하는 네트워크 인터페이스에 Amazon EKS가 할당하는 보안 그룹을 하나 이상 지정하는 경우 [securityGroup](#) 옵션을 지정합니다.

보안 그룹 선택 여부에 관계없이 Amazon EKS는 클러스터와 VPC 간에 통신을 사용 설정할 수 있는 보안 그룹을 생성합니다. Amazon EKS는 이 보안 그룹과 사용자가 선택한 보안 그룹을 생성하는 네트워크 인터페이스에 연결합니다. Amazon EKS에서 생성하는 클러스터 보안 그룹에 대한 자세한 내용을 알아보려면 [the section called “보안 그룹 요구 사항”](#) 부분을 참조하세요. Amazon EKS가 생성하는 클러스터 보안 그룹에서 규칙을 수정할 수 있습니다.

- 가 서비스 IP 주소를 할당하는 IPv4 CIDR(Classless Inter-Domain Routing) 블록을 지정하려면 [serviceIPv4CIDR](#) 옵션을 지정합니다.

자체 범위를 지정하면 VPC에 피어링되거나 연결된 Kubernetes 서비스와 기타 네트워크 사이의 충돌을 방지할 수 있습니다. CIDR 표기법으로 범위를 입력합니다. 예를 들면 10.2.0.0/16입니다.

CIDR 블록은 다음 요구 사항을 충족해야 합니다.

- 다음 범위 10.0.0.0/8, 172.16.0.0/12 또는 192.168.0.0/16 중 하나 내에 있어야 합니다..
- 최소 크기는 /24고 최대 크기는 /12입니다.
- Amazon EKS 리소스의 VPC 범위와 겹치지 않습니다.

이 옵션은 IPv4 주소 패밀리를 사용할 때 및 클러스터 생성 시에만 지정할 수 있습니다. 이 옵션을 지정하지 않은 경우 Kubernetes는 10.100.0.0/16 또는 172.20.0.0/16 CIDR 블록 중 하나에서 서비스 IP 주소를 할당합니다.

- 클러스터를 만들 때 클러스터가 포트와 서비스에 IPv4 주소 대신 IPv6 주소를 할당하도록 하려면 [ipFamily](#) 옵션을 지정하세요.

Kubernetes는 기본적으로 포트와 서비스에 IPv4 주소를 할당합니다. IPv6 패밀리를 사용하기로 결정하기 전에 [the section called “VPC 요구 사항 및 고려 사항”](#), [the section called “서브넷 요구 사항 및 고려 사항”](#), [the section called “보안 그룹 요구 사항”](#) 및 [the section called “IPv6”](#) 주제의 모든 고려 사항 및 요구 사항을 숙지해야 합니다. IPv6 패밀리를 선택하면 IPv4 패밀리처럼 Kubernetes가 IPv6 서비스 주소를 할당할 주소 범위를 지정할 수 없습니다. Kubernetes는 고유한 로컬 주소 범위 (fc00::/7)에서 서비스 주소를 할당합니다.

클러스터 생성 - AWS 콘솔

- [Amazon EKS 콘솔](#)을 엽니다.
- 클러스터 추가를 선택하고 생성을 선택합니다.
- 구성 옵션에서 사용자 지정 구성을 선택합니다.
 - EKS Auto Mode를 사용하여 클러스터를 빠르게 생성하는 방법에 대한 자세한 내용은 [the section called “관리 콘솔”](#) 섹션을 참조하세요.
- EKS Auto Mode에서 EKS Auto Mode 사용을 끕니다.
 - 사용자 지정 구성을 사용하여 EKS Auto Mode 클러스터를 생성하는 방법에 대한 자세한 내용은 [the section called “자동 클러스터 생성”](#) 섹션을 참조하세요.
- 클러스터 구성 페이지에서 다음 필드를 입력합니다.
 - 이름 - 클러스터의 이름 이름에는 영숫자(대소문자 구분)와 하이픈, 밑줄만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다.
 - 클러스터 IAM 역할-생성한 Amazon EKS 클러스터 IAM 역할을 선택하여 Kubernetes 컨트롤 플레인 사용자가 대신 AWS 리소스를 관리하게 할 수 있습니다.
 - Kubernetes 버전(Kubernetes version) - 클러스터에 대해 사용할 Kubernetes 버전 이전 버전이 필요한 경우가 아니면 최신 버전을 선택하는 것이 좋습니다.
 - 지원 유형 - 클러스터에 설정하려는 Kubernetes 버전 정책입니다. 클러스터가 표준 지원 버전에 서만 실행되도록 하려면 표준 지원을 선택할 수 있습니다. 버전에 대한 표준 지원이 종료될 때 클러스터가 추가 지원을 시작하도록 하려면 추가 지원을 선택할 수 있습니다. 현재 확장 지원 중인 Kubernetes 버전을 선택한 경우에는 표준 지원을 옵션으로 선택할 수 없습니다.
 - 비밀 암호화(Secrets encryption) - (선택 사항) KMS 키를 사용하여 Kubernetes 비밀의 비밀 암호화를 사용 설정하도록 선택합니다. 클러스터를 생성한 후에도 이 기능을 사용 설정할 수 있습니다.

다. 이 기능을 사용 설정하기 전에 [the section called “비밀 암호화 활성화”](#)의 정보를 숙지해야 합니다.

- 태그-(선택사항) 클러스터에 태그를 추가합니다. 자세한 내용은 [the section called “리소스에 태그”](#) 섹션을 참조하세요.
- ARC 영역 전환-(선택 사항) Route53 Application Recovery 컨트롤러를 사용하여 손상된 가용 영역을 완화할 수 있습니다. 자세한 내용은 [the section called “구역별 교대에 대해 알아보기”](#) 섹션을 참조하세요.

6. 클러스터 구성 페이지의 클러스터 액세스 섹션에 다음 필드를 입력합니다.

- 부트스트랩 클러스터 관리자 액세스-클러스터 생성자는 자동으로 Kubernetes 관리자입니다. 이를 비활성화하려면 클러스터 관리자 액세스 허용 안 함을 선택합니다.
- 클러스터 인증 모드-IAM 사용자 및 역할에 Kubernetes API에 대한 액세스 권한을 부여하는 방법을 결정합니다. 자세한 내용은 [the section called “클러스터 인증 모드 설정”](#) 섹션을 참조하세요.

이 페이지를 모두 완료하면 다음을 선택합니다.

7. 네트워킹 지정 페이지에서 다음 필드의 값을 선택합니다.

- VPC-클러스터를 생성하려면 [Amazon EKS VPC 요구 사항](#)을 충족하는 기존 VPC를 선택합니다. VPC를 선택하기 전에 [the section called “VPC 및 서브넷 요구 사항”](#)의 요구 사항 및 고려 사항 모두를 숙지하는 것이 좋습니다. 클러스터 생성 후에는 사용하려는 VPC를 변경할 수 없습니다. 목록에 기존 VPC가 없는 경우 먼저 VPC를 생성해야 합니다. 자세한 내용은 [the section called “VPC 생성”](#) 섹션을 참조하세요.
- 서브넷-기본적으로 이전 필드에서 지정한 VPC에서 사용 가능한 모든 서브넷이 사전 선택됩니다. 두 개 이상을 선택해야 합니다.

선택한 서브넷은 [Amazon EKS 서브넷 요구 사항](#)을 충족해야 합니다. 서브넷을 선택하기 전에 [Amazon EKS VPC 및 서브넷 요구 사항 및 고려 사항](#) 모두를 숙지하는 것이 좋습니다.

보안 그룹 - (선택사항) 생성하는 네트워크 인터페이스에 Amazon EKS가 연결하려는 보안 그룹을 하나 이상 지정합니다.

보안 그룹 선택 여부에 관계없이 Amazon EKS는 클러스터와 VPC 간에 통신을 사용 설정할 수 있는 보안 그룹을 생성합니다. Amazon EKS는 이 보안 그룹과 사용자가 선택한 보안 그룹을 생성하는 네트워크 인터페이스에 연결합니다. Amazon EKS에서 생성하는 클러스터 보안 그룹에 대한 자세한 내용을 알아보려면 [the section called “보안 그룹 요구 사항”](#) 부분을 참조하세요. Amazon EKS가 생성하는 클러스터 보안 그룹에서 규칙을 수정할 수 있습니다.

- 클러스터 IP 주소 패밀리 선택 - IPv4와 IPv6 중 하나를 선택할 수 있습니다.

Kubernetes는 기본적으로 포트와 서비스에 IPv4 주소를 할당합니다. IPv6 제품군 사용을 결정하기 전에 [VPC 요구 사항 및 고려 사항](#), [the section called “서브넷 요구 사항 및 고려 사항”](#), [the section called “보안 그룹 요구 사항”](#) 및 [the section called “IPv6”](#) 항목의 모든 고려 사항과 요구 사항을 숙지하고 있는지 확인하세요. IPv6 패밀리를 선택하면 IPv4 패밀리처럼 Kubernetes가 IPv6 서비스 주소를 할당할 주소 범위를 지정할 수 없습니다. Kubernetes는 고유한 로컬 주소 범위(fc00::/7)에서 서비스 주소를 할당합니다.

- (선택 사항) Kubernetes 서비스 IP 주소 범위 구성(Configure Kubernetes Service IP address range)을 선택하고 서비스 **IPv4** 범위(Service range)를 지정합니다.

자체 범위를 지정하면 VPC에 피어링되거나 연결된 Kubernetes 서비스와 기타 네트워크 사이의 충돌을 방지할 수 있습니다. CIDR 표기법으로 범위를 입력합니다. 예를 들면 10.2.0.0/16입니다.

CIDR 블록은 다음 요구 사항을 충족해야 합니다.

- 다음 범위 10.0.0.0/8, 172.16.0.0/12 또는 192.168.0.0/16 중 하나 내에 있어야 합니다..
- 최소 크기는 /24고 최대 크기는 /12입니다.
- Amazon EKS 리소스의 VPC 범위와 겹치지 않습니다.

이 옵션은 IPv4 주소 패밀리를 사용할 때 및 클러스터 생성 시에만 지정할 수 있습니다. 이 옵션을 지정하지 않은 경우 Kubernetes는 10.100.0.0/16 또는 172.20.0.0/16 CIDR 블록 중 하나에서 서비스 IP 주소를 할당합니다.

- 클러스터 엔드포인트 액세스에서 옵션을 선택합니다. 클러스터를 생성한 후에는 이 옵션을 변경할 수 있습니다. 기본값이 아닌 옵션을 선택하기 전에 옵션과 그 의미를 숙지해야 합니다. 자세한 내용은 [the section called “클러스터 엔드포인트 액세스”](#) 섹션을 참조하세요.

이 페이지를 모두 완료하면 다음을 선택합니다.

8. (선택사항) 관찰성 구성 페이지에서 설정할 지표 및 컨트롤 플레인 로깅 옵션을 선택합니다. 기본적으로 각 로그 유형은 해제되어 있습니다.

- Prometheus 지표 옵션에 자세한 내용은 [the section called “1단계: Prometheus 지표 켜기”](#) 섹션을 참조하세요.
- 컨트롤 플레인 로깅 옵션에 대한 자세한 내용은 [the section called “컨트롤 플레인 로그”](#) 섹션을 참조하세요.

이 페이지를 모두 완료하면 다음을 선택합니다.

9. 추가 기능 선택 페이지에서 클러스터에 추가할 추가 기능을 선택합니다. 특정 추가 기능은 미리 선택되어 있습니다. 필요한 만큼 Amazon EKS 애드온 및 AWS 마켓플레이스 애드온을 선택할 수 있습니다. 설치하려는 AWS 마켓플레이스 추가 기능이 목록에 없는 경우 페이지 번호를 클릭하여 추가 페이지 결과를 확인하거나 검색창에 텍스트를 입력하여 사용 가능한 AWS 마켓플레이스 추가 기능을 검색할 수 있습니다. 카테고리, 공급업체 또는 가격 모델별로 필터링한 다음 검색 결과 중에서 추가 기능을 선택할 수도 있습니다. 클러스터를 생성할 때 [the section called “Pod Identity”](#)에 설명된 대로 EKS Pod Identity를 지원하는 추가 기능을 보고, 선택하고, 설치할 수 있습니다.

이 페이지를 모두 완료하면 다음을 선택합니다.

Amazon VPC CNI, CoreDNS, kube-proxy와 같은 일부 추가 기능은 기본적으로 설치됩니다. 기본 추가 기능을 비활성화하면 Kubernetes 애플리케이션 실행 기능에 영향을 미칠 수 있습니다.

10. 선택한 추가 기능 설정 구성 페이지에서 설치할 버전을 선택합니다. 클러스터 생성 후 언제든지 최신 버전으로 업데이트할 수 있습니다.

EKS Pod Identity를 지원하는 추가 기능의 경우 콘솔을 사용하여 추가 기능에 대해 미리 채워진 이름, AWS 관리형 정책, 신뢰 정책에 따라 역할을 자동으로 생성할 수 있습니다. 지원되는 추가 기능에 기존 역할을 재사용하거나 새 역할을 생성할 수 있습니다. 콘솔을 사용하여 EKS Pod Identity를 지원하는 추가 기능에 대한 역할을 생성하는 단계는 [the section called “추가 기능 생성\(AWS 콘솔\)”](#) 섹션을 참조하세요. 추가 기능이 EKS Pod Identity를 지원하지 않는 경우 클러스터가 생성된 후 마법사를 사용하여 서비스 계정에 대한 IAM 역할(IRSA)을 생성하는 지침이 포함된 메시지가 표시됩니다.

클러스터 생성 후 각 추가 기능의 구성을 업데이트할 수 있습니다. 추가 기능 구성에 대한 자세한 내용은 [the section called “추가 기능 업데이트”](#) 섹션을 참조하세요. 이 페이지를 모두 완료하면 다음을 선택합니다.

11. 검토 및 생성 페이지에서 이전 페이지에서 입력하거나 선택한 정보를 검토합니다. 변경해야 하는 경우 편집을 선택합니다. 만족하는 경우 생성을 선택합니다. 클러스터가 프로비저닝되는 동안 상태 필드에는 생성 중이 표시됩니다.

Note

요청의 가용 영역 중 하나에 Amazon EKS 클러스터를 생성하는 데 충분한 용량이 없다는 오류가 표시될 수 있습니다. 이 경우 오류 출력에는 새 클러스터를 지원할 수 있는 가용 영역이 포함됩니다. 사용자 계정의 지원 가용 영역에 있는 2개 이상의 서브넷을 사용하여 클러스터를 다시 생성합니다. 자세한 내용은 [the section called “용량 부족”](#) 섹션을 참조하세요.

클러스터 프로비저닝에는 몇 분 정도 걸립니다.

12. [the section called “3단계: kubeconfig 업데이트”](#) 계속 진행

클러스터 생성-AWS CLI

1. 다음 명령을 사용하여 클러스터를 생성합니다. 명령을 실행하기 전에 다음과 같은 바꾸기를 합니다.

- **region-code**를 클러스터를 생성할 AWS 리전으로 바꿉니다.
- **my-cluster**를 클러스터의 이름으로 바꿉니다. 이름에는 영숫자(대소문자 구분)와 하이픈, 밑줄만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다.
- **1.32**를 [Amazon EKS 지원 버전](#)으로 바꿉니다.
- **111122223333**을 계정 ID로, **myAmazonEKSClusterRole**을 클러스터 IAM 역할 이름으로 바꿉니다.
- subnetIds의 값을 고유한 값으로 바꿉니다. 또한 추가 ID를 추가할 수 있습니다. 2개 이상의 서브넷 ID를 지정해야 합니다.

선택한 서브넷은 [Amazon EKS 서브넷 요구 사항](#)을 충족해야 합니다. 서브넷을 선택하기 전에 [Amazon EKS VPC 및 서브넷 요구 사항 및 고려 사항](#) 모두를 숙지하는 것이 좋습니다.

- 보안 그룹 ID를 지정하지 않으려면 명령에서, securityGroupIds=sg-<ExampleID1>을 제거합니다. 하나 이상의 보안 그룹 ID를 지정하려는 경우 securityGroupIds의 값을 고유한 값으로 바꿉니다. 또한 추가 ID를 추가할 수 있습니다.

보안 그룹 선택 여부에 관계없이 Amazon EKS는 클러스터와 VPC 간에 통신을 사용 설정할 수 있는 보안 그룹을 생성합니다. Amazon EKS는 이 보안 그룹과 사용자가 선택한 보안 그룹을 생성하는 네트워크 인터페이스에 연결합니다. Amazon EKS에서 생성하는 클러스터 보안 그룹에 대한 자세한 내용을 알아보려면 [the section called “보안 그룹 요구 사항”](#) 부분을 참조하세요. Amazon EKS가 생성하는 클러스터 보안 그룹에서 규칙을 수정할 수 있습니다.

```
aws eks create-cluster --region region-code --name my-cluster --kubernetes-version
1.32 \
  --role-arn arn:aws:iam::111122223333:role/myAmazonEKSClusterRole \
  --resources-vpc-config subnetIds=subnet-ExampleID1,subnet-
ExampleID2,securityGroupIds=sg-ExampleID1
```

Note

요청의 가용 영역 중 하나에 Amazon EKS 클러스터를 생성하는 데 충분한 용량이 없다는 오류가 표시될 수 있습니다. 이 경우 오류 출력에는 새 클러스터를 지원할 수 있는 가용 영역이 포함됩니다. 사용자 계정의 지원 가용 영역에 있는 2개 이상의 서브넷을 사용하여 클러스터를 다시 생성합니다. 자세한 내용은 [the section called “용량 부족”](#) 섹션을 참조하세요.

다음은 필요한 경우 이전 명령에 추가해야 하는 선택 가능한 설정입니다. 클러스터 생성 후가 아닌 생성할 때만 이러한 옵션을 사용 설정할 수 없습니다.

- 기본적으로 EKS는 클러스터 생성 중에 여러 네트워킹 추가 기능을 설치합니다. 여기에는 Amazon VPC CNI, CoreDNS, kube-proxy가 포함되어 있습니다.

이러한 기본 네트워킹 추가 기능의 설치를 비활성화하려면 아래 파라미터를 사용하세요. 이는 Cilium과 같은 대체 CNI에 사용될 수 있습니다. 자세한 내용은 [EKS API 참조](#)를 검토하세요.

```
aws eks create-cluster --bootstrapSelfManagedAddons false
```

- Kubernetes가 서비스 IP 주소를 할당하는 IPv4 Classless Inter-Domain Routing(CIDR) 블록을 지정하려는 경우 다음 명령에 `--kubernetes-network-config serviceIpv4Cidr=<cidr-block>`을 추가하여 해당 블록을 지정해야 합니다.

자체 범위를 지정하면 VPC에 피어링되거나 연결된 Kubernetes 서비스와 기타 네트워크 사이의 충돌을 방지할 수 있습니다. CIDR 표기법으로 범위를 입력합니다. 예를 들면 10.2.0.0/16입니다.

CIDR 블록은 다음 요구 사항을 충족해야 합니다.

- 다음 범위 10.0.0.0/8, 172.16.0.0/12 또는 192.168.0.0/16 중 하나 내에 있어야 합니다..
- 최소 크기는 /24고 최대 크기는 /12입니다.
- Amazon EKS 리소스의 VPC 범위와 겹치지 않습니다.

이 옵션은 IPv4 주소 패밀리를 사용할 때 및 클러스터 생성 시에만 지정할 수 있습니다. 이 옵션을 지정하지 않은 경우 Kubernetes는 10.100.0.0/16 또는 172.20.0.0/16 CIDR 블록 중 하나에서 서비스 IP 주소를 할당합니다.

- 클러스터를 생성하고 클러스터가 IPv4 주소 대신 포드와 서비스에 IPv6 주소를 할당하도록 하려면 다음 명령에 `--kubernetes-network-config ipFamily=ipv6`를 추가합니다.

Kubernetes는 기본적으로 포드와 서비스에 IPv4 주소를 할당합니다. IPv6 제품군 사용을 결정하기 전에 [VPC 요구 사항 및 고려 사항](#), [the section called “서브넷 요구 사항 및 고려 사항”](#), [the section called “보안 그룹 요구 사항”](#) 및 [the section called “IPv6”](#) 항목의 모든 고려 사항과 요구 사항을 숙지하고 있는지 확인하세요. IPv6 패밀리를 선택하면 IPv4 패밀리처럼 Kubernetes가 IPv6 서비스 주소를 할당할 주소 범위를 지정할 수 없습니다. Kubernetes는 고유한 로컬 주소 범위(`fc00::/7`)에서 서비스 주소를 할당합니다.

2. 클러스터를 프로비저닝하는 데 몇 분 정도 걸립니다. 다음 명령을 사용하여 클러스터의 상태를 쿼리할 수 있습니다.

```
aws eks describe-cluster --region region-code --name my-cluster --query
"cluster.status"
```

반환되는 출력이 ACTIVE가 되면 다음 단계로 진행하세요.

3. [the section called “3단계: kubeconfig 업데이트”](#) 계속 진행

3단계: kubeconfig 업데이트

1. `eksctl`을 사용하여 클러스터를 생성한 경우 이 단계를 건너뛸 수 있습니다. 이는 `eksctl`이 사용자 대신 이 단계를 이미 완료했기 때문입니다. `kubectl config` 파일에 새 컨텍스트를 추가하여 `kubectl`이 클러스터와 통신하도록 사용 설정합니다. 파일 생성 및 업데이트 방법에 대한 자세한 내용을 알아보려면 [the section called “kubectl을 사용하여 클러스터에 액세스”](#) 부분을 참조하세요.

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

예제 출력은 다음과 같습니다.

```
Added new context arn:aws:eks:region-code:111122223333:cluster/my-cluster to /home/
username/.kube/config
```

2. 다음 명령을 실행하여 클러스터와의 통신을 확인합니다.

```
kubectl get svc
```

예제 출력은 다음과 같습니다.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.100.0.1	<none>	443/TCP	28h

4단계: 클러스터 설정

1. (권장 사항) 일부 Amazon EKS 추가 기능을 사용하거나 개별 Kubernetes 워크로드가 특정 AWS Identity and Access Management(IAM) 권한을 갖도록 하려면 클러스터에 대한 [IAM OpenID Connect\(OIDC\) 제공자](#)를 만드세요. 클러스터에 대해 IAM OIDC 공급자를 한 번만 생성하면 됩니다. Amazon EKS 추가 기능에 대한 자세한 내용은 [the section called “Amazon EKS 추가 기능”](#) 부분을 참조하세요. 워크로드에 특정 IAM 권한을 할당하는 방법에 대한 자세한 내용은 [the section called “IRSA를 사용한 자격 증명”](#) 부분을 참조하세요.
2. (권장 사항) Amazon EC2 노드를 클러스터에 배포하기 전에 Kubernetes용 Amazon VPC CNI 플러그인에 대한 클러스터를 구성합니다. 기본적으로 플러그 인은 클러스터와 함께 설치되어 있습니다. 클러스터에 Amazon EC2 노드를 추가하면 추가하는 각 Amazon EC2 노드에 플러그 인이 자동으로 배포됩니다. 플러그 인을 사용하려면 다음 IAM 정책 중 하나를 IAM 역할에 연결해야 합니다. 클러스터가 IPv4 제품군을 사용하는 경우 [AmazonEKS_CNI_Policy](#) 관리형 IAM 정책을 사용합니다. 클러스터에서 IPv6 제품군을 사용하는 경우 [사용자가 만든 IAM 정책](#)을 사용합니다.

정책을 연결하는 IAM 역할은 노드 IAM 역할이거나 플러그 인에만 사용되는 전용 역할일 수 있습니다. 이 역할에 정책을 연결하는 것이 좋습니다. 역할 생성에 대한 자세한 내용은 [the section called “IRSA에 대한 구성”](#) 또는 [Amazon EKS 노드 IAM 역할 섹션](#)을 참조하세요.

3. AWS Management Console을 사용하여 클러스터를 배포한 경우 이 단계를 건너뛸 수 있습니다. AWS Management Console는 기본적으로 Kubernetes용 Amazon VPC CNI 플러그인, CoreDNS 및 kube-proxy Amazon EKS 추가 기능을 배포합니다.

eksctl 또는 AWS CLI 중 하나를 사용하여 클러스터를 배포하는 경우 Kubernetes용 Amazon VPC CNI 플러그인, CoreDNS 및 kube-proxy 자체 관리형 추가 기능이 배포됩니다. 클러스터와 함께 배포되는 Kubernetes용 Amazon VPC CNI 플러그인, CoreDNS 및 kube-proxy 자체 관리형 추가 기능을 Amazon EKS 추가 기능으로 마이그레이션할 수 있습니다. 자세한 내용은 [the section called “Amazon EKS 추가 기능”](#) 섹션을 참조하세요.

4. (선택 사항) 아직 수행하지 않은 경우 클러스터의 Prometheus 지표를 활성화할 수 있습니다. 자세한 내용은 Prometheus용 Amazon 관리형 서비스 사용 설명서의 [스크래이퍼 생성](#)을 참조하세요.
5. Amazon EBS 볼륨을 사용하는 클러스터에 워크로드를 배포할 계획인 경우 워크로드를 배포하기 전에 클러스터에 [Amazon EBS CSI](#)를 설치해야 합니다.

다음 단계

- 클러스터를 생성한 [IAM 보안 주체](#)는 클러스터에 대한 액세스 권한이 있는 유일한 보안 주체입니다. 클러스터에 액세스할 수 있도록 [다른 IAM 보안 주체에 권한을 부여](#)합니다.
- 클러스터를 생성한 IAM 보안 주체에 사전 조건에서 참조하는 최소 IAM 권한만 있는 경우 해당 보안 주체에 대한 Amazon EKS 권한을 추가할 수 있습니다. IAM 보안 주체에 Amazon EKS 권한 부여에 대한 자세한 내용을 알아보려면 [the section called “IAM 참조”](#) 부분을 참조하세요.
- 클러스터를 생성한 IAM 위탁자 또는 다른 위탁자가 Amazon EKS 콘솔에서 Kubernetes 리소스를 보도록 하려면 엔터티에 [필수 권한](#)을 부여합니다.
- 노드와 IAM 보안 주체가 VPC 내에서 클러스터에 액세스하도록 하려면 클러스터에 대한 프라이빗 엔드포인트를 활성화합니다. 퍼블릭 엔드포인트는 기본적으로 활성화되어 있습니다. 원하는 경우 프라이빗 엔드포인트를 활성화한 후 퍼블릭 엔드포인트를 비활성화할 수 있습니다. 자세한 내용은 [the section called “클러스터 엔드포인트 액세스”](#) 섹션을 참조하세요.
- [클러스터에서 보안 암호 암호화 활성화](#)
- [클러스터에서 로깅 구성](#)
- [클러스터에 노드 추가](#)

클러스터 인사이트를 사용한 Kubernetes 버전 업그레이드 준비 및 잘못된 구성 문제 해결

Amazon EKS 클러스터 인사이트는 문제를 탐지하고 이를 해결하기 위한 권장 사항을 제공하여 클러스터 관리를 돕습니다. 모든 Amazon EKS 클러스터는 Amazon EKS에서 선별한 인사이트 목록을 대상으로 자동 반복 검사를 거칩니다. 이러한 인사이트 검사는 Amazon EKS에서 모두 관리되며, 결과를 처리하는 방법에 대한 권장 사항을 제공합니다.

Amazon EKS는 구성 인사이트와 업그레이드 인사이트라는 2가지 유형의 인사이트를 제공합니다. 구성 인사이트는 EKS Hybrid Nodes 설정에서 클러스터나 워크로드의 기능을 손상시킬 수 있는 잘못된 구성을 식별합니다. 업그레이드 인사이트는 새 버전의 Kubernetes로 업그레이드하는 기능에 영향을 미칠 수 있는 문제를 식별합니다.

수행된 인사이트 검사 목록과 Amazon EKS에서 식별한 관련 문제를 확인하려면 AWS Management Console, AWS CLI, AWS SDK 및 Amazon EKS ListInsights API 작업에서 모양을 직접적으로 호출합니다. 단계는 [the section called “클러스터 인사이트 보기”](#)를 참조하세요.

빈도 Amazon EKS는 24시간마다 클러스터 인사이트를 새로 고칩니다. 수동으로 클러스터 인사이트를 새로 고칠 수 없습니다. 클러스터 문제를 해결한 경우 클러스터 인사이트가 업데이트되는 데 다소 시간

이 걸립니다. 문제가 해결되었는지 확인하려면 변경 사항이 배포된 시간을 인사이트 검사의 ‘마지막 새로 고침 시간’과 비교하세요.

권한 Amazon EKS는 모든 EKS 클러스터의 클러스터 인사이트에 대한 클러스터 액세스 항목을 자동으로 생성합니다. 이 항목은 EKS에 클러스터에 대한 정보를 볼 수 있는 권한을 부여합니다. EKS는 이 정보를 사용하여 인사이트를 생성합니다. 자세한 내용은 [the section called “AmazonEKSClusterInsightsPolicy”](#) 섹션을 참조하세요.

클러스터 인사이트를 사용한 Kubernetes 버전 업그레이드 준비

Important

Amazon EKS는 특정 클러스터 인사이트 문제가 있을 때 `--force` 플래그를 사용하여 클러스터를 업그레이드해야 하는 기능을 일시적으로 롤백했습니다. 자세한 내용은 GitHub의 [Temporary rollback of enforcing upgrade insights on update cluster version](#)을 참조하세요. 클라이언트 업데이트에 대한 자세한 내용은 [the section called “3단계: 클러스터 제어 영역 업데이트”](#) 섹션을 참조하세요.

업그레이드 인사이트는 클러스터 인사이트 내 특정 유형의 인사이트 검사입니다. 이 검사는 Kubernetes 버전 업그레이드 준비와 관련된 인사이트를 반환합니다. Amazon EKS는 모든 EKS 클러스터에서 업그레이드 인사이트 검사를 실행합니다.

- 클러스터 Kubernetes 버전을 업데이트하기 전에 [Amazon EKS 콘솔](#)에서 관찰성 대시보드의 클러스터 인사이트 탭을 확인합니다.
- 클러스터에서 문제가 식별된 경우 문제를 검토하고 적절하게 수정하세요. 문제에는 Amazon EKS 및 Kubernetes 설명서에 대한 링크가 포함됩니다.
- 문제를 해결한 후에는 클러스터 인사이트가 새로 고쳐질 때까지 기다리세요. 모든 문제가 해결되었으면 [클러스터를 업데이트](#)하세요.

Amazon EKS는 Kubernetes 버전 업그레이드 준비와 관련된 인사이트를 반환합니다. 업그레이드 인사이트는 Kubernetes 클러스터 업그레이드에 영향을 미칠 수 있는 잠재적 문제를 식별합니다. 이를 통해 관리자가 업그레이드를 준비하는 데 드는 노력을 최소화하고 최신 Kubernetes 버전에서 애플리케이션의 신뢰성을 높일 수 있습니다. Amazon EKS는 문제에 영향을 줄 수 있는 가능한 Kubernetes 버전 업그레이드 목록을 대상으로 클러스터를 자동으로 스캔합니다. Amazon EKS는 각 Kubernetes 버전 릴리스의 변경 사항 검토를 기반으로 인사이트 검사 목록을 자주 업데이트합니다.

Amazon EKS 업그레이드 인사이트는 새 버전에 대한 테스트 및 확인 프로세스를 가속화합니다. 또한 클러스터 관리자 및 애플리케이션 개발자는 우려 사항을 강조하고 문제 해결 조언을 제공하여 최신 Kubernetes 기능을 활용할 수 있습니다.

클러스터 인사이트로 잘못된 구성 문제 해결

EKS 클러스터 인사이트는 하이브리드 노드가 있는 Amazon EKS 클러스터를 자동으로 스캔하여 Kubernetes 컨트롤 플레인과 웹훅 간 통신, exec 및 로그와 같은 kubectl 명령 등을 손상시키는 구성 문제를 식별합니다. 구성 인사이트는 문제를 표면화하고 문제 해결 권장 사항을 제공하여 완벽하게 작동하는 하이브리드 노드 설정에 걸리는 시간을 단축합니다.

클러스터 인사이트 보기

Amazon EKS는 구성 인사이트와 업그레이드 인사이트라는 2가지 유형의 인사이트를 제공합니다. 구성 인사이트는 EKS Hybrid Nodes 설정에서 클러스터나 워크로드의 기능을 손상시킬 수 있는 잘못된 구성을 식별합니다. 업그레이드 인사이트는 새 버전의 Kubernetes로 업그레이드하는 기능에 영향을 미칠 수 있는 문제를 식별합니다.

수행된 인사이트 검사 목록과 Amazon EKS에서 식별한 관련 문제를 확인하려면 AWS Management Console, AWS CLI, AWS SDK 및 Amazon EKS ListInsights API 작업에서 모양을 직접적으로 호출합니다.

구성 인사이트 보기(콘솔)

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 클러스터 목록에서 인사이트를 확인하려는 Amazon EKS 클러스터 이름을 선택합니다.
3. 클러스터 모니터링을 선택합니다.
4. 클러스터 상태 탭을 선택합니다.
5. 구성 인사이트 테이블에는 다음 열이 표시됩니다.
 - 이름 - 클러스터를 대상으로 Amazon EKS에서 수행한 검사입니다.
 - 인사이트 상태 - 상태가 Error인 인사이트는 클러스터 기능에 영향을 미칠 수 있는 잘못된 구성이 있음을 의미합니다. 상태가 Warning인 인사이트는 구성이 문서화된 접근 방식과 일치하지 않지만 의도적으로 구성한 경우 클러스터 기능이 작동할 수 있음을 의미합니다. Passing 상태의 인사이트는 Amazon EKS가 클러스터에서 이 인사이트 검사와 관련된 문제를 발견하지 못했음을 의미합니다.
 - 설명 - 알림 및 문제 해결을 위한 권장 작업을 포함하는 인사이트 검사 정보입니다.

- 마지막 새로 고침 시간-이 클러스터에서 인사이트 상태를 마지막으로 새로 고침 시간입니다.

업그레이드 인사이트 보기(콘솔)

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 클러스터 목록에서 인사이트를 확인하려는 Amazon EKS 클러스터 이름을 선택합니다.
3. 대시보드 보기를 선택합니다.
4. 클러스터 인사이트 탭을 선택합니다.
5. 업그레이드 인사이트 테이블에는 다음 열이 표시됩니다.
 - 이름 - 클러스터를 대상으로 Amazon EKS에서 수행한 검사입니다.
 - 인사이트 상태 - '오류' 상태의 인사이트는 일반적으로 영향을 받는 Kubernetes 버전이 현재 클러스터 버전의 N+1임을 의미하고, '경고' 상태는 인사이트가 향후 Kubernetes 버전 N+2 이상에 적용됨을 의미합니다. '통과' 상태의 인사이트는 Amazon EKS가 클러스터에서 이 인사이트 검사와 관련된 문제를 발견하지 못했음을 의미합니다. '알 수 없음' 상태의 인사이트는 Amazon EKS가 클러스터가 이 인사이트 검사의 영향을 받는지 여부를 판단할 수 없음을 의미합니다.
 - 버전 - 인사이트가 잠재적 문제가 있는지 검사한 Kubernetes 버전입니다.
 - 마지막 새로 고침 시간-이 클러스터에서 인사이트 상태를 마지막으로 새로 고침 시간입니다.
 - 마지막 전환 시간-이 인사이트의 상태가 마지막으로 전환된 시간입니다.
 - 설명 - 알림 및 문제 해결을 위한 권장 작업을 포함하는 인사이트 검사 정보입니다.

클러스터 인사이트 보기(AWS CLI)

1. 인사이트를 확인하려는 클러스터를 결정합니다. 다음 명령은 지정된 클러스터에 대한 인사이트 정보를 나열합니다. 필요에 따라 명령을 다음과 같이 수정한 다음에 수정한 명령을 실행합니다.
 - AWS 리전을 *region-code*로 바꿉니다.
 - *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks list-insights --region region-code --cluster-name my-cluster
```

예제 출력은 다음과 같습니다.

```
{
  "insights":
```

```
[
  {
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "name": "Deprecated APIs removed in Kubernetes vX.XX",
    "category": "UPGRADE_READINESS",
    "kubernetesVersion": "X.XX",
    "lastRefreshTime": 1734557315.000,
    "lastTransitionTime": 1734557309.000,
    "description": "Checks for usage of deprecated APIs that are scheduled
for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
updated APIs supported by vX.XX could cause application impact.",
    "insightStatus":
    {
      "status": "PASSING",
      "reason": "No deprecated API usage detected within the last 30
days.",
    },
  },
  {
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
    "name": "Kubelet version skew",
    "category": "UPGRADE_READINESS",
    "kubernetesVersion": "X.XX",
    "lastRefreshTime": 1734557309.000,
    "lastTransitionTime": 1734557309.000,
    "description": "Checks for kubelet versions of worker nodes in the
cluster to see if upgrade would cause non compliance with supported Kubernetes
kubelet version skew policy.",
    "insightStatus":
    {
      "status": "UNKNOWN",
      "reason": "Unable to determine status of node kubelet versions.",
    },
  },
  {
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "name": "Deprecated APIs removed in Kubernetes vX.XX",
    "category": "UPGRADE_READINESS",
    "kubernetesVersion": "X.XX",
    "lastRefreshTime": 1734557315.000,
    "lastTransitionTime": 1734557309.000,
    "description": "Checks for usage of deprecated APIs that are scheduled
for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
updated APIs supported by vX.XX could cause application impact.",
```

```

        "insightStatus":
        {
            "status": "PASSING",
            "reason": "No deprecated API usage detected within the last 30
days.",
        },
    },
    {
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaaa",
        "name": "Cluster health issues",
        "category": "UPGRADE_READINESS",
        "kubernetesVersion": "X.XX",
        "lastRefreshTime": 1734557314.000,
        "lastTransitionTime": 1734557309.000,
        "description": "Checks for any cluster health issues that prevent
successful upgrade to the next Kubernetes version on EKS.",
        "insightStatus":
        {
            "status": "PASSING",
            "reason": "No cluster health issues detected.",
        },
    },
    {
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEbbbbb",
        "name": "EKS add-on version compatibility",
        "category": "UPGRADE_READINESS",
        "kubernetesVersion": "X.XX",
        "lastRefreshTime": 1734557314.000,
        "lastTransitionTime": 1734557309.000,
        "description": "Checks version of installed EKS add-ons to ensure they
are compatible with the next version of Kubernetes. ",
        "insightStatus": { "status": "PASSING", "reason": "All installed EKS
add-on versions are compatible with next Kubernetes version." },
    },
    {
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEccccc",
        "name": "kube-proxy version skew",
        "category": "UPGRADE_READINESS",
        "kubernetesVersion": "X.XX",
        "lastRefreshTime": 1734557314.000,
        "lastTransitionTime": 1734557309.000,
        "description": "Checks version of kube-proxy in cluster to see if
upgrade would cause non compliance with supported Kubernetes kube-proxy version
skew policy.",
    },

```

```

        "insightStatus":
        {
            "status": "PASSING",
            "reason": "kube-proxy versions match the cluster control plane
version.",
        },
    },
    {
        "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEddddd",
        "name": "Deprecated APIs removed in Kubernetes vX.XX",
        "category": "UPGRADE_READINESS",
        "kubernetesVersion": "X.XX",
        "lastRefreshTime": 1734557315.000,
        "lastTransitionTime": 1734557309.000,
        "description": "Checks for usage of deprecated APIs that are scheduled
for removal in Kubernetes vX.XX. Upgrading your cluster before migrating to the
updated APIs supported by vX.XX could cause application impact.",
        "insightStatus":
        {
            "status": "PASSING",
            "reason": "No deprecated API usage detected within the last 30
days.",
        },
    },
],
"nextToken": null,
}

```

2. 인사이트에 대한 설명 정보를 보려면 다음 명령을 실행합니다. 필요에 따라 명령을 다음과 같이 수정한 다음에 수정한 명령을 실행합니다.

- AWS 리전을 *region-code*로 바꿉니다.
- *a1b2c3d4-5678-90ab-cdef-EXAMPLE22222*를 클러스터 인사이트 목록에서 검색된 인사이트 ID로 바꿉니다.
- *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks describe-insight --region region-code --id a1b2c3d4-5678-90ab-cdef-EXAMPLE22222 --cluster-name my-cluster
```

예제 출력은 다음과 같습니다.

```
{
```

```

"insight":
{
  "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "name": "Kubelet version skew",
  "category": "UPGRADE_READINESS",
  "kubernetesVersion": "1.27",
  "lastRefreshTime": 1734557309.000,
  "lastTransitionTime": 1734557309.000,
  "description": "Checks for kubelet versions of worker nodes in the cluster
to see if upgrade would cause non compliance with supported Kubernetes kubelet
version skew policy.",
  "insightStatus":
  {
    "status": "UNKNOWN",
    "reason": "Unable to determine status of node kubelet versions.",
  },
  "recommendation": "Upgrade your worker nodes to match the Kubernetes version
of your cluster control plane.",
  "additionalInfo":
  {
    "Kubelet version skew policy": "https://kubernetes.io/releases/version-
skew-policy/#kubelet",
    "Updating a managed node group": "https://docs.aws.amazon.com/eks/latest/
userguide/update-managed-node-group.html",
  },
  "resources": [],
  "categorySpecificSummary":
  { "deprecationDetails": [], "addonCompatibilityDetails": [] },
},
}

```

기존 클러스터를 새 Kubernetes 버전으로 업데이트

Amazon EKS에서 새 Kubernetes 버전을 사용할 수 있는 경우 Amazon EKS 클러스터를 최신 버전으로 업데이트할 수 있습니다.

⚠ Important

클러스터를 업그레이드한 후에는 이전 버전으로 다운그레이드할 수 없습니다. 새 Kubernetes 버전으로 업데이트하기 전에 [the section called “Kubernetes 버전”](#)의 정보와 이 주제의 업데이트 단계를 검토하는 것이 좋습니다.

새로운 Kubernetes 버전에는 때로는 큰 변화가 도입되기도 합니다. 따라서 프로덕션 클러스터를 업데이트하기 전에 새 Kubernetes 버전에 대해 애플리케이션의 동작을 테스트하는 것이 좋습니다. 새 Kubernetes 버전으로 이동하기 전에 애플리케이션 동작을 테스트하기 위한 지속적인 통합 워크플로를 구축하여 이 작업을 수행할 수 있습니다.

업데이트 프로세스는 Amazon EKS에서 업데이트된 Kubernetes 버전으로 새 API 서버 노드를 시작해 기존 노드를 대체합니다. Amazon EKS는 이러한 새 노드에서 네트워크 트래픽의 표준 인프라 및 준비 상태 검사를 수행하여 예상대로 작동하고 있는지 확인합니다. 하지만 클러스터 업그레이드를 시작한 후에는 일시 중지하거나 중지할 수 없습니다. 이러한 검사 중 하나라도 실패하면 Amazon EKS는 인프라 배포를 되돌리고, 클러스터는 이전 Kubernetes 버전으로 남아 있게 됩니다. 실행 중인 애플리케이션은 영향을 받지 않으며, 클러스터가 불확정적 또는 복구 불가능 상태로 남아 있는 일은 없습니다. Amazon EKS는 모든 관리형 클러스터를 정기적으로 백업하고, 필요할 경우 클러스터를 복구하는 메커니즘이 있습니다. Kubernetes 인프라 관리 프로세스를 지속적으로 평가 및 개선하고 있습니다.

클러스터를 업데이트하려면 Amazon EKS에서 클러스터를 생성할 때 지정된 서브넷의 사용 가능한 IP 주소 최대 5개가 필요합니다. Amazon EKS에서는 지정한 서브넷에 새 클러스터 탄력적 네트워크 인터페이스(네트워크 인터페이스)를 생성합니다. 네트워크 인터페이스는 기존 네트워크 인터페이스가 있는 다른 서브넷에 생성될 수 있으므로 보안 그룹 규칙에서 클러스터를 생성할 때 지정한 모든 서브넷에 대해 [필수 클러스터 통신](#)을 허용해야 합니다. 클러스터를 생성할 때 지정한 서브넷이 없거나, 해당 서브넷에 사용 가능한 IP 주소가 충분하지 않거나 필요한 클러스터 통신을 허용하는 보안 그룹 규칙이 없는 경우 업데이트가 실패할 수 있습니다.

Amazon EKS는 클러스터의 API 서버 엔드포인트에 항상 액세스할 수 있도록 가용성 높은 Kubernetes 컨트롤 플레인을 제공하고 업데이트 작업 중에 API 서버 인스턴스의 롤링 업데이트를 수행합니다. Kubernetes API 서버 엔드포인트를 지원하는 API 서버 인스턴스의 IP 주소 변경을 고려하려면 API 서버 클라이언트가 재연결을 효과적으로 관리하도록 해야 합니다. kubectl의 최신 버전과 공식적으로 지원되는 Kubernetes 클라이언트 [라이브러리](#), 재연결 프로세스는 이 재연결 프로세스를 투명하게 수행합니다.

Note

클러스터 업데이트에 대한 자세한 내용은 EKS 모범 사례 가이드의 [Best Practices for Cluster Upgrades](#)를 참조하세요. 이 리소스는 업그레이드를 계획하고 클러스터 업그레이드 전략을 이해하는 데 도움이 됩니다.

Amazon EKS Auto Mode 고려 사항

- Amazon EKS Auto Mode의 컴퓨팅 기능은 노드의 Kubernetes 버전을 제어합니다. 컨트롤 플레인을 업그레이드하면 EKS Auto Mode가 관리형 노드를 점진적으로 업데이트하기 시작합니다. EKS Auto Mode는 포드 중단 예산을 준수합니다.
- 컴퓨팅 오토 스케일링, 블록 스토리지, 로드 밸런싱 기능을 포함하여 Amazon EKS Auto Mode의 기능을 수동으로 업그레이드할 필요가 없습니다.

요약

Amazon EKS 클러스터 업그레이드 프로세스의 대략적인 요약은 다음과 같습니다.

1. 클러스터가 업그레이드를 지원하는 상태인지 확인합니다. 여기에는 클러스터에 배포된 리소스에서 사용하는 Kubernetes API를 검사하여 클러스터에 상태 문제가 없는지 확인하는 작업이 포함됩니다. 클러스터의 업그레이드 준비 상태를 평가할 때 Amazon EKS 업그레이드 인사이트를 사용해야 합니다.
2. 다음 마이너 버전으로 컨트롤 플레인을 업그레이드합니다(예: 1.31에서 1.32로).
3. 컨트롤 플레인의 노드와 일치하도록 데이터 플레인의 노드를 업그레이드합니다.
4. 클러스터에서 실행되는 추가 애플리케이션(예: `cluster-autoscaler`)을 업그레이드합니다.
5. 기본적으로 포함된 추가 기능과 같이 Amazon EKS에서 제공하는 추가 기능을 업그레이드합니다.
 - [Amazon VPC CNI 권장 버전](#)
 - [CoreDNS 권장 버전](#)
 - [kube-proxy 권장 버전](#)
6. 클러스터와 통신하는 클라이언트를 업그레이드합니다(예: `kubectl`).

1단계: 업그레이드 준비

클러스터 제어 영역의 Kubernetes 버전과 사용자 노드의 Kubernetes 버전을 비교합니다.

- 클러스터 컨트롤 플레인의 Kubernetes 버전을 가져옵니다.

```
kubectl version
```

- 노드의 Kubernetes 버전을 가져옵니다. 이 명령은 자체 관리형 및 관리형 Amazon EC2, Fargate 및 하이브리드 노드를 모두 반환합니다. 각 Fargate 포드는 자체 노드로 나열됩니다.

```
kubectl get nodes
```

컨트롤 플레인을 새 Kubernetes 버전으로 업데이트하기 전에 클러스터에 있는 관리형 노드 및 Fargate 노드 모두의 Kubernetes 마이너 버전이 컨트롤 플레인의 현재 버전과 동일한지 확인합니다. 예를 들어, 컨트롤 플레인에서 버전 1.29를 실행 중이고 노드 중 하나에서 버전 1.28을 실행 중인 경우에는 컨트롤 플레인을 1.30으로 업데이트하기 전에 노드를 버전 1.29로 업데이트해야 합니다. 또한 컨트롤 플레인을 업데이트하기 전에 자체 관리형 노드와 하이브리드 노드를 컨트롤 플레인과 동일한 버전으로 업데이트하는 것이 좋습니다. 자세한 내용은 [the section called “업데이트”](#), [the section called “업데이트 방법”](#), [the section called “하이브리드 노드 업그레이드”](#) 섹션을 참조하세요. 컨트롤 플레인 버전보다 낮은 마이너 버전이 적용된 Fargate 노드가 있는 경우 먼저 이 노드로 대표되는 포드를 삭제합니다. 그런 다음 컨트롤 플레인을 업데이트합니다. 나머지 포드는 모두 다시 배포한 후 새 버전으로 업데이트됩니다.

2단계: 업그레이드 고려 사항 검토

Amazon EKS 클러스터 인사이트는 더 이상 사용되지 않는 Kubernetes API 사용 등의 문제에 영향을 미치는 가능한 Kubernetes 버전 업그레이드 목록을 대상으로 클러스터를 자동으로 스캔합니다. Amazon EKS는 Kubernetes 프로젝트의 변경 사항 평가를 기반으로 수행할 인사이트 검사 목록을 주기적으로 업데이트합니다. 또한 Amazon EKS 서비스에 새 버전과 함께 변경 사항이 도입되면 인사이트 검사 목록도 업데이트됩니다. 자세한 내용은 [the section called “클러스터 인사이트”](#) 섹션을 참조하세요.

Kubernetes Documentation의 [Deprecated API Migration Guide](#)를 검토합니다.

클러스터 인사이트 검토

Amazon EKS 클러스터 인사이트를 사용하여 문제 식별

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 클러스터 목록에서 인사이트를 확인하려는 Amazon EKS 클러스터 이름을 선택합니다.
3. 대시보드 보기를 선택합니다.

4. 클러스터 인사이트 탭을 선택합니다.
5. 업그레이드 인사이트 테이블에는 다음 열이 표시됩니다.
 - 이름 - 클러스터를 대상으로 Amazon EKS에서 수행한 검사입니다.
 - 인사이트 상태 - '오류' 상태의 인사이트는 일반적으로 영향을 받는 Kubernetes 버전이 현재 클러스터 버전의 N+1임을 의미하고, '경고' 상태는 인사이트가 향후 Kubernetes 버전 N+2 이상에 적용됨을 의미합니다. '통과' 상태의 인사이트는 Amazon EKS가 클러스터에서 이 인사이트 검사와 관련된 문제를 발견하지 못했음을 의미합니다. '알 수 없음' 상태의 인사이트는 Amazon EKS가 클러스터가 이 인사이트 검사의 영향을 받는지 여부를 판단할 수 없음을 의미합니다.
 - 버전 - 인사이트가 잠재적 문제가 있는지 검사한 Kubernetes 버전입니다.
 - 마지막 새로 고침 시간-이 클러스터에서 인사이트 상태를 마지막으로 새로 고침 시간입니다.
 - 마지막 전환 시간-이 인사이트의 상태가 마지막으로 전환된 시간입니다.
 - 설명 - 알림 및 문제 해결을 위한 권장 작업을 포함하는 인사이트 검사 정보입니다.

세부 고려 사항

- Amazon EKS는 가용성 높은 제어 영역을 실행하므로 한 번에 하나의 마이너 버전만 업데이트할 수 있습니다. 이 요구 사항에 대한 자세한 내용을 알아보려면 [Kubernetes 버전 및 버전 차이 지원 정책](#)을 참조하세요. 현재 클러스터 버전이 1.28 버전이고 1.30 버전으로 업데이트하려는 경우를 예로 들어보겠습니다. 먼저 1.28 버전 클러스터를 1.29 버전으로 업데이트한 다음에 1.29 버전 클러스터를 1.30 버전으로 업데이트해야 합니다.
- 노드의 Kubernetes kube-apiserver와 kubelet 사이에서 버전 차이를 검토합니다.
 - Kubernetes 버전 1.28부터 kubelet은 kube-apiserver보다 오래된 최대 3개의 마이너 버전에 해당할 수 있습니다. [Kubernetes upstream version skew policy](#)를 참조하세요.
 - Fargate 노드와 관리형 노드의 kubelet이 Kubernetes 버전 1.25 이상인 경우 kubelet 버전을 업데이트하지 않고도 클러스터를 최대 3개 버전까지 미리 업데이트할 수 있습니다. 예를 들어, kubelet 버전이 1.25인 경우 kubelet 버전은 1.25로 유지하면서 Amazon EKS 클러스터 버전을 1.25에서 1.26, 1.27, 1.28로 업데이트할 수 있습니다.
- 업데이트를 시작하기 전에 모범 사례는 노드의 kubelet이 컨트롤 플레인과 동일한 Kubernetes 버전인지 확인하는 것입니다.
- 클러스터가 1.8.0 이전 버전의 Kubernetes용 Amazon VPC CNI 플러그인으로 구성된 경우에는 클러스터를 업데이트하기 전에 플러그인을 최신 버전으로 업데이트하는 것이 좋습니다. 플러그인을 업데이트하려면 [the section called “Amazon VPC CNI”](#)을 참조하세요.

3단계: 클러스터 제어 영역 업데이트

Important

Amazon EKS는 특정 클러스터 인사이트 문제가 있을 때 `--force` 플래그를 사용하여 클러스터를 업그레이드해야 하는 기능을 일시적으로 롤백했습니다. 자세한 내용은 GitHub의 [Temporary rollback of enforcing upgrade insights on update cluster version](#)을 참조하세요. Amazon EKS는 '마지막 새로 고침 시간'으로부터 24시간 후에 클러스터 인사이트를 새로 고칩니다. 문제를 해결한 시간을 클러스터 인사이트의 '마지막 새로 고침 시간'과 비교할 수 있습니다. 또한 더 이상 사용되지 않는 API 사용을 처리한 후 인사이트 상태가 업데이트되는 데 최대 30일이 걸릴 수 있습니다. 업그레이드 인사이트는 항상 30일 동안 더 이상 사용되지 않는 API 사용을 찾습니다.

다음을 사용하여 EKS 컨트롤 플레인 버전 업그레이드 요청을 제출할 수 있습니다.

- [eksctl](#)
- [AWS 콘솔](#)
- [AWS CLI](#)

클러스터 업데이트-eksctl

이 절차에는 eksctl 버전 0.210.0 이상이 필요합니다. 버전은 다음 명령을 통해 확인할 수 있습니다.

```
eksctl version
```

eksctl 설치 또는 업데이트에 대한 지침은 eksctl 설명서에서 [설치](#)를 참조하세요.

Amazon EKS 컨트롤 플레인의 Kubernetes 버전을 업데이트합니다. <cluster-name>을 클러스터 이름으로 교체합니다. <version-number>를 클러스터를 업데이트하려는 Amazon EKS에서 지원하는 버전 번호로 바꿉니다. 지원되는 버전 번호 목록은 [the section called "Kubernetes 버전"](#) 섹션을 참조하세요.

```
eksctl upgrade cluster --name <cluster-name> --version <version-number> --approve
```

업데이트를 완료하는 데 몇 분 정도 걸립니다.

계속해서 [the section called “4단계: 클러스터 구성 요소 업데이트”](#)로 이동하세요.

클러스터 업데이트 - AWS 콘솔

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 업그레이드하려는 클러스터에 대해 지금 업그레이드를 선택합니다.
3. 클러스터를 업데이트할 버전을 선택하고 업그레이드를 선택합니다.
4. 업데이트를 완료하는 데 몇 분 정도 걸립니다. 계속해서 [the section called “4단계: 클러스터 구성 요소 업데이트”](#)로 이동하세요.

클러스터 업데이트 - AWS CLI

1. AWS CLI가 설치되어 있고 로그인했는지 확인합니다. 자세한 내용은 [최신 버전의 AWS CLI 설치 또는 업데이트](#)를 참조하세요.
2. 다음 AWS CLI명령을 사용하여 Amazon EKS 클러스터를 업데이트합니다. 업그레이드하려는 클러스터의 <cluster-name>와 <region-code>을 바꿉니다. <version-number>를 클러스터를 업데이트하려는 Amazon EKS에서 지원하는 버전 번호로 바꿉니다. 지원되는 버전 번호 목록은 [the section called “Kubernetes 버전”](#) 섹션을 참조하세요.

```
aws eks update-cluster-version --name <cluster-name> \
  --kubernetes-version <version-number> --region <region-code>
```

예제 출력은 다음과 같습니다.

```
{
  "update": {
    "id": "<update-id>",
    "status": "InProgress",
    "type": "VersionUpdate",
    "params": [
      {
        "type": "Version",
        "value": "<version-number>"
      },
      {
        "type": "PlatformVersion",
        "value": "eks.1"
      }
    ]
  }
}
```

```
[...]
  "errors": []
}
```

- 업데이트를 완료하는 데 몇 분 정도 걸립니다. 다음 명령을 사용하여 클러스터의 업데이트 상태를 모니터링할 수 있습니다. 동일한 <cluster-name>과 <region-code>를 사용하는 것 외에도 이전 명령에서 반환한 <update-id>를 사용합니다.

```
aws eks describe-update --name <cluster-name> \
  --region <region-code> --update-id <update-id>
```

Successful 상태가 표시되면 업데이트가 완료됩니다.

- 계속해서 [the section called “4단계: 클러스터 구성 요소 업데이트”](#)로 이동하세요.

4단계: 클러스터 구성 요소 업데이트

- 클러스터 업데이트가 완료된 후 업데이트한 클러스터와 동일한 Kubernetes 마이너 버전으로 노드를 업데이트합니다. 자세한 내용은 [the section called “업데이트 방법”](#), [the section called “업데이트”](#), [the section called “하이브리드 노드 업그레이드”](#) 섹션을 참조하세요. Fargate에서 시작된 새 포드에는 클러스터 버전과 일치하는 kubelet 버전이 있습니다. 기존 Fargate 포드는 변경되지 않습니다.
- (선택 사항) 클러스터를 업데이트하기 전에 Kubernetes Cluster Autoscaler를 클러스터에 배포한 경우 업그레이드한 Kubernetes 메이저 및 마이너 버전과 일치하는 최신 버전으로 Cluster Autoscaler를 업데이트합니다.
 - 웹 브라우저에서 Cluster Autoscaler [릴리스](#) 페이지를 열고 클러스터의 Kubernetes 메이저 및 마이너 버전과 일치하는 최신 Cluster Autoscaler 버전을 검색합니다. 예를 들어 클러스터의 Kubernetes 버전이 1.30이라면 1.30으로 시작하는 Cluster Autoscaler 릴리스를 검색합니다. 다음 단계에서 사용할 해당 릴리스의 의미 체계 버전 번호(예: 1.30.n)를 기록합니다.
 - 다음 명령을 사용하여 Cluster Autoscaler 이미지 태그를 이전 단계에서 적어 둔 버전으로 설정합니다. 필요한 경우 X.XX.X을 고유 값으로 바꿉니다.

```
kubectl -n kube-system set image deployment.apps/cluster-autoscaler cluster-autoscaler=registry.k8s.io/autoscaling/cluster-autoscaler:vX.XX.X
```

- (GPU 노드가 있는 클러스터만 해당) 클러스터에 GPU 지원이 포함된 노드 그룹이 있는 경우(예: p3.2xlarge) 클러스터에서 [Kubernetes용 NVIDIA 디바이스 플러그인](#) DaemonSet을 업데이트해

야 합니다. 다음 명령을 실행하기 전에 <vX.X.X>을(를) 원하는 [NVIDIA/k8s-device-plugin](#) 버전으로 교체합니다.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/<vX.X.X>/deployments/static/nvidia-device-plugin.yml
```

4. Kubernetes용 Amazon VPC CNI 플러그인, CoreDNS 및 kube-proxy 추가 기능을 업데이트합니다. [서비스 계정 토큰](#)에 나열된 최소 버전으로 추가 기능을 업데이트하는 것이 좋습니다.
 - Amazon EKS 추가 기능을 사용하는 경우 Amazon EKS 콘솔에서 클러스터를 선택한 다음 왼쪽 검색 창에서 업데이트한 클러스터의 이름을 선택합니다. 콘솔에 알림이 표시됩니다. 사용 가능한 업데이트가 있는 추가 기능마다 새 버전을 사용할 수 있다는 알림이 표시됩니다. 추가 기능을 업데이트하려면 추가 기능 탭을 선택합니다. 사용 가능한 업데이트가 있는 추가 기능의 상자 중 하나에서 지금 업데이트를 선택하고 사용 가능한 버전을 선택한 다음 업데이트를 선택합니다.
 - AWS CLI 또는 eksctl을 사용하여 추가 기능을 업데이트할 수도 있습니다. 자세한 내용은 [the section called “추가 기능 업데이트”](#) 섹션을 참조하세요.
5. 필요하면 kubectl의 버전을 업데이트합니다. Amazon EKS 클러스터 제어 영역과 마이너 버전이 하나 다른 kubectl 버전을 사용해야 합니다.

Amazon EKS 클러스터에 대한 Kubernetes 버전 다운그레이드

Amazon EKS 클러스터의 Kubernetes는 다운그레이드할 수 없습니다. 대신 이전 Amazon EKS 버전에서 새 클러스터를 생성하고 워크로드를 마이그레이션하세요.

클러스터 삭제

Amazon EKS 클러스터 사용을 완료한 경우 불필요한 비용이 발생하지 않도록 클러스터와 연결된 리소스를 삭제해야 합니다.

eksctl, AWS Management Console 또는 AWS CLI를 사용하여 클러스터를 삭제할 수 있습니다.

고려 사항

- 클러스터의 활성 서비스가 로드 밸런서와 연결된 경우 로드 밸런서가 적절하게 삭제되도록 클러스터 삭제 전에 해당 서비스를 삭제해야 합니다. 그렇지 않으면 VPC에 분리된 리소스가 발생하여 VPC를 삭제할 수 없게 됩니다.
- 클러스터 생성자가 제거되어 오류가 발생하는 경우 문제를 해결하려면 [이 문서](#)를 참조하세요.

- Prometheus용 Amazon 관리형 서비스 리소스는 클러스터 수명 주기를 벗어나며 클러스터에 독립적으로 유지 관리되어야 합니다. 클러스터를 삭제할 때 해당 스크레이퍼도 모두 삭제하여 관련 비용을 줄여야 합니다. 자세한 내용은 Prometheus용 Amazon 관리형 서비스 사용자 가이드의 [스크레이퍼 찾기 및 삭제](#)를 참조하세요.
- 연결된 클러스터를 제거하려면 [the section called “클러스터 등록 취소”](#) 섹션을 참조하세요.

EKS Auto Mode 고려 사항

- EC2 관리형 인스턴스를 포함하여 모든 EKS Auto Mode 노드가 삭제됩니다.
- 모든 로드 밸런서가 삭제됩니다.

자세한 내용은 [the section called “EKS Auto Mode 비활성화”](#) 섹션을 참조하세요.

클러스터 삭제(eksctl)

이 절차에는 eksctl 버전 0.210.0 이상이 필요합니다. 버전은 다음 명령을 통해 확인할 수 있습니다.

```
eksctl version
```

eksctl 설치 또는 업데이트에 대한 지침은 eksctl 설명서의 [설치](#)를 참조하세요.

1. 클러스터에서 실행 중인 모든 서비스를 나열합니다.

```
kubectl get svc --all-namespaces
```

- a. EXTERNAL-IP 값과 연결된 모든 서비스를 삭제합니다. 이러한 서비스는 Elastic Load Balancing 로드 밸런서에 의해 설정되고, Kubernetes에서 이를 삭제하여 로드 밸런서 및 연결된 리소스가 적절하게 릴리스되어야 합니다. 설명과 같이 **service-name**을 나열된 각 서비스의 이름으로 바꿉니다.

```
kubectl delete svc service-name
```

2. 다음 명령으로 클러스터 및 연결된 노드를 삭제하되, **prod**를 클러스터 이름으로 변경합니다.

```
eksctl delete cluster --name prod
```

출력:

```
[#] using region region-code
[#] deleting EKS cluster "prod"
[#] will delete stack "eksctl-prod-nodegroup-standard-nodes"
[#] waiting for stack "eksctl-prod-nodegroup-standard-nodes" to get deleted
[#] will delete stack "eksctl-prod-cluster"
[#] the following EKS cluster resource(s) for "prod" will be deleted: cluster. If in
doubt, check CloudFormation console
```

클러스터 삭제(AWS 콘솔)

1. 클러스터에서 실행 중인 모든 서비스를 나열합니다.

```
kubectl get svc --all-namespaces
```

2. EXTERNAL-IP 값과 연결된 모든 서비스를 삭제합니다. 이러한 서비스는 Elastic Load Balancing 로드 밸런서에 의해 설정되고, Kubernetes에서 이를 삭제하여 로드 밸런서 및 연결된 리소스가 적절하게 릴리스되어야 합니다. 설명과 같이 *service-name*을 나열된 각 서비스의 이름으로 바꿉니다.

```
kubectl delete svc service-name
```

3. 모든 노드 그룹 및 Fargate 프로필을 삭제합니다.

- a. [Amazon EKS 콘솔](#)을 엽니다.
- b. 왼쪽 탐색 창에서 Amazon EKS 클러스터를 선택한 다음 탭으로 구분된 클러스터 목록에서 삭제할 클러스터의 이름을 선택합니다.
- c. 컴퓨팅탭을 선택하고 삭제할 노드 그룹을 선택합니다. 삭제를 선택하고 노드 그룹의 이름을 입력한 다음 삭제를 선택합니다. 클러스터의 모든 노드 그룹을 삭제합니다.

Note

[관리형 노드 그룹](#)만 나열됩니다.

- d. 삭제할 Fargate 프로필에 이어 삭제를 선택하고 프로필의 이름을 입력한 다음 삭제를 선택합니다. 클러스터의 모든 Fargate 프로필을 삭제합니다.
4. 모든 자체 관리형 노드 AWS CloudFormation 스택을 삭제합니다.
 - a. [AWS CloudFormation 콘솔](#)을 엽니다.
 - b. 삭제할 노드 스택을 선택하고 삭제를 선택합니다.

- c. 스택 삭제 확인 대화 상자에서 스택 삭제를 선택합니다. 클러스터의 모든 자체 관리형 노드 스택을 삭제합니다.
5. 클러스터를 삭제합니다.
 - a. [Amazon EKS 콘솔](#)을 엽니다.
 - b. 삭제할 클러스터를 선택하고 삭제를 선택합니다.
 - c. 클러스터 삭제 확인 화면에서 삭제를 선택합니다.
6. (선택사항) VPC AWS CloudFormation 스택을 삭제합니다.
 - a. [AWS CloudFormation 콘솔](#)을 엽니다.
 - b. 삭제할 VPC 스택을 선택하고 삭제를 선택합니다.
 - c. 스택 삭제 확인 대화 상자에서 스택 삭제를 선택합니다.

클러스터를 삭제(AWS CLI)

1. 클러스터에서 실행 중인 모든 서비스를 나열합니다.

```
kubectl get svc --all-namespaces
```

2. EXTERNAL-IP 값과 연결된 모든 서비스를 삭제합니다. 이러한 서비스는 Elastic Load Balancing 로드 밸런서에 의해 설정되고, Kubernetes에서 이를 삭제하여 로드 밸런서 및 연결된 리소스가 적절하게 릴리스되어야 합니다. 설명과 같이 *service-name*을 나열된 각 서비스의 이름으로 바꿉니다.

```
kubectl delete svc service-name
```

3. 모든 노드 그룹 및 Fargate 프로필을 삭제합니다.
 - a. 다음 명령을 사용하여 클러스터의 노드 그룹을 나열합니다.

```
aws eks list-nodegroups --cluster-name my-cluster
```

Note

[관리형 노드 그룹](#)만 나열됩니다.

- b. 다음 명령을 사용하여 각 노드 그룹을 삭제합니다. 클러스터의 모든 노드 그룹을 삭제합니다.

```
aws eks delete-nodegroup --nodegroup-name my-nodegroup --cluster-name my-cluster
```

- c. 다음 명령을 사용하여 클러스터에 Fargate 프로필을 나열합니다.

```
aws eks list-fargate-profiles --cluster-name my-cluster
```

- d. 다음 명령을 사용하여 각 Fargate 프로필을 삭제합니다. 클러스터의 모든 Fargate 프로필을 삭제합니다.

```
aws eks delete-fargate-profile --fargate-profile-name my-fargate-profile --cluster-name my-cluster
```

4. 모든 자체 관리형 노드 AWS CloudFormation 스택을 삭제합니다.

- a. 다음 명령으로 사용 가능한 AWS CloudFormation 스택을 나열합니다. 결과 출력에서 노드 템플릿의 이름을 찾습니다.

```
aws cloudformation list-stacks --query "StackSummaries[].StackName"
```

- b. 다음 명령으로 각 노드 스택을 삭제하되, *node-stack*을 노드 스택 이름으로 변경합니다. 클러스터의 모든 자체 관리형 노드 스택을 삭제합니다.

```
aws cloudformation delete-stack --stack-name node-stack
```

5. 다음 명령으로 클러스터를 삭제하되, *my-cluster*를 사용할 클러스터 이름으로 변경합니다.

```
aws eks delete-cluster --name my-cluster
```

6. (선택사항) VPC AWS CloudFormation 스택을 삭제합니다.

- a. 다음 명령으로 사용 가능한 AWS CloudFormation 스택을 나열합니다. 결과 출력에서 VPC 템플릿의 이름을 찾습니다.

```
aws cloudformation list-stacks --query "StackSummaries[].StackName"
```

- b. 다음 명령으로 VPC 스택을 삭제하되, *my-vpc-stack*을 사용할 VPC 스택의 이름으로 변경합니다.

```
aws cloudformation delete-stack --stack-name my-vpc-stack
```

클러스터 API 서버 엔드포인트

이 주제에서는 Amazon EKS 클러스터의 Kubernetes API 서버 엔드포인트에 대한 프라이빗 액세스를 활성화하거나 인터넷에서 퍼블릭 액세스를 완전히 비활성화하는 방법을 설명합니다.

새 클러스터를 생성할 때 Amazon EKS에서는 클러스터와 통신하는 데 사용하는 관리형 Kubernetes API 서버에 대한 엔드포인트를 생성합니다(kubectl과 같은 Kubernetes 관리 도구 사용). 기본적으로 이 API 서버 엔드포인트는 인터넷에 공개되어 있으며, API 서버에 대한 액세스는 AWS Identity and Access Management(IAM) 및 기본 Kubernetes [역할 기반 액세스 제어\(RBAC\)](#)의 조합을 통해 보호됩니다. 이 엔드포인트를 클러스터 퍼블릭 엔드포인트라고 합니다. 클러스터 프라이빗 엔드포인트도 있습니다. 클러스터 프라이빗 엔드포인트에 대한 자세한 내용은 다음 [the section called “클러스터 프라이빗 엔드포인트”](#) 섹션을 참조하세요.

IPv6 클러스터 엔드포인트 형식

EKS는 2024년 10월 이후에 만들어진 새 IPv6 클러스터에 대해 다음과 같은 형식으로 고유한 듀얼 스택 엔드포인트를 생성합니다. IPv6 클러스터는 클러스터의 IP 패밀리(ipFamily) 설정에서 IPv6를 선택하는 클러스터입니다.

Example

AWS

EKS 클러스터 퍼블릭/프라이빗 엔드포인트: eks-cluster.*region*.api.aws

AWS GovCloud (US)

EKS 클러스터 퍼블릭/프라이빗 엔드포인트: eks-cluster.*region*.api.aws

Amazon Web Services in China

EKS 클러스터 퍼블릭/프라이빗 엔드포인트: eks-cluster.*region*.api.amazonwebservices.com.cn

Note

이중 스택 클러스터 엔드포인트는 2024년 10월에 도입되었습니다. IPv6 클러스터에 대한 자세한 내용은 [the section called “IPv6”](#)을 참조하세요. 2024년 10월 이전에 생성된 클러스터는 대신 다음 엔드포인트 형식을 사용합니다.

IPv4 클러스터 엔드포인트 형식

EKS는 클러스터의 IP 패밀리(ipFamily) 설정에서 IPv4를 선택하는 각 클러스터에 대해 다음 형식으로 고유한 엔드포인트를 생성합니다.

Example

AWS

EKS 클러스터 퍼블릭/프라이빗 엔드포인트 `eks-cluster.region.eks.amazonaws.com`

AWS GovCloud (US)

EKS 클러스터 퍼블릭/프라이빗 엔드포인트 `eks-cluster.region.eks.amazonaws.com`

Amazon Web Services in China

EKS 클러스터 퍼블릭/프라이빗 엔드포인트 `eks-cluster.region.amazonwebservices.com.cn`

Note

2024년 10월 이전에 IPv6 클러스터는 이 엔드포인트 형식도 사용했습니다. 이러한 클러스터의 경우 퍼블릭 엔드포인트와 프라이빗 엔드포인트 모두 이 엔드포인트에서 확인된 IPv4 주소만 있습니다.

클러스터 프라이빗 엔드포인트

노드와 API 서버 간의 모든 통신이 VPC 내에 유지되도록 Kubernetes API 서버에 대한 프라이빗 액세스를 활성화할 수 있습니다. 인터넷에서 API 서버로 액세스하는 IP 주소를 제한하거나 API 서버로의 인터넷 액세스를 완전히 비활성화할 수 있습니다.

Note

이 엔드포인트는 Kubernetes API 서버이며 AWS API와 통신하기 위한 기존 AWS PrivateLink 엔드포인트가 아니므로 Amazon VPC 콘솔에서 엔드포인트로 표시되지 않습니다.

클러스터에 대해 엔드포인트 프라이빗 액세스를 활성화하면 Amazon EKS에서는 사용자 대신 Route 53 프라이빗 호스팅 영역을 생성하고 클러스터의 VPC에 연결합니다. 이 프라이빗 호스팅 영역은

Amazon EKS에서 관리하며, 사용자 계정의 Route 53 리소스에는 표시되지 않습니다. 프라이빗 호스팅 영역이 API 서버로 트래픽을 올바르게 라우팅하려면 VPC에서 `enableDnsHostnames` 및 `enableDnsSupport`가 `true`로 설정되어야 하고, VPC에 설정된 DHCP 옵션이 도메인 이름 서버 목록에 `AmazonProvidedDNS`를 포함해야 합니다. 자세한 내용은 Amazon VPC 사용 설명서의 [VPC에 대한 DNS 지원 업데이트](#)를 참조하세요.

새 클러스터를 생성할 때 API 서버 엔드포인트 액세스 요구 사항을 정의할 수 있으며, 언제든지 클러스터에 대한 API 서버 엔드포인트 액세스를 업데이트할 수 있습니다.

클러스터 엔드포인트 액세스 수정

이 섹션의 절차를 통해 기존 클러스터에 대한 엔드포인트 액세스를 수정합니다. 다음 표에는 지원되는 API 서버 엔드포인트 액세스 조합과 연결된 동작이 나와 있습니다.

엔드포인트 퍼블릭 액세스	엔드포인트 프라이빗 액세스	동작
활성화됨	비활성	• 새 Amazon EKS 클러스터의 기본 동작입니다. • 클러스터의 VPC 내에서 비롯된 Kubernetes API 요청(예: 컨트롤 플레인 통신에 대한 노드)은 VPC에서 벗어나지만, Amazon의 네트워크에서는 벗어나지 않습니다. • 인터넷에서 클러스터 API 서버에 액세스할 수 있습니다. 선택적으로 퍼블릭 엔드포인트에 액세스할 수 있는 CIDR 블록을 제한할 수 있습니다. 특정 CIDR 블록에 대한 액세스를 제한하는 경우 프라이빗 엔드포인트를 활성화하거나 지정된 CIDR 블록에 노드와 Fargate 포드(사용하는 경우)가 퍼블릭 엔드포인트에 액세스하는 주소를 포함합니다
활성화됨	활성화됨	• 클러스터의 VPC 내 Kubernetes API 요청(예: 컨트롤 플레인 통신에 대한 노드)은 프라이빗 VPC 엔드포인트를 사용합니다. • 인터넷에서 클러스터 API 서버에 액세스할 수 있습니다. 선택적으로 퍼블릭 엔드포인트에 액세스할 수 있는 CIDR 블록을 제한할 수 있습니다. • Amazon EKS 클러스터에서 하이브리드 노드를 사용하는 경우 퍼블릭 및 프라이빗 클러스터 엔드포인트 액세스

엔드포인트 퍼블릭 액세스	엔드포인트 프라이빗 액세스	동작
		스를 모두 활성화하는 것은 권장되지 않습니다. 하이브리드 노드는 VPC 외부에서 실행되므로 클러스터 엔드포인트를 퍼블릭 IP 주소로 확인합니다. 하이브리드 노드가 있는 클러스터에는 퍼블릭 또는 프라이빗 클러스터 엔드포인트 액세스를 사용하는 것이 좋습니다.
비활성	활성화됨	- 클러스터 API 서버에 대한 모든 트래픽은 클러스터의 VPC 또는 연결된 네트워크 내에서 비롯되어야 합니다. - 인터넷에서 API 서버로 퍼블릭 액세스할 수 없습니다. 모든 kubectl 명령은 VPC 또는 연결된 네트워크 내에서 가져와야 합니다. 연결 옵션은 the section called “프라이빗 전용 API 서버 액세스” 섹션을 참조하세요. - 클러스터의 API 서버 엔드포인트는 퍼블릭 DNS 서버에 의해 VPC의 프라이빗 IP 주소로 확인됩니다. 이전에는 VPC 내에서만 엔드포인트를 확인할 수 있었습니다. 엔드포인트가 VPC 내부에서 기존 클러스터에 대한 프라이빗 IP 주소로 확인되지 않는 경우 다음을 수행할 수 있습니다. - 퍼블릭 액세스를 활성화한 다음 다시 비활성화합니다. 클러스터에 대해 한 번만 수행하면 엔드포인트가 해당 시점부터 프라이빗 IP 주소로 확인됩니다. - 클러스터를 업데이트합니다.

퍼블릭 엔드포인트의 CIDR 블록(IPv6 클러스터)

퍼블릭 엔드포인트는 듀얼 스택이므로 IPv6 클러스터의 퍼블릭 엔드포인트에 IPv6 및 IPv4 CIDR 블록을 추가할 수 있습니다. 이는 2024년 10월 이후에 생성하고 ipFamily가 IPv6로 설정된 새 클러스터에만 적용됩니다. 새 엔드포인트 도메인 이름인 api.aws로 이러한 클러스터를 식별할 수 있습니다.

퍼블릭 엔드포인트의 CIDR 블록(IPv4 클러스터)

IPv4 클러스터의 퍼블릭 엔드포인트에 IPv4 CIDR 블록을 추가할 수 있습니다. IPv4 클러스터의 퍼블릭 엔드포인트에 IPv6 CIDR 블록을 추가할 수 없습니다. 시도하면 EKS에서 다음 오류 메시지를 반환합니다. The following CIDRs are invalid in publicAccessCidrs

퍼블릭 엔드포인트의 CIDR 블록(2024년 10월 이전에 만들어진 **IPv6** 클러스터)

2024년 10월 이전에 만든 이전 IPv6 클러스터의 퍼블릭 엔드포인트에 IPv4 CIDR 블록을 추가할 수 있습니다. eks.amazonaws.com 엔드포인트로 이러한 클러스터를 식별할 수 있습니다. 2024년 10월 이전에 만든 이러한 이전 IPv6 클러스터의 퍼블릭 엔드포인트에 IPv6 CIDR 블록을 추가할 수 없습니다. 시도하면 EKS에서 다음 오류 메시지를 반환합니다. The following CIDRs are invalid in publicAccessCidrs

프라이빗 전용 API 서버 액세스

클러스터의 Kubernetes API 서버 엔드포인트에 대해 퍼블릭 액세스를 비활성화하면 VPC 또는 [연결된 네트워크](#) 내에서만 API 서버에 액세스할 수 있습니다. Kubernetes API 서버 엔드포인트에 액세스하는 방법은 다음과 같습니다.

연결된 네트워크

[AWS 전송 게이트웨이](#) 또는 기타 [연결](#) 옵션을 사용하여 네트워크를 VPC에 연결한 다음 연결된 네트워크의 컴퓨터를 사용합니다. Amazon EKS 제어 영역 보안 그룹에 연결된 네트워크의 포트 443에서 수신 트래픽을 허용하는 규칙이 포함되어 있는지 확인해야 합니다.

Amazon EC2 Bastion Host

Amazon EC2 인스턴스를 클러스터의 VPC에 있는 퍼블릭 서브넷으로 시작한 다음 SSH를 통해 해당 인스턴스에 로그인하여 kubectl 명령을 실행할 수 있습니다. 자세한 내용은 [AWS 기반 Linux Bastion 호스트](#)를 참조하세요. Amazon EKS 제어 영역 보안 그룹에 Bastion 호스트의 포트 443에서 수신 트래픽을 허용하는 규칙이 포함되어 있는지 확인해야 합니다. 자세한 내용은 [the section called “보안 그룹 요구 사항”](#) 섹션을 참조하세요.

Bastion Host에 대해 kubectl을 구성할 때 클러스터의 RBAC 구성에 이미 매핑된 AWS 자격 증명을 사용하거나, 엔드포인트 퍼블릭 액세스를 제거하기 전에 Bastion에서 사용할 [IAM 보안 주체](#)를 RBAC 구성에 추가해야 합니다. 자세한 내용은 [the section called “Kubernetes API 액세스”](#) 및 [the section called “권한이 없거나 액세스가 거부됨\(kubectl\)”](#) 섹션을 참조하세요.

AWS Cloud9 IDE

AWS Cloud9는 브라우저만으로 코드를 작성, 실행 및 디버깅할 수 있는 클라우드 기반 통합 개발 환경(IDE)입니다. 클러스터의 VPC에 AWS Cloud9 IDE를 생성할 수 있으며, IDE를 사용하여 클러스터와 통신할 수 있습니다. 자세한 내용은 [AWS Cloud9에서 환경 생성](#)을 참조하세요. Amazon

EKS 제어 영역 보안 그룹에 IDE 보안 그룹의 포트 443에서 수신 트래픽을 허용하는 규칙이 포함되어 있는지 확인해야 합니다. 자세한 내용은 [the section called “보안 그룹 요구 사항”](#) 섹션을 참조하세요.

AWS Cloud9 IDE에 대해 `kubectl`을 구성할 때 클러스터의 RBAC 구성에 이미 매핑된 AWS 자격 증명을 사용하거나, 엔드포인트 퍼블릭 액세스를 제거하기 전에 IDE에서 사용할 IAM 보안 주체를 RBAC 구성에 추가해야 합니다. 자세한 내용은 [the section called “Kubernetes API 액세스”](#) 및 [the section called “권한이 없거나 액세스가 거부됨\(kubectl\)”](#)(을)를 참조하세요.

[GitHub에서 이 페이지 편집](#)

클러스터 API 서버 엔드포인트에 대한 네트워크 액세스 구성

다음 섹션에서 AWS Management Console 또는 AWS CLI를 사용하여 클러스터 API 서버 엔드포인트 액세스를 수정할 수 있습니다.

엔드포인트 액세스 구성 - AWS 콘솔

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 클러스터 이름을 선택하여 클러스터 정보를 표시합니다.
3. 네트워킹 탭과 엔드포인트 액세스 관리를 차례대로 선택합니다.
4. 프라이빗 액세스에서 클러스터의 Kubernetes API 서버 엔드포인트에 대해 프라이빗 액세스를 활성화할지 아니면 비활성화할지를 선택합니다. 프라이빗 액세스를 활성화하면 클러스터의 VPC 내에서 비로트된 Kubernetes API 요청이 프라이빗 VPC 엔드포인트를 사용합니다. 퍼블릭 액세스를 비활성화하려면 프라이빗 액세스를 활성화해야 합니다.
5. 퍼블릭 액세스에서 클러스터의 Kubernetes API 서버 엔드포인트에 대해 퍼블릭 액세스를 활성화할지 아니면 비활성화할지를 선택합니다. 퍼블릭 액세스를 비활성화하면 클러스터의 Kubernetes API 서버가 클러스터의 VPC 내에서 비로트된 요청만 수신할 수 있습니다.
6. (선택사항) 퍼블릭 액세스를 활성화한 경우, 인터넷에서 퍼블릭 엔드포인트와 통신할 수 있는 주소를 지정할 수 있습니다. 고급 설정을 선택합니다. CIDR 블록(예: **203.0.113.5/32**)을 입력합니다. 블록에는 [예약된 주소](#)가 포함될 수 없습니다. Add Source(소스 추가)를 선택하여 추가 블록을 입력할 수 있습니다. 지정할 수 있는 최대 CIDR 블록 수가 있습니다. 자세한 내용은 [the section called “Service quotas”](#) 섹션을 참조하세요. 블록을 지정하지 않으면 퍼블릭 API 서버 엔드포인트는 듀얼 스택 IPv6 클러스터에 대한 IPv4(0.0.0.0/0)와 IPv6(:::/0)의 모든 IP 주소에서 요청을 수신합니다. CIDR 블록을 사용하여 퍼블릭 엔드포인트에 대한 액세스를 제한하는 경우, 노드와 Fargate Pods(사용하는 경우)가 클러스터와 통신할 수 있도록 프라이빗 엔드포인트 액세스도 활성화하는 것이 좋습니다. 프라이빗 엔드포인트를 활성화하지 않은 경우, 퍼블릭 액세스 엔드포인트 CIDR 소

스에는 VPC의 송신 소스가 포함되어야 합니다. 예를 들어, NAT 게이트웨이를 통해 인터넷과 통신하는 프라이빗 서브넷에 노드가 있는 경우, 퍼블릭 엔드포인트에서 허용하는 CIDR 블록의 일부로 NAT 게이트웨이의 아웃바운드 IP 주소를 추가해야 합니다.

7. 업데이트를 선택하여 완료합니다.

엔드포인트 액세스 구성 - AWS CLI

AWS CLI 버전 1.27.160 이상을 사용하여 다음 단계를 완료합니다. `aws --version`을 사용하여 현재 버전을 확인할 수 있습니다. AWS CLI를 설치하거나 업그레이드하려면 [AWS CLI 설치](#)를 참조하세요.

1. 다음 AWS CLI 명령을 통해 클러스터 API 서버 엔드포인트 액세스를 업데이트합니다. 클러스터 이름과 원하는 엔드포인트 액세스 값을 대체합니다. `endpointPublicAccess=true`를 설정한 경우 선택적으로 단일 CIDR 블록 또는 `publicAccessCidrs`에 대한 쉼표로 구분된 CIDR 블록 목록을 입력할 수 있습니다. 블록에는 [예약된 주소](#)가 포함될 수 없습니다. CIDR 블록을 지정한 경우 퍼블릭 API 서버 엔드포인트는 나열된 블록에서만 요청을 수신합니다. 지정할 수 있는 최대 CIDR 블록 수가 있습니다. 자세한 내용은 [the section called "Service quotas"](#) 섹션을 참조하세요. CIDR 블록을 사용하여 퍼블릭 엔드포인트에 대한 액세스를 제한하는 경우, 노드와 Fargate Pods(사용하는 경우)가 클러스터와 통신할 수 있도록 프라이빗 엔드포인트 액세스도 활성화하는 것이 좋습니다. 프라이빗 엔드포인트를 활성화하지 않은 경우, 퍼블릭 액세스 엔드포인트 CIDR 소스에는 VPC의 송신 소스가 포함되어야 합니다. 예를 들어, NAT 게이트웨이를 통해 인터넷과 통신하는 프라이빗 서브넷에 노드가 있는 경우, 퍼블릭 엔드포인트에서 허용하는 CIDR 블록의 일부로 NAT 게이트웨이의 아웃바운드 IP 주소를 추가해야 합니다. CIDR 블록을 지정하지 않으면 퍼블릭 API 서버 엔드포인트는 듀얼 스택 IPv6 클러스터에 대한 모든 (0.0.0.0/0) IP 주소와 IPv6(::/0)에서 요청을 수신합니다.

Note

다음 명령은 API 서버 엔드포인트에 대한 단일 IP 주소에서 프라이빗 액세스 및 퍼블릭 액세스를 활성화합니다. **203.0.113.5/32**를 단일 CIDR 블록 또는 네트워크 액세스를 제한할 쉼표로 구분된 CIDR 블록 목록으로 바꿉니다.

```
aws eks update-cluster-config \
  --region region-code \
  --name my-cluster \
  --resources-vpc-config
  endpointPublicAccess=true,publicAccessCidrs="203.0.113.5/32",endpointPrivateAccess=true
```

예제 출력은 다음과 같습니다.

```
{
  "update": {
    "id": "e6f0905f-a5d4-4a2a-8c49-EXAMPLE000000",
    "status": "InProgress",
    "type": "EndpointAccessUpdate",
    "params": [
      {
        "type": "EndpointPublicAccess",
        "value": "true"
      },
      {
        "type": "EndpointPrivateAccess",
        "value": "true"
      },
      {
        "type": "publicAccessCidrs",
        "value": "[\"203.0.113.5/32\"]"
      }
    ],
    "createdAt": 1576874258.137,
    "errors": []
  }
}
```

- 이전 명령에서 반환된 클러스터 이름과 업데이트 ID를 사용하여 다음 명령으로 엔드포인트 액세스 업데이트의 상태를 모니터링합니다. 상태가 `Successful`로 표시되면 업데이트가 완료된 것입니다.

```
aws eks describe-update \
  --region region-code \
  --name my-cluster \
  --update-id e6f0905f-a5d4-4a2a-8c49-EXAMPLE000000
```

예제 출력은 다음과 같습니다.

```
{
  "update": {
    "id": "e6f0905f-a5d4-4a2a-8c49-EXAMPLE000000",
    "status": "Successful",
    "type": "EndpointAccessUpdate",
```

```

    "params": [
      {
        "type": "EndpointPublicAccess",
        "value": "true"
      },
      {
        "type": "EndpointPrivateAccess",
        "value": "true"
      },
      {
        "type": "publicAccessCidrs",
        "value": "[\"203.0.113.5/32\"]"
      }
    ],
    "createdAt": 1576874258.137,
    "errors": []
  }
}

```

[GitHub에서 이 페이지 편집](#)

EKS 클러스터에 Windows 노드 배포

Amazon EKS 클러스터에 대한 Windows 지원을 활성화하고 관리하여 Linux 컨테이너와 함께 Windows 컨테이너를 실행하는 방법을 알아봅니다.

고려 사항

Windows 노드를 배포하기 전에 다음 사항을 고려해야 합니다.

- EKS Auto Mode는 Windows 노드를 지원하지 않습니다.
- HostProcess 포드로 Windows 노드에서 호스트 네트워킹을 사용할 수 있습니다. 자세한 내용은 Kubernetes Documentation의 [Create a Windows HostProcessPod](#)를 참조하세요.
- Amazon EKS 클러스터에는 CoreDNS와 같이 Linux에서만 실행되는 코어 시스템 포드를 실행하기 위한 Linux 또는 Fargate 노드가 하나 이상 있어야 합니다.
- kubelet 및 kube-proxy 이벤트 로그는 EKS Windows 이벤트 로그로 리디렉션되며 200MB 제한으로 설정됩니다.
- Windows 노드에서 실행 중인 포드가 있는 [개별 포드에 보안 그룹 할당](#)을 사용할 수 없습니다.

- Windows 노드에서는 [사용자 지정 네트워킹](#)을 사용할 수 없습니다.
- Windows 노드에서는 IPv6를 사용할 수 없습니다.
- Windows 노드는 노드당 하나의 탄력적 네트워크 인터페이스를 지원합니다. 기본적으로 Windows 노드당 실행할 수 있는 포트 수는 노드의 인스턴스 유형에 대해 탄력적 네트워크 인터페이스당 사용 가능한 IP 주소 수에서 1을 뺀 값과 같습니다. 자세한 내용은 Amazon EC2 사용 설명서의 [IP addresses per network interface per instance type](#)를 참조하세요.
- Amazon EKS 클러스터에서 로드 밸런서가 있는 단일 서비스는 최대 1,024개의 백엔드 포드를 지원할 수 있습니다. 각 포트에는 고유한 IP 주소가 있습니다. [OS 빌드 17763.2746](#)부터 [Windows Server 업데이트](#) 후 이전의 포트 64개 제한이 더 이상 적용되지 않습니다.
- Windows 컨테이너는 Fargate의 Amazon EKS 포트에 대해 지원되지 않습니다.
- Amazon EKS Hybrid Nodes를 Windows와 함께 호스트의 운영 체제로 사용할 수 없습니다.
- vpc-resource-controller 포트에서 로그를 검색할 수 없습니다. 이전에는 컨트롤러를 데이터 영역에 배포할 때 가능했습니다.
- IPv4 주소가 새 포트에 할당되기 전에 휴지 기간이 있습니다. 이렇게 하면 오래된 kube-proxy 규칙으로 인해 동일한 IPv4 주소를 사용하는 이전 포드로 트래픽이 흐르는 것을 방지할 수 있습니다.
- 컨트롤러의 소스는 GitHub에서 관리됩니다. 컨트롤러에 기여하거나 컨트롤러에 대한 문제를 제기하려면 GitHub의 [project](#)를 방문하세요.
- Windows 관리형 노드 그룹에 사용자 지정 AMI ID를 지정할 때는 AWS IAM Authenticator 구성 맵에 eks:kube-proxy-windows를 추가합니다. 자세한 내용은 [the section called “AMI ID 지정 시 제한과 조건”](#) 섹션을 참조하세요.
- 사용 가능한 IPv4 주소를 유지하는 것이 서브넷에 중요한 경우 [EKS Best Practices Guide - Windows Networking IP Address Management](#)를 참조하세요.
- EKS 액세스 항목에 대한 고려 사항
 - Windows 인스턴스에 다른 노드 IAM 역할을 사용하는 경우 EKS는 필요한 Windows 액세스 항목을 자동으로 생성합니다.
 - Windows 노드와 함께 사용할 액세스 항목에는 EC2_WINDOWS 유형이 필요합니다. 자세한 내용은 [the section called “액세스 항목 생성”](#) 섹션을 참조하세요.

Windows 노드에 대한 액세스 항목을 생성하려면 다음을 수행하세요.

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/<role-name> --type EC2_Windows
```

사전 조건

- 기존 클러스터가 있어야 합니다.
- CoreDNS를 실행하려면 클러스터에 Linux 노드 또는 Fargate 포드가 1개 이상(최소 2개 권장) 있어야 합니다. 레거시 Windows 지원을 사용 설정하는 경우 Linux 노드(Fargate 포드 사용 불가)를 사용하여 CoreDNS를 실행해야 합니다.
- 기존 [Amazon EKS 클러스터 IAM 역할](#).

Windows 지원 사용

1. 클러스터에 Amazon Linux 노드가 없고 포드에 보안 그룹을 사용하는 경우 다음 단계로 건너됩니다. 그렇지 않으면 AmazonEKSVPCResourceController 관리형 정책이 [클러스터 역할](#)에 연결되어 있는지 확인합니다. `eksClusterRole`을 클러스터 역할 이름으로 바꿉니다.

```
aws iam list-attached-role-policies --role-name eksClusterRole
```

예제 출력은 다음과 같습니다.

```
{
  "AttachedPolicies": [
    {
      "PolicyName": "AmazonEKSClusterPolicy",
      "PolicyArn": "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
    },
    {
      "PolicyName": "AmazonEKSVPCResourceController",
      "PolicyArn": "arn:aws:iam::aws:policy/AmazonEKSVPCResourceController"
    }
  ]
}
```

이전 출력에서와 같이 정책이 연결된 경우 다음 단계를 건너됩니다.

2. [AmazonEKSVPCResourceController](#) 관리형 정책을 [Amazon EKS 클러스터 IAM 역할](#)에 연결합니다. `eksClusterRole`을 클러스터 역할 이름으로 바꿉니다.

```
aws iam attach-role-policy \
  --role-name eksClusterRole \
```

```
--policy-arn arn:aws:iam::aws:policy/AmazonEKSVPCResourceController
```

3. VPC CNI ConfigMap을 업데이트하여 Windows IPAM을 활성화합니다.

- a. 다음 내용이 포함된 *vpc-resource-controller-configmap.yaml*이라는 파일을 생성합니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: amazon-vpc-cni
  namespace: kube-system
data:
  enable-windows-ipam: "true"
```

- b. 클러스터에 ConfigMap 적용.

```
kubectl apply -f vpc-resource-controller-configmap.yaml
```

4. 클러스터에 aws-auth configmap을 활성화하도록 인증 모드가 설정된 경우

- aws-auth ConfigMap에 Windows 노드의 인스턴스 역할에 대한 매핑이 포함되어 `eks:kube-proxy-windows` RBAC 권한 그룹을 포함하는지 확인합니다. 다음 명령을 실행하여 확인할 수 있습니다.

```
kubectl get configmap aws-auth -n kube-system -o yaml
```

예제 출력은 다음과 같습니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      - eks:kube-proxy-windows # This group is required for Windows DNS resolution
        to work
      rolearn: arn:aws:iam::111122223333:role/eksNodeRole
      username: system:node:{{EC2PrivateDNSName}}
```


[...]

그룹 아래에 `eks:kube-proxy-windows`가 표시될 것입니다. 그룹이 지정되지 않은 경우 ConfigMap을 업데이트하거나 생성하여 필수 그룹을 포함해야 합니다. `aws-auth` ConfigMap에 대한 자세한 내용은 [the section called “클러스터에 aws-authConfigMap 적용”](#) 섹션을 참조하세요.

5. 클러스터에 `aws-auth configmap`을 비활성화하도록 인증 모드가 설정된 경우 EKS 액세스 항목을 사용할 수 있습니다. Windows 인스턴스와 함께 사용할 새 노드 역할을 생성하면 EKS가 EC2_WINDOWS 유형의 액세스 항목을 자동으로 생성합니다.

Windows 포드 배포

클러스터에 포드를 배포할 때 노드 유형을 혼합하여 실행하는 경우 사용되는 운영 체제를 지정해야 합니다.

Linux 포드의 경우 매니페스트에서 다음 노드 선택기 텍스트를 사용합니다.

```
nodeSelector:
  kubernetes.io/os: linux
  kubernetes.io/arch: amd64
```

Windows 포드의 경우 매니페스트에서 다음 노드 선택기 텍스트를 사용합니다.

```
nodeSelector:
  kubernetes.io/os: windows
  kubernetes.io/arch: amd64
```

[샘플 애플리케이션](#)을 배포하여 사용 중인 노드 선택기를 볼 수 있습니다.

Windows 노드에서 더 높은 포드 밀도 지원

Amazon EKS에서 각 포드에는 VPC의 IPv4 주소가 할당됩니다. 이로 인해 노드에서 더 많은 포드를 실행하기에 충분한 리소스가 있더라도 노드에 배포할 수 있는 포드 수는 사용 가능한 IP 주소에 의해 제한됩니다. Windows 노드에서는 탄력적 네트워크 인터페이스를 하나만 지원하므로 기본적으로 Windows 노드에서 사용 가능한 최대 IP 주소 수는 다음과 같습니다.

```
Number of private IPv4 addresses for each interface on the node - 1
```

하나의 IP 주소는 네트워크 인터페이스의 기본 IP 주소로 사용되므로 포드에 할당할 수 없습니다.

IP 접두사 위임을 활성화하여 Windows 노드에서 더 높은 포트 밀도를 활성화할 수 있습니다. 이 기능을 사용하면 보조 IPv4 주소를 할당하는 대신 기본 네트워크 인터페이스에 /28 IPv4 접두사를 할당할 수 있습니다. IP 접두사를 할당하면 노드에서 사용 가능한 최대 IPv4 주소가 다음으로 증가합니다.

```
(Number of private IPv4 addresses assigned to the interface attached to the node - 1) * 16
```

이렇게 사용 가능한 IP 주소의 수가 상당히 많기 때문에 사용 가능한 IP 주소가 노드의 포트 수를 확장하는 기능을 제한하지는 않습니다. 자세한 내용은 [the section called “IP 주소 증가”](#) 섹션을 참조하세요.

Windows 지원 사용 중지

1. 클러스터에 Amazon Linux 노드가 포함되어 있고 [포트에 보안 그룹](#)을 함께 사용하는 경우 이 단계를 건너뛰니다.

[클러스터 역할](#)에서 AmazonVPCResourceController 관리형 IAM 정책을 제거합니다. **eksClusterRole**을 클러스터 역할의 이름으로 바꿉니다.

```
aws iam detach-role-policy \
  --role-name eksClusterRole \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSVPCResourceController
```

2. amazon-vpc-cni ConfigMap에서 Windows IPAM을 사용 중지합니다.

```
kubectl patch configmap/amazon-vpc-cni \
  -n kube-system \
  --type merge \
  -p '{"data":{"enable-windows-ipam":"false"}}'
```

인터넷 액세스가 제한된 프라이빗 클러스터 배포

이 주제에서는 아웃바운드 인터넷 액세스 권한이 없는 AWS 클라우드에 대해, Amazon EKS 클러스터를 배포하는 방법을 설명합니다. AWS Outposts에 로컬 클러스터가 있는 경우 이 항목 대신 [the section called “노드”](#)를 참조하세요.

Amazon EKS 네트워킹에 익숙하지 않은 경우 [Amazon EKS 작업자 노드에 대한 클러스터 네트워킹 설명](#)을 참조하세요. 클러스터에 아웃바운드 인터넷 액세스 권한이 없는 경우에는 다음과 같은 요구 사항을 충족해야 합니다.

- 클러스터에서는 VPC에 있는 컨테이너 레지스트리에서 이미지를 가져와야 합니다. VPC에서 Amazon Elastic Container Registry를 생성하고 노드에서 가져올 컨테이너 이미지를 복사할 수 있습니다. 자세한 내용은 [the section called “리포지토리에 이미지 복사”](#) 섹션을 참조하세요.
- 클러스터에 활성화된 엔드포인트 프라이빗 액세스 권한이 있어야 합니다. 노드에서 클러스터 엔드포인트에 등록하려면 이 권한이 필요합니다. 엔드포인트 퍼블릭 액세스는 선택 사항입니다. 자세한 내용은 [the section called “클러스터 엔드포인트 액세스”](#) 단원을 참조하십시오.
- 자체 관리형 Linux 노드와 Windows 노드에서는 시작하기 전에 다음과 같은 부트스트랩 인수가 포함되어야 합니다. 이러한 인수에서는 Amazon EKS 내부 검사를 우회하며 VPC 내에서 Amazon EKS API에 액세스하는 권한이 필요하지 않습니다.
 - a. 다음과 같은 명령으로 클러스터의 엔드포인트 값을 결정합니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks describe-cluster --name my-cluster --query cluster.endpoint --output text
```

예제 출력은 다음과 같습니다.

```
https://EXAMPLE108C897D9B2F1B21D5EXAMPLE.sk1.region-code.eks.amazonaws.com
```

- b. 다음과 같은 명령으로 클러스터의 인증 기관 값을 결정합니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks describe-cluster --name my-cluster --query cluster.certificateAuthority --output text
```

반환된 출력은 긴 문자열입니다.

- c. 다음과 같은 명령의 *cluster-endpoint*와 *certificate-authority*를 이전 명령의 출력에서 반환된 값으로 바꿉니다. 자체 관리형 노드 시작 시 부트스트랩 인수 지정에 대한 자세한 내용은 [the section called “Amazon Linux”](#) 및 [the section called “Windows”](#) 부분을 참조하세요.

- Linux 노드의 경우:

```
--apiserver-endpoint cluster-endpoint --b64-cluster-ca certificate-authority
```

추가 인수는 GitHub에서 [부트스트랩 스크립트](#)를 참조하세요.

- Windows 노드의 경우:

Note

사용자 지정 서비스 CIDR을 사용하는 경우 -ServiceCIDR 파라미터를 사용하여 CIDR을 지정해야 합니다. 그렇지 않으면 클러스터의 포드에 대한 DNS 확인이 실패합니다.

```
-APIServerEndpoint cluster-endpoint -Base64ClusterCA certificate-authority
```

추가 인수는 [the section called “부트스트랩 스크립트 구성 파라미터”](#) 부분을 참조하세요.

- 클러스터의 aws-auth ConfigMap은 VPC 내에서 생성해야 합니다. 항목을 생성하고 aws-auth ConfigMap에 추가하는 방법에 대한 자세한 내용을 보려면 터미널에 `eksctl create iamidentitymapping --help`를 입력합니다. ConfigMap이 서버에 없는 경우 eksctl에서는 명령을 사용해 ID 매핑을 추가할 때 해당 항목을 생성합니다.
- [서비스 계정에 대한 IAM 역할](#)로 구성된 포드에서는 AWS Security Token Service(AWS STS) API 직접 호출에서 자격 증명을 취득합니다. 아웃바운드 인터넷 액세스 권한이 없는 경우 VPC에서 AWS STS VPC 엔드포인트를 생성하고 사용해야 합니다. 대부분의 AWS v1 SDK는 기본적으로 글로벌 AWS STS 엔드포인트를 사용하며(`sts.amazonaws.com`), AWS STS VPC 엔드포인트는 사용하지 않습니다. AWS STS VPC 엔드포인트를 사용하려면 리전별 AWS 엔드포인트(`sts.region-code.amazonaws.com`)를 사용하도록 SDK를 구성해야 할 수도 있습니다. 자세한 내용은 [the section called “STS 엔드포인트”](#) 단원을 참조하십시오.
- 클러스터의 VPC 서브넷에는 포드에서 액세스해야 하는 모든 AWS 서비스에 대한 VPC 인터페이스 엔드포인트가 있어야 합니다. 자세한 내용은 [인터페이스 VPC 엔드포인트를 사용하여 AWS 서비스에 액세스](#)를 참조하세요. 일반적으로 사용하는 몇 가지 서비스와 엔드포인트가 아래 표에 나열되어 있습니다. 엔드포인트의 전체 목록은 [AWS PrivateLink 가이드](#)에서 [AWS PrivateLink와 통합되는 AWS 서비스](#)를 참조하세요.

VPC 엔드포인트의 [프라이빗 DNS 이름을 활성화](#)하는 것이 좋습니다. 이렇게 하면 워크로드가 문제 없이 퍼블릭 AWS 서비스 엔드포인트를 계속 사용할 수 있습니다.

Service	엔드포인트
Amazon EC2	<code>com.amazonaws.region-code.ec2</code>

Service	엔드포인트
Amazon Elastic Container Registry(컨테이너 이미지 가져오기용)	com.amazonaws. <i>region-code</i> .ecr.api, com.amazonaws. <i>region-code</i> .ecr.dkr, and com.amazonaws. <i>region-code</i> .s3
Application Load Balancer 및 Network Load Balancer	com.amazonaws. <i>region-code</i> .elasticloadbalancing
AWS X-Ray	com.amazonaws. <i>region-code</i> .xray
Amazon CloudWatch Logs	com.amazonaws. <i>region-code</i> .logs
AWS 보안 토큰 서비스(서비스 계정에 IAM 역할 사용 시 필요)	com.amazonaws. <i>region-code</i> .sts
Amazon EKS Auth(Pod Identity 연결을 사용할 때 필요)	com.amazonaws. <i>region-code</i> .eks-auth
Amazon EKS	com.amazonaws. <i>region-code</i> .eks

- 자체 관리형 노드는 필요한 VPC 인터페이스 엔드포인트가 있는 서브넷에 배포해야 합니다. 관리형 노드 그룹을 생성하는 경우 VPC 인터페이스 엔드포인트 보안 그룹에서 서브넷에 대한 CIDR을 허용하거나 본인이 생성된 노드 보안 그룹을 VPC 인터페이스 엔드포인트 보안 그룹에 추가해야 합니다.
- 포드에서 Amazon EFS 볼륨을 사용하는 경우 [Amazon EFS로 저장된 탄력적 파일 시스템](#)을 배포하기 전에 컨테이너 이미지가 Amazon EKS 클러스터와 동일한 AWS 리전을 사용하도록 설정하도록 드라이버의 [kustomization.yaml](#) 파일을 변경해야 합니다.
- Route53는 AWS PrivateLink를 지원하지 않습니다. 프라이빗 Amazon EKS 클러스터에서는 Route53 DNS 레코드를 관리할 수 없습니다. 이는 Kubernetes [external-dns](#)에 영향을 미칩니다.
- EKS 최적화 AMI를 사용하는 경우 위 표의 ec2 엔드포인트를 활성화해야 합니다. 또는 노드 DNS 이름을 수동으로 설정할 수 있습니다. 최적화된 AMI는 EC2 API를 사용하여 노드 DNS 이름을 자동으로 설정합니다.
- [AWS 로드 밸런서 컨트롤러](#)를 사용하여 AWS 애플리케이션 로드 밸런서(ALB) 및 네트워크 로드 밸런서를 프라이빗 클러스터에 배포할 수 있습니다. 배포할 때 [명령줄 플래그](#)를 사용하여 enable-shield, enable-waf 및 enable-wafv2를 false로 설정해야 합니다. 수신 객체의 호스트 이름을 사용한 [인증서 검색](#)은 지원되지 않습니다. 이는 컨트롤러가 VPC 인터페이스 엔드포인트가 없는 AWS 인증서 관리자에 연결해야 하기 때문입니다.

컨트롤러는 Fargate와 함께 사용하는 데 필요한 IP 대상이 있는 Network Load Balancer를 지원합니다. 자세한 내용은 [the section called “애플리케이션 로드 밸런싱”](#) 및 [the section called “Network Load Balancer 생성”](#) 섹션을 참조하세요.

- [Cluster Autoscaler](#)가 지원됩니다. Cluster Autoscaler 포드를 배포할 때 명령줄이 `--aws-use-static-instance-list=true`를 포함하는지 확인하세요. 자세한 내용은 GitHub에서 [정적 인스턴스 목록 사용](#)을 참조하세요. 작업자 노드 VPC에는 AWS STS VPC 엔드포인트 및 자동 크기 조정 VPC 엔드포인트도 포함되어야 합니다.
- 일부 컨테이너 소프트웨어 제품은 사용량을 모니터링하는 AWS Marketplace 미터링 서비스에 액세스하는 API 호출을 사용합니다. 프라이빗 클러스터에서는 이러한 호출이 허용되지 않으므로 이러한 컨테이너 유형은 프라이빗 클러스터에서 사용할 수 없습니다.

EKS의 Kubernetes 버전 수명 주기 이해

Kubernetes는 새로운 기능, 디자인 업데이트, 버그 수정으로 빠르게 진화하고 있습니다. 커뮤니티에서는 평균 4개월에 한 번씩 새로운 Kubernetes 마이너 버전(예: 1.32)을 릴리스합니다. Amazon EKS는 마이너 버전의 업스트림 릴리스 및 지원 중단 주기를 따릅니다. Amazon EKS에서 새로운 Kubernetes 버전을 사용할 수 있게 되면 미리 클러스터를 업데이트하여 최신 버전을 사용하는 것이 좋습니다.

마이너 버전은 릴리스 후 처음 14개월 동안 Amazon EKS에서 표준 지원을 받습니다. 버전 표준 지원 종료일이 지나면 다음 12개월 동안의 추가 지원이 시작됩니다. 확장 지원을 통해 클러스터 시간당 추가 비용을 지불하고 특정 Kubernetes 버전을 더 오래 사용할 수 있습니다. 추가 지원 기간이 종료되기 전에 클러스터를 업데이트하지 않은 경우 클러스터는 현재 지원되는 가장 오래된 추가 버전으로 자동 업그레이드됩니다.

추가 지원은 기본적으로 활성화되어 있습니다. [the section called “추가 지원 비활성화”](#)

Amazon EKS에서 지원하는 사용 가능한 최신 Kubernetes 버전으로 클러스터를 생성하는 것이 좋습니다. 애플리케이션에 특정 버전의 Kubernetes 버전이 필요한 경우 이전 버전을 선택할 수 있습니다. 표준 또는 추가 지원으로 제공되는 모든 버전에서 새 Amazon EKS 클러스터를 생성할 수 있습니다.

추가 지원으로 사용 가능한 버전

현재 Amazon EKS 표준 지원으로 사용할 수 있는 Kubernetes 버전은 다음과 같습니다.

- 1.33

- 1.32
- 1.31
- 1.30

표준 지원의 각 버전에 대해 알아야 할 중요한 변경 사항은 [the section called “표준 지원 버전”](#)을(를) 참조하세요.

확장 지원으로 사용 가능한 버전

현재 Amazon EKS 확장 지원으로 사용할 수 있는 Kubernetes 버전은 다음과 같습니다.

- 1.29
- 1.28
- 1.27
- 1.26

추가 지원의 각 버전에 대해 알아야 할 중요한 변경 사항은 [the section called “추가 지원 버전”](#)을(를) 참조하세요.

Amazon EKS Kubernetes 릴리스 일정

다음 표에는 각 Kubernetes 버전에서 고려해야 할 중요한 릴리스 및 지원 날짜가 나와 있습니다. 추가 지원에 대한 청구는 해당 버전에 대한 표준 지원이 종료되는 날부터 시작됩니다.

Note

월과 연도만 있는 날짜는 대략적인 날짜이며 알 수 있는 정확한 날짜로 업데이트됩니다.

이 특정 설명서 페이지의 모든 소스 파일 변경 사항에 대한 알림을 받으려면 RSS 리더를 사용하여 다음 URL을 구독하세요.

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/clusters/kubernetes-versions.adoc.atom
```

Kubernetes 버전	업스트림 릴리스	Amazon EKS 릴리스	표준 지원 종료일	추가 지원 종료일
1.33	2025년 4월 23일	2025년 5월 29일	2026년 7월 29일	2027년 7월 29일
1.32	2024년 12월 11일	2025년 1월 23일	2026년 3월 23일	2027년 3월 23일
1.31	2024년 8월 13일	2024년 9월 26일	2025년 11월 26일	2026년 11월 26일
1.30	2024년 4월 17일	2024년 5월 23일	2025년 7월 23일	2026년 7월 23일
1.29	2023년 12월 13일	2024년 1월 23일	2025년 3월 23일	2026년 3월 23일
1.28	2023년 8월 15일	2023년 9월 26일	2024년 11월 26일	2025년 11월 26일
1.27	2023년 4월 11일	2023년 5월 24일	2024년 7월 24일	2025년 7월 24일
1.26	2022년 12월 9일	2023년 4월 11일	2024년 6월 11일	2025년 6월 11일

AWS CLI를 사용하여 버전 정보 가져오기

AWS CLI를 사용하여 표준 지원의 종료 날짜와 같이 EKS에서 사용 가능한 Kubernetes 버전에 대한 정보를 가져올 수 있습니다.

AWS CLI를 사용하여 EKS에서 사용 가능한 Kubernetes 버전에 대한 정보를 검색하려면

1. 터미널을 엽니다.
2. AWS CLI가 설치되고 구성되어 있는지 확인합니다. 자세한 내용은 [최신 버전의 CLI 설치 또는 업데이트](#)를 참조하세요.
3. 다음 명령 실행:

```
aws eks describe-cluster-versions
```


4. 명령은 사용 가능한 클러스터 버전에 대한 세부 정보가 포함된 JSON 출력을 반환합니다. 다음은 출력 예시입니다.

```
{
  "clusterVersions": [
    {
      "clusterVersion": "1.31",
      "clusterType": "eks",
      "defaultPlatformVersion": "eks.21",
      "defaultVersion": true,
      "releaseDate": "2024-09-25T17:00:00-07:00",
      "endOfStandardSupportDate": "2025-11-25T16:00:00-08:00",
      "endOfExtendedSupportDate": "2026-11-25T16:00:00-08:00",
      "status": "STANDARD_SUPPORT",
      "kubernetesPatchVersion": "1.31.3"
    }
  ]
}
```

출력은 각 클러스터 버전에 대해 다음 정보를 제공합니다.

- `clusterVersion`: EKS 클러스터의 Kubernetes 버전
- `clusterType`: 클러스터 유형(예: 'eks')
- `defaultPlatformVersion`: 기본 EKS 플랫폼 버전
- `defaultVersion`: 기본 버전 여부
- `releaseDate`: 이 버전이 릴리스된 날짜
- `endOfStandardSupportDate`: 표준 지원이 종료되는 날짜
- `endOfExtendedSupportDate`: 확장 지원이 종료되는 날짜
- `status`: 버전의 현재 지원 상태(STANDARD_SUPPORT 또는 EXTENDED_SUPPORT)
- `kubernetesPatchVersion`: 특정 Kubernetes 패치 버전

Amazon EKS 버전 FAQ

표준 지원으로 사용할 수 있는 Kubernetes 버전은 몇 개인가요?

Kubernetes 버전에 대한 Kubernetes 커뮤니티 지원에 발맞춰 Amazon EKS는 언제든지 세 가지 Kubernetes 버전에 대한 지원을 제공하기 위해 최선을 다하고 있습니다. 지정된 Kubernetes 마이

너 버전의 표준 지원 종료일은 최소 60일 전에 공지할 예정입니다. 새로운 Kubernetes 버전에 대한 Amazon EKS 인증 및 릴리스 프로세스로 인해, Amazon EKS의 Kubernetes 버전 표준 지원 종료일은 Kubernetes 프로젝트가 버전 업스트림 지원을 중지하는 날짜 이후가 됩니다.

Kubernetes는 Amazon EKS의 표준 지원을 얼마나 오래 받나요?

Kubernetes 버전은 Amazon EKS에서 처음 제공된 후 14개월 동안 표준 지원을 받습니다. 업스트림 Kubernetes가 Amazon EKS에서 제공되는 버전을 더 이상 지원하지 않는 경우에도 마찬가지입니다. Amazon EKS에서 지원되는 Kubernetes 버전에 적용할 수 있는 보안 패치를 백포트하고 있습니다.

Amazon EKS에서 Kubernetes 버전에 대한 지원이 종료되면 알림을 받을 수 있나요?

예. 계정의 클러스터에서 지원 종료 시점이 거의 도래한 버전을 실행 중인 경우 Amazon EKS는 Kubernetes 버전이 Amazon EKS에서 릴리스된 후 약 12개월되는 시점에 AWSHealth Dashboard을(를) 통해 알림을 보냅니다. 알림에는 지원 종료 날짜가 포함됩니다. 공지일로부터 최소 60일이 소요됩니다.

Amazon EKS에서 지원하는 Kubernetes 기능은 무엇인가요?

Amazon EKS는 Kubernetes API의 모든 상용(GA) 기능을 지원합니다. 새 베타 API가 기본적으로 클러스터에서 활성화되어 있지 않습니다. 하지만 이전 기존 베타 API와 기존 베타 API의 새 버전이 기본적으로 계속 활성화되어 있습니다. 알파 기능은 지원되지 않습니다.

Amazon EKS 관리형 노드 그룹은 클러스터 컨트롤 플레인 버전과 함께 자동으로 업데이트되나요?

아니요. 관리형 노드 그룹은 사용자 계정에 Amazon EC2 인스턴스를 생성합니다. 이러한 인스턴스는 사용자 또는 Amazon EKS가 컨트롤 플레인을 업데이트할 때 자동으로 업그레이드되지 않습니다. 자세한 내용은 [the section called “업데이트”](#) 섹션을 참조하세요. 제어 플레인과 노드에 동일한 Kubernetes 버전을 유지하는 것이 좋습니다.

자체 관리형 노드 그룹은 클러스터 컨트롤 플레인 버전과 함께 자동으로 업데이트되나요?

아니요. 자체 관리형 노드 그룹은 사용자 계정에 Amazon EC2 인스턴스를 포함합니다. 이러한 인스턴스는 사용자 또는 Amazon EKS가 컨트롤 플레인 버전을 업데이트할 때 자동으로 업그레이드되지 않습니다. 자체 관리형 노드 그룹에는 콘솔에 업데이트가 필요하다는 표시가 없습니다. 클러스터의 개요 탭에 있는 노드 목록에서 노드를 선택하여 노드에 설치된 kubelet 버전을 확인하면 업데이트가 필요한 노드를 확인할 수 있습니다. 노드를 수동으로 업데이트해야 합니다. 자세한 내용은 [the section called “업데이트 방법”](#) 섹션을 참조하세요.

Kubernetes 프로젝트는 최대 3개의 마이너 버전에 대한 컨트롤 플레인과 노드 간의 호환성을 테스트합니다. 예를 들어, 1.29 노드는 1.32 컨트롤 플레인에 의해 오케스트레이션될 경우 계

속 작동합니다. 그러나 컨트롤 플레인보다 지속적으로 세 버전이 낮은 마이너 버전이 있는 노드가 있는 클러스터를 실행하는 것은 권장되지 않습니다. 자세한 내용은 [Kubernetes Version and Version Skew Support Policy](#)를 참조하십시오. 제어 플레인과 노드에 동일한 Kubernetes 버전을 유지하는 것이 좋습니다.

Fargate에서 실행되는 포드는 자동 클러스터 컨트롤 플레인 버전 업그레이드로 자동 업그레이드되나요?

아니요. Kubernetes 배포와 같은 복제 컨트롤러의 일부로 Fargate 포드를 실행하는 것이 좋습니다. 그런 다음 모든 Fargate 포드의 롤링을 재시작하세요. Fargate 포드의 새 버전은 업데이트된 클러스터 컨트롤 플레인 버전과 동일한 kubelet 버전을 사용하여 배포됩니다. 자세한 내용은 Kubernetes 문서의 [배포](#)를 참조하세요.

Important

컨트롤 플레인을 업데이트하는 경우 Fargate 노드를 직접 업데이트해야 합니다. Fargate 노드를 업데이트하려면 노드가 나타내는 Fargate 포드를 삭제하고 포드를 다시 배포합니다. 새 포드는 클러스터와 동일한 kubelet 버전을 사용하여 배포됩니다.

하이브리드 노드에서 지원되는 Kubernetes 버전

Amazon EKS Hybrid Nodes는 표준 및 확장 Kubernetes 버전 지원을 포함하여 다른 노드 컴퓨팅 유형과 함께 Amazon EKS 클러스터와 동일한 Kubernetes 버전을 지원합니다. 컨트롤 플레인 버전을 업그레이드하면 하이브리드 노드가 자동으로 업그레이드되지 않으며 하이브리드 노드의 업그레이드는 사용자의 책임입니다. 자세한 내용은 [the section called “하이브리드 노드 업그레이드”](#) 섹션을 참조하세요.

Amazon EKS 추가 지원 FAQ

표준 지원 및 추가 지원이라는 용어가 생소합니다. 이 용어들은 무엇을 의미하나요?

Amazon EKS의 Kubernetes 버전에 대한 표준 지원은 Amazon EKS에서 Kubernetes 버전이 릴리스될 때 시작되며 릴리스 날짜로부터 14개월 후에 종료됩니다. Kubernetes 버전에 대한 확장 지원은 표준 지원이 종료된 직후에 시작되며 이후 12개월 후에 종료됩니다. 예를 들어 Amazon EKS 버전에서 버전 1.23에 대한 표준 지원은 2023년 10월 11일에 종료됩니다. 버전 1.23에 대한 추가 지원은 2023년 10월 12일에 시작되었으며 2024년 10월 11일에 종료되었습니다.

Amazon EKS 클러스터에 대한 추가 지원을 받으려면 어떻게 해야 하나요?

클러스터 업그레이드 정책을 EXTENDED로 변경하여 클러스터에 대한 확장 지원([the section called “추가 지원 활성화”](#) 참조)을 활성화해야 합니다. 기본적으로 모든 새 클러스터와 기존 클러스터의 경우 달리 지정하지 않는 한 업그레이드 정책은 확장됨으로 설정됩니다. 클러스터의 업그레이드 정책을 보려면 [the section called “업그레이드 정책 보기”](#)을 참조하세요. 표준 지원은 Amazon EKS에 Kubernetes 버전이 릴리스될 때 시작되며 릴리스 날짜로부터 14개월 후에 종료됩니다. Kubernetes 버전에 대한 확장 지원은 표준 지원이 종료된 직후에 시작되며 이후 12개월 후에 종료됩니다.

어떤 Kubernetes 버전에 대해 확장 지원을 받을 수 있나요?

해당 버전에 대한 표준 지원이 종료된 후 최대 12개월 동안 모든 버전에서 클러스터를 실행할 수 있습니다. 즉, Amazon EKS에서 각 버전이 26개월(표준 지원 14개월 + 추가 지원 12개월) 동안 지원됩니다.

추가 지원을 사용하고 싶지 않으면 어떻게 하나요?

확장 지원에 자동으로 등록되는 것을 원하지 않는 경우 클러스터를 표준 Amazon EKS 지원의 Kubernetes 버전으로 업그레이드할 수 있습니다. 확장 지원을 비활성화하는 방법은 [the section called “추가 지원 비활성화”](#)을 참조하세요. 참고: 확장 지원을 사용하지 않도록 설정하면 표준 지원이 종료될 때 클러스터가 자동으로 업그레이드됩니다.

12개월의 추가 지원이 종료되면 어떻게 되나요?

26개월의 수명 주기(14개월의 표준 지원 + 12개월의 확장 지원)를 마친 Kubernetes 버전에서 실행되는 클러스터는 다음 버전으로 자동 업그레이드됩니다. 자동 업그레이드에는 Kubernetes 컨트롤 플레인만 포함됩니다. EKS Auto Mode 노드가 있는 경우 노드가 자동으로 업데이트될 수 있습니다. 자체 관리형 노드와 EKS 관리형 노드 그룹은 이전 버전에 유지됩니다.

추가 지원 종료 날짜에는 지원되지 않는 버전으로 새 Amazon EKS 클러스터를 더 이상 생성할 수 없게 됩니다. 지원 종료 날짜 이후 점진적인 배포 프로세스를 통해 Amazon EKS에 의해 기존 컨트롤 플레인이 지원되는 최신 버전으로 자동 업데이트됩니다. 자동 컨트롤 플레인 업데이트 후에는 클러스터 추가 기능 및 Amazon EC2 노드를 수동으로 업데이트해야 합니다. 자세한 내용은 [the section called “Kubernetes 버전 업데이트”](#) 섹션을 참조하세요.

추가 지원 종료 날짜 이후에 컨트롤 플레인이 정확히 언제 자동으로 업데이트되나요?

Amazon EKS는 특정 기간을 제공할 수 없습니다. 자동 업데이트는 추가 지원 종료 날짜 이후 언제든지 발생할 수 있습니다. 업데이트 전에 알림이 제공되지 않습니다. Amazon EKS 자동 업데이트 프로세스에 의존하지 않고 선제적으로 컨트롤 플레인을 업데이트하는 것이 좋습니다. 자세한 내용은 [the section called “Kubernetes 버전 업데이트”](#) 섹션을 참조하세요.

컨트롤 플레인을 Kubernetes 버전에 무기한 그대로 둘 수 있나요?

아니요. AWS에서는 클라우드 보안을 가장 중요하게 생각합니다. 특정 시점(보통 1년)이 지난 후 Kubernetes 커뮤니티는 일반 취약성 및 노출(CVE) 패치 릴리스를 중단하고 지원되지 않는 버전의 CVE 제출을 금지합니다. 즉, Kubernetes의 이전 버전과 관련된 취약성은 보고되지 않을 수도 있습니다. 이렇게 하면 취약성의 경우 통지 및 수정 옵션 없이 클러스터를 노출합니다. 이를 고려하여 Amazon EKS는 컨트롤 플레인이 추가 지원 종료에 도달한 버전을 유지하도록 허용하지 않습니다.

추가 지원을 받으려면 추가 비용이 드나요?

예. 추가 지원으로 실행되는 Amazon EKS 클러스터에는 추가 비용이 부과됩니다. 가격 책정에 대한 자세한 내용은 AWS 블로그 또는 [가격 책정 페이지](#)에서 [Kubernetes 버전에 대한 Amazon EKS 확장 지원 가격](#)을 참조하세요.

추가 지원에는 무엇이 포함되나요?

확장 지원의 Amazon EKS 클러스터는 Kubernetes 컨트롤 플레인에 대한 지속적인 보안 패치를 받습니다. 또한 Amazon EKS는 확장 지원 버전의 Amazon VPC CNI, kube-proxy 및 CoreDNS 추가 기능에 대한 패치를 릴리스할 예정입니다. Amazon EKS는 Amazon Linux, Bottlerocket 및 Windows용 AWS 게시 Amazon EKS 최적화 AMI용 패치와 해당 버전용 Amazon EKS Fargate 노드도 출시할 예정입니다. 추가 지원의 모든 클러스터는 계속해서 AWS의 기술 지원을 받을 수 있습니다.

확장 지원에서 Kubernetes 구성 요소가 아닌 패치에 적용되는 제한 사항이 있나요?

확장 지원은 AWS의 모든 Kubernetes 특정 구성 요소를 포함하지만 항상 Amazon Linux, Bottlerocket 및 Windows용 AWS 게시 Amazon EKS 최적화 AMI에 대한 지원만 제공합니다. 즉, 추가 지원을 사용하는 동안 Amazon EKS 최적화 AMI에 새로운 구성 요소(예: OS 또는 커널)가 있을 수 있습니다. 예를 들어, Amazon Linux 2의 [수명 주기가 2025년에 종료](#)되면 Amazon EKS 최적화 Amazon Linux AMI는 최신 Amazon Linux OS를 사용하여 구축됩니다. Amazon EKS는 각 Kubernetes 버전에 대해 이와 같은 중요한 지원 수명 주기 불일치 사항을 발표하고 문서화합니다.

추가 지원 버전을 사용하여 새 클러스터를 생성할 수 있습니까?

예.

표준 지원의 Kubernetes 버전에 대한 릴리스 노트 검토

이 주제에서는 표준 지원의 각 Kubernetes 버전에 대해 알아야 할 중요한 변경 사항을 제공합니다. 업그레이드할 때는 클러스터의 이전 버전과 새 버전 간에 발생한 변경 사항을 주의 깊게 검토하세요.

Kubernetes 1.33

Kubernetes 1.33은 이제 Amazon EKS에서 사용할 수 있습니다. Kubernetes 1.33에 대한 자세한 내용은 [공식 릴리스 발표](#)를 참조하세요.

Important

- 동적 리소스 할당 베타 Kubernetes API가 활성화되었습니다.
 - 이 베타 API는 GPU 등의 리소스가 필요한 워크로드를 예약하고 모니터링하는 환경을 개선합니다.
 - 베타 API는 Kubernetes 커뮤니티에서 정의되며 향후 버전의 Kubernetes에서 변경될 수 있습니다.
 - 베타 API 사용의 의미를 이해하려면 Kubernetes 문서의 [기능 게이트](#)를 주의 깊게 검토하세요.
- AWS는 Kubernetes 1.33용 EKS 최적화 Amazon Linux 2 AMI를 릴리스하지 않습니다.
 - AWS에서는 Amazon Linux 2023으로 마이그레이션을 권장합니다. [the section called “AL2023으로 업그레이드”](#) 방법에 대해 알아보십시오.
 - 자세한 내용은 [the section called “Amazon Linux 2 AMI 사용 중단”](#) 섹션을 참조하세요.
- 인플레이스 포드 리소스 크기 조정(베타): 인플레이스 리소스 크기 조정이 베타로 승격되어 기존 포드의 CPU 및 메모리 리소스를 재시작하지 않고도 동적으로 업데이트할 수 있습니다. 이를 통해 가동 중지 시간 없이 상태 저장 워크로드의 수직적 스케일링과 트래픽 패턴에 따른 원활한 리소스 조정이 가능합니다.
- 사이드카 컨테이너 안정화: 사이드카 컨테이너가 안정화되어, 애플리케이션 컨테이너보다 먼저 시작하고, 포드 수명 주기 전체에서 실행되고, 운영 상태 신호를 위한 프로브를 지원하는 restartPolicy: Always를 사용하여 특수 init 컨테이너로 사이드카를 구현했습니다.
 - 자세한 내용은 Kubernetes Documentation의 [Sidecar Containers](#)를 참조하세요.
- 엔드포인트 API 사용 중단: 엔드포인트 API는 이제 공식적으로 사용 중단되었으며 액세스 시 경고를 반환합니다. 대신 듀얼 스택 네트워킹과 같은 최신 기능을 지원하고 서비스당 여러 EndpointSlices를 처리하는 EndpointSlices API를 사용하도록 워크로드와 스크립트를 마이그레이션하세요.
 - 자세한 내용은 Kubernetes Blog의 [Kubernetes v1.33: Continuing the transition from Endpoints to EndpointSlice](#)를 참조하세요.
- Elastic Fabric Adapter 지원: 이제 Amazon EKS 클러스터의 기본 보안 그룹에서 Elastic Fabric Adapter(EFA) 트래픽을 지원합니다. 기본 보안 그룹에 동일한 보안 그룹의 대상을 가진 EFA 트래픽

을 허용하는 새로운 아웃바운드 규칙이 있습니다. 이를 통해 클러스터 내에서 EFA 트래픽이 허용됩니다.

- 자세한 내용은 Amazon EC2 사용 설명서의 [Amazon EC2의 AI/ML 및 HPC 워크로드를 위한 Elastic Fabric Adapter](#)를 참조하세요.

전체 Kubernetes 1.33 변경 로그는 <https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.33.md>를 참조하세요.

Kubernetes 1.32

Kubernetes 1.32는 이제 Amazon EKS에서 사용할 수 있습니다. Kubernetes 1.32에 대한 자세한 내용은 [공식 릴리스 발표](#)를 참조하세요.

Important

- 버전 1.32에서 FlowSchema 및 PriorityLevelConfiguration의 `flowcontrol.apiserver.k8s.io/v1beta3` API 버전이 제거되었습니다. 이 API를 사용하는 경우 업그레이드하기 전에 지원되는 최신 버전을 사용하도록 구성을 업데이트해야 합니다.
- `ServiceAccount metadata.annotations[kubernetes.io/enforce-mountable-secrets]`는 버전 1.32에서 더 이상 사용되지 않으며 향후 Kubernetes 마이너 버전 릴리스에서 제거될 예정입니다. 탑재된 보안 암호에 대한 액세스를 격리하려면 별도의 네임스페이스를 사용하는 것이 좋습니다.
- Kubernetes 버전 1.32는 Amazon EKS가 Amazon Linux 2(AL2) AMI를 출시하는 마지막 버전입니다. 버전 1.33부터 Amazon EKS는 Amazon Linux 2023(AL2023) 및 Bottlerocket 기반 AMI를 계속 출시합니다.
- Memory Manager 기능은 Kubernetes 버전 1.32의 정식 버전(GA) 상태로 전환되었습니다. 이 개선 사항은 컨테이너화된 애플리케이션에 더 효율적이고 예측 가능한 메모리 할당을 제공하며, 특히 특정 메모리 요구 사항이 있는 워크로드에 유용합니다.
- 이제 StatefulSets에서 생성한 PersistentVolumeClaims(PVC)에 자동 정리 기능이 포함됩니다. PVC가 더 이상 필요하지 않으면 StatefulSet의 업데이트 및 노드 유지 관리 작업 중에 데이터 지속성을 유지하면서 해당 PVC가 자동으로 삭제됩니다. 이 기능은 스토리지 관리를 간소화하고 클러스터에서 분리된 PVC가 생기지 않도록 하는 데 도움이 됩니다.

- 사용자 지정 리소스 필드 선택기 기능이 도입되어 개발자가 사용자 지정 리소스에 필드 선택기를 추가할 수 있습니다. 이 기능은 기본 제공 Kubernetes 객체에서 사용자 지정 리소스에 사용할 수 있는 것과 동일한 필터링 기능을 제공하므로 보다 정확하고 효율적인 리소스 필터링을 지원하고 더 나은 API 설계 사례를 촉진할 수 있습니다.

전체 Kubernetes 1.32 변경 로그는 <https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.32.md>를 참조하세요.

익명 인증 관련 변경 사항

Amazon EKS 1.32부터 익명 인증은 다음 API 서버 상태 확인 엔드포인트로 제한됩니다.

- /healthz
- /livez
- /readyz

system:unauthenticated 사용자를 사용하는 다른 엔드포인트에 대한 요청은 401 Unauthorized HTTP 응답을 수신합니다. 이러한 보안 강화는 잘못 구성된 RBAC 정책으로 인해 발생할 수 있는 의도하지 않은 클러스터 액세스를 방지하는 데 도움이 됩니다.

Note

public-info-viewer RBAC 역할은 위에 나열된 상태 확인 엔드포인트에 계속 적용됩니다.

Amazon Linux 2 AMI 사용 중단

Kubernetes 버전 1.33 이상의 경우 EKS는 사전 빌드된 최적화된 Amazon Linux 2(AL2) Amazon Machine Image(AMI)를 제공하지 않습니다.

AWS는 EKS 자율 모드를 채택하거나 Amazon Linux 2023(AL2023) 또는 Bottlerocket과 같은 최신 운영 체제로 마이그레이션할 것을 제안합니다.

- [the section called “MNG에서 마이그레이션”](#)
- [the section called “AL2023으로 업그레이드”](#)
- [the section called “Bottlerocket”](#)

Note

이 업데이트는 EKS 최적화 AL2 AMI에 적용됩니다. 운영 체제 자체에 대한 자세한 내용은 [Amazon Linux 2 FAQ](#)를 참조하세요.

Kubernetes 1.31

Kubernetes 1.31은 이제 Amazon EKS에서 사용할 수 있습니다. Kubernetes 1.31에 대한 자세한 내용은 [공식 릴리스 발표](#)를 참조하세요.

Important

- 2017년 이후 더 이상 사용되지 않는 `--keep-terminated-pod-volumes` 플래그 kubelet는 버전 1.31 릴리스의 일환으로 제거되었습니다. 이 변경 사항은 종료된 포드 볼륨이 kubelet에서 처리되는 방식에 영향을 미칩니다. 노드 구성에서 이 플래그를 사용하는 경우 업그레이드 전에 부트스트랩 스크립트와 시작 템플릿을 업데이트하여 제거해야 합니다.
- 베타 VolumeAttributesClass 기능 게이트 및 API 리소스는 Amazon EKS 버전 1.31에서 활성화됩니다. 이 기능을 통해 클러스터 운영자는 Amazon EBS CSI 드라이버를 비롯하여 호환 가능한 CSI 드라이버로 관리되는 영구 볼륨(PV)의 변경 가능한 속성을 수정할 수 있습니다. 이 기능을 활용하려면 CSI 드라이버가 해당 VolumeAttributesClass 기능을 지원하는지 확인하세요 (Amazon EBS CSI 드라이버의 경우, 1.35.0 이상으로 업그레이드하여 기능을 자동으로 활성화). VolumeAttributesClass 객체를 생성하여 볼륨 유형 및 처리량과 같은 원하는 볼륨 속성을 정의하고 이를 영구 볼륨 클레임(PVC)과 연결할 수 있습니다. 자세한 내용은 [공식 Kubernetes 설명서](#) 및 CSI 드라이버 설명서를 참조하세요.
- Amazon EBS CSI 드라이버에 대한 자세한 내용은 [the section called “Amazon EBS”](#)을 참조하세요.
- [AppArmor](#)에 대한 Kubernetes의 지원이 안정화되었으며 이제 일반 대중에게 공개되었습니다. 이 기능을 사용하면 컨테이너의 securityContext에 appArmorProfile.type 필드를 설정하여 AppArmor로 컨테이너를 보호할 수 있습니다. Kubernetes 버전 1.30 이전에는 주석으로 AppArmor를 제어했습니다. 버전 1.30부터 필드를 사용하여 제어합니다. 이 기능을 활용하려면 주석에서 벗어나 appArmorProfile.type 필드를 사용하여 워크로드가 호환되는지 확인하는 것이 좋습니다.
- PersistentVolume 마지막 단계 트랜지션 시간 기능이 안정화되었으며, 이제 Kubernetes 버전 1.31에서 일반 공개로 사용할 수 있게 되었습니다. 이 기능은 PersistentVolume이 다른 단

계로 마지막으로 전환된 시점의 타임스탬프를 제공하는 PersistentVolumeStatus의 새로운 `.status.lastTransitionTime` 필드를 도입합니다. 이 개선 사항을 통해 특히 볼륨의 수명 주기를 이해하는 것이 중요한 시나리오에서 PersistentVolumes을 더 잘 추적하고 관리할 수 있습니다.

전체 Kubernetes 1.31 변경 로그는 <https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.31.md>를 참조하세요.

Kubernetes 1.30

Kubernetes 1.30은 이제 Amazon EKS에서 사용할 수 있습니다. Kubernetes 1.30에 대한 자세한 내용은 [공식 릴리스 발표](#)를 참조하세요.

Important

- Amazon EKS 버전 1.30 이상부터 새로 생성되는 모든 관리형 노드 그룹은 자동으로 Amazon Linux 2023(AL2023)을 기본 노드 운영 체제로 사용합니다. 이전에는 새 노드 그룹이 Amazon Linux 2(AL2)를 기본 노드 운영 체제로 사용했습니다. 새 노드 그룹을 생성할 때 AL2를 AMI 유형으로 선택하여 계속 사용할 수 있습니다.
 - AL2에서 AL2023로의 마이그레이션에 대한 자세한 내용은 [the section called “AL2023으로 업그레이드”](#) 섹션을 참조하세요.
 - Amazon Linux에 대한 자세한 정보는 [AL2 및 AL 2023 비교](#)를 참조하세요.
 - 관리형 노드 그룹의 운영 체제 지정에 대한 자세한 내용은 [the section called “생성”](#) 섹션을 참조하세요.
- Amazon EKS 1.30을 사용하면 `topology.k8s.aws/zone-id` 레이블이 작업자 노드에 추가됩니다. 가용 영역 ID(AZ ID)를 사용하면 다른 계정의 리소스를 기준으로 한 계정의 리소스 위치를 확인할 수 있습니다. 자세한 내용은 AWS IAM 사용 설명서의 [AWS 리소스에 대한 가용 영역 ID\(AZ ID\)](#)를 참조하세요.
 - 1.30부터 Amazon EKS는 새로 생성된 클러스터에 적용된 gp2 StorageClass 리소스에 대한 default 주석을 더 이상 포함하지 않습니다. 이 스토리지 클래스를 이름으로 참조하는 경우에는 영향을 주지 않습니다. 클러스터에서 기본값 StorageClass 사용에 의존하고 있다면 조치를 취해야 합니다. 이름 gp2를 기준으로 StorageClass를 참조해야 합니다. 아니면 버전 1.31.0 설치 시 또는 나중에 `aws-ebs-csi-driver` add-on을 설치할 때 `defaultStorageClass.enabled` 파라미터를 true로 설정하여 Amazon EBS 권장 기본 스토리지 클래스를 배포할 수 있습니다.

- Amazon EKS 클러스터 IAM 역할에 필요한 최소 IAM 정책이 변경되었습니다.
ec2:DescribeAvailabilityZones 작업은 필수입니다. 자세한 내용은 [the section called “클러스터 IAM 역할”](#) 섹션을 참조하세요.

전체 Kubernetes 1.30 변경 로그는 <https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.30.md>을 참조하세요.

확장 지원의 Kubernetes 버전에 대한 릴리스 노트 검토

Amazon EKS는 Amazon EKS에서 릴리스된 시점으로부터 14개월 동안 Kubernetes 마이너 버전에 대한 표준 지원과 추가 12개월 동안 Kubernetes 마이너 버전에 대한 추가 지원(버전당 총 26개월)을 통해 업스트림에서 지원되는 것보다 긴 Kubernetes 버전을 지원합니다.

이 주제에서는 추가 지원의 각 Kubernetes 버전에 대해 알아야 할 중요한 변경 사항을 제공합니다. 업그레이드할 때는 클러스터의 이전 버전과 새 버전 간에 발생한 변경 사항을 주의 깊게 검토하세요.

Kubernetes 1.29

Kubernetes 1.29는 이제 Amazon EKS에서 사용할 수 있습니다. Kubernetes 1.29에 대한 자세한 내용은 [공식 릴리스 발표](#)를 참조하세요.

Important

- FlowSchema 및 PriorityLevelConfiguration의 사용 중단된
flowcontrol.apiserver.k8s.io/v1beta2 API 버전은 Kubernetes 버전 1.29에서 더 이상 제공되지 않습니다. 사용 중단된 베타 API 그룹을 사용하는 매니페스트 또는 클라이언트 소프트웨어가 있는 경우 버전 1.29로 업그레이드하기 전에 이를 변경해야 합니다.
- Node 객체의 .status.kubeProxyVersion 필드는 이제 사용 중단되었으며 Kubernetes 프로젝트는 향후 릴리스에서 해당 필드를 제거할 것을 제안하고 있습니다. 사용 중단된 필드는 정확하지 않으며 kube-proxy 버전을 실제로 알지 못하거나 kube-proxy가 실행 중인지 여부조차 모르는 kubelet에 의해 관리되었습니다. 클라이언트 소프트웨어에서 이 필드를 사용했다면 중지하세요. 정보를 신뢰할 수 없으며 해당 필드는 이제 사용 중단되었습니다.
- 잠재적인 공격 표면을 줄이기 위해 Kubernetes 1.29에서 LegacyServiceAccountTokenCleanUp 기능은 레거시 자동 생성 비밀 기반 토큰이 오랫동안(기본값으로 1년) 사용되지 않으면 유효하지 않은 것으로 레이블을 지정하고, 유효하지 않은 것으로 표

시된 후 오랫동안(기본값으로 추가 1년) 사용을 시도하지 않으면 자동으로 제거합니다. 이러한 토큰을 식별하려면 다음을 실행합니다.

```
kubectl get cm kube-apiserver-legacy-service-account-token-tracking -n kube-system
```

전체 Kubernetes 1.29 변경 로그는 <https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.29.md#changelog-since-v1280>을 참조하세요.

Kubernetes 1.28

Kubernetes 1.28은 이제 Amazon EKS에서 사용할 수 있습니다. Kubernetes 1.28에 대한 자세한 내용은 [공식 릴리스 발표](#)를 참조하세요.

- Kubernetes v1.28은 코어 노드와 컨트롤 플레인 구성 요소 간에 지원되는 스큐를 n-2에서 n-3으로 하나의 마이너 버전으로 확장하여 지원되는 가장 오래된 마이너 버전의 노드 구성 요소(kubelet 및 kube-proxy)가 지원되는 최신 마이너 버전의 컨트롤 플레인 구성 요소(kube-apiserver, kube-scheduler, kube-controller-manager, cloud-controller-manager)와 함께 작동할 수 있도록 했습니다.
- Pod GC Controller의 지표 `force_delete_pods_total` 및 `force_delete_pod_errors_total`은 모든 강제 포드 삭제를 처리하도록 향상되었습니다. 포드가 종료됐거나, 끊겼거나, 서비스 중단 테인트로 종료되었거나, 종료 및 예정되지 않았기 때문에 포드가 강제 삭제됐는지 여부를 나타내기 위해 지표에 이유가 추가됩니다.
- `storageClassName`이 설정되지 않은 바인딩되지 않은 `PersistentVolumeClaim`에 기본 `StorageClass`를 자동으로 할당하도록 `PersistentVolume (PV)` 컨트롤러가 수정되었습니다. 또한 API 서버 내의 `PersistentVolumeClaim` 승인 검증 메커니즘이 설정되지 않은 상태에서 실제 `StorageClass` 이름으로 값을 변경할 수 있도록 조정되었습니다.

전체 Kubernetes 1.28 변경 로그는 <https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.28.md#changelog-since-v1270>을 참조하세요.

Kubernetes 1.27

Kubernetes 1.27은 이제 Amazon EKS에서 사용할 수 있습니다. Kubernetes 1.27에 대한 자세한 내용은 [공식 릴리스 발표](#)를 참조하세요.

⚠ Important

- 알파 seccomp 주석 `seccomp.security.alpha.kubernetes.io/pod` 및 `container.seccomp.security.alpha.kubernetes.io` 주석에 대한 지원이 제거되었습니다. 알파 seccomp 주석은 1.19에서 더 이상 사용되지 않으며 1.27에서 제거됨에 따라 seccomp 주석이 있는 Pods에 대해 seccomp 필드가 더 이상 자동으로 채워지지 않습니다. 대신 Pods 또는 컨테이너에 `securityContext.seccompProfile` 필드를 사용하여 seccomp 프로필을 구성합니다. 클러스터에서 더 이상 사용되지 않는 알파 seccomp 주석을 사용하고 있는지 확인하려면 다음 명령을 실행합니다.

```
kubectl get pods --all-namespaces -o json | grep
-E 'seccomp.security.alpha.kubernetes.io/pod|
container.seccomp.security.alpha.kubernetes.io'
```

- kubelet에 대한 `--container-runtime` 명령줄 인수가 제거되었습니다. Amazon EKS의 기본 컨테이너 런타임은 1.24부터 containerd가 되어 컨테이너 런타임을 지정할 필요가 없습니다. 1.27부터 Amazon EKS는 부트스트랩 스크립트에 전달된 `--container-runtime` 인수를 무시합니다. 노드 부트스트랩 프로세스 중 오류를 방지하기 위해 이 인수를 `--kubelet-extra-args`에 전달하지 않는 것이 중요합니다. 모든 노드 생성 워크플로 및 빌드 스크립트에서 `--container-runtime` 인수를 제거해야 합니다.
- Kubernetes 1.27의 kubelet은 기본 kubeAPIQPS를 50으로, kubeAPIBurst를 100으로 늘렸습니다. 이러한 향상된 기능을 통해 kubelet은 더 많은 양의 API 쿼리를 처리하여 응답 시간과 성능을 개선할 수 있습니다. 조정 요구 사항으로 인해 Pods에 대한 수요가 증가하면 수정된 기본값을 통해 kubelet이 증가한 워크로드를 효율적으로 관리할 수 있습니다. 결과적으로 Pod 실행이 더 빨라지고 클러스터 작업이 더 효과적입니다.
- 보다 세분화된 Pod 토폴로지를 사용하여 minDomain과 같은 정책을 분산할 수 있습니다. 이 파라미터는 Pods가 분산되어야 하는 최소 도메인 수를 지정할 수 있는 기능을 제공합니다. `nodeAffinityPolicy` 및 `nodeTaintPolicy`를 사용하면 더 세부적으로 Pod 배포를 관리할 수 있습니다. 이는 노드 친화도, 테인트 및 Pod's 사양의 `topologySpreadConstraints`에 있는 `matchLabelKeys` 필드에 따른 것입니다. 이를 통해 롤링 업그레이드 후 계산을 분산하기 위해 Pods를 선택할 수 있습니다.
- Kubernetes 1.27은 `PersistentVolumeClaims(PVCs)`의 수명을 제어하는 `StatefulSets`에 대한 새로운 정책 메커니즘을 베타 버전으로 승격했습니다. 새로운 PVC 보존 정책을 사용하

면 StatefulSet가 삭제되거나 StatefulSet의 복제본이 스케일 다운될 때 StatefulSet 사양 템플릿에서 생성된 PVCs를 자동으로 삭제할지 아니면 유지할지 지정할 수 있습니다.

- Kubernetes API 서버의 [goaway-chance](#) 옵션은 연결을 무작위로 종료하여 HTTP/2 클라이언트 연결이 단일 API 서버 인스턴스에서 중단되는 것을 방지합니다. 연결이 닫히면 클라이언트가 다시 연결을 시도하고 로드 밸런싱의 결과로 다른 API 서버에 도달할 가능성이 높습니다. Amazon EKS 버전 1.27은 goaway-chance 플래그를 활성화했습니다. Amazon EKS 클러스터에서 실행 중인 워크로드가 [HTTP GOAWAY](#)와 호환되지 않는 클라이언트를 사용하는 경우, 연결 종료 시 다시 연결하여 GOAWAY를 처리하도록 클라이언트를 업데이트하는 것이 좋습니다.

전체 Kubernetes 1.27 변경 로그는 <https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.27.md#changelog-since-v1260>을 참조하세요.

Kubernetes 1.26

Kubernetes 1.26은 이제 Amazon EKS에서 사용할 수 있습니다. Kubernetes 1.26에 대한 자세한 내용은 [공식 릴리스 발표](#)를 참조하세요.

Important

Kubernetes 1.26은 더 이상 CRI v1alpha2를 지원하지 않습니다. 이로 인해 컨테이너 런타임이 CRI v1을 지원하지 않는 경우 kubelet이 더 이상 노드를 등록하지 않습니다. 또한 이는 Kubernetes 1.26이 containerd 마이너 버전 1.5 이하를 지원하지 않음을 의미합니다. containerd를 사용하는 경우 노드를 Kubernetes 1.26으로 업그레이드하기 전에 containerd 버전 1.6.0 이상으로 업그레이드해야 합니다. v1alpha2만 지원하는 다른 컨테이너 런타임도 업그레이드해야 합니다. 자세한 내용은 컨테이너 런타임 공급업체에 문의하세요. 기본적으로 Amazon Linux와 Bottlerocket AMI에는 containerd 버전 1.6.6이 포함되어 있습니다.

- Kubernetes 1.26으로 업그레이드하기 전에 Kubernetes용 Amazon VPC CNI 플러그인을 버전 1.12 이상으로 업그레이드합니다. Kubernetes용 Amazon VPC CNI 플러그인 1.12 이상으로 업그레이드하지 않으면 Kubernetes용 Amazon VPC CNI 플러그인이 충돌합니다. 자세한 내용은 [the section called “Amazon VPC CNI”](#) 섹션을 참조하세요.
- Kubernetes API 서버의 [goaway-chance](#) 옵션은 연결을 무작위로 종료하여 HTTP/2 클라이언트 연결이 단일 API 서버 인스턴스에서 중단되는 것을 방지합니다. 연결이 닫히면 클라이언트가 다시 연결을 시도하고 로드 밸런싱의 결과로 다른 API 서버에 도달할 가능성이 높습니다. Amazon EKS 버전 1.26은 goaway-chance 플래그를 활성화했습니다. Amazon EKS 클러스터에서 실행 중인 워크

로드가 [HTTP GOAWAY](#)와 호환되지 않는 클라이언트를 사용하는 경우, 연결 종료 시 다시 연결하여 GOAWAY를 처리하도록 클라이언트를 업데이트하는 것이 좋습니다.

전체 Kubernetes 1.26 변경 로그는 <https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG/CHANGELOG-1.26.md#changelog-since-v1250>을 참조하세요.

현재 클러스터 지원 기간 보기

AWS 콘솔의 클러스터 지원 기간 섹션에는 클러스터가 현재 표준 지원 중인지 추가 지원 중인지가 표시됩니다. 클러스터 지원 기간이 추가 지원인 경우 EKS 추가 지원 요금이 부과됩니다.

표준 및 추가 지원에 대한 자세한 내용은 [the section called “Kubernetes 버전”](#) 섹션을 참조하세요.

1. AWS 콘솔의 EKS 섹션에서 클러스터 페이지로 이동합니다. 콘솔이 검토하려는 클러스터와 동일한 AWS 리전으로 설정되어 있는지 확인합니다.
2. 지원 기간 열을 검토합니다. 값이 표준 지원까지...인 경우 현재 추가 지원 요금이 부과되지 않습니다. 표준 지원 기간 내에 있습니다. 값이 추가 지원...인 경우 현재 이 클러스터에는 추가 지원 요금이 부과됩니다.

Note

AWS API 또는 CLI로 지원 기간을 검색할 수 없습니다.

현재 클러스터 업그레이드 정책 보기

클러스터 업그레이드 정책은 표준 지원 기간이 종료될 때 클러스터가 어떻게 되는지 결정합니다. 업그레이드 정책이 EXTENDED인 경우 클러스터는 자동으로 업그레이드되지 않으며 추가 지원을 시작합니다. 업그레이드 정책이 STANDARD인 경우 자동으로 업그레이드됩니다.

Kubernetes 버전 정책에 대한 Amazon EKS 제어를 사용하면 EKS 클러스터에 대한 표준 지원 동작의 종료를 선택할 수 있습니다. 이러한 제어 기능을 사용하면 확장 지원을 시작해야 하는 클러스터와 Kubernetes 버전에 대한 표준 지원이 끝나면 자동으로 업그레이드해야 하는 클러스터를 결정할 수 있습니다.

마이너 버전은 릴리스 후 처음 14개월 동안 Amazon EKS에서 표준 지원을 받습니다. 버전 표준 지원 종료일이 지나면 다음 12개월 동안의 추가 지원이 시작됩니다. 확장 지원을 통해 클러스터 시간당 추가

비용을 지불하고 특정 Kubernetes 버전을 더 오래 사용할 수 있습니다. EKS 클러스터에 대한 추가 지원을 활성화하거나 비활성화할 수 있습니다. 추가 지원을 비활성화하면 AWS는 표준 지원이 종료될 때 자동으로 클러스터를 다음 버전으로 업그레이드합니다. 추가 지원을 활성화하면 제한된 기간 동안 추가 비용으로 현재 버전을 계속 사용할 수 있습니다. 확장 지원을 사용하는 경우에도 Kubernetes 클러스터를 정기적으로 업그레이드할 계획을 세우세요.

supportType 속성을 사용하여 새 클러스터와 기존 클러스터 모두에 대한 버전 정책을 설정할 수 있습니다. 버전 지원 정책을 설정하는 데 사용할 수 있는 두 가지 옵션이 있습니다.

- **STANDARD** - EKS 클러스터는 표준 지원 종료 시 자동 업그레이드가 가능합니다. 이 설정을 사용하면 확장 지원 요금이 발생하지 않지만 EKS 클러스터는 표준 지원에서 다음 지원 Kubernetes 버전으로 자동 업그레이드됩니다.
- **EXTENDED** - Kubernetes 버전의 표준 지원이 종료되면 EKS 클러스터가 확장 지원을 시작합니다. 이 설정을 사용하면 추가 지원 요금이 부과됩니다. 클러스터를 표준 지원 Kubernetes 버전으로 업그레이드하여 확장 지원 요금이 발생하지 않도록 할 수 있습니다. 추가 지원으로 실행되는 클러스터는 추가 지원 종료 시 자동 업그레이드를 받을 수 있습니다.

새 클러스터와 기존 클러스터에는 기본적으로 추가 지원이 활성화되어 있습니다. AWS Management Console의 클러스터에 대해 추가 지원이 활성화되었는지 확인하거나 AWS CLI를 사용하여 확인할 수 있습니다.

Important

확장 지원 기간을 활용하기 위해 클러스터를 현재 Kubernetes 버전으로 유지하려면 표준 지원 기간이 종료되기 전에 확장 지원 업그레이드 정책을 활성화해야 합니다.

표준 지원의 Kubernetes 버전에서 실행되는 동안에만 클러스터에 대한 버전 지원 정책을 설정할 수 있습니다. 버전이 추가 지원에 들어가면 표준 지원 버전을 실행할 때까지 이 설정을 변경할 수 없습니다.

예를 들어, 버전 지원 정책을 standard로 설정한 경우 클러스터에서 실행 중인 Kubernetes 버전의 표준 지원이 종료된 후에는 이 설정을 변경할 수 없습니다. 버전 지원 정책을 extended로 설정한 경우 클러스터에서 실행 중인 Kubernetes 버전의 표준 지원이 종료된 후에는 이 설정을 변경할 수 없습니다. 버전 지원 정책 설정을 변경하려면 클러스터가 표준 지원 Kubernetes 버전에서 실행되고 있어야 합니다.

클러스터 업그레이드 정책 보기(AWS 콘솔)

1. AWS 콘솔의 EKS 섹션에서 클러스터 페이지로 이동합니다. 콘솔이 검토하려는 클러스터와 동일한 AWS 리전으로 설정되어 있는지 확인합니다.
2. 업그레이드 정책 열을 검토합니다. 값이 표준 지원인 경우 클러스터는 추가 지원을 시작하지 않습니다. 값이 추가 지원인 경우 클러스터는 추가 지원을 시작합니다.

클러스터 업그레이드 정책 보기(AWS CLI)

1. AWS CLI가 설치되어 있고 로그인했는지 확인합니다. [AWS CLI를 업데이트하고 설치하는 방법을 알아봅니다.](#)
2. EKS 클러스터의 이름을 확인합니다. CLI를 EKS 클러스터와 동일한 AWS 리전으로 설정합니다.
3. 다음 명령 실행:

```
aws eks describe-cluster \
--name <cluster-name> \
--query "cluster.upgradePolicy.supportType"
```

4. 값이 STANDARD인 경우 클러스터는 추가 지원을 시작하지 않습니다. 값이 EXTENDED인 경우 클러스터는 추가 지원을 시작합니다.

EKS 추가 지원을 활성화하여 Kubernetes 버전 업그레이드 계획에 유연성 더하기

이 항목에서는 추가 지원을 활성화하기 위해 EKS 클러스터의 업그레이드 정책을 설정하는 방법을 설명합니다. EKS 클러스터의 업그레이드 정책은 클러스터의 표준 지원 기간이 끝나면 어떻게 되는지 결정합니다. 클러스터 업그레이드 정책에서 추가 지원을 활성화한 경우 표준 지원 기간이 끝나면 추가 지원 기간이 시작됩니다. 표준 지원 기간이 종료되어도 클러스터는 자동으로 업그레이드되지 않습니다.

실제로 지원 기간이 추가된 클러스터에서는 더 높은 비용이 발생합니다. 클러스터에 추가 지원을 활성화하도록 업그레이드 정책만 설정되어 있고 그렇지 않은 경우에는 표준 지원 기간에 있는 경우 표준 비용이 발생합니다.

AWS 콘솔에서 클러스터를 생성하는 경우 업그레이드 정책이 설정되어 확장 지원을 비활성화합니다. 다른 방식으로 클러스터를 생성하는 경우 확장 지원을 활성화하도록 업그레이드 정책이 설정됩니다. 예를 들어 AWS API로 생성된 클러스터는 확장 지원을 활성화했습니다.

업그레이드 정책에 대한 자세한 내용은 [the section called “업그레이드 정책 보기”](#) 섹션을 참조하세요

⚠ Important

확장 지원 기간을 활용하기 위해 클러스터를 현재 Kubernetes 버전으로 유지하려면 표준 지원 기간이 종료되기 전에 확장 지원 업그레이드 정책을 활성화해야 합니다.
추가 지원을 활성화하지 않은 경우 클러스터가 자동으로 업그레이드됩니다.

EKS 추가 지원 활성화(AWS 콘솔)

1. AWS 콘솔에서 EKS 클러스터로 이동합니다. 클러스터 정보 페이지에서 개요 탭을 선택합니다.
2. Kubernetes 버전 설정 섹션에서 관리를 선택합니다.
3. 추가 지원을 선택한 다음 변경 사항 저장을 선택합니다.

EKS 추가 지원 활성화(AWS CLI)

1. AWS CLI가 설치되어 있고 로그인했는지 확인합니다. [AWS CLI를 업데이트하고 설치하는 방법을 알아봅니다.](#)
2. EKS 클러스터의 이름을 확인합니다.
3. 다음 명령 실행:

```
aws eks update-cluster-config \
  --name <cluster-name> \
  --upgrade-policy supportType=EXTENDED
```

EKS 추가 지원을 비활성화하여 클러스터 비용 증가 방지

이 항목에서는 추가 지원을 비활성화하기 위해 EKS 클러스터의 업그레이드 정책을 설정하는 방법을 설명합니다. EKS 클러스터의 업그레이드 정책은 클러스터의 표준 지원 기간이 끝나면 어떻게 되는지 결정합니다. 클러스터 업그레이드 정책에서 확장 지원이 비활성화된 경우 자동으로 다음 Kubernetes 버전으로 업그레이드됩니다.

업그레이드 정책에 대한 자세한 내용은 [the section called “업그레이드 정책 보기”](#) 섹션을 참조하세요

⚠ Important

클러스터에서 추가 지원이 시작된 후에는 추가 지원을 비활성화할 수 없습니다. 표준 지원에서만 클러스터에 대한 추가 지원을 비활성화할 수 있습니다.

AWS에서는 표준 지원 기간에 클러스터를 버전으로 업그레이드할 것을 권장합니다.

EKS 추가 지원 비활성화(AWS 콘솔)

1. AWS 콘솔에서 EKS 클러스터로 이동합니다. 클러스터 정보 페이지에서 개요 탭을 선택합니다.
2. Kubernetes 버전 설정 섹션에서 관리를 선택합니다.
3. 표준 지원을 선택한 다음 변경 사항 저장을 선택합니다.

EKS 추가 지원 비활성화(AWS CLI)

1. AWS CLI가 설치되어 있고 로그인했는지 확인합니다. [AWS CLI를 업데이트하고 설치하는 방법을 알아봅니다.](#)
2. EKS 클러스터의 이름을 확인합니다.
3. 다음 명령 실행:

```
aws eks update-cluster-config \
--name <cluster-name> \
--upgrade-policy supportType=STANDARD
```

각 Kubernetes 버전에 대한 Amazon EKS 플랫폼 버전 보기

Amazon EKS 플랫폼 버전은 현재 Kubernetes 패치 버전뿐만 아니라 활성화된 Kubernetes API 서버 플러그 등, Amazon EKS 클러스터 제어 영역의 기능을 나타냅니다. 각 Kubernetes 마이너 버전에는 하나 이상의 연결된 Amazon EKS 플랫폼 버전이 있습니다. 서로 다른 Kubernetes 마이너 버전에 대한 플랫폼 버전은 독립적입니다. AWS CLI 또는 AWS Management Console을 사용하여 [클러스터의 현재 플랫폼 버전을 검색](#)할 수 있습니다. AWS Outposts에 로컬 클러스터가 있는 경우 이 주제 대신 [the section called “EKS 플랫폼 버전”](#)을 참조하세요.

Amazon EKS에서 새 Kubernetes 마이너 버전(예: 1.32)을 사용할 수 있는 경우 해당 Kubernetes 마이너 버전에 대한 초기 Amazon EKS 플랫폼 버전은 eks.1에서 시작합니다. 그러나 Amazon EKS에서는 새 Kubernetes 제어 영역 설정을 활성화하고 보안 수정 사항을 제공하기 위해 새 플랫폼 버전을 주기적으로 릴리스합니다.

마이너 버전에 대해 새 Amazon EKS 플랫폼 버전을 사용할 수 있게 된 경우 다음과 같은 사항이 발생합니다.

- Amazon EKS 플랫폼 버전 번호가 증가합니다(eks.<n+1>).
- Amazon EKS에서 기존의 모든 클러스터를 해당하는 Kubernetes 마이너 버전에 대한 최신 Amazon EKS 플랫폼 버전으로 자동으로 업그레이드합니다. 기존 Amazon EKS 플랫폼 버전의 자동 업그레이드는 증분 방식으로 돌아옵니다. 이 롤아웃 프로세스는 다소 시간이 걸릴 수 있습니다. 최신 Amazon EKS 플랫폼 버전 기능이 즉시 필요한 경우 새 Amazon EKS 클러스터를 생성해야 합니다.

클러스터가 현재 플랫폼 버전보다 2 이상의 플랫폼 버전인 경우 Amazon EKS가 클러스터를 자동으로 업데이트하지 못했을 수 있습니다. 이 문제의 원인에 대한 자세한 내용은 [the section called “Amazon EKS 플랫폼 버전이 현재 플랫폼 버전보다 두 버전 이상 뒤떨어져 있음”](#)을 참조하세요.

- Amazon EKS에서는 해당 패치 버전으로 새 AMI를 게시할 수 있습니다. 그러나 모든 패치 버전은 지정된 Kubernetes 마이너 버전에 대한 노드 AMI와 EKS 제어 영역 간에 호환됩니다.

새 Amazon EKS 플랫폼 버전은 주요 변경 사항을 도입하거나 서비스 종단을 초래하지 않습니다.

클러스터는 항상 지정된 Kubernetes 버전에 사용 가능한 최신 Amazon EKS 플랫폼 버전(eks.<n>)으로 생성됩니다. 클러스터를 새 Kubernetes 마이너 버전으로 업데이트하면 클러스터가 업데이트한 Kubernetes 마이너 버전에 대한 현재 Amazon EKS 플랫폼 버전을 수신합니다.

현재 및 최근 Amazon EKS 플랫폼 버전은 다음 표에 설명되어 있습니다.

Note

AWS는 최근 2024년 6월에 게시된 일부 플랫폼 버전을 비활성화했습니다. 플랫폼 버전에 안정성 문제가 있었습니다. 별도의 작업은 필요 없습니다.

이 특정 설명서 페이지의 모든 소스 파일 변경 사항에 대한 알림을 받으려면 RSS 리더를 사용하여 다음 URL을 구독하세요.

<https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/clusters/platform-versions.adoc.atom>

Kubernetes 버전 1.33

다음 승인 컨트롤러는 모든 1.33 플랫폼 버전(NodeRestriction, ExtendedResourceToleration, NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, PodSecurity, Priority, DefaultTolerationSeconds, DefaultStorageClass, StorageObjectInUseProtection,

PersistentVolumeClaimResize, RuntimeClass, CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota)에 대해 활성화됩니다.

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.33.1	eks.5	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 6월 11일
1.33.1	eks.4	EKS에 대한 Kubernetes 버전 1.33의 최초 릴리스. 자세한 내용은 the section called “Kubernetes 1.33” 섹션을 참조하세요.	2025년 5월 28일

Kubernetes 버전 1.32

다음 승인 컨트롤러는 모든 1.32 플랫폼 버전(NodeRestriction, ExtendedResourceToleration, NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, PodSecurity, Priority, DefaultTolerationSeconds, DefaultStorageClass, StorageObjectInUseProtection, PersistentVolumeClaimResize, RuntimeClass, CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota)에 대해 활성화됩니다.

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.32.5	eks.12	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 6월 11일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.32.5	eks.11	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 5월 30일
1.32.3	eks.10	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 5월 16일
1.32.3	eks.9	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 29일
1.32.3	eks.8	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 18일
1.32.3	eks.7	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 18일
1.32.3	eks.6	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 2일
1.32.2	eks.5	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 3월 17일
1.32.2	eks.4	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 3월 4일
1.32.1	eks.3	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 2월 24일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.32.0	eks.2	EKS에 대한 Kubernetes 버전 1.32의 최초 릴리스. 자세한 내용은 the section called “Kubernetes 1.32” 섹션을 참조하세요.	2025년 1월

Kubernetes 버전 1.31

다음 승인 컨트롤러는 모든 1.31 플랫폼 버전(NodeRestriction, ExtendedResourceToleration, NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, PodSecurity, Priority, DefaultTolerationSeconds, DefaultStorageClass, StorageObjectInUseProtection, PersistentVolumeClaimResize, RuntimeClass, CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota, ObjectCount)에 대해 활성화됩니다.

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.31.9	eks.28	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 6월 11일
1.31.9	eks.27	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 5월 30일
1.31.7	eks.26	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 5월 16일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.31.7	eks.25	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 29일
1.31.7	eks.24	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 18일
1.31.7	eks.23	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 18일
1.31.7	eks.22	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 2일
1.31.6	eks.21	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 3월 17일
1.31.6	eks.20	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 3월 4일
1.31.5	eks.19	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 2월 24일
1.31.5	eks.18	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 2월 24일
1.31.4	eks.17	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 1월 17일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.31.2	eks.12	새 플랫폼 버전은 Amazon EKS Hybrid Nodes 지원이 포함되고 컨트롤 플레인 관찰성이 개선되었습니다. the section called “하이브리드 노드” 와 Amazon EKS의 성능 관찰성 개선 섹션을 각각 참조하세요.	2024년 11월 15일
1.31.1	eks.6	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 10월 21일
1.31.0	eks.4	EKS에 대한 Kubernetes 버전 1.31의 최초 릴리스. 자세한 내용은 the section called “Kubernetes 1.31” 섹션을 참조하세요.	2024년 9월 26일

Kubernetes 버전 1.30

다음 승인 컨트롤러는 모든 1.30 플랫폼 버전(NodeRestriction, ExtendedResourceToleration, NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, PodSecurity, Priority, DefaultTolerationSeconds, DefaultStorageClass, StorageObjectInUseProtection, PersistentVolumeClaimResize, RuntimeClass, CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota)에 대해 활성화됩니다.

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.30.13	eks.36	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 6월 11일
1.30.13	eks.35	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 5월 30일
1.30.11	eks.34	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 5월 16일
1.30.11	eks.33	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 29일
1.30.11	eks.32	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 18일
1.30.11	eks.31	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 18일
1.30.11	eks.30	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 2일
1.30.10	eks.29	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 3월 17일
1.30.10	eks.28	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 3월 4일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.30.9	eks.27	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 2월 24일
1.30.8	eks.25	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 1월 17일
1.30.8	eks.24	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 1월 3일
1.30.7	eks.23	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 12월 13일
1.30.6	eks.22	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 12월 13일
1.30.6	eks.21	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 12월 13일
1.30.6	eks.20	새 플랫폼 버전은 Amazon EKS Hybrid Nodes 지원이 포함되고 컨트롤 플레인 관찰성이 개선되었습니다. the section called “하이브리드 노드” 와 Amazon EKS의 성능 관찰성 개선 섹션을 각각 참조하세요.	2024년 11월 15일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.30.5	eks.12	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 10월 21일
1.30.4	eks.8	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 9월 3일
1.30.3	eks.7	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 8월 28일
1.30.3	eks.6	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 8월 9일
1.30.2	eks.5	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 7월 2일
1.30.0	eks.2	EKS에 대한 Kubernetes 버전 1.30의 최초 릴리스. 자세한 내용은 the section called “Kubernetes 1.30” 섹션을 참조하세요.	2024년 5월 23일

Kubernetes 버전 1.29

다음 승인 컨트롤러는 모든 1.29 플랫폼 버전(NodeRestriction, ExtendedResourceToleration, NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, PodSecurity, Priority, DefaultTolerationSeconds, DefaultStorageClass, StorageObjectInUseProtection, PersistentVolumeClaimResize, RuntimeClass, CertificateApproval,

CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota)에 대해 활성화됩니다.

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.29.15	eks.39	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 6월 11일
1.29.15	eks.38	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 5월 30일
1.29.15	eks.37	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 5월 16일
1.29.15	eks.36	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 29일
1.29.15	eks.35	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 18일
1.29.15	eks.34	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 18일
1.29.15	eks.33	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 2일
1.29.14	eks.32	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 3월 17일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.29.14	eks.31	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 3월 4일
1.29.13	eks.30	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 2월 24일
1.29.13	eks.29	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 2월 24일
1.29.12	eks.28	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 1월 20일
1.29.12	eks.27	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 1월 3일
1.29.11	eks.26	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 12월 13일
1.29.10	eks.25	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 12월 13일
1.29.10	eks.24	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 12월 13일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.29.10	eks.23	새 플랫폼 버전은 Amazon EKS Hybrid Nodes 지원이 포함되고 컨트롤 플레인 관찰성이 개선되었습니다. the section called “하이브리드 노드” 와 Amazon EKS의 성능 관찰성 개선 섹션을 각각 참조하세요.	2024년 11월 15일
1.29.9	eks.17	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 10월 21일
1.29.8	eks.13	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 9월 3일
1.29.7	eks.12	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 8월 28일
1.29.7	eks.11	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 8월 9일
1.29.6	eks.10	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 7월 2일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.29.4	eks.7	CoreDNS Auto Scaling, 보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전. CoreDNS Auto Scaling에 대한 자세한 내용은 the section called “높은 트래픽을 위해 규모 조정” 섹션을 참조하세요.	2024년 5월 16일
1.29.3	eks.6	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 4월 18일
1.29.1	eks.5	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 3월 29일
1.29.1	eks.4	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 3월 20일
1.29.1	eks.3	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 3월 12일
1.29.0	eks.1	EKS에 대한 Kubernetes 버전 1.29의 최초 릴리스. 자세한 내용은 the section called “Kubernetes 1.29” 섹션을 참조하세요.	2024년 1월 23일

Kubernetes 버전 1.28

다음 승인 컨트롤러는 모든 1.28 플랫폼 버전(NodeRestriction, ExtendedResourceToleration, NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, PodSecurity, Priority, DefaultTolerationSeconds, DefaultStorageClass, StorageObjectInUseProtection, PersistentVolumeClaimResize, RuntimeClass, CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota)에 대해 활성화됩니다.

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.28.15	eks.45	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 6월 11일
1.28.15	eks.44	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 5월 30일
1.28.15	eks.43	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 5월 16일
1.28.15	eks.42	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 29일
1.28.15	eks.41	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 18일
1.28.15	eks.40	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 18일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.28.15	eks.39	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 2일
1.28.15	eks.38	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 3월 17일
1.28.15	eks.37	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 3월 4일
1.28.15	eks.36	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 2월 24일
1.28.15	eks.34	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 1월 17일
1.28.15	eks.33	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 1월 3일
1.28.15	eks.32	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 12월 13일
1.28.15	eks.31	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 12월 13일
1.28.15	eks.30	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 12월 13일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.28.15	eks.29	새 플랫폼 버전은 Amazon EKS Hybrid Nodes 지원이 포함되고 컨트롤 플레인 관찰성이 개선되었습니다. the section called “하이브리드 노드” 와 Amazon EKS의 성능 관찰성 개선 섹션을 각각 참조하세요.	2024년 11월 15일
1.28.14	eks.23	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 10월 21일
1.28.13	eks.19	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 9월 3일
1.28.12	eks.18	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 8월 28일
1.28.11	eks.17	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 8월 9일
1.28.11	eks.16	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 7월 2일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.28.9	eks.13	CoreDNS Auto Scaling, 보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전. CoreDNS Auto Scaling에 대한 자세한 내용은 the section called “높은 트래픽을 위해 규모 조정” 섹션을 참조하세요.	2024년 5월 16일
1.28.8	eks.12	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 4월 18일
1.28.7	eks.11	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 3월 29일
1.28.7	eks.10	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 3월 20일
1.28.6	eks.9	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 3월 12일
1.28.5	eks.7	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 1월 17일
1.28.4	eks.6	액세스 항목 , 보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 12월 14일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.28.4	eks.5	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 12월 12일
1.28.3	eks.4	EKS pod identity가 포드에 AWS 서비스에 대한 액세스 권한을 부여하는 방법 알아보기 , 보안 수정 및 개선 사항이 포함된 새로운 플랫폼 버전.	2023년 11월 10일
1.28.3	eks.3	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 11월 3일
1.28.2	eks.2	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 10월 16일
1.28.1	eks.1	EKS에 대한 Kubernetes 버전 1.28의 최초 릴리스. 자세한 내용은 the section called “Kubernetes 1.28” 섹션을 참조하세요.	2023년 9월 26일

Kubernetes 버전 1.27

다음 승인 컨트롤러는 모든 1.27 플랫폼 버전(NodeRestriction, ExtendedResourceToleration, NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, PodSecurity, Priority, DefaultTolerationSeconds, DefaultStorageClass, StorageObjectInUseProtection, PersistentVolumeClaimResize, RuntimeClass, CertificateApproval,

CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota)에 대해 활성화됩니다.

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.27.16	eks.49	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 6월 11일
1.27.16	eks.48	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 5월 30일
1.27.16	eks.47	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 5월 16일
1.27.16	eks.46	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 29일
1.27.16	eks.45	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 18일
1.27.16	eks.44	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 18일
1.27.16	eks.43	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 2일
1.27.16	eks.42	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 3월 17일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.27.16	eks.41	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 3월 4일
1.27.16	eks.40	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 3월 4일
1.27.16	eks.38	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 1월 17일
1.27.16	eks.37	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 1월 3일
1.27.16	eks.36	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 12월 13일
1.27.16	eks.35	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 12월 13일
1.27.16	eks.34	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 12월 13일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.27.16	eks.33	새 플랫폼 버전은 Amazon EKS Hybrid Nodes 지원, 보안 수정 및 개선 사항이 포함됩니다. Amazon EKS Hybrid Nodes에 대한 자세한 내용은 the section called “하 이브리드 노드” 섹션을 참조하세요.	2024년 11월 15일
1.27.16	eks.27	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 10월 21일
1.27.16	eks.23	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 9월 3일
1.27.16	eks.22	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 8월 28일
1.27.16	eks.21	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 8월 9일
1.27.15	eks.20	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 7월 2일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.27.13	eks.17	CoreDNS Auto Scaling, 보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전. CoreDNS Auto Scaling에 대한 자세한 내용은 the section called “높은 트래픽을 위해 규모 조정” 섹션을 참조하세요.	2024년 5월 16일
1.27.12	eks.16	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 4월 18일
1.27.11	eks.15	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 3월 29일
1.27.11	eks.14	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 3월 20일
1.27.10	eks.13	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 3월 12일
1.27.9	eks.11	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 1월 17일
1.27.8	eks.10	액세스 항목 , 보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 12월 14일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.27.8	eks.9	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 12월 12일
1.27.7	eks.8	EKS pod identity가 포드에 AWS 서비스에 대한 액세스 권한을 부여하는 방법 알아보기 , 보안 수정 및 개선 사항이 포함된 새로운 플랫폼 버전.	2023년 11월 10일
1.27.7	eks.7	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 11월 3일
1.27.6	eks.6	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 10월 16일
1.27.4	eks.5	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 8월 30일
1.27.4	eks.4	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 7월 30일
1.27.3	eks.3	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 6월 30일
1.27.2	eks.2	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 6월 9일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.27.1	eks.1	EKS에 대한 Kubernetes 버전 1.27의 최초 릴리스. 자세한 내용은 the section called “Kubernetes 1.27” 섹션을 참조하세요.	2023년 5월 24일

Kubernetes 버전 1.26

다음 승인 컨트롤러는 모든 1.26 플랫폼 버전(NodeRestriction, ExtendedResourceToleration, NamespaceLifecycle, LimitRanger, ServiceAccount, TaintNodesByCondition, PodSecurity, Priority, DefaultTolerationSeconds, DefaultStorageClass, StorageObjectInUseProtection, PersistentVolumeClaimResize, RuntimeClass, CertificateApproval, CertificateSigning, CertificateSubjectRestriction, DefaultIngressClass, MutatingAdmissionWebhook, ValidatingAdmissionWebhook, ResourceQuota)에 대해 활성화됩니다.

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.26.15	eks.51	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 6월 11일
1.26.15	eks.50	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 5월 30일
1.26.15	eks.49	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 5월 16일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.26.15	eks.48	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 29일
1.26.15	eks.47	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 18일
1.26.15	eks.46	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 18일
1.26.15	eks.45	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 4월 2일
1.26.15	eks.44	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 3월 17일
1.26.15	eks.43	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 3월 4일
1.26.15	eks.42	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 2월 24일
1.26.15	eks.40	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 1월 17일
1.26.15	eks.39	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2025년 1월 3일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.26.15	eks.38	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 12월 13일
1.26.15	eks.37	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 12월 13일
1.26.15	eks.36	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 12월 13일
1.26.15	eks.35	새 플랫폼 버전은 Amazon EKS Hybrid Nodes 지원, 보안 수정 및 개선 사항이 포함됩니다. Amazon EKS Hybrid Nodes에 대한 자세한 내용은 the section called “하이브리드 노드” 섹션을 참조하세요.	2024년 11월 15일
1.26.15	eks.28	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 10월 21일
1.26.15	eks.24	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 9월 3일
1.26.15	eks.23	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 8월 28일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.26.15	eks.22	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 8월 9일
1.26.15	eks.21	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 7월 2일
1.26.15	eks.18	CoreDNS Auto Scaling, 보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전. CoreDNS Auto Scaling에 대한 자세한 내용은 the section called “높은 트래픽을 위해 규모 조정” 섹션을 참조하세요.	2024년 5월 16일
1.26.15	eks.17	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 4월 18일
1.26.14	eks.16	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 3월 29일
1.26.14	eks.15	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 3월 20일
1.26.13	eks.14	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 3월 12일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.26.12	eks.12	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 1월 17일
1.26.11	eks.11	액세스 항목 , 보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 12월 14일
1.26.11	eks.10	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 12월 12일
1.26.10	eks.9	EKS pod identity가 포드에 AWS 서비스에 대한 액세스 권한을 부여하는 방법 알아보기 , 보안 수정 및 개선 사항이 포함된 새로운 플랫폼 버전.	2023년 11월 10일
1.26.10	eks.8	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 11월 3일
1.26.9	eks.7	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 10월 16일
1.26.7	eks.6	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 8월 30일
1.26.7	eks.5	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 7월 30일

Kubernetes 버전	EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.26.6	eks.4	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 6월 30일
1.26.5	eks.3	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 6월 9일
1.26.4	eks.2	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2023년 5월 5일
1.26.2	eks.1	EKS에 대한 Kubernetes 버전 1.26의 최초 릴리스. 자세한 내용은 the section called “Kubernetes 1.26” 섹션을 참조하세요.	2023년 4월 11일

현재 플랫폼 버전 가져오기

1. Amazon EKS 콘솔을 엽니다.
2. 탐색 창에서 클러스터를 선택합니다.
3. 클러스터 목록에서 플랫폼 버전을 확인할 클러스터 이름을 선택합니다.
4. 개요 탭을 선택합니다.
5. 플랫폼 버전은 세부 정보 섹션에서 확인할 수 있습니다.
6. 플랫폼 버전을 확인하려는 클러스터의 이름을 결정합니다.
7. 다음 명령 실행:

```
aws eks describe-cluster --name my-cluster --query cluster.platformVersion
```

예제 출력은 다음과 같습니다.


```
"eks.10"
```

플랫폼 버전 변경

EKS 클러스터의 플랫폼 버전은 변경할 수 없습니다. Kubernetes 버전에 대해 새 Amazon EKS 플랫폼 버전을 사용할 수 있게 되면 EKS는 모든 기존 클러스터를 해당 Kubernetes 버전의 최신 Amazon EKS 플랫폼 버전으로 자동으로 업그레이드합니다. 기존 Amazon EKS 플랫폼 버전의 자동 업그레이드는 중 분 방식으로 돌아옵니다. AWS 콘솔이나 CLI를 사용하여 플랫폼 버전을 변경할 수 없습니다.

Kubernetes 버전을 업그레이드하면 클러스터가 Kubernetes 버전의 최신 플랫폼 버전으로 이동합니다.

Karpenter 및 Cluster Autoscaler를 사용한 클러스터 컴퓨팅 규모 조정

자동 크기 조정은 변화하는 요구 사항에 맞게 리소스 규모를 자동으로 조정하는 기능입니다. 이는 Kubernetes의 주요 기능이며 그렇지 않으면 수동으로 수행하기 위한 광범위한 인적 자원을 필요로 할 것입니다.

EKS Auto Mode

Amazon EKS Auto Mode는 클러스터 컴퓨팅 리소스의 규모를 자동으로 조정합니다. 포드를 기존 노드에 맞출 수 없는 경우 EKS Auto Mode가 새 포드를 생성합니다. 또한 EKS Auto Mode는 워크로드를 통합하고 노드를 삭제합니다. EKS Auto Mode는 Karpenter를 기반으로 빌드됩니다.

자세한 내용은 다음을 참조하세요.

- [EKS Auto Mode](#)
- [the section called “노드 풀 생성”](#)
- [the section called “워크로드 팅창 배포”](#)

추가 솔루션

Amazon EKS는 두 가지 자동 크기 조정 제품을 추가로 지원합니다.

Karpenter

Karpenter는 애플리케이션 가용성과 클러스터 효율성을 개선하는 데 도움이 되는 유연한 고성능 Kubernetes Cluster Autoscaler입니다. Karpenter는 1분 이내에 애플리케이션 로드 변화에 대응하여 적절한 크기의 컴퓨팅 리소스(예: Amazon EC2 인스턴스)를 시작합니다. Kubernetes를 AWS와 통합함으로써 Karpenter는 워크로드의 요구 사항을 정확하게 충족하는 JIT(Just-In-Time) 컴퓨팅 리소스를 프로비저닝할 수 있습니다. Karpenter는 클러스터 워크로드의 특정 요구 사항을 기반으로 새로운 컴퓨팅 리소스를 자동으로 프로비저닝합니다. 여기에는 컴퓨팅, 스토리지, 가속화 및 예약 요구 사항이 포함됩니다. Amazon EKS는 Karpenter를 사용하는 클러스터를 지원하지만 Karpenter는 호환되는 모든 Kubernetes 클러스터에서 작동합니다. 자세한 내용은 [Karpenter](#) 설명서를 참조하세요.

Important

Karpenter는 AWS 고객이 Kubernetes 클러스터에서 설치, 구성 및 관리할 책임이 있는 오픈 소스 소프트웨어입니다. Karpenter가 Amazon EKS 클러스터의 호환 버전을 사용하여 수정되지 않은 상태로 실행될 때 기술 지원을 AWS 제공합니다. 고객이 다른 고객 관리형 소프트웨어와 마찬가지로 Karpenter 컨트롤러를 업그레이드하거나 실행 중인 Kubernetes 클러스터를 업그레이드할 때 Karpenter 컨트롤러의 가용성 및 보안과 적절한 테스트 절차를 유지하는 것이 중요합니다. Karpenter에 대한 AWS 서비스 수준 계약(SLA)은 없으며 고객은 Karpenter가 시작한 EC2 인스턴스가 비즈니스 요구 사항을 충족하는지 확인할 책임이 있습니다.

Cluster Autoscaler

Kubernetes Cluster Autoscaler는 포드가 실패하거나 다른 노드로 다시 예약될 때 클러스터의 노드 수를 자동으로 조정합니다. Cluster Autoscaler는 Auto Scaling을 사용합니다. 자세한 내용은 [AWS의 Cluster Autoscaler](#)를 참조하세요.

Amazon EKS에서 Amazon 애플리케이션 복구 컨트롤러(ARC)의 구역 이동에 대해 알아보기

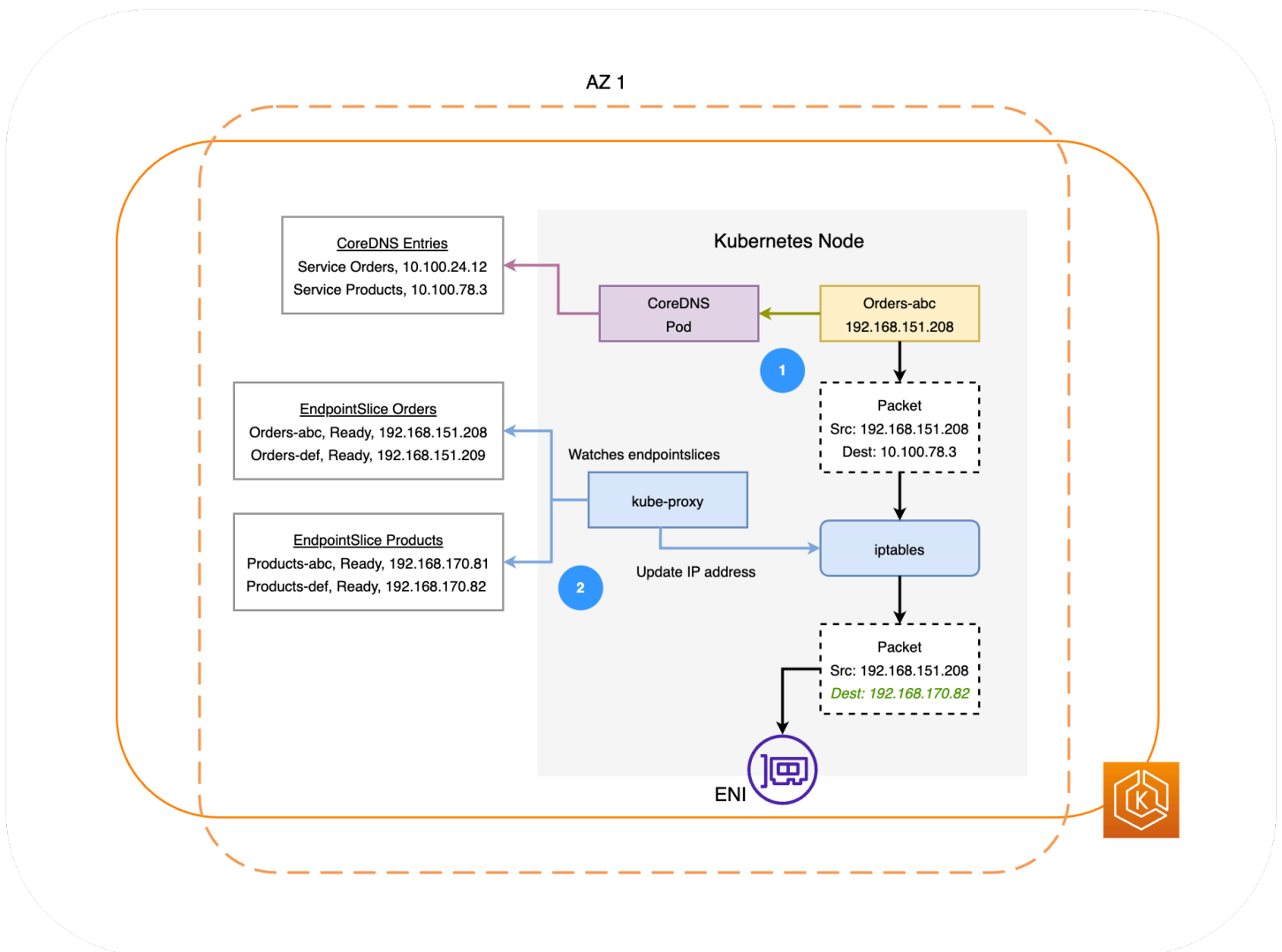
Kubernetes에는 가용 영역(AZ)의 상태 저하 또는 손상과 같은 이벤트에 대해 애플리케이션을 보다 탄력적으로 만들 수 있는 기본 기능이 있습니다. Amazon EKS 클러스터에서 워크로드를 실행할 때 [Amazon Application Recovery Controller\(ARC\) 영역 이동](#) 또는 [영역 자동 이동](#)을 사용하여 애플리케이션 환경의 내결함성 및 애플리케이션 복구를 더욱 개선할 수 있습니다. ARC 구역 이동은 구역 이동

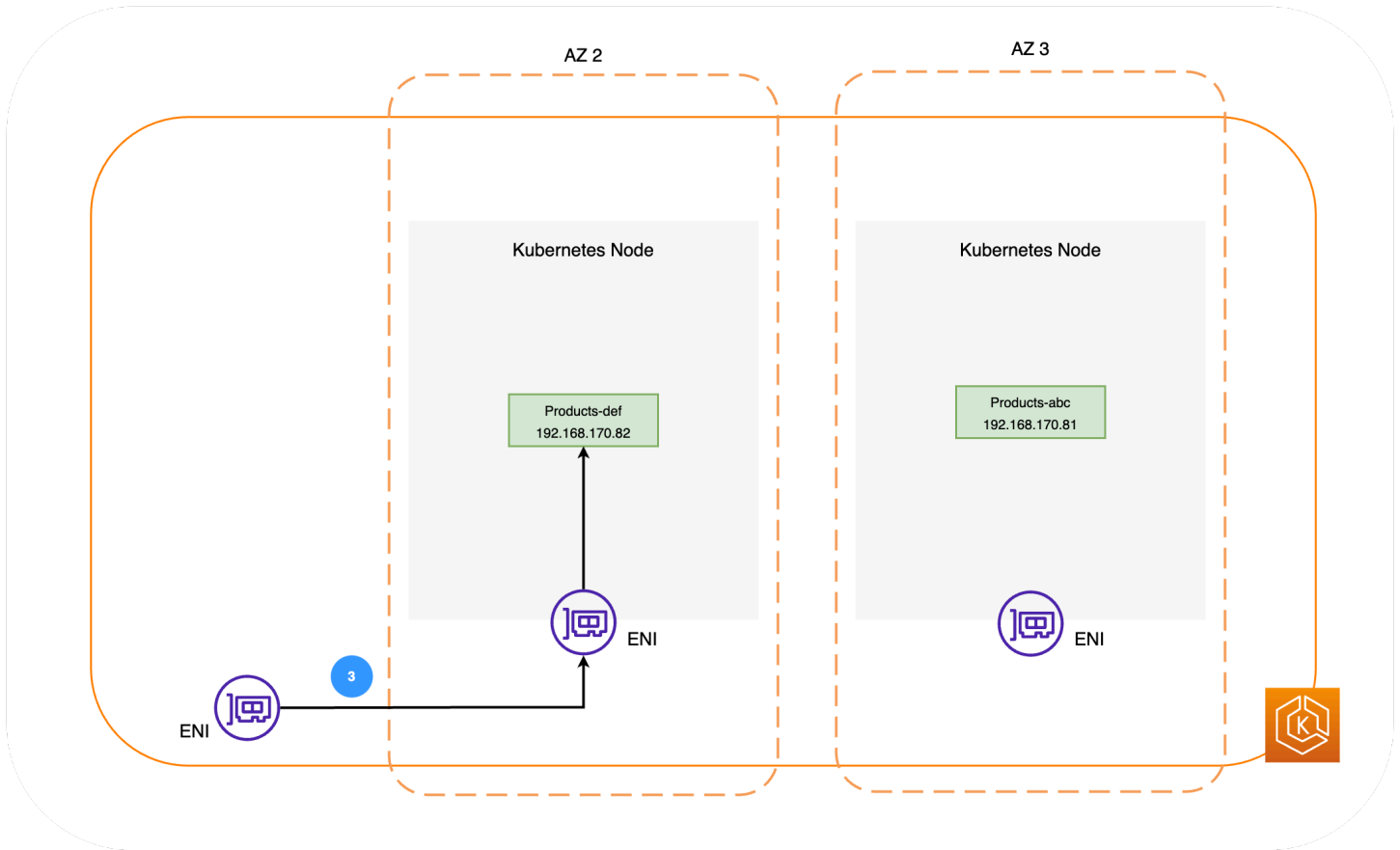
이 만료되거나 취소할 때까지 리소스에 대한 트래픽을 장애가 발생한 AZ에서 멀리 이동할 수 있는 임시 조치로 설계되었습니다. 필요한 경우 구역 이동을 연장할 수 있습니다.

EKS 클러스터에 대한 영역별 전환을 시작하거나 영역 자동 전환을 활성화하여 AWS가 대신 수행할 수 있도록 허용할 수 있습니다. 이 변경은 클러스터의 동서 네트워크 트래픽 흐름을 업데이트하여 정상 AZ의 워커 노드에서 실행 중인 포드에 대한 네트워크 엔드포인트만 고려하도록 합니다. 또한 EKS 클러스터의 애플리케이션에 대한 인그레스 트래픽을 처리하는 모든 ALB 또는 NLB는 트래픽을 정상 AZ의 타겟으로 자동 라우팅합니다.고가용성 목표를 추구하는 고객의 경우, AZ가 장애가 발생하는 경우 복구될 때까지 모든 트래픽을 장애가 발생한 AZ로부터 멀리 이동시키는 것이 중요할 수 있습니다. 이를 위해 [ARC 영역 전환을 사용하여 ALB 또는 NLB를 활성화](#)할 수도 있습니다.

포드 간 동서 네트워크 트래픽 흐름 이해하기

다음 다이어그램은 주문과 제품이라는 두 가지 워크로드 예시를 보여줍니다. 이 예시의 목적은 서로 다른 AZ의 워크로드와 포드가 통신하는 방법을 보여주기 위한 것이다.





1. Orders가 Products과 통신하려면 먼저 대상 서비스의 DNS 이름을 확인해야 합니다. Orders는 CoreDNS와 통신하여 해당 서비스의 가상 IP 주소(클러스터 IP)를 가져옵니다. Orders가 제품 서비스 이름을 확인하면 해당 대상 IP로 트래픽을 전송합니다.
2. kube-proxy는 클러스터의 모든 노드에서 실행되며 Service용 [EndpointSlices](#)를 지속적으로 감시합니다. 서비스가 생성되면 EndpointSlice 컨트롤러가 백그라운드에서 EndpointSlice를 생성하고 관리합니다. 각 EndpointSlice에는 엔드포인트가 실행 중인 노드와 함께 포트 주소의 하위 집합을 포함하는 엔드포인트 목록 또는 테이블이 있습니다. kube-proxy는 노드에서 iptables를 사용하여 이러한 각 Pod 엔드포인트에 대한 라우팅 규칙을 설정합니다. 또한 kube-proxy는 서비스의 클러스터 IP로 향하는 트래픽을 대신 포드의 IP 주소로 직접 전송하도록 리디렉션하여 기본적인 형태의 로드 밸런싱을 담당합니다. kube-proxy는 발신 연결에서 대상 IP를 다시 작성하여 이를 수행합니다.
3. 그런 다음 네트워크 패킷은 위 다이어그램에 표시된 대로 각 노드의 ENI를 통해 AZ 2의 제품 포드로 전송됩니다.

Amazon EKS의 ARC 영역 전환 이해

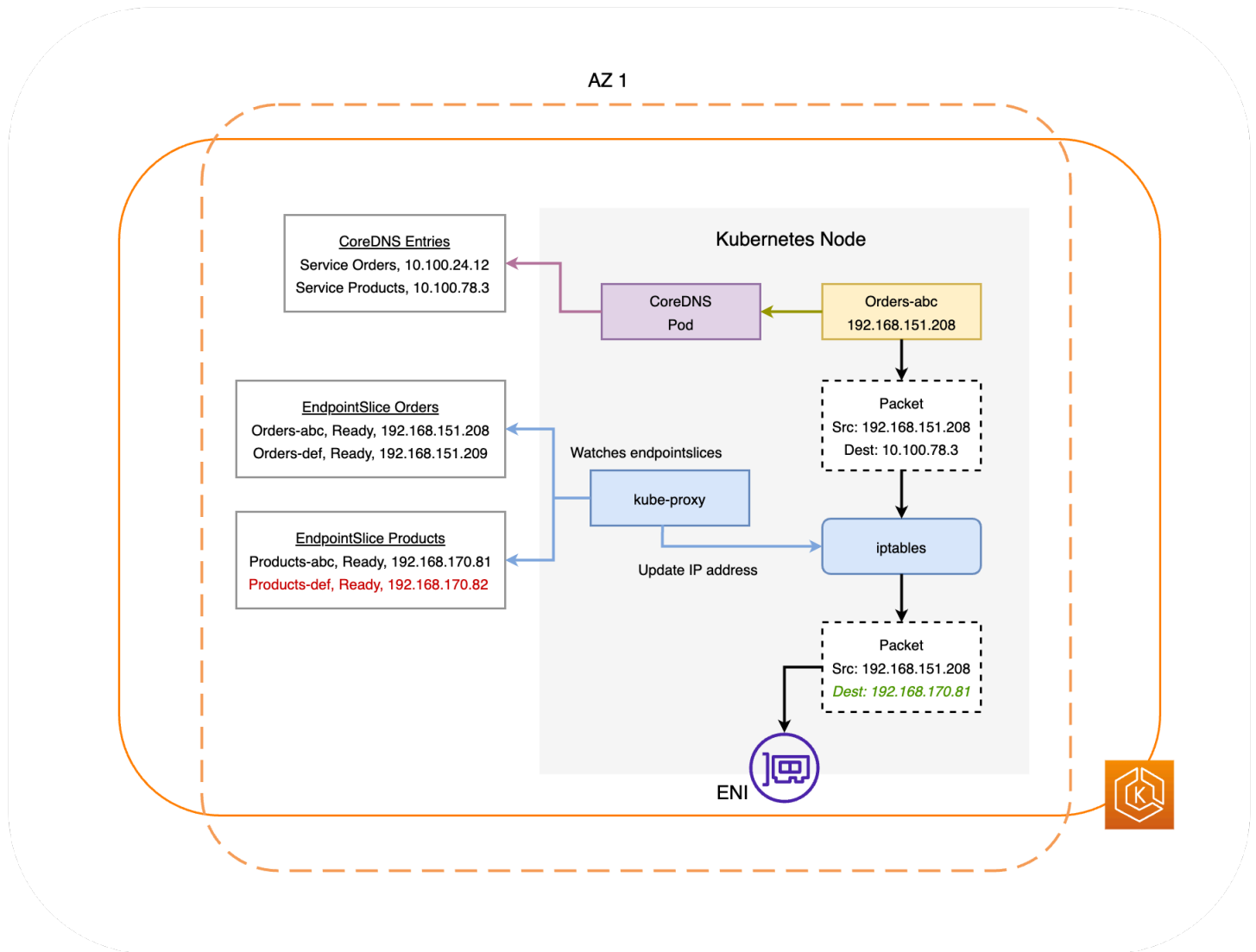
사용자 환경에 AZ 장애가 있는 경우 EKS 클러스터 환경에 대한 영역 이동을 시작할 수 있습니다. 또는 AWS가 영역 자동 전환을 사용하여 이를 관리하도록 허용할 수 있습니다. 영역 자동 전환을 사용하면

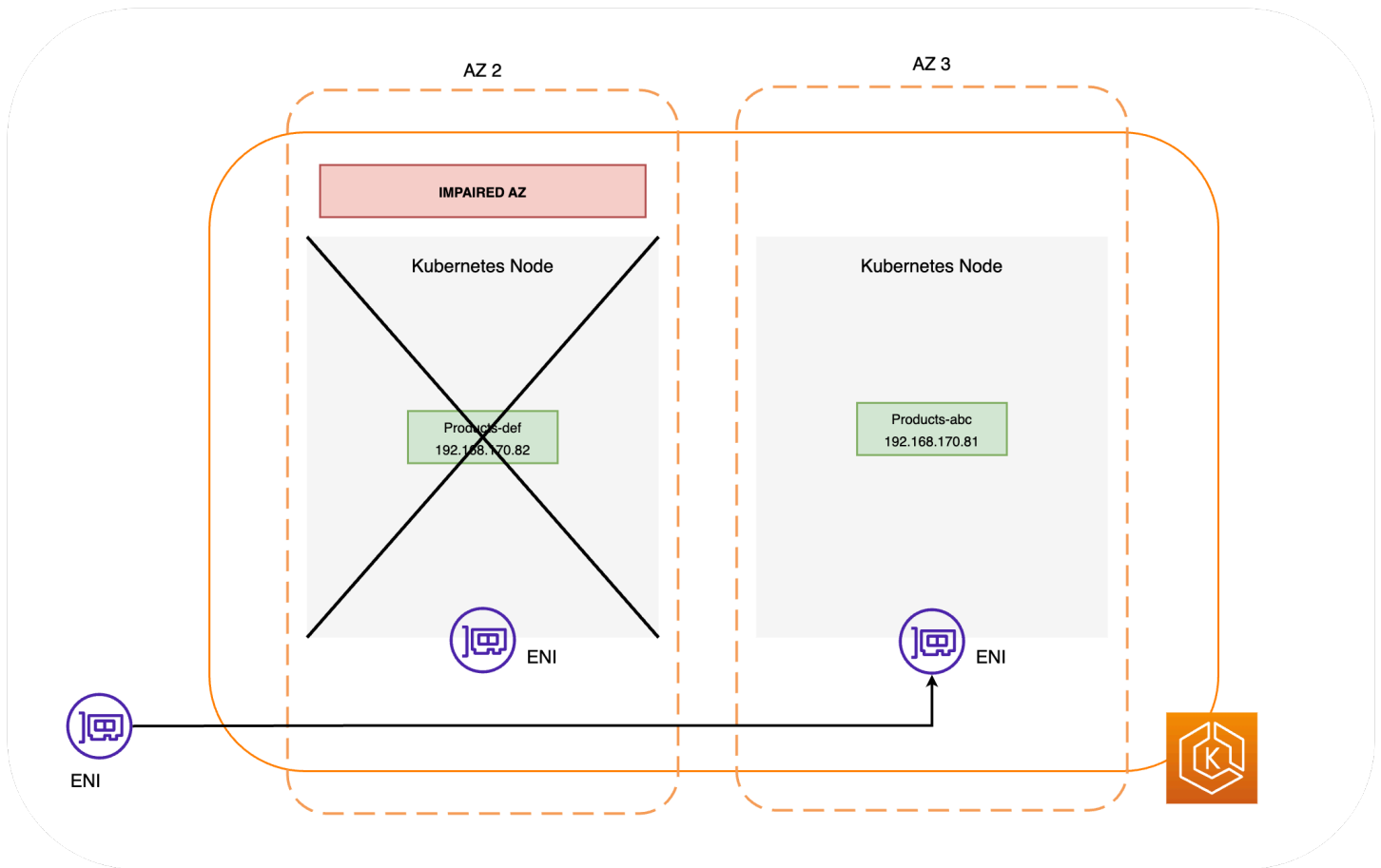
AWS는 클러스터 환경의 손상된 AZ에서 트래픽을 자동으로 전환하여 전체 AZ 상태를 모니터링하고 잠재적인 AZ 장애에 대응합니다.

Amazon EKS 클러스터에 ARC로 영역 전환이 활성화되면 ARC 콘솔, AWS CLI 또는 영역 전환 및 구역 자동 이동 API를 사용하여 영역 전환을 트리거하거나 구역 자동 이동을 활성화할 수 있습니다. EKS 구역 교대 근무 중에는 다음 작업이 자동으로 수행됩니다:

- 영향을 받는 AZ의 모든 노드가 폐쇄됩니다. 이렇게 하면 Kubernetes 스케줄러가 건강하지 않은 AZ의 노드에 새 포드를 스케줄링하지 못하게 된다.
- [관리형 노드 그룹](#)을 사용하는 경우, [가용 영역 리밸런싱](#)이 일시 중지되고 Auto Scaling Group(ASG)이 업데이트되어 새 EKS Data Plane 노드가 정상 AZs에서만 시작되도록 합니다.
- 비정상적인 AZ의 노드는 종료되지 않으며 이러한 노드에서 포드가 퇴출되지 않습니다. 이는 구역 전환 근무가 만료되거나 취소될 때 트래픽이 아직 수용 인원이 남아 있는 AZ로 안전하게 복귀할 수 있도록 하기 위한 것입니다.
- EndpointSlice 컨트롤러는 손상된 AZ에서 모든 Pod 엔드포인트를 찾아 관련 EndpointSlices에서 제거합니다. 이렇게 하면 정상 AZ에 있는 포드 엔드포인트만 네트워크 트래픽을 수신하도록 타겟팅됩니다. 영역 이동이 취소되거나 만료되면, EndpointSlice 컨트롤러는 복원된 AZ에 엔드포인트를 포함하도록 EndpointSlices를 업데이트합니다.

아래 다이어그램은 EKS 영역 이동이 클러스터 환경에서 건강한 포드 엔드포인트만 대상으로 삼도록 하는 방법에 대한 높은 수준의 흐름을 보여줍니다.





EKS 구역 교대 근무 요건

EKS에서 구역 이동이 성공적으로 작동하려면 AZ 장애에 탄력적으로 대응할 수 있도록 클러스터 환경을 미리 설정해야 합니다. 다음은 따라야 하는 단계의 목록입니다.

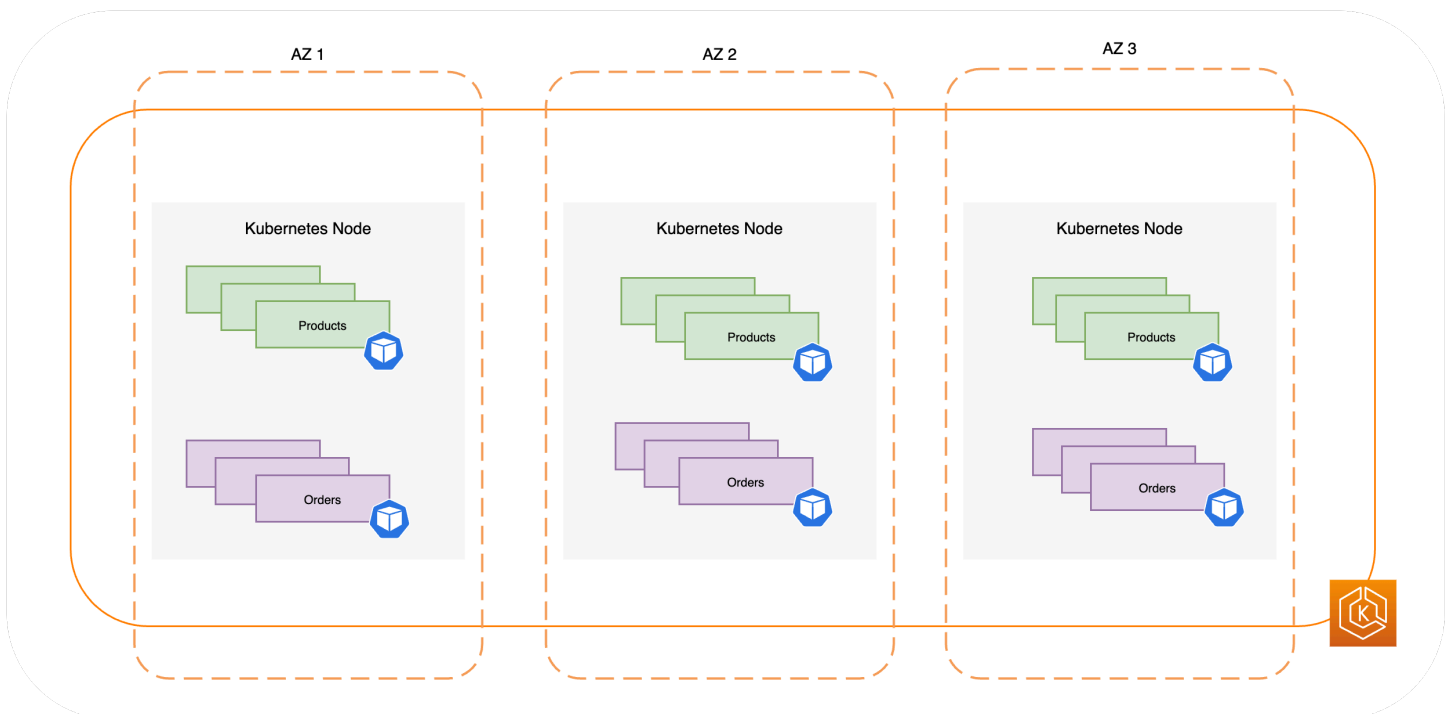
- 여러 AZ에 걸쳐 클러스터의 워커 노드 프로비저닝하기
- 단일 AZ의 제거를 견딜 수 있도록 충분한 컴퓨팅 용량 프로비저닝
- 모든 AZ에서 포드(CoreDNS 포함) 사전 스케일링
- 모든 AZ에 여러 포드 복제본을 분산하여 단일 AZ에서 이동해도 충분한 용량을 확보할 수 있도록 하세요.
- 동일한 AZ에 상호 의존적이거나 관련된 포드를 공동 배치합니다.
- 수동으로 영역 전환을 시작하여 AZ가 하나 줄어들었을 때 클러스터 환경이 예상대로 작동하는지 테스트합니다. 또는 구역별 자동 교대 근무를 활성화하고 자동 교대 근무 연습 실행에 응답할 수 있습니다. 영역 교대 근무가 EKS에서 작동하기 위해 반드시 필요한 것은 아니지만 강력히 권장됩니다.

여러 AZ에 걸쳐 EKS 워커 노드 프로비저닝하기

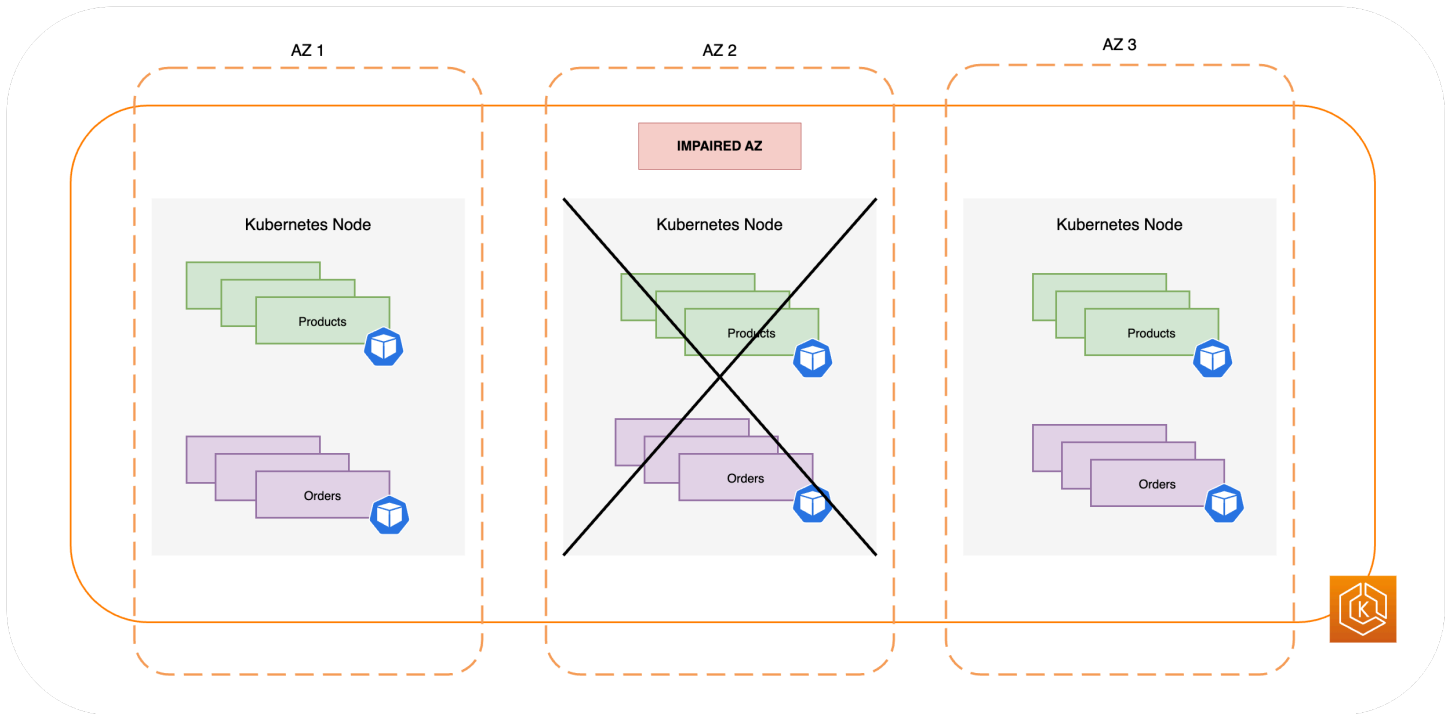
AWS 리전에는 가용 영역(AZ)이라고 하는 물리적 데이터 센터가 있는 여러 개의 별도 위치가 있습니다. AZ는 리전 전체에 영향을 미칠 수 있는 동시 영향을 피하기 위해 서로 물리적으로 격리되도록 설계되었습니다. EKS 클러스터를 프로비저닝할 때는 리전 내 여러 AZ에 워커 노드를 배포해야 합니다. 이렇게 하면 클러스터 환경이 단일 AZ의 장애에 더 탄력적으로 대응하고 다른 AZ에서 실행 중인 애플리케이션의 고가용성(HA)을 유지할 수 있습니다. 영향을 받는 AZ에서 구역 이동을 시작하면 클러스터의 고가용성 상태를 유지하면서 정상 AZ만 사용하도록 EKS 환경의 클러스터 내 네트워크가 자동으로 업데이트됩니다.

EKS 환경에 이러한 다중 AZ를 설정하면 시스템의 전반적인 안정성을 향상시킬 수 있습니다. 그러나 multi-AZ 환경은 애플리케이션 데이터가 전송되고 처리되는 방식에 중요한 역할을 할 수 있으며, 이는 결국 환경의 네트워크 요금에 영향을 미칠 수 있습니다. 특히 잦은 이그레스 크로스존 트래픽(AZ 간에 분산되는 트래픽)은 네트워크 관련 비용에 큰 영향을 미칠 수 있습니다. 다양한 전략을 적용하여 EKS 클러스터의 포드 간 영역 간 트래픽 양을 제어하고 관련 비용을 절감할 수 있습니다. 고가용성 EKS 환경을 실행할 때 네트워크 비용을 최적화하는 방법에 대한 자세한 내용은 [이 모범 사례 가이드](#)를 참조하세요.

아래 다이어그램은 3개의 정상 AZ가 있는 고가용성 EKS 환경을 보여줍니다.



아래 다이어그램은 3개의 AZ가 있는 EKS 환경이 AZ 장애에도 복원력이 있고 다른 2개의 정상 AZ로 인해 높은 가용성을 유지하는 방법을 보여 줍니다.



단일 AZ 제거를 견딜 수 있는 충분한 컴퓨팅 용량 프로비저닝

EKS 데이터 플레인에서 컴퓨팅 인프라의 리소스 사용률과 비용을 최적화하려면 워크로드 요구 사항에 따라 컴퓨팅 용량을 조정하는 것이 가장 좋습니다. 그러나 모든 워커 노드가 최대 용량에 도달하면 새 포드를 예약하기 전에 새 워커 노드를 EKS 데이터 플레인에 추가해야 합니다. 중요한 워크로드를 실행할 때는 일반적으로 갑작스러운 부하 증가, 노드 상태 문제 등과 같은 비상 상황에 대처하기 위해 항상 온라인에서 여분의 용량을 확보하고 실행하는 것이 좋습니다. 영역 이동을 사용하려는 경우 전체 AZ의 용량을 제거할 계획이므로 AZ가 오프라인 상태에서도 부하를 충분히 처리할 수 있도록 여분의 컴퓨팅 용량을 조정해야 합니다.

컴퓨터를 확장할 시 EKS Data Plane에 새 노드를 추가하는 프로세스에는 시간이 다소 걸리며, 이는 특히 영역 장애가 발생할 경우 애플리케이션의 실시간 성능과 가용성에 영향을 미칠 수 있습니다. 최종 사용자나 클라이언트의 경험 저하를 방지하기 위해 EKS 환경은 AZ 손실로 인한 부하를 흡수할 수 있는 탄력성을 갖춰야 합니다. 즉, 새 포드가 필요한 시점과 워커 노드에서 실제로 예약되는 시점 사이의 지연을 최소화하거나 없애야 합니다.

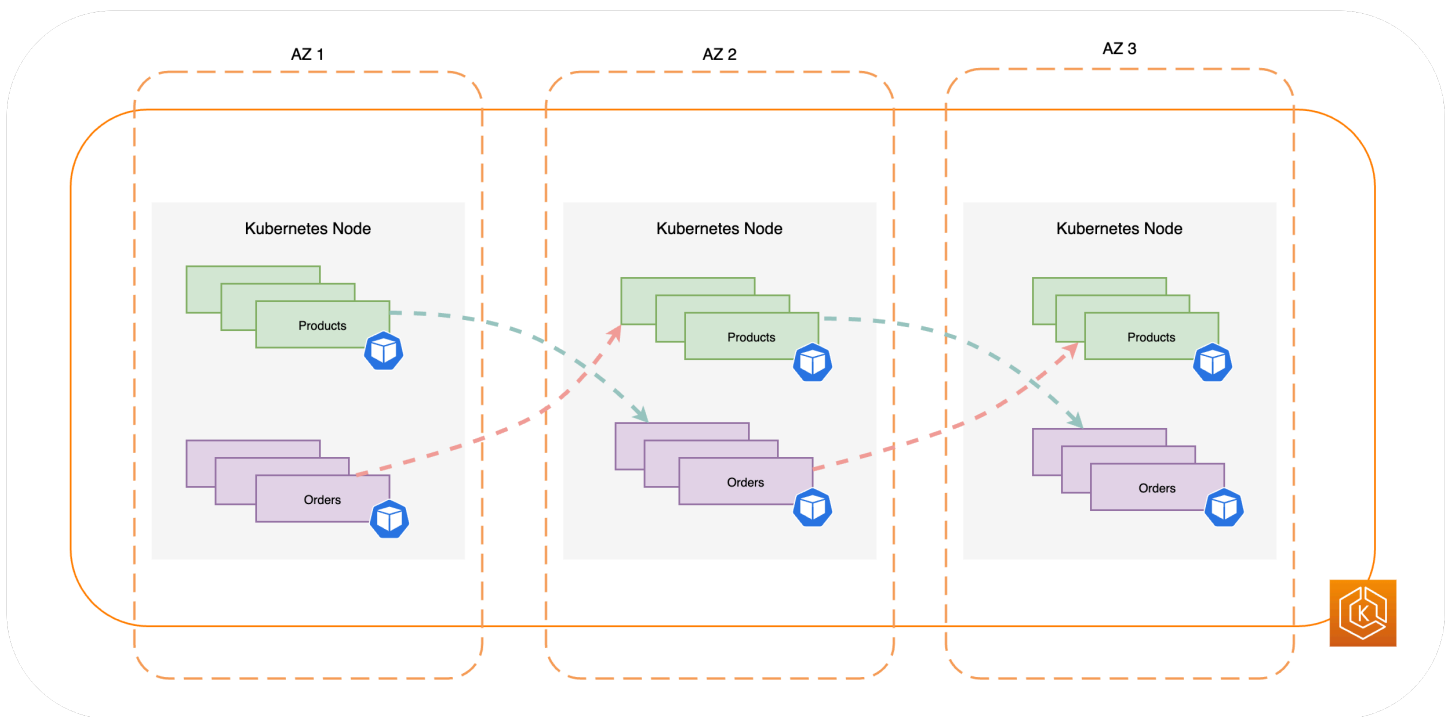
또한 영역 장애가 발생하는 경우, 잠재적인 컴퓨팅 용량 제약으로 인해 새로 필요한 노드를 정상 AZ의 EKS 데이터 플레인에 추가할 수 없는 위험을 완화해야 합니다.

이를 위해, 특히 사용자 환경에 AZ가 하나 더 적은 경우, 각 AZ의 일부 워커 노드에서 컴퓨팅 용량을 오버프로비저닝하여 쿠버네티스 스케줄러가 새로운 포드 배치를 위해 기존 용량을 사용할 수 있도록 해야 합니다.

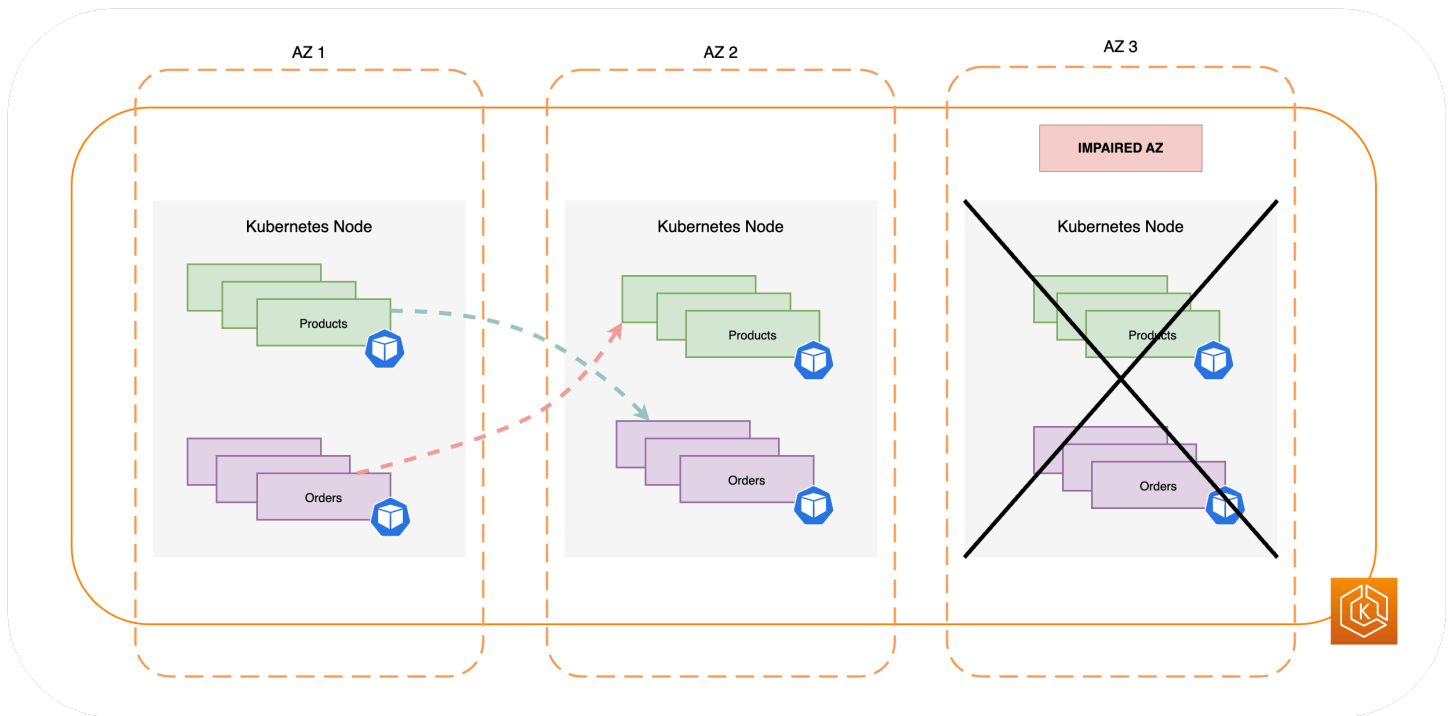
여러 포드 복제본을 AZ에서 실행 및 확산하기

Kubernetes를 사용하면 단일 애플리케이션의 여러 인스턴스(포드 복제본)를 실행하여 워크로드를 사전 확장할 수 있습니다. 애플리케이션에 대해 여러 포드 복제본을 실행하면 단일 장애 지점을 제거하고 단일 복제본의 리소스 부담을 줄임으로써 전반적인 성능을 향상시킬 수 있습니다. 그러나 애플리케이션의고가용성과 내결함성을 모두 확보하려면 애플리케이션의 여러 복제본을 여러 장애 도메인(이 경우, 토폴로지 도메인이라고도 함)에 걸쳐 실행하고 분산시켜야 합니다(이 경우, AZ). [토폴로지 분산 제약 조건](#)을 사용하면 기존 정적 안정성을 갖도록 애플리케이션을 설정할 수 있으므로 AZ 장애가 발생할 경우 정상 AZ에 추가 트래픽 급증 또는 급증을 즉시 처리할 수 있는 충분한 복제본을 보유할 수 있습니다.

아래 다이어그램은 모든 AZ가 정상일 때 동서 방향 트래픽 흐름이 있는 EKS 환경을 보여줍니다.



아래 다이어그램은 단일 AZ에 장애가 발생하여 구역 이동을 시작할 때 동서 방향 트래픽 흐름이 있는 EKS 환경을 보여 줍니다.



아래 코드 스니펫은 이 Kubernetes 기능으로 워크로드를 설정하는 방법의 예시입니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orders
spec:
  replicas: 9
  selector:
    matchLabels:
      app: orders
  template:
    metadata:
      labels:
        app: orders
        tier: backend
    spec:
      topologySpreadConstraints:
        - maxSkew: 1
          topologyKey: "topology.kubernetes.io/zone"
          whenUnsatisfiable: ScheduleAnyway
          labelSelector:
            matchLabels:
              app: orders
```

가장 중요한 것은 DNS 서버 소프트웨어(CoreDNS/kube-dns)의 여러 복제본을 실행하고 기본적으로 구성되지 않은 경우, 유사한 토폴로지 확산 제약 조건을 적용해야 한다는 것입니다. 이렇게 하면 단일 AZ 장애가 발생하는 경우 클러스터의 다른 통신하는 포드에 대한 서비스 검색 요청을 계속 처리할 수 있도록 정상 AZ에 충분한 DNS 포드가 있는지 확인하는 데 도움이 됩니다. [CoreDNS EKS 추가 기능](#)에는 사용 가능한 AZs가 여러 개 있는 경우 클러스터의 가용 영역에 CoreDNS 포드가 분산되도록 하는 기본 설정이 있습니다. 이러한 기본 설정을 사용자 지정 구성으로도 바꿀 수 있습니다.

[Helm에 CoreDNS](#)를 설치할 때 [value.yaml 파일](#)의 replicaCount를 업데이트하여 각 AZ에 충분한 수의 복제본이 있는지 확인할 수 있습니다. 또한 이러한 복제본이 클러스터 환경의 여러 AZ에 분산되도록 하려면 동일한 values.yaml 파일에서 topologySpreadConstraints 속성을 업데이트해야 합니다. 아래 코드 스니펫은 이를 위해 CoreDNS를 구성하는 방법을 보여줍니다.

CoreDNS Helm values.yaml

```
replicaCount: 6
topologySpreadConstraints:
  - maxSkew: 1
    topologyKey: topology.kubernetes.io/zone
    whenUnsatisfiable: ScheduleAnyway
    labelSelector:
      matchLabels:
        k8s-app: kube-dns
```

AZ 장애가 발생하는 경우, CoreDNS용 자동 확장 시스템을 사용하여 CoreDNS 포드의 증가된 부하를 흡수할 수 있습니다. 필요한 DNS 인스턴스 수는 클러스터에서 실행 중인 워크로드 수에 따라 달라집니다. CoreDNS는 [Horizontal Pod Autoscaler\(HPA\)](#)를 사용하여 CPU를 기반으로 규모를 조정할 수 있는 CPU 바인딩입니다. 아래는 필요에 맞게 수정할 수 있는 예시입니다.

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: coredns
  namespace: default
spec:
  maxReplicas: 20
  minReplicas: 2
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: coredns
  targetCPUUtilizationPercentage: 50
```

또는 EKS는 CoreDNS의 EKS 애드온 버전에서 CoreDNS 배포의 자동 확장을 관리할 수 있습니다. 이 CoreDNS 오토스케일러는 노드 수와 CPU 코어 수를 포함하여 클러스터 상태를 지속적으로 모니터링합니다. 해당 정보를 기반으로 컨트롤러는 EKS 클러스터에서 CoreDNS 배포의 복제본 수를 동적으로 조정합니다.

[CoreDNS EKS 추가 기능에서 자동 크기 조정 구성](#)을 활성화하려면 다음과 같은 선택적 구성 설정을 추가해야 합니다.

```
{
  "autoScaling": {
    "enabled": true
  }
}
```

[NodeLocal DNS](#) 또는 [클러스터 비레 오토 스케일러](#)를 사용하여 CoreDNS 를 조정할 수도 있습니다. 여기에서 [CoreDNS 수평 확장](#)에 대한 자세한 내용을 확인할 수 있습니다.

동일한 AZ에 상호 의존적인 포드 배치하기

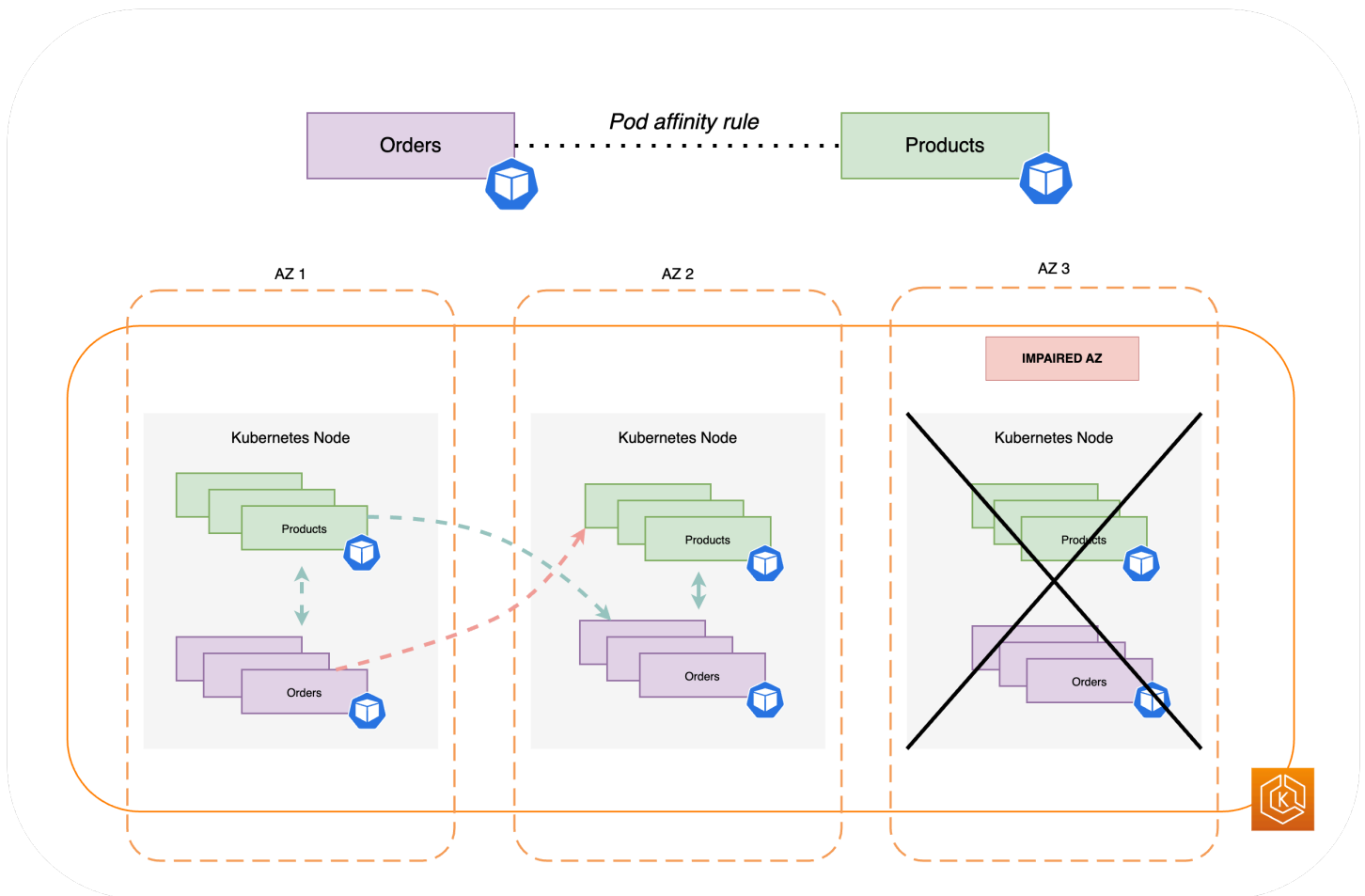
대부분의 경우 엔드투엔드 프로세스의 성공적인 실행을 위해 서로 통신해야 하는 별개의 워크로드를 실행하고 있을 수 있습니다. 별개의 애플리케이션이 서로 다른 AZ에 분산되어 있지만 동일한 AZ에 배치되어 있지 않은 경우 단일 AZ 장애가 기본 엔드투엔드 프로세스에 영향을 미칠 수 있습니다. 예를 들어, 애플리케이션 A가 AZ 1과 AZ 2에 여러 복제본을 가지고 있지만 애플리케이션 B는 모든 복제본을 AZ 3에 가지고 있는 경우, AZ 3의 손실은 이 두 워크로드(애플리케이션 A와 B) 간의 모든 엔드투엔드 프로세스에 영향을 미칩니다. 토폴로지 확산 제약 조건과 포드 선호도를 결합하면 모든 AZ에 포드를 확산하고 특정 포드 간의 관계를 구성하여 함께 코로케이션되도록 함으로써 애플리케이션의 복원력을 향상시킬 수 있습니다.

[포드 선호도 규칙](#)을 사용하면 워크로드 간의 관계를 정의하여 Kubernetes 스케줄러의 동작에 영향을 주어 동일한 워커 노드 또는 동일한 AZ에서 포드를 공동 배치할 수 있습니다. 이러한 예약 제약 조건을 얼마나 엄격하게 설정할지 구성할 수도 있습니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: products
  namespace: ecommerce
  labels:
    app.kubernetes.io/version: "0.1.6"
```

```
spec:
  serviceAccountName: graphql-service-account
  affinity:
    podAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
              - key: app
                operator: In
                values:
                  - orders
            topologyKey: "kubernetes.io/hostname"
```

다 다이어그램은 포드 선호도 규칙을 사용하여 동일한 노드에 공동 배치된 포드를 보여줍니다.



클러스터 환경이 AZ 손실을 처리할 수 있는지 테스트하기

위의 요구 사항을 완료한 후 다음 중요한 단계는 AZ 손실을 처리할 수 있는 충분한 컴퓨팅 및 워크로드 용량을 보유하고 있는지 테스트하는 것입니다. EKS에서 영역 이동을 수동으로 트리거하여 이 작업

을 수행할 수 있습니다. 또는 영역 자동 시프트를 활성화하고 연습 실행을 구성하여 클러스터 환경에서 AZ를 하나 줄인 상태에서 애플리케이션이 예상대로 작동하는지 테스트할 수 있습니다.

FAQ

왜 이 기능을 사용해야 하나요?

EKS 클러스터에서 ARC 구역 이동 또는 구역 자동 이동을 사용하면 클러스터 내 네트워크 트래픽을 손상된 AZ에서 멀리 이동시키는 빠른 복구 프로세스를 자동화하여 Kubernetes 애플리케이션 가용성을 더 잘 유지할 수 있습니다. ARC를 사용하면 장애 발생 시 복구 기간이 길어지는 길고 복잡한 단계를 피할 수 있습니다.

이 기능은 다른 AWS 서비스에서 어떻게 작동하나요?

EKS는 AWS에서 복구 작업을 수행할 수 있는 기본 인터페이스를 제공하는 ARC와 통합됩니다. 클러스터 내 트래픽이 손상된 AZ에서 적절하게 라우팅되도록 하기 위해, Kubernetes 데이터 플레인에서 실행되는 포드의 네트워크 엔드포인트 목록을 수정합니다. 외부 트래픽을 클러스터로 라우팅하기 위해 AWS 로드 밸런서를 사용하는 경우, 이미 로드 밸런서를 ARC에 등록하고 영역 이동을 트리거하여 성능 저하 영역으로 트래픽이 유입되는 것을 방지할 수 있습니다. 이 기능은 또한 EKS 관리형 노드 그룹(MNG)에 의해 생성된 Amazon EC2 Auto Scaling 그룹(ASG)과 상호 작용합니다. 손상된 AZ가 새로운 Kubernetes 포드 또는 노드 실행에 사용되지 않도록 하기 위해, EKS는 손상된 AZ를 ASG에서 제거합니다.

이 기능은 기본 Kubernetes 보호와 어떻게 다른가요?

이 기능은 고객이 복원력을 유지하는 데 도움이 되는 여러 가지 Kubernetes 기본 제공 보호 기능과 함께 작동합니다. 포드가 트래픽을 수신해야 하는 시기를 결정하는 포드 준비 상태 및 활성 프로브를 구성할 수 있습니다. 이러한 프로브가 실패하면 쿠버네티스는 이러한 포드를 서비스의 대상으로 제거하고 트래픽은 더 이상 포드로 전송되지 않는다. 이 기능은 유용하지만, 고객이 이러한 상태 검사를 구성하여 영역이 저하될 때 실패하도록 하는 것은 간단하지 않습니다. ARC 구역 이동 기능은 Kubernetes의 기본 보호 기능으로 충분하지 않은 경우 성능 저하된 AZ를 완전히 격리하는 데 도움이 되는 추가적인 안전망을 제공합니다. 또한 아키텍처의 운영 준비 상태와 복원력을 쉽게 테스트할 수 있는 방법을 제공합니다.

AWS가 저를 대신하여 구역 이동을 트리거할 수 있나요?

예, ARC 구역 전환을 완전히 자동화된 방식으로 사용하려면 ARC 구역 자동 전환을 사용하도록 설정할 수 있습니다. 구역별 자동 교대 근무를 사용하면 AWS를 사용하여 EKS 클러스터의 AZ 상태를 모니터링하고 AZ 장애가 감지되면 자동으로 교대 근무를 트리거할 수 있습니다.

이 기능을 사용하는데 워커 노드와 워크로드가 사전 확장되지 않은 경우 어떻게 되나요?

사전 확장하지 않고 영역 이동 중에 추가 노드 또는 포드 프로비저닝에 의존하는 경우 복구가 지연될 위험이 있습니다. Kubernetes Data Plane에 새 노드를 추가하는 프로세스에는 시간이 다소 걸리며, 이는 특히 영역 장애가 발생할 경우 애플리케이션의 실시간 성능과 가용성에 영향을 미칠 수 있습니다. 또한 영역 장애가 발생하면 잠재적인 컴퓨팅 용량 제약이 발생하여 새로 필요한 노드를 정상 AZ에 추가하지 못할 수도 있습니다.

워크로드가 사전 확장되지 않고 클러스터의 모든 AZ에 분산되어 있는 경우, 영역 장애는 영향을 받는 AZ의 워커 노드에서만 실행되는 애플리케이션의 가용성에 영향을 미칠 수 있습니다. 애플리케이션의 완전한 가용성 중단 위험을 완화하기 위해 EKS는 워크로드의 모든 엔드포인트가 비정상적인 AZ에 있는 경우 트래픽이 손상된 영역의 포드 엔드포인트로 전송되지 않도록 하는 장애 세이프 기능을 제공합니다. 그러나 영역별 문제 발생 시 가용성을 유지하기 위해 애플리케이션을 사전 확장하고 모든 AZ에 분산하는 것이 좋습니다.

스테이플 애플리케이션을 실행하면 어떻게 되나요?

상태 저장 애플리케이션을 실행하는 경우 사용 사례와 아키텍처에 따라 내결함성을 평가해야 합니다. 액티브/대기 아키텍처 또는 패턴이 있는 경우 액티브가 손상된 AZ에 있는 경우가 있을 수 있습니다. 애플리케이션 수준에서 대기 모드가 활성화되어 있지 않으면 애플리케이션에 문제가 발생할 수 있습니다. 정상 AZs에서 새 Kubernetes Pod가 시작되면 손상된 AZ에 한정된 영구 볼륨에 연결할 수 없으므로 문제가 발생할 수도 있습니다.

이 기능은 Karpenter와 함께 작동하나요?

현재 EKS의 ARC 구역 전환 및 영역 자동 전환에서는 Karpenter 지원을 사용할 수 없습니다. AZ가 손상된 경우, 새 워커 노드가 정상 AZ에서만 시작되도록 비정상적인 AZ를 제거하여 관련 Karpenter 노드 풀 구성을 조정할 수 있습니다.

이 기능은 EKS Fargate에서 작동하나요?

이 기능은 EKS Fargate에서 작동하지 않습니다. 기본적으로 EKS Fargate가 영역 상태 이벤트를 인식하면 포드는 다른 AZ에서 실행되는 것을 선호합니다.

EKS 관리형 Kubernetes 컨트롤 플레인이 영향을 받나요?

아니요, 기본적으로 Amazon EKS는 고가용성을 보장하기 위해 여러 AZ에서 Kubernetes 컨트롤 플레인을 실행하고 확장합니다. ARC 영역 시프트 및 영역 자동 시프트는 Kubernetes 데이터 플레인에서만 작동합니다.

이 새로운 기능과 관련된 비용이 있습니까?

EKS 클러스터에서 추가 비용 없이 ARC 구역 교대근무 및 구역 자동 교대근무를 사용할 수 있습니다. 그러나 프로비저닝된 인스턴스에 대한 비용은 계속 지불해야 하며, 이 기능을 사용하기 전에 Kubernetes 데이터 플레인을 사전 확장하는 것이 좋습니다. 비용과 애플리케이션 가용성 간의 적절한 균형을 고려해야 합니다.

추가 리소스

- [영역 전환이 작동하는 방식](#)
- [ARC의 영역 전환 모범 사례](#)
- [영역 전환 및 영역 자동 전환에 지원되는 리소스 및 시나리오](#)
- [Amazon EKS에서 탄력적인 워크로드 운영](#)
- [포트 우선순위 및 오버프로비저닝으로 Kubernetes 노드 스케일링 지연 제거](#)
- [높은 DNS 트래픽을 위한 CoreDNS 포트 확장](#)

손상된 가용 영역을 방지하려면 EKS 영역 전환을 활성화합니다.

Amazon Application Recovery Controller(ARC)는 가용 영역(AZs)에서 애플리케이션에 대한 복구를 관리 및 조정하는 데 도움이 되며 Amazon EKS를 비롯한 많은 서비스와 함께 작동합니다. ARC 영역 전환에 대한 EKS 지원을 통해 클러스터 내 네트워크 트래픽을 손상된 AZ에서 멀리 이동할 수 있습니다. 또한 AZs의 상태를 AWS 모니터링하고 사용자를 대신하여 비정상 AZ에서 네트워크 트래픽을 일시적으로 이동할 수 있는 권한을 부여할 수 있습니다.

EKS 영역 전환을 사용하는 방법:

1. Amazon Application Recovery Controller(ARC)를 사용하여 EKS 클러스터를 활성화합니다. 이는 Amazon EKS 콘솔, AWS CLI, CloudFormation 또는 eksctl을 사용하여 클러스터 수준에서 수행됩니다.
2. 활성화되면 ARC 콘솔, AWS CLI 또는 영역 전환 및 영역 자동 전환 APIs.

EKS 클러스터를 ARC에 등록한 후에도 ARC를 구성해야 합니다. 예를 들어, ARC 콘솔을 사용하여 Zonal Autoshift를 구성할 수 있습니다.

EKS Zonal Shift의 작동 방식과 손상된 가용 영역을 처리하도록 워크로드를 설계하는 방법에 대한 자세한 내용은 [the section called “구역별 교대에 대해 알아보기”](#)을 참조하세요.

고려 사항:

- EKS Auto Mode는 Amazon Application Recovery Controller, Zonal Shift, Zonal Autoshift를 지원하지 않습니다.

Amazon Application Recovery Controller란 무엇입니까?

Amazon Application Recovery Controller(ARC)는 AWS에서 실행 중인 애플리케이션의 복구를 준비하고 더 빠르게 수행할 수 있도록 도와줍니다. 영역 전환을 사용하면 지원되는 리소스의 트래픽을 AZ에서 AWS 리전의 정상 AZ로 일시적으로 이동하여 가용 영역(AZ) 장애에서 빠르게 복구할 수 있습니다.

[Amazon Application Recovery Controller\(ARC\)에 대해 자세히 알아보기](#)

영역 전환이란 무엇입니까?

영역 전환은 EKS 클러스터 또는 Elastic Load Balancer와 같은 리소스에 대한 트래픽을 AWS 리전의 가용 영역에서 멀리 이동하여 문제를 빠르게 완화하고 애플리케이션을 빠르게 복구할 수 있는 ARC의 기능입니다. 예를 들어, 잘못된 배포로 인해 지연 시간이 발생하거나 가용 영역이 손상되어 트래픽을 이동하도록 선택할 수 있습니다. 영역 전환에는 사전 구성 단계가 필요하지 않습니다.

[ARC Zonal shift에 대해 자세히 알아보기](#)

영역 자동 전환이란 무엇입니까?

영역 자동 이동은 AWS가 고객을 대신하여 지원되는 리소스에 대한 트래픽을 AZ에서 AWS 리전의 정상 AZ로 이동하도록 승인할 수 있도록 설정할 수 있는 ARC의 기능입니다. AWS는 내부 원격 분석 결과 리전 내 한 AZ에 잠재적으로 고객에게 영향을 미칠 수 있는 장애가 있는 것으로 나타나면 자동 전환을 시작합니다. 내부 원격 측정에는 AWS 네트워크, Amazon EC2 및 Elastic Load Balancing 서비스를 비롯한 멀티 소스의 지표가 통합되어 있습니다.

AWS는 표시기에 더 이상 문제 또는 잠재적 문제가 없는 것으로 표시되면 자동 전환을 종료합니다.

[ARC Zonal Autoshift에 대해 자세히 알아보기](#)

EKS는 자동 전환 중에 무엇을 합니까?

EKS는 네트워킹 구성을 업데이트하여 트래픽이 손상된 AZ로 연결되지 않도록 합니다. 또한 관리형 노드 그룹을 사용하는 경우 EKS는 영역 전환 중에만 정상 AZs의 새 노드를 시작합니다. 교대가 만료되거나 취소되면 이전에 비정상적으로 감지된 AZ를 포함하도록 네트워킹 구성이 복원됩니다.

[EKS Zonal shift에 대해 자세히 알아보기](#)

Amazon Application Recovery Controller(ARC)에 EKS 클러스터 등록(AWS 콘솔)

1. ARC에 등록할 EKS 클러스터의 이름과 리전을 찾습니다.
2. 해당 리전의 [EKS 콘솔](#)로 이동하여 클러스터를 선택합니다.
3. 클러스터 정보 페이지에서 개요 탭을 선택합니다.
4. 영역 전환 제목에서 관리 버튼을 선택합니다.
5. EKS Zonal Shift에 대해 활성화 또는 비활성화를 선택합니다.

이제 EKS 클러스터가 ARC에 등록됩니다.

AWS가 가용성 장애 구역을 감지하고 이를 피하도록 하려면 ARC 구역 자동 이동을 구성해야 합니다. 예를 들어, ARC 콘솔에서 이 작업을 수행할 수 있습니다.

다음 단계

- [영역 자동 전환을 활성화](#)하는 방법 알아보기
- 수동으로 [영역별 교대근무 시작하기](#) 방법 알아보기

Amazon EKS에서 액세스 제어가 작동하는 방식 알아보기

Amazon EKS 클러스터에 대한 액세스를 관리하는 방법을 알아봅니다. Amazon EKS를 사용하려면 Kubernetes 및 AWS Identity and Access Management(AWS IAM) 액세스 처리 방법을 모두 알고 있어야 합니다.

이 섹션에는 다음 내용이 포함되어 있습니다.

[the section called “Kubernetes API 액세스”](#) - 애플리케이션 또는 사용자가 Kubernetes API에 인증할 수 있도록 설정하는 방법을 알아봅니다. 액세스 항목, aws-auth ConfigMap 또는 외부 OIDC 공급자를 사용할 수 있습니다.

[the section called “클러스터 리소스 액세스”](#) - Amazon EKS 클러스터와 통신하도록 AWS Management Console을 구성하는 방법을 알아봅니다. 콘솔을 사용하여 클러스터의 Kubernetes 리소스(예: 네임스페이스, 노드 및 포드)를 확인합니다.

[the section called “kubectl을 사용하여 클러스터에 액세스”](#) - Amazon EKS 클러스터와 통신하도록 kubectl을 구성하는 방법을 알아봅니다. AWS CLI를 사용하여 kubeconfig 파일을 생성할 수 있습니다.

[the section called “AWS에 대한 워크로드 액세스”](#) - Kubernetes 서비스 계정과 AWS IAM 역할을 연결하는 방법을 알아봅니다. Pod Identity 또는 서비스 계정에 대한 IAM 역할(IRSA)을 사용할 수 있습니다.

일반적인 작업

- 개발자에게 Kubernetes API에 대한 액세스 권한을 부여합니다. AWS Management Console에서 Kubernetes 리소스를 봅니다.
 - 해결 방법: [액세스 항목을 사용](#)하여 Kubernetes RBAC 권한을 AWS IAM 사용자 또는 역할과 연결합니다.
- AWS 자격 증명을 사용하여 Amazon EKS 클러스터와 통신하도록 kubectl을 구성합니다.
 - 해결 방법: AWS CLI를 사용하여 [kubeconfig 파일을 생성](#)합니다.
- Ping Identity와 같은 외부 ID 제공업체를 사용하여 Kubernetes API에 대해 사용자를 인증합니다.
 - 해결 방법: [외부 OIDC 공급자를 연결](#)합니다.
- Kubernetes 클러스터의 워크로드에 AWS API를 직접적으로 호출하는 기능을 부여합니다.
 - 해결 방법: [Pod Identity를 사용](#)하여 AWS IAM 역할을 Kubernetes 서비스 계정에 연결합니다.

배경

- [Kubernetes 서비스 계정의 작동 방식을 알아봅니다.](#)
- [Kubernetes 역할 기반 액세스 제어\(RBAC\) 모델 검토](#)
- AWS 리소스에 대한 액세스를 관리하는 방법에 자세한 내용은 [AWS IAM 사용 설명서](#)를 참조하세요. 또는 [AWS IAM 사용에 대한 무료 입문 교육](#)을 수강할 수도 있습니다.

EKS Auto Mode 고려 사항

EKS Auto Mode는 EKS Pod Identity 및 EKS EKS 액세스 항목과 통합됩니다.

- EKS Auto Mode는 액세스 항목을 사용하여 EKS 컨트롤 플레인 Kubernetes 권한을 부여합니다. 예를 들어 액세스 정책은 EKS Auto Mode가 네트워크 엔드포인트 및 서비스에 대한 정보를 읽을 수 있게 합니다.
- EKS Auto Mode 클러스터에서는 액세스 항목을 비활성화할 수 없습니다.
- 선택적으로 aws-auth ConfigMap을 활성화할 수 있습니다.
- EKS Auto Mode의 액세스 항목은 자동으로 구성됩니다. 이러한 액세스 항목은 볼 수 있지만 수정할 수는 없습니다.
- NodeClass를 사용하여 사용자 지정 노드 IAM 역할을 생성하는 경우 AmazonEKSAutoNodePolicy 액세스 정책을 사용하여 역할에 대한 액세스 항목을 생성해야 합니다.
- AWS 서비스에 대한 워크로드 권한을 부여하려면 EKS Pod Identity를 사용합니다.
- EKS Auto Mode 클러스터에 Pod Identity 에이전트를 설치할 필요가 없습니다.

IAM 사용자 및 역할에 Kubernetes API에 대한 액세스 권한 부여

클러스터에는 Kubernetes API 엔드포인트가 있습니다. Kubectl은 이 API를 사용합니다. 다음 두 유형의 ID를 사용하여 이 API에 인증할 수 있습니다.

- AWS ID 및 액세스 관리(IAM) 보안 주체(역할 또는 사용자) - 이 유형에는 IAM에 대한 인증이 필요합니다. 자격 증명 소스를 통해 제공된 보안 인증 정보를 사용하여 [IAM](#) 사용자 또는 [페더레이션 ID](#)로 AWS에 로그인할 수 있습니다. 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정한 경우 페더레이션형 ID로만 로그인할 수 있습니다. 페더레이션을 사용하여 AWS에 액세스하면 간접적으로 [역할을 수입](#)합니다. 이 유형의 ID를 사용하는 경우 다음을 수행할 수 있습니다.

- 클러스터의 Kubernetes 객체에서 작업할 수 있도록 Kubernetes 권한을 할당할 수 있습니다. 클러스터의 Kubernetes 객체에 액세스할 수 있도록 IAM 위탁자에 권한을 할당하는 방법에 대한 자세한 내용은 [the section called “권한 부여”](#) 섹션을 참조하세요.
- Amazon EKS API, AWS CLI, AWS CloudFormation, AWS Management Console 또는 eksctl을 사용하여 Amazon EKS 클러스터 및 해당 리소스에서 작업할 수 있도록 IAM 권한을 할당할 수 있습니다. 자세한 내용은 서비스 권한 부여 참조에서 [Amazon Elastic Kubernetes Service에서 정의한 작업](#)을 참조하세요.
- 노드는 IAM 역할을 수입하여 클러스터에 조인합니다. IAM 보안 주체를 사용하는 클러스터에 액세스하는 기능은 [Kubernetes용 AWS IAM Authenticator](#)에서 제공되며, Amazon EKS 컨트롤 플레인에서 실행됩니다.
- 자체 OpenID Connect(OIDC) 제공자의 사용자 - 이 유형은 [OIDC](#) 제공자에 대한 인증이 필요합니다. Amazon EKS 클러스터에서 자체 OIDC 제공자를 설정하는 방법에 대한 자세한 내용은 [the section called “OIDC 제공자 연결”](#) 섹션을 참조하세요. 이 유형의 ID를 사용하는 경우 다음을 수행할 수 있습니다.
- 클러스터의 Kubernetes 객체에서 작업할 수 있도록 Kubernetes 권한을 할당할 수 있습니다.
- Amazon EKS API, AWS CLI, AWS CloudFormation, AWS Management Console 또는 eksctl을 사용하여 Amazon EKS 클러스터 및 해당 리소스에서 작업할 수 있도록 IAM 권한을 할당할 수 없습니다.

클러스터에서는 두 가지 유형의 ID를 모두 사용할 수 있습니다. IAM 인증 방법은 비활성화할 수 없습니다. OIDC 인증 방법은 선택 사항입니다.

IAM 자격 증명을 Kubernetes 권한과 연결

[Kubernetes용 AWS IAM Authenticator](#)는 클러스터의 컨트롤 플레인에 설치됩니다. [AWS Identity and Access Management\(IAM\)](#) 위탁자(역할 및 사용자)가 클러스터의 Kubernetes 리소스에 액세스할 수 있습니다. 다음 방법 중 하나를 사용하여 IAM 위탁자가 클러스터의 Kubernetes 객체에 액세스하도록 허용할 수 있습니다.

- 액세스 항목 생성 - 클러스터가 클러스터의 Kubernetes 버전의 [사전 조건](#) 섹션에 나열된 플랫폼 버전 이상인 경우 이 옵션을 사용하는 것이 좋습니다.

액세스 항목을 사용하여 클러스터 외부에서 IAM 위탁자의 Kubernetes 권한을 관리합니다. EKS API, AWS 명령줄 인터페이스, AWS SDK, AWS CloudFormation 및 AWS Management Console를 사용하여 클러스터에 대한 액세스를 추가하고 관리할 수 있습니다. 즉, 클러스터를 생성할 때 사용한 것과 동일한 도구로 사용자를 관리할 수 있습니다.

시작하려면 [인증 모드를 액세스 항목을 사용하도록 변경](#)에 따라 [기존 aws-auth ConfigMap 항목을 액세스 항목으로 마이그레이션](#)합니다.

- **aws-auth ConfigMap**에 항목 추가-클러스터의 플랫폼 버전이 [사전 조건](#) 섹션에 나열된 버전 이하인 경우 이 옵션을 사용해야 합니다. 클러스터의 플랫폼 버전이 클러스터의 Kubernetes 버전의 [사전 조건](#) 섹션에 나열된 플랫폼 버전 이상이고 ConfigMap에 항목을 추가한 경우 해당 항목을 액세스 항목으로 마이그레이션하는 것이 좋습니다. 그러나 관리형 노드 그룹 또는 Fargate 프로필과 함께 사용되는 IAM 역할 항목과 같이 Amazon EKS가 ConfigMap에 추가한 항목은 마이그레이션할 수 없습니다. 자세한 내용은 [the section called “Kubernetes API 액세스”](#) 섹션을 참조하세요.
- aws-auth ConfigMap 옵션을 사용해야 하는 경우 `eksctl create iamidentitymapping` 명령을 사용하여 ConfigMap에 항목을 추가할 수 있습니다. 자세한 내용은 eksctl 설명서의 [IAM 사용자 및 역할 관리](#)를 참조하세요.

클러스터 인증 모드 설정

각 클러스터에는 인증 모드가 있습니다. 인증 모드는 IAM 위탁자가 클러스터의 Kubernetes 객체에 액세스하도록 허용하는 데 사용할 수 있는 방법을 결정합니다. 세 가지 인증 모드가 있습니다.

Important

액세스 항목 방법을 활성화한 후에는 비활성화할 수 없습니다.

클러스터 생성 중에 ConfigMap 방법을 활성화하지 않으면 나중에 활성화할 수 없습니다. 액세스 항목이 도입되기 전에 생성된 모든 클러스터에는 ConfigMap 메서드가 활성화되어 있습니다.

클러스터에서 하이브리드 노드를 사용하는 경우 API 또는 API_AND_CONFIG_MAP 클러스터 인증 모드를 사용해야 합니다.

클러스터 내부 aws-auth ConfigMap

Amazon EKS 클러스터의 원래 인증 모드입니다. 클러스터를 생성한 IAM 보안 주체는 kubectl을 사용하여 클러스터에 액세스할 수 있는 초기 사용자입니다. 초기 사용자는 aws-auth ConfigMap에서 목록에 다른 사용자를 추가하고 클러스터 내 다른 사용자에게 영향을 주는 권한을 할당해야 합니다. ConfigMap에 관리할 항목이 없기 때문에 이러한 다른 사용자는 초기 사용자를 관리하거나 제거할 수 없습니다.

ConfigMap 및 액세스 항목 모두

이 인증 모드에서는 두 가지 방법을 모두 사용하여 클러스터에 IAM 보안 주체를 추가할 수 있습니다. 각 방법에서는 별도의 항목을 저장합니다. 예를 들어, AWS CLI에서 액세스 항목을 추가하는 경우 `aws-auth ConfigMap`이 업데이트되지 않습니다.

액세스 항목만

이 인증 모드에서는 EKS API, AWS 명령줄 인터페이스, AWS SDK, AWS CloudFormation 및 AWS Management Console을 사용하여 IAM 보안 주체의 클러스터에 대한 액세스를 관리할 수 있습니다.

각 액세스 항목에는 유형이 있으며, 보안 주체를 특정 네임스페이스로 제한하는 액세스 범위와 사전 구성된 재사용 가능한 권한 정책을 설정하는 액세스 정책을 조합하여 사용할 수 있습니다. 또는 STANDARD 유형 및 Kubernetes RBAC 그룹을 사용하여 사용자 지정 권한을 할당할 수 있습니다.

인증 모드	메서드
ConfigMap 만 해당(CONFIG_MAP)	<code>aws-auth ConfigMap</code>
EKS API 및 ConfigMap (API_AND_CONFIG_MAP)	EKS API, AWS 명령줄 인터페이스, AWS SDKs, AWS CloudFormation 및 AWS Management Console 및 <code>aws-auth ConfigMap</code> 의 액세스 항목
EKS API만 해당(API)	EKS API, AWS 명령줄 인터페이스, AWS SDKs, AWS CloudFormation 및 AWS Management Console의 액세스 항목

Note

Amazon EKS Auto Mode에는 액세스 항목이 필요합니다.

EKS 액세스 항목을 사용한 IAM 사용자에게 Kubernetes에 대한 액세스 권한 부여

EKS 액세스 항목이란?

EKS 액세스 항목은 사용자에게 Kubernetes API에 대한 액세스 권한을 부여하는 가장 좋은 방법입니다. 예를 들어 액세스 항목을 사용하여 개발자에게 kubectl을 사용할 수 있는 액세스 권한을 부여할 수 있습니다.

기본적으로 EKS 액세스 항목은 Kubernetes 권한 세트를 IAM 역할과 같은 IAM 자격 증명에 연결합니다. 예를 들어 개발자는 IAM 역할을 수임하고 이를 사용하여 EKS 클러스터에 대해 인증할 수 있습니다.

두 가지 방법으로 액세스 항목에 Kubernetes 권한을 연결할 수 있습니다.

- 액세스 정책을 추가합니다. 액세스 정책은 AWS에서 유지 관리하는 사전 정의된 Kubernetes 권한 템플릿입니다. 자세한 내용은 [the section called “정책 액세스 검토”](#) 단원을 참조하십시오.
- Kubernetes 그룹을 참조합니다. IAM 자격 증명을 Kubernetes 그룹에 연결하는 경우 그룹 권한을 부여하는 Kubernetes 리소스를 생성할 수 있습니다. 자세한 내용은 Kubernetes 문서의 [Using RBAC Authorization\(RBAC 승인 사용\)](#)을 참조하십시오.

장점

Amazon EKS 클러스터 액세스 관리를 사용하면 Amazon EKS API를 통해 직접 Kubernetes 클러스터에 대한 인증 및 권한 부여를 제어할 수 있습니다. 이 기능은 사용자 권한을 관리할 때 AWS 및 Kubernetes API 사이를 전환하지 않아도 되므로 액세스 관리를 간소화합니다. 액세스 항목 및 액세스 정책을 사용하여 클러스터 생성자로부터 클러스터 관리자 권한을 수정하거나 취소하는 기능을 비롯해 AWS IAM 위탁자에 대한 세분화된 권한을 정의할 수 있습니다.

이 기능은 AWS CloudFormation, Terraform 및 AWS CDK와 같은 코드형 인프라(IaC) 도구와 통합되어 클러스터 생성 중에 액세스 구성을 정의할 수 있습니다. 잘못 구성된 경우 직접 Kubernetes API에 액세스하지 않고도 Amazon EKS API를 통해 클러스터 액세스를 복원할 수 있습니다. 이러한 중앙 집중식 접근 방식은 CloudTrail 감사 로깅 및 다중 인증과 같은 기존의 AWS IAM 기능을 활용하여 운영 오버헤드를 줄이고 보안을 개선합니다.

시작하기

1. 사용할 IAM 자격 증명 및 액세스 정책을 결정하세요.
 - [the section called “정책 액세스 검토”](#)
2. 클러스터에서 EKS 액세스 항목을 활성화하세요. 지원되는 플랫폼 버전이 있는지 확인하세요.
 - [the section called “인증 모드”](#)
3. IAM 자격 증명을 Kubernetes 권한에 연결하는 액세스 항목을 생성하세요.

- [the section called “액세스 항목 생성”](#)
4. IAM 자격 증명을 사용하여 클러스터에 대해 인증하세요.
- [the section called “AWS CLI 설정”](#)
 - [the section called “kubectl 및 eksctl 설정”](#)

레거시 클러스터 액세스 구성

이 기능을 도입하기 전에 생성된 클러스터(플랫폼 버전 요구 사항에 지정된 것보다 이전 버전의 초기 플랫폼 버전을 사용하는 클러스터)에서 EKS 액세스 항목을 활성화하는 경우 EKS는 기존 권한을 반영하는 액세스 항목을 자동으로 생성합니다. 이 액세스 항목은 다음을 표시합니다.

- 원래 클러스터를 생성한 IAM 자격 증명
- 클러스터 생성 중에 해당 자격 증명에 부여된 관리 권한

Note

이전에는 이 관리 액세스 권한은 자동으로 부여되었으며 수정할 수 없었습니다. 이제 EKS 액세스 항목을 활성화하면 이 레거시 액세스 구성을 보고 삭제할 수 있습니다.

액세스 정책을 액세스 항목과 연결

STANDARD 유형의 액세스 항목에 하나 이상의 액세스 정책을 할당할 수 있습니다. Amazon EKS는 클러스터에서 제대로 작동하는 데 필요한 권한을 다른 유형의 액세스 항목에 자동으로 부여합니다. Amazon EKS 액세스 정책에는 IAM 권한이 아니라 Kubernetes 권한이 포함됩니다. 액세스 정책을 액세스 항목에 연결하기 전에 각 액세스 정책에 포함된 Kubernetes 권한에 친숙해야 합니다. 자세한 내용은 [the section called “정책 액세스 검토”](#) 단원을 참조하십시오. 요구 사항을 충족하는 액세스 정책이 없는 경우 액세스 정책을 액세스 항목에 연결하지 않습니다. 대신 액세스 항목에 대해 하나 이상의 그룹 이름을 지정하고 Kubernetes 역할 기반 액세스 제어 객체를 생성하고 관리합니다. 자세한 내용은 [the section called “액세스 항목 생성”](#) 단원을 참조하십시오.

- 기존 액세스 항목. 파일을 만들려면 [the section called “액세스 항목 생성”](#) 섹션을 참조하세요.
- ListAccessEntries, DescribeAccessEntry, UpdateAccessEntry, ListAccessPolicies, AssociateAccessPolicy, DisassociateAccessPolicy 권한을 보유한 AWS ID 및 액세스 관리 역할 또는 사용자. 자세한 내용은 서비스 권한 부여 참조에서 [Amazon Elastic Kubernetes Service에서 정의한 작업](#)을 참조하세요.

액세스 정책을 액세스 항목에 연결하기 전에 다음 요구 사항을 고려하세요.

- 각 액세스 항목에 여러 액세스 정책을 연결할 수 있지만 각 정책은 액세스 항목에 한 번만 연결할 수 있습니다. 여러 액세스 정책을 연결하는 경우 액세스 항목의 IAM 보안 주체는 연결된 모든 액세스 정책에 모든 권한을 포함합니다.
- 클러스터의 모든 리소스에 대한 액세스 정책 범위를 지정하거나 하나 이상의 Kubernetes 네임스페이스 이름을 지정하여 범위를 지정할 수 있습니다. 네임스페이스 이름에 와일드카드 문자를 사용할 수 있습니다. 예를 들어, dev-로 시작하는 모든 네임스페이스로 액세스 정책 범위를 지정하려는 경우 네임스페이스 이름으로 dev-*를 지정할 수 있습니다. 클러스터에 네임스페이스가 존재하고 철자가 클러스터의 실제 네임스페이스 이름과 일치하는지 확인합니다. Amazon EKS는 클러스터에 있는 네임스페이스의 철자나 존재 여부를 확인하지 않습니다.
- 액세스 항목에 연결한 후에 액세스 정책의 액세스 범위를 변경할 수 있습니다. 액세스 정책의 범위를 Kubernetes 네임스페이스로 지정한 경우 필요에 따라 연결을 위해 네임스페이스를 추가 및 제거할 수 있습니다.
- 그룹 이름도 지정된 액세스 항목에 액세스 정책을 연결하는 경우 IAM 보안 주체는 연결된 모든 액세스 정책에 모든 권한을 보유합니다. 또한 그룹 이름을 지정하는 Kubernetes Role 및 RoleBinding 객체에 지정된 Kubernetes Role 또는 ClusterRole 객체에 모든 권한을 보유합니다.
- `kubectl auth can-i --list` 명령을 실행하면 명령을 실행할 때 사용한 IAM 위탁자의 액세스 항목에 연결된 액세스 정책에서 할당한 Kubernetes 권한이 표시되지 않습니다. 이 명령은 액세스 항목에 지정한 그룹 이름 또는 사용자 이름에 바인딩하는 Role 또는 ClusterRole 객체에 Kubernetes 권한을 부여하는 경우에만 권한을 표시합니다.
- `kubectl` 명령을 `--as username` 또는 `--as-group group-name` 과 함께 사용하는 등 클러스터에서 Kubernetes 객체와 상호 작용할 때 Kubernetes 사용자 또는 그룹을 가장하면 Kubernetes RBAC 인증을 강제로 사용합니다. 따라서 IAM 보안 주체는 액세스 항목에 연결된 액세스 정책에서 할당한 권한을 보유하지 않습니다. IAM 위탁자가 가장하는 사용자 또는 그룹에 있는 유일한 Kubernetes 권한은 해당 그룹 이름 또는 사용자 이름에 바인딩하는 Kubernetes Role 또는 ClusterRole 객체에 부여한 Kubernetes 권한입니다. IAM 위탁자가 연결된 액세스 정책의 권한을 보유하려면 Kubernetes 사용자 또는 그룹을 가장하지 않습니다. 또한 IAM 위탁자는 액세스 항목에 지정한 그룹 이름 또는 사용자 이름에 바인딩하는 Kubernetes Role 또는 ClusterRole 객체에 부여한 권한도 계속 보유합니다. 자세한 내용은 Kubernetes Documentation의 [User impersonation](#)을 참조하세요.

AWS Management Console 또는 AWS CLI를 사용하여 액세스 정책을 액세스 항목에 연결할 수 있습니다.

AWS Management Console

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 액세스 정책을 연결할 액세스 항목이 있는 클러스터 이름을 선택합니다.
3. 액세스 탭을 선택합니다.
4. 액세스 항목 유형이 표준인 경우 Amazon EKS 액세스 정책을 연결하거나 연결을 해제할 수 있습니다. 액세스 항목 유형이 표준 이외의 항목인 경우 이 옵션을 사용할 수 없습니다.
5. 액세스 정책 연결을 선택합니다.
6. 정책 이름에서 IAM 보안 주체에 부여하려는 권한이 있는 정책을 선택합니다. 이러한 권한이 포함된 정책을 보려면 [the section called “정책 액세스 검토”](#) 섹션을 참조하세요.
7. 액세스 범위에서 액세스 범위를 선택합니다. 클러스터를 선택하면 액세스 정책의 권한이 모든 Kubernetes 네임스페이스의 리소스에 대한 IAM 위탁자에 부여됩니다. Kubernetes 네임스페이스를 선택한 경우 새 네임스페이스 추가를 선택할 수 있습니다. 나타나는 네임스페이스 필드에 클러스터의 Kubernetes 네임스페이스 이름을 입력할 수 있습니다. IAM 보안 주체가 여러 네임스페이스에 대한 권한을 보유하게 하려면 네임스페이스를 여러 개 입력하면 됩니다.
8. 액세스 정책 추가를 선택합니다.

AWS CLI

1. 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.
2. 사용 가능한 액세스 정책을 봅니다.

```
aws eks list-access-policies --output table
```

예제 출력은 다음과 같습니다.

```
-----
|                                     ListAccessPolicies
|
```

```

+-----+
+
||                                     accessPolicies
||
||                                     ||
|+-----+
+-----+|
||                                     arn
name                                     ||
|+-----+
+-----+|
|| {arn-aws}eks::aws:cluster-access-policy/AmazonEKSAAdminPolicy      |
AmazonEKSAAdminPolicy              ||
|| {arn-aws}eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy |
AmazonEKSClusterAdminPolicy        ||
|| {arn-aws}eks::aws:cluster-access-policy/AmazonEKSEditPolicy         |
AmazonEKSEditPolicy                 ||
|| {arn-aws}eks::aws:cluster-access-policy/AmazonEKSVIEWPolicy         |
AmazonEKSVIEWPolicy                 ||
|+-----+
+-----+|

```

이러한 권한이 포함된 정책을 보려면 [the section called “정책 액세스 검토”](#) 섹션을 참조하세요.

3. 기존 액세스 항목을 확인합니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks list-access-entries --cluster-name my-cluster
```

예제 출력은 다음과 같습니다.

```

{
  "accessEntries": [
    "arn:aws:iam::111122223333:role/my-role",
    "arn:aws:iam::111122223333:user/my-user"
  ]
}

```

4. 액세스 정책을 액세스 항목에 연결합니다. 다음 예제에서는 AmazonEKSVIEWPolicy 액세스 정책을 액세스 항목에 연결합니다. *my-role* IAM 역할이 클러스터의 Kubernetes 객체에 액세스를 시도할 때마다 Amazon EKS는 정책에 있는 권한을 사용하여 *my-namespace1* 및 *my-namespace2* Kubernetes 네임스페이스의 Kubernetes 객체에만 액세스할 수 있는 권한을 역할에 부여합니다. *my-cluster*를 클러스터 이름으로, *111122223333*을 AWS 계정 ID로, *my-role*을 Amazon EKS가 Kubernetes 클러스터 객체에 대한 액세스를 승인하려는 IAM 역할의 이름으로 바꿉니다.

```
aws eks associate-access-policy --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/my-role \
    --access-scope type=namespace,namespaces=my-namespace1,my-namespace2 --policy-arn
arn:aws:eks::aws:cluster-access-policy/AmazonEKSVViewPolicy
```

IAM 보안 주체가 클러스터 전체 권한을 갖게 하려면 `type=namespace,namespaces=`*my-namespace1,my-namespace2* 를 `type=cluster`로 바꿉니다. 여러 액세스 정책을 액세스 항목에 연결하려면 각각 고유한 액세스 정책을 사용하여 명령을 여러 번 실행합니다. 연결된 각 액세스 정책의 범위는 고유합니다.

Note

나중에 연결된 액세스 정책의 범위를 변경하려면 새 범위를 사용하여 이전 명령을 다시 실행합니다. 예를 들어, *my-namespace2*를 제거하려는 경우 `type=namespace,namespaces=`*my-namespace1* 만 사용하여 명령을 다시 실행합니다. 범위를 namespace에서 cluster로 변경하려면 `type=cluster`를 사용하여 다시 명령을 실행해 `type=namespace,namespaces=`*my-namespace1,my-namespace2* 를 제거합니다.

5. 액세스 항목에 연결된 액세스 정책을 결정합니다.

```
aws eks list-associated-access-policies --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/my-role
```

예제 출력은 다음과 같습니다.

```
{
  "clusterName": "my-cluster",
  "principalArn": "arn:aws:iam::111122223333",
  "associatedAccessPolicies": [
    {
      "policyArn": "arn:aws:eks::aws:cluster-access-policy/
AmazonEKSVViewPolicy",
      "accessScope": {
        "type": "cluster",
        "namespaces": []
      },
      "associatedAt": "2023-04-17T15:25:21.675000-04:00",
      "modifiedAt": "2023-04-17T15:25:21.675000-04:00"
    }
  ]
}
```

```

    },
    {
      "policyArn": "arn:aws:eks::aws:cluster-access-policy/
AmazonEKSAAdminPolicy",
      "accessScope": {
        "type": "namespace",
        "namespaces": [
          "my-namespace1",
          "my-namespace2"
        ]
      },
      "associatedAt": "2023-04-17T15:02:06.511000-04:00",
      "modifiedAt": "2023-04-17T15:02:06.511000-04:00"
    }
  ]
}

```

이전 예제에서 이 액세스 항목의 IAM 위탁자는 클러스터의 모든 네임스페이스에 대한 보기 권한과 두 Kubernetes 네임스페이스에 대한 관리자 권한을 보유합니다.

6. 액세스 항목에서 액세스 정책 연결을 해제합니다. 이 예제에서 AmazonEKSAAdminPolicy 정책은 액세스 항목과 연결이 해제되었습니다. 하지만 IAM 보안 주체는 *my-namespace1* 및 *my-namespace2* 네임스페이스의 객체에 대한 AmazonEKSVIEWPolicy 액세스 정책에 있는 권한을 보유합니다. 해당 액세스 정책이 액세스 항목과 연결 해제되지 않았기 때문입니다.

```

aws eks disassociate-access-policy --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/my-role \
  --policy-arn arn:aws:eks::aws:cluster-access-policy/AmazonEKSAAdminPolicy

```

사용 가능한 액세스 정책을 나열하려면 [the section called “정책 액세스 검토”](#) 섹션을 참조하세요.

기존 **aws-auth ConfigMap** 항목을 액세스 항목으로 마이그레이션

클러스터의 aws-auth ConfigMap에 항목을 추가한 경우 aws-auth ConfigMap의 기존 항목에 대한 액세스 항목을 생성하는 것이 좋습니다. 액세스 항목을 생성한 후 해당 항목을 ConfigMap에서 제거할 수 있습니다. [액세스 정책](#)을 aws-auth ConfigMap의 항목에 연결할 수 없습니다. 액세스 정책을 IAM 보안 주체에 연결하려면 액세스 항목을 생성합니다.

⚠ Important

클러스터에 [관리형 노드 그룹](#) 또는 [Fargate 프로필](#)을 추가할 때 Amazon EKS에서 생성한 기존 aws-auth ConfigMap 항목을 제거하지 않습니다. Amazon EKS가 ConfigMap에서 생성한 항목을 제거하면 클러스터가 제대로 작동하지 않습니다. 하지만 [자체 관리형](#) 노드 그룹에 대한 액세스 항목을 생성한 후에는 해당 항목을 제거할 수 있습니다.

사전 조건

- 액세스 항목 및 액세스 정책에 익숙해야 합니다. 자세한 내용은 [the section called “권한 부여”](#) 및 [the section called “액세스 정책 연결”](#)(을)를 참조하세요.
- 기존 클러스터의 플랫폼 버전이 [the section called “권한 부여”](#) 주제의 전제 조건에 나열된 버전 이상입니다.
- 장치에 설치된 eksctl 명령줄 도구의 버전 0.210.0 이상 또는 AWS CloudShell이 필요합니다. eksctl을 설치 또는 업그레이드하려면 eksctl 설명서에서 [설치](#)를 참조하세요.
- kube-system 네임스페이스에서 aws-auth ConfigMap을 수정할 Kubernetes 권한.
- CreateAccessEntry 및 ListAccessEntries 권한을 보유한 AWS ID 및 액세스 관리 역할 또는 사용자. 자세한 내용은 서비스 권한 부여 참조에서 [Amazon Elastic Kubernetes Service에서 정의한 작업을](#) 참조하세요.

eksctl

1. aws-auth ConfigMap에서 기존 항목을 봅니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
eksctl get iamidentitymapping --cluster my-cluster
```

예제 출력은 다음과 같습니다.

ARN	USERNAME	GROUPS
	ACCOUNT	
arn:aws:iam::111122223333:role/EKS-my-cluster-Admins	Admins	system:masters
arn:aws:iam::111122223333:role/EKS-my-cluster-my-namespace-Viewers	my-namespace-Viewers	Viewers


```
arn:aws:iam::111122223333:role/EKS-my-cluster-self-managed-ng-1
    system:node:{{EC2PrivateDNSName}}    system:bootstrappers,system:nodes
arn:aws:iam::111122223333:user/my-user
    my-user
arn:aws:iam::111122223333:role/EKS-my-cluster-fargateprofile1
    system:node:{{SessionName}}
    system:bootstrappers,system:nodes,system:node-proxier
arn:aws:iam::111122223333:role/EKS-my-cluster-managed-ng
    system:node:{{EC2PrivateDNSName}}    system:bootstrappers,system:nodes
```

2. 생성 후 이전 출력에서 반환된 모든 ConfigMap 항목에 대해 [the section called “액세스 항목 생성”](#)합니다. 액세스 항목을 생성하는 경우 출력에서 반환된 ARN, USERNAME, GROUPS 및 ACCOUNT에 대해 동일한 값을 지정해야 합니다. 예제 출력에서는 Amazon EKS가 Fargate 프로파일 및 관리형 노드 그룹에 대한 액세스 항목을 생성했기 때문에 마지막 두 항목을 제외한 모든 항목에 대한 액세스 항목을 생성합니다.
3. 생성한 모든 액세스 항목에 대한 항목을 ConfigMap에서 삭제합니다. ConfigMap에서 항목을 삭제하지 않으면 IAM 보안 주체 ARN의 액세스 항목 설정이 ConfigMap 항목보다 우선합니다. **111122223333**을 사용자 AWS 계정 ID로, **EKS-my-cluster-my-namespace-Viewers**를 ConfigMap 항목의 역할 이름으로 바꿉니다. IAM 역할이 아닌 IAM 사용자에게 대한 항목을 제거하는 경우 role을 user로, **EKS-my-cluster-my-namespace-Viewers**를 사용자 이름으로 바꿉니다.

```
eksctl delete iamidentitymapping --arn arn:aws:iam::111122223333:role/EKS-my-cluster-my-namespace-Viewers --cluster my-cluster
```

액세스 정책 권한 검토

액세스 정책에는 Kubernetes verbs(권한) 및 resources가 포함된 rules가 있습니다. 액세스 정책에는 IAM 권한 또는 리소스가 없습니다. Kubernetes Role 및 ClusterRole 객체와 마찬가지로 액세스 정책에는 allow rules만 포함됩니다. 액세스 정책의 콘텐츠는 수정할 수 없습니다. 자체 액세스 정책을 생성할 수 없습니다. 액세스 정책의 권한이 요구 사항에 맞지 않는 경우 Kubernetes RBAC 객체를 생성하고 액세스 항목의 그룹 이름을 지정합니다. 자세한 내용은 [the section called “액세스 항목 생성”](#) 섹션을 참조하세요. 액세스 정책에 포함된 권한은 Kubernetes 사용자 대면 클러스터 역할의 권한과 유사합니다. 자세한 내용은 Kubernetes Documentation의 [User-facing roles](#)를 참조하세요.

해당 콘텐츠를 보려는 액세스 정책을 선택합니다. 각 액세스 정책에 있는 각 테이블의 각 행은 별도의 규칙입니다.

AmazonEKSAAdminPolicy

이 액세스 정책에는 리소스에 대한 대부분의 권한을 IAM 보안 주체에 부여하는 권한이 포함됩니다. 액세스 항목에 연결된 경우 액세스 범위는 일반적으로 하나 이상의 Kubernetes 네임스페이스입니다. IAM 보안 주체가 클러스터의 모든 리소스에 대한 관리자 액세스 권한을 보유하려면 대신 [the section called “AmazonEKSClusterAdminPolicy”](#) 액세스 정책을 액세스 항목에 연결합니다.

ARN- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSAAdminPolicy`

Kubernetes API 그룹	Kubernetes 리소스	Kubernetes 동사(권한)
apps	daemonsets , deployments , deployments/rollback , deployments/scale , replicaset , replicaset/scale , statefulsets , statefulsets/scale	create, delete, deletecollection , patch, update
apps	controllerrevisions , daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , replicaset , replicaset/scale , replicaset/status , statefulsets , statefulsets/scale , statefulsets/status	get, list, watch
authorization.k8s.io	localsubjectaccessreviews	create
autoscaling	horizontalpodautoscalers	create, delete, deletecollection , patch, update
autoscaling	horizontalpodautoscalers , horizonta	get, list, watch

Kubernetes API 그룹	Kubernetes 리소스	Kubernetes 동사(권한)
	lppodautoscalers/status	
batch	cronjobs, jobs	create, delete, deletecollection , patch, update
batch	cronjobs, cronjobs/status , jobs, jobs/status	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch
extensions	daemonsets , deployments , deployments/rollback , deployments/scale , ingresses , networkpolicies , replicaset , replicaset/scale , replicationcontrollers/scale	create, delete, deletecollection , patch, update
extensions	daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , ingresses , ingresses/status , networkpolicies , replicaset , replicaset/scale , replicaset/status , replicationcontrollers/scale	get, list, watch
networking.k8s.io	ingresses , ingresses/status , networkpolicies	get, list, watch

Kubernetes API 그룹	Kubernetes 리소스	Kubernetes 동사(권한)
networking.k8s.io	ingresses , networkpolicies	create, delete, deletecollection , patch, update
policy	poddisruptionbudgets	create, delete, deletecollection , patch, update
policy	poddisruptionbudgets , poddisruptionbudgets/status	get, list, watch
rbac.authorization.k8s.io	rolebindings , roles	create, delete, deletecollection , get, list, patch, update, watch
	configmaps , endpoints , persistentvolumeclaims , persistentvolumeclaims/status , pods, replicationcontrollers , replicationcontrollers/scale , serviceaccounts , services, services/status	get, list, watch
	pods/attach , pods/exec , pods/portforward , pods/proxy , secrets, services/proxy	get, list, watch

Kubernetes API 그룹	Kubernetes 리소스	Kubernetes 동사(권한)
	configmaps , events, persistentvolumeclaims , replicationcontrollers , replicationcontrollers/scale , secrets, serviceaccounts , services, services/proxy	create, delete, deletecollection , patch, update
	pods, pods/attach , pods/exec , pods/port forward , pods/proxy	create, delete, deletecollection , patch, update
	serviceaccounts	impersonate
	bindings, events, limitranges , namespaces/status , pods/log, pods/status , replicationcontrollers/status , resourcequotas , resourcequotas/status	get, list, watch
	namespaces	get, list, watch

AmazonEKSClusterAdminPolicy

이 액세스 정책에는 클러스터에 대한 액세스 권한을 IAM 보안 주체 관리자에게 부여하는 권한이 포함됩니다. 액세스 항목에 연결된 경우 액세스 범위는 일반적으로 Kubernetes 네임스페이스가 아닌 클러스터입니다. IAM 보안 주체의 관리 범위를 더 제한하려면 대신 [the section called “AmazonEKSAAdminPolicy”](#) 액세스 정책을 액세스 항목에 연결하는 방법을 고려합니다.

ARN- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterAdminPolicy`

Kubernetes API 그룹	Kubernetes nonResourceURLs	Kubernetes 리소스	Kubernetes 동사(권한)
*		*	*
	*		*

AmazonEKSAAdminViewPolicy

이 액세스 정책에는 클러스터의 모든 리소스를 나열하고 볼 수 있는 권한을 IAM 보안 주체에 부여하는 권한이 포함되어 있습니다. 여기에는 [Kubernetes 보안 암호](#)가 포함됩니다.

ARN- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSAAdminViewPolicy`

Kubernetes API 그룹	Kubernetes 리소스	Kubernetes 동사(권한)
*	*	get, list, watch

AmazonEKSEditPolicy

이 액세스 정책에는 IAM 위탁자가 대부분의 Kubernetes 리소스를 편집할 수 있는 권한이 포함되어 있습니다.

ARN- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSEditPolicy`

Kubernetes API 그룹	Kubernetes 리소스	Kubernetes 동사(권한)
apps	daemonsets , deployments , deployments/rollback , deployments/scale , replicaset , replicaset/scale, statefulsets , statefulsets/scale	create, delete, deletecollection , patch, update
apps	controllerrevisions , daemonsets , daemonset	get, list, watch

Kubernetes API 그룹	Kubernetes 리소스	Kubernetes 동사(권한)
	s/status , deployments , deployments/scale , deployments/status , replicaset , replicaset/scale , replicaset/status , statefulsets , statefulsets/scale , statefulsets/status	
autoscaling	horizontalpodautoscalers , horizontalpodautoscalers/status	get, list, watch
autoscaling	horizontalpodautoscalers	create, delete, deletecollection , patch, update
batch	cronjobs, jobs	create, delete, deletecollection , patch, update
batch	cronjobs, cronjobs/status , jobs, jobs/status	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch
extensions	daemonsets , deployments , deployments/rollback , deployments/scale , ingresses , networkpolicies , replicaset , replicaset/scale , replicationcontrollers/scale	create, delete, deletecollection , patch, update

Kubernetes API 그룹	Kubernetes 리소스	Kubernetes 동사(권한)
extensions	daemonsets , daemonsets/status , deployments , deployments/scale , deployments/status , ingresses , ingresses/status , networkpolicies , replicaset , replicaset/scale , replicaset/status , replicationcontrollers/scale	get, list, watch
networking.k8s.io	ingresses , networkpolicies	create, delete, deletecollection , patch, update
networking.k8s.io	ingresses , ingresses/status , networkpolicies	get, list, watch
policy	poddisruptionbudgets	create, delete, deletecollection , patch, update
policy	poddisruptionbudgets , poddisruptionbudgets/status	get, list, watch
	namespaces	get, list, watch
	pods/attach , pods/exec , pods/portforward , pods/proxy , secrets, services/proxy	get, list, watch
	serviceaccounts	impersonate

Kubernetes API 그룹	Kubernetes 리소스	Kubernetes 동사(권한)
	pods, pods/attach , pods/exec , pods/port forward , pods/proxy	create, delete, deletecol lection , patch, update
	configmaps , events, persistentvolumecl aims , replicati oncontrollers , replicationcontrol lers/scale , secrets, serviceaccounts , services, services/ proxy	create, delete, deletecol lection , patch, update
	configmaps , endpoints , persistentvolumecl aims , persisten tvolumeclaims/stat us , pods, replicati oncontrollers , replicationcontrol lers/scale , serviceac counts , services, services/status	get, list, watch
	bindings, events, limitranges , namespace s/status , pods/log, pods/status , replicati oncontrollers/stat us , resourcequotas , resourcequotas/sta tus	get, list, watch

AmazonEKSVIEWPolicy

이 액세스 정책에는 IAM 위탁자가 대부분의 Kubernetes 리소스를 볼 수 있는 권한이 포함되어 있습니다.

ARN- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSVIEWPolicy`

Kubernetes API 그룹	Kubernetes 리소스	Kubernetes 동사(권한)
apps	controllerrevisions , daemonsets , daemonset s/status , deployments ts , deployments/scale , deployments/status , replicasets , replicase ts/scale , replicase ts/status , statefuls ets , statefulsets/ scale , statefulsets/ status	get, list, watch
autoscaling	horizontalpodautos calers , horizonta lpodautoscalers/st atus	get, list, watch
batch	cronjobs, cronjobs/ status , jobs, jobs/stat us	get, list, watch
discovery.k8s.io	endpointslices	get, list, watch
extensions	daemonsets , daemonset s/status , deploymen ts , deployments/scale , deployments/status , ingresses , ingresses /status , networkpo licies , replicasets ,	get, list, watch

Kubernetes API 그룹	Kubernetes 리소스	Kubernetes 동사(권한)
	replicasets/scale , replicasets/status , replicationcontrol lers/scale	
networking.k8s.io	ingresses , ingresses /status , networkpo licies	get, list, watch
policy	poddisruptionbudgets , poddisruptionbudge ts/status	get, list, watch
	configmaps , endpoints , persistentvolumecl aims , persisten tvolumeclaims/stat us , pods, replicati oncontrollers , replicationcontrol lers/scale , serviceac counts , services, services/status	get, list, watch
	bindings, events, limitranges , namespace s/status , pods/log, pods/status , replicati oncontrollers/stat us , resourcequotas , resourcequotas/status	get, list, watch
	namespaces	get, list, watch

AmazonEKSAutoNodePolicy

ARN- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSAutoNodePolicy`

이 정책에는 Amazon EKS 구성 요소가 다음 태스크를 완료할 수 있도록 허용하는 다음과 같은 권한이 포함되어 있습니다.

- kube-proxy-네트워크 엔드포인트 및 서비스를 모니터링하고 관련 이벤트를 관리합니다. 이렇게 하면 클러스터 전반의 네트워크 프록시 기능이 활성화됩니다.
- ipamd - AWS VPC 네트워킹 리소스 및 컨테이너 네트워크 인터페이스(CNI)를 관리합니다. 이렇게 하면 IP 주소 관리 대몬이 포드 네트워킹을 처리할 수 있습니다.
- coredns-엔드포인트 및 서비스와 같은 서비스 검색 리소스에 액세스합니다. 이렇게 하면 클러스터 내에서 DNS 확인이 활성화됩니다.
- ebs-csi-driver-Amazon EBS 볼륨에 대한 스토리지 관련 리소스를 사용합니다. 이를 통해 영구 볼륨의 동적 프로비저닝 및 연결이 가능합니다.
- neuron - AWS Neuron 디바이스의 노드 및 포드를 모니터링합니다. 이를 통해 AWS Inferentia 및 Trainium 액셀러레이터를 관리할 수 있습니다.
- node-monitoring-agent-노드 진단 및 이벤트에 액세스합니다. 이렇게 하면 클러스터 상태 모니터링 및 진단 수집이 활성화됩니다.

각 구성 요소는 전용 서비스 계정을 사용하며 특정 함수에 필요한 권한으로만 제한됩니다.

NodeClass에서 노드 IAM 역할을 수동으로 지정하는 경우 새 노드 IAM 역할이 액세스 정책과 연결하는 액세스 항목을 생성해야 합니다.

AmazonEKSBlockStoragePolicy

Note

이 정책은 AWS 서비스 연결 역할에만 지정되며 고객 관리형 역할에는 사용할 수 없습니다.

ARN- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSBlockStoragePolicy`

이 정책에는 Amazon EKS가 스토리지 작업에 대한 리더 선택 및 조정 리소스를 관리하는 권한이 포함되어 있습니다.

- coordination.k8s.io-리더 선택을 위해 임대 객체를 생성하고 관리합니다. 이에 따라 EKS 스토리지 구성 요소는 리더 선택 메커니즘을 통해 클러스터 전체에서 활동을 조정할 수 있습니다.

이 정책은 클러스터의 다른 조정 리소스에 대한 액세스 충돌을 방지하기 위해 EKS 스토리지 구성 요소가 사용하는 특정 임대 리소스로 범위가 지정됩니다.

Amazon EKS는 Auto Mode가 활성화되면 클러스터 IAM 역할에 대한 이 액세스 정책으로 액세스 항목을 자동으로 생성하여 블록 스토리지 기능이 제대로 작동하는 데 필요한 권한이 있는지 확인합니다.

AmazonEKSLoadBalancingPolicy

ARN- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSLoadBalancingPolicy`

이 정책에는 Amazon EKS가 로드 밸런싱을 위한 리더 선택 리소스를 관리하는 권한이 포함되어 있습니다.

- `coordination.k8s.io`-리더 선택을 위해 임대 객체를 생성하고 관리합니다. 이를 통해 EKS 로드 밸런싱 구성 요소는 리더를 선택하여 여러 복제본에서 활동을 조정할 수 있습니다.

이 정책은 특히 로드 밸런싱 임대 리소스에 적용되어 클러스터의 다른 임대 리소스에 대한 액세스를 방지하면서 적절한 조정을 보장합니다.

Amazon EKS는 Auto Mode가 활성화되면 클러스터 IAM 역할에 대한 이 액세스 정책으로 액세스 항목을 자동으로 생성하여 네트워킹 기능이 제대로 작동하는 데 필요한 권한이 있는지 확인합니다.

AmazonEKSNetworkingPolicy

ARN- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSNetworkingPolicy`

이 정책에는 Amazon EKS가 네트워킹을 위한 리더 선택 리소스를 관리하는 권한이 포함되어 있습니다.

- `coordination.k8s.io`-리더 선택을 위해 임대 객체를 생성하고 관리합니다. 이를 통해 EKS 네트워킹 구성 요소는 리더를 선택하여 IP 주소 할당 활동을 조정할 수 있습니다.

이 정책은 특히 네트워킹 임대 리소스에 적용되어 클러스터의 다른 임대 리소스에 대한 액세스를 방지하면서 적절한 조정을 보장합니다.

Amazon EKS는 Auto Mode가 활성화되면 클러스터 IAM 역할에 대한 이 액세스 정책으로 액세스 항목을 자동으로 생성하여 네트워킹 기능이 제대로 작동하는 데 필요한 권한이 있는지 확인합니다.

AmazonEKSComputePolicy

Note

이 정책은 AWS 서비스 연결 역할에만 지정되며 고객 관리형 역할에는 사용할 수 없습니다.

ARN- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSComputePolicy`

이 정책에는 Amazon EKS가 컴퓨팅 작업을 위한 리더 선택 리소스를 관리하는 권한이 포함되어 있습니다.

- `coordination.k8s.io`-리더 선택을 위해 임대 객체를 생성하고 관리합니다. 이를 통해 EKS 컴퓨팅 구성 요소는 리더를 선택하여 노드 조정 활동을 조율할 수 있습니다.

이 정책은 클러스터의 모든 임대 리소스에 대한 기본 읽기 액세스(get, watch)를 허용하는 동시에 컴퓨팅 관리 임대 리소스에 특별히 적용됩니다.

Amazon EKS는 Auto Mode가 활성화되면 클러스터 IAM 역할에 대한 이 액세스 정책으로 액세스 항목을 자동으로 생성하여 네트워킹 기능이 제대로 작동하는 데 필요한 권한이 있는지 확인합니다.

AmazonEKSBlockStorageClusterPolicy

Note

이 정책은 AWS 서비스 연결 역할에만 지정되며 고객 관리형 역할에는 사용할 수 없습니다.

ARN- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSBlockStorageClusterPolicy`

이 정책은 Amazon EKS Auto Mode의 블록 스토리지 기능에 필요한 권한을 부여합니다. 이를 통해 Amazon EKS 클러스터 내에서 블록 스토리지 리소스를 효율적으로 관리할 수 있습니다. 다음 정책에는 이러한 권한이 포함됩니다.

CSI 드라이버 관리:

- 특히 블록 스토리지를 위한 CSI 드라이버를 생성, 읽기, 업데이트 및 삭제합니다.

볼륨 관리:

- 영구 볼륨을 나열, 감시, 생성, 업데이트, 패치, 삭제합니다.
- 영구 볼륨 클레임을 나열, 감시, 업데이트합니다.
- 영구 볼륨 클레임 상태를 패치합니다.

노드 및 포드 상호 작용:

- 노드 및 포드 정보를 읽습니다.
- 스토리지 작업과 관련된 이벤트를 관리합니다.

스토리지 클래스 및 속성:

- 스토리지 클래스 및 CSI 노드를 읽습니다.
- 볼륨 속성 클래스를 읽습니다.

볼륨 연결:

- 볼륨 연결 및 해당 상태를 나열, 감시, 수정합니다.

스냅샷 작업:

- 볼륨 스냅샷, 스냅샷 콘텐츠, 스냅샷 클래스를 관리합니다.
- 볼륨 그룹 스냅샷 및 관련 리소스에 대한 작업을 처리합니다.

이 정책은 Auto Mode에서 실행되는 Amazon EKS 클러스터 내에서 포괄적인 블록 스토리지 관리를 지원하도록 설계되었습니다. 블록 스토리지 볼륨의 프로비저닝, 연결, 크기 조정, 스냅샷 생성 등 다양한 작업에 대한 권한을 결합합니다.

Amazon EKS는 Auto Mode가 활성화되면 클러스터 IAM 역할에 대한 이 액세스 정책으로 액세스 항목을 자동으로 생성하여 블록 스토리지 기능이 제대로 작동하는 데 필요한 권한이 있는지 확인합니다.

AmazonEKSClusterPolicy

Note

이 정책은 AWS 서비스 연결 역할에만 지정되며 고객 관리형 역할에는 사용할 수 없습니다.

ARN- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSComputeClusterPolicy`

이 정책은 Amazon EKS Auto Mode의 컴퓨팅 관리 기능에 필요한 권한을 부여합니다. Amazon EKS 클러스터 내에서 컴퓨팅 리소스를 효율적으로 오케스트레이션하고 확장할 수 있습니다. 다음 정책에는 이러한 권한이 포함됩니다.

노드 관리:

- NodePools 및 NodeClaims의 상태를 생성, 읽기, 업데이트, 삭제, 관리합니다.
- 생성, 수정, 삭제를 포함하여 NodeClasses를 관리합니다.

예약 및 리소스 관리:

- 포드, 노드, 영구 볼륨, 영구 볼륨 클레임, 복제 컨트롤러, 네임스페이스에 대한 읽기 액세스입니다.
- 스토리지 클래스, CSI 노드, 볼륨 연결에 대한 읽기 액세스입니다.
- 배포, 대몬 세트, 복제본 세트, 상태 저장 세트를 나열하고 감시합니다.
- 포드 중단 예산을 읽습니다.

이벤트 처리:

- 클러스터 이벤트를 생성, 읽기, 관리합니다.

노드 디프로비저닝 및 포드 제거:

- 노드를 업데이트, 패치, 삭제합니다.
- 포드 제거를 생성하고 필요한 경우 포드를 삭제합니다.

사용자 지정 리소스 정의(CRD) 관리:

- 새 CRD를 생성합니다.
- 노드 관리와 관련된 특정 CRD(NodeClasses, NodePools, NodeClaims, NodeDiagnostics)를 관리합니다.

이 정책은 Auto Mode에서 실행되는 Amazon EKS 클러스터 내에서 포괄적인 컴퓨팅 관리를 지원하도록 설계되었습니다. 노드 프로비저닝, 예약, 조정, 리소스 최적화를 비롯한 다양한 작업에 대한 권한을 결합합니다.

Amazon EKS는 Auto Mode가 활성화되면 클러스터 IAM 역할에 대한 이 액세스 정책으로 액세스 항목을 자동으로 생성하여 컴퓨팅 관리 기능이 제대로 작동하는 데 필요한 권한이 있는지 확인합니다.

AmazonEKSLoadBalancingClusterPolicy

ARN- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSLoadBalancingClusterPolicy`

이 정책은 Amazon EKS Auto Mode의 로드 밸런싱 기능에 필요한 권한을 부여합니다. Amazon EKS 클러스터 내에서 로드 밸런싱 리소스를 효율적으로 관리하고 구성할 수 있습니다. 다음 정책에는 이러한 권한이 포함됩니다.

이벤트 및 리소스 관리:

- 이벤트를 생성하고 패치합니다.
- 포드, 노드, 엔드포인트, 네임스페이스에 대한 읽기 액세스입니다.
- 포드 상태를 업데이트합니다.

서비스 및 수신 관리:

- 서비스 및 해당 상태를 전체적으로 관리합니다.
- 수신 및 상태를 포괄적으로 제어합니다.
- 엔드포인트 조각 및 수신 클래스에 대한 읽기 액세스입니다.

대상 그룹 바인딩:

- 대상 그룹 바인딩과 해당 상태를 생성하고 수정합니다.
- 수신 클래스 파라미터에 대한 읽기 액세스입니다.

사용자 지정 리소스 정의(CRD) 관리:

- 모든 CRD를 생성하고 읽습니다.
- `targetgroupbindings.eks.amazonaws.com` 및 `ingressclassparams.eks.amazonaws.com` CRD에 대한 특정 관리입니다.

웹훅 구성:

- 웹훅의 변경 및 검증 구성을 생성하고 읽습니다.
- eks-load-balancing-webhook 구성을 관리합니다.

이 정책은 Auto Mode에서 실행되는 Amazon EKS 클러스터 내에서 포괄적인 로드 밸런싱 관리를 지원하도록 설계되었습니다. 서비스 노출, 수신 라우팅, AWS 로드 밸런싱 서비스와의 통합을 비롯한 다양한 작업에 대한 권한을 결합합니다.

Amazon EKS는 Auto Mode가 활성화되면 클러스터 IAM 역할에 대한 이 액세스 정책으로 액세스 항목을 자동으로 생성하여 로드 밸런싱 기능이 제대로 작동하는 데 필요한 권한이 있는지 확인합니다.

AmazonEKSNetworkingClusterPolicy

ARN- `arn:aws:eks::aws:cluster-access-policy/AmazonEKSNetworkingClusterPolicy`

AmazonEKSNetworkingClusterPolicy

이 정책은 Amazon EKS Auto Mode의 네트워킹 기능에 필요한 권한을 부여합니다. Amazon EKS 클러스터 내에서 네트워킹 리소스를 효율적으로 관리하고 구성할 수 있습니다. 다음 정책에는 이러한 권한이 포함됩니다.

노드 및 포드 관리:

- NodeClasses 및 해당 상태에 대한 읽기 액세스입니다.
- NodeClaims 및 해당 상태에 대한 읽기 액세스입니다.
- 포드에 대한 읽기 액세스입니다.

CNI 노드 관리:

- 생성, 읽기, 업데이트, 삭제, 패치를 포함한 CNINodes 및 해당 상태에 대한 권한입니다.

사용자 지정 리소스 정의(CRD) 관리:

- 모든 CRD를 생성하고 읽습니다.
- `cninodes.eks.amazonaws.com` CRD에 대한 특정 관리(업데이트, 패치, 삭제)입니다.

이벤트 관리:

- 이벤트를 생성하고 패치합니다.

이 정책은 Auto Mode에서 실행되는 Amazon EKS 클러스터 내에서 포괄적인 네트워킹 관리를 지원하도록 설계되었습니다. 노드 네트워킹 구성, CNI(컨테이너 네트워크 인터페이스) 관리, 관련 사용자 지정 리소스 처리를 비롯한 다양한 작업에 대한 권한을 결합합니다.

이 정책은 네트워킹 구성 요소가 노드 관련 리소스와 상호 작용하고, CNI별 노드 구성을 관리하고, 클러스터의 네트워킹 작업에 중요한 사용자 지정 리소스를 처리하도록 허용합니다.

Amazon EKS는 Auto Mode가 활성화되면 클러스터 IAM 역할에 대한 이 액세스 정책으로 액세스 항목을 자동으로 생성하여 네트워킹 기능이 제대로 작동하는 데 필요한 권한이 있는지 확인합니다.

AmazonEKSHybridPolicy

Note

이 정책은 AWS 서비스 연결 역할에만 지정되며 고객 관리형 역할에는 사용할 수 없습니다.

이 액세스 정책에는 클러스터의 노드에 대한 EKS 액세스 권한을 부여하는 권한이 포함되어 있습니다. 액세스 항목에 연결된 경우 액세스 범위는 일반적으로 Kubernetes 네임스페이스가 아닌 클러스터입니다. 이 정책은 Amazon EKS Hybrid Nodes에서 사용됩니다.

ARN-arn:aws:eks::aws:cluster-access-policy/AmazonEKSHybridPolicy

Kubernetes API 그룹	Kubernetes nonResourceURLs	Kubernetes 리소스	Kubernetes 동사(권한)
*		nodes	list

AmazonEKSClusterInsightsPolicy

Note

이 정책은 AWS 서비스 연결 역할에만 지정되며 고객 관리형 역할에는 사용할 수 없습니다.

ARN- arn:aws:eks::aws:cluster-access-policy/AmazonEKSClusterInsightsPolicy

이 정책은 Amazon EKS Cluster Insights 기능에 대한 읽기 전용 권한을 부여합니다. 다음 정책에는 이러한 권한이 포함됩니다.

노드 액세스: - 클러스터 노드 나열 및 보기 - 노드 상태 정보 읽기

DaemonSet 액세스: - kube-proxy 구성에 대한 읽기 액세스

이 정책은 Cluster Insights용 EKS 서비스에 의해 자동으로 관리됩니다. 자세한 내용은 [the section called “클러스터 인사이트”](#) 섹션을 참조하세요.

액세스 정책 업데이트

액세스 정책이 도입된 이후 액세스 정책에 대한 업데이트 세부 정보를 확인합니다. 이 페이지의 변경 사항에 대한 자동 알림을 받아보려면 [문서 기록](#)에서 RSS 피드를 구독하세요.

변경 사항	설명	날짜
EKS Cluster Insights에 대한 정책 추가	AmazonEKSClusterInsightsPolicy 게시	2024년 12월 2일
Amazon EKS 하이브리드에 대한 정책 추가	AmazonEKSHybridPolicy 게시	2024년 12월 2일
Amazon EKS Auto Mode에 대한 정책 추가	이러한 액세스 정책은 클러스터 IAM 역할 및 노드 IAM 역할에 Kubernetes API를 직접 호출 권한을 부여합니다. AWS는 이를 사용하여 스토리지, 컴퓨팅, 네트워킹 리소스에 대한 일상적인 작업을 자동화합니다.	2024년 12월 2일
AmazonEKSAAdminViewPolicy 추가	비밀과 같은 리소스를 포함하여 확장된 보기 액세스에 대한 새 정책을 추가합니다.	2024년 4월 23일
액세스 정책이 도입되었습니다.	Amazon EKS에서 액세스 정책을 도입했습니다.	2023년 5월 29일

액세스 항목을 사용하도록 인증 모드 변경

액세스 항목 사용을 시작하려면 클러스터의 인증 모드를 API_AND_CONFIG_MAP 또는 API 모드로 변경해야 합니다. 그러면 액세스 항목에 대한 API가 추가됩니다.

AWS 콘솔

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 관리형 노드 그룹을 생성하려는 클러스터의 이름을 선택합니다.
3. 액세스 탭을 선택합니다.
4. 인증 모드는 클러스터의 현재 인증 모드를 표시합니다. EKS API 모드로 표시되면 이미 액세스 항목을 추가할 수 있으며 나머지 단계는 건너뛰어도 됩니다.
5. 액세스 관리를 선택합니다.
6. 클러스터 인증 모드에서 EKS API로 모드를 선택합니다. EKS API 및 액세스 항목을 제거한 모드로 인증 모드를 다시 변경할 수 없습니다.
7. 변경 사항 저장을 선택합니다. Amazon EKS가 클러스터 업데이트를 시작하고 클러스터가 업데이트 중 상태로 변경되며 변경 사항은 업데이트 기록 탭에 기록됩니다.
8. 클러스터가 활성 상태로 돌아갈 때까지 기다립니다. 클러스터가 활성 상태이면 [the section called “액세스 항목 생성”](#)의 단계에 따라 IAM 위탁자에서 클러스터에 대한 액세스를 추가할 수 있습니다.

AWS CLI

1. AWS 명령줄 인터페이스 사용 설명서의 [설치](#)에 설명된 대로 AWS CLI를 설치합니다.
2. 다음 명령을 실행합니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다. 이 ConfigMap 방법을 영구적으로 비활성화하려면 API_AND_CONFIG_MAP을 API로 바꿉니다.

Amazon EKS가 클러스터 업데이트를 시작하고 클러스터가 UPDATING 상태로 변경되며 변경 사항은 `aws eks list-updates`에 기록됩니다.

```
aws eks update-cluster-config --name my-cluster --access-config
authenticationMode=API_AND_CONFIG_MAP
```

3. 클러스터가 활성 상태로 돌아갈 때까지 기다립니다. 클러스터가 활성 상태이면 [the section called “액세스 항목 생성”](#)의 단계에 따라 IAM 위탁자에서 클러스터에 대한 액세스를 추가할 수 있습니다.

필요한 플랫폼 버전

액세스 항목을 사용하려면 클러스터의 플랫폼 버전이 다음 표에 나열된 버전과 동일한 버전 또는 이후 버전이거나 Kubernetes 버전이 표에 나열된 버전보다 이후 버전이어야 합니다. Kubernetes 버전이 나열되지 않은 경우 모든 플랫폼 버전이 액세스 항목을 지원합니다.

Kubernetes 버전	플랫폼 버전
1.30	eks.2
1.29	eks.1
1.28	eks.6
1.27	eks.10
1.26	eks.11

자세한 내용은 [the section called “플랫폼 버전”](#) 섹션을 참조하세요.

액세스 항목 생성

액세스 항목을 생성하기 전에 다음을 고려합니다.

- 올바르게 설정된 인증 모드입니다. [the section called “인증 모드”](#)을(를) 참조하세요.
- 액세스 항목에는 기존 IAM 보안 주체 하나의 Amazon 리소스 이름(ARN)만 포함됩니다. IAM 보안 주체는 둘 이상의 액세스 항목에 포함될 수 없습니다. 지정하는 ARN에 대한 추가 고려 사항:
 - IAM 모범 사례에서는 장기 보안 인증 정보를 보유한 IAM 사용자 대신, 단기 보안 인증 정보를 보유한 IAM 역할을 사용하여 클러스터에 액세스하도록 권장합니다. 자세한 내용은 IAM 사용 설명서에서 [임시 자격 증명을 사용하여 AWS에 액세스하려면 인간 사용자가 ID 공급자와의 페더레이션을 사용하도록 요구](#)를 참조하세요.
 - IAM 역할에 대한 ARN인 경우 경로를 포함할 수 있습니다. aws-auth ConfigMap 항목의 ARN은 경로를 포함할 수 없습니다. 예를 들어, ARN은 `arn:aws:iam::<111122223333>:role/<development/apps/my-role>` 또는 `arn:aws:iam::<111122223333>:role/<my-role>`일 수 있습니다.
 - 액세스 항목 유형이 STANDARD 이외의 유형인 경우,(유형에 대한 다음 고려 사항 참조) ARN은 클러스터가 속한 같은 AWS 계정에 있어야 합니다. 유형이 STANDARD인 경우 ARN은 클러스터가 있는 계정과 같거나 다른 AWS 계정에 있을 수 있습니다.

- 액세스 항목이 생성된 후에는 IAM 보안 주체를 변경할 수 없습니다.
- 이 ARN으로 IAM 보안 주체를 삭제해도 액세스 항목은 자동으로 삭제되지 않습니다. 삭제하는 IAM 보안 주체에 대한 ARN을 사용하여 액세스 항목을 삭제하는 것이 좋습니다. 액세스 항목을 삭제하지 않고 IAM 보안 주체를 다시 생성하면 ARN이 같아도 액세스 항목이 작동하지 않습니다. 다시 생성된 IAM 보안 주체에서 ARN이 동일하더라도 roleID 또는 userID(`aws sts get-caller-identity` AWS CLI 명령으로 확인할 수 있음)는 다시 생성된 IAM 보안 주체의 해당 항목과 다릅니다(원래 IAM 보안 주체의 항목임). 액세스 항목의 IAM 보안 주체 roleID 또는 userID가 표시되지 않더라도 Amazon EKS는 액세스 항목과 함께 이를 저장합니다.
- 각 액세스 항목에는 유형이 있습니다. 유형을 지정하지 않으면 Amazon EKS에서 자동으로 유형을 STANDARD로 설정합니다.
 - EC2_LINUX - Linux 또는 Bottlerocket 자체 관리형 노드와 함께 사용되는 IAM 역할의 경우
 - EC2_WINDOWS - Windows 자체 관리형 노드와 함께 사용되는 IAM 역할의 경우
 - FARGATE_LINUX - AWS Fargate(Fargate)와 함께 사용되는 IAM 역할의 경우
 - HYBRID_LINUX - 하이브리드 노드와 함께 사용되는 IAM 역할의 경우
 - STANDARD - 지정되지 않은 경우 기본 유형
 - EC2 - EKS 자동 모드 사용자 지정 노드 클래스의 경우. 자세한 내용은 [the section called “노드 클래스 액세스 항목 생성”](#) 섹션을 참조하세요.
- 액세스 항목을 생성한 후에는 유형을 변경할 수 없습니다.
- 관리형 노드 그룹 또는 Fargate 프로파일에 사용되는 IAM 역할에 대한 액세스 항목을 생성하지 않아도 됩니다. EKS는 액세스 항목을 생성하거나(활성화된 경우) 인증 구성 맵을 업데이트합니다(액세스 항목을 사용할 수 없는 경우).
- 액세스 항목 유형이 STANDARD인 경우 액세스 항목의 사용자 이름을 지정할 수 있습니다. 사용자 이름 값을 지정하지 않는 경우, Amazon EKS는 지정한 IAM 보안 주체(IAM 역할 또는 IAM 사용자) 및 액세스 항목의 유형에 따라 다음 값 중 하나를 자동으로 설정합니다. 고유한 사용자 이름을 지정해야 하는 특별한 이유가 없는 한, 사용자 이름을 지정하지 말고 Amazon EKS에서 자동으로 생성하게 두는 것이 좋습니다. 고유한 사용자 이름을 지정하는 경우:
 - `system:`, `eks:`, `aws:`, `amazon:` 또는 `iam:`으로 시작할 수 없습니다.
 - IAM 역할에 대한 사용자 이름인 경우 사용자 이름 끝에 `{{SessionName}}` 또는 `{{SessionNameRaw}}`를 추가하는 것이 좋습니다. 사용자 이름에 `{{SessionName}}` 또는 `{{SessionNameRaw}}`를 추가하는 경우 사용자 이름에서 `{{SessionName}}` 앞에 콜론을 포함해야 합니다. 이 역할을 수임하면 역할을 수임할 때 지정한 AWS STS 세션 이름이 클러스터에 자동으로 전달되고 CloudTrail 로그에 표시됩니다. 예를 들어, `john{{SessionName}}`의 사용자 이름을 포함할 수 없습니다. 사용자 이름은 `:john{{SessionName}}` 또는 `jo:hn{{SessionName}}`이어야 합니다. 콜론은 `{{SessionName}}` 앞에 와야 합니다. 다음 테

이블에서 Amazon EKS가 생성한 사용자 이름에는 ARN이 포함됩니다. ARN에는 콜론이 포함되므로 이 요구 사항을 충족합니다. 사용자 이름에 {{SessionName}}을 포함하지 않는 경우 콜론은 필요하지 않습니다. {{SessionName}}에서는 세션 이름의 특수 문자 '@'가 '-'로 대체됩니다. {{SessionNameRaw}}는 세션 이름의 모든 특수 문자를 유지합니다.

IAM 보안 주체 유형	유형	Amazon EKS가 자동으로 설정하는 사용자 이름 값
User	STANDARD	사용자의 ARN. 예시: arn:aws:iam::<111122223333>:user/<my-user>
역할	STANDARD	역할을 수입한 경우 STS ARN. Amazon EKS가 역할에 {{SessionName}}을 추가합니다. 예시: arn:aws:sts::<111122223333>:assumed-role/<my-role>/{{SessionName}} 지정한 역할의 ARN에 경로가 포함된 경우 Amazon EKS는 생성된 사용자 이름에서 해당 경로를 제거합니다.
역할	EC2_LINUX 또는 EC2_Windows	system:node:{{EC2PrivateDNSName}}
역할	FARGATE_LINUX	system:node:{{SessionName}}
역할	HYBRID_LINUX	system:node:{{SessionName}}

액세스 항목을 생성한 후 사용자 이름을 변경할 수 있습니다.

- 액세스 항목 유형이 STANDARD이고 Kubernetes RBAC 권한 부여를 사용하려는 경우 액세스 항목에 하나 이상의 그룹 이름을 추가할 수 있습니다. 액세스 항목을 생성한 후 그룹 이름을 추가 및 제거할 수 있습니다. IAM 위탁자가 클러스터의 Kubernetes 객체에 액세스할 수 있으려면 Kubernetes 역할 기반 권한 부여(RBAC) 객체를 생성하고 관리해야 합니다. 클러스터에서 그룹 이름을 kind: Group의 subject로 지정하는 Kubernetes RoleBinding 또는 ClusterRoleBinding 객체를 생성합니다. Kubernetes는 Kubernetes Role 또는 ClusterRole 객체에서 지정한 모든 클러스터 객체에 대한 IAM 주체 액세스를 승인하고 바인딩의 roleRef에서도 이를 지정합니다. 그룹 이름을 지정하는 경우 Kubernetes 역할 기반 인증(RBAC) 객체에 익숙해지는 것이 좋습니다. 자세한 내용은 Kubernetes 문서의 [Using RBAC Authorization\(RBAC 승인 사용\)](#)을 참조하십시오.

Important

Amazon EKS는 클러스터에 있는 Kubernetes RBAC 객체에 지정한 그룹 이름이 포함되었는지 확인하지 않습니다. 예를 들어 현재 존재하지 않는 그룹에 대한 액세스 항목을 생성하는 경우 EKS는 오류를 반환하는 대신 해당 그룹을 생성합니다.

Kubernetes에서 클러스터의 Kubernetes 객체에 대한 액세스 권한을 IAM 위탁자에 부여하는 대신 또는 이 방법 외에도 Amazon EKS 액세스 정책을 액세스 항목에 연결할 수 있습니다. Amazon EKS는 IAM 위탁자에 액세스 정책의 권한과 함께 클러스터의 Kubernetes 객체에 액세스할 수 있는 권한을 부여합니다. 지정한 Kubernetes 네임스페이스로 액세스 정책의 권한 범위를 지정할 수 있습니다. 액세스 정책을 사용할 경우 Kubernetes RBAC 객체를 관리할 필요가 없습니다. 자세한 내용은 [the section called “액세스 정책 연결”](#) 섹션을 참조하세요.

- 유형이 EC2_LINUX 또는 EC2_Windows인 액세스 항목을 생성하는 경우 액세스 항목을 생성하는 IAM 보안 주체에 iam:PassRole 권한이 있어야 합니다. 자세한 내용은 IAM 사용 설명서에서 [사용자에게 AWS 서비스 역할을 전달할 수 있는 권한 부여](#)를 참조하십시오.
- 표준 [IAM 동작](#)과 마찬가지로, 액세스 항목 생성 및 업데이트는 실질적으로 일관되며 초기 API 직접 호출이 성공적으로 반환된 후 효력이 발생하는 데 몇 초가 걸릴 수 있습니다. 이러한 잠재적 지연을 고려하도록 애플리케이션을 설계해야 합니다. 액세스 항목 생성 또는 업데이트는 애플리케이션의 중요한 고가용성 코드 경로에 포함하지 않는 것이 좋습니다. 대신 자주 실행하지 않는 별도의 초기화 루틴이나 설정 루틴에서 이를 변경하세요. 또한 프로덕션 워크플로우에서 변경 사항을 적용하기 전에 변경 사항이 전파되었는지 확인하세요.

- 액세스 항목은 [서비스 연결 역할](#)을 지원하지 않습니다. 보안 주체 ARN이 서비스 연결 역할인 경우 액세스 항목을 생성할 수 없습니다. `arn:aws:iam::*:role/aws-service-role/*` 형식의 ARN으로 서비스 연결 역할을 식별할 수 있습니다.

AWS Management Console 또는 AWS CLI를 사용하여 액세스 항목을 생성할 수 있습니다.

AWS Management Console

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 관리형 노드 그룹을 생성하려는 클러스터의 이름을 선택합니다.
3. 액세스 탭을 선택합니다.
4. 액세스 항목 생성을 선택합니다.
5. IAM 보안 주체의 경우 기존 IAM 역할 또는 사용자를 선택합니다. IAM 모범 사례에서는 장기 보안 인증 정보를 보유한 IAM 사용자 대신, 단기 보안 인증 정보를 보유한 IAM 역할을 사용하여 클러스터에 액세스하도록 권장합니다. 자세한 내용은 IAM 사용 설명서에서 [임시 자격 증명을 사용하여 AWS에 액세스하려면 인간 사용자가 ID 공급자와의 페더레이션을 사용하도록 요구](#)를 참조하세요.
6. 자체 관리형 Amazon EC2 노드에 사용되는 노드 역할에 대한 액세스 항목인 경우 유형에서 EC2 Linux 또는 EC2 Windows를 선택합니다. 그렇지 않으면 기본값(표준)을 그대로 수락합니다.
7. 선택한 유형이 표준이고 사용자 이름을 지정하려면 사용자 이름을 입력합니다.
8. 선택한 유형이 표준이고 IAM 위탁자에 대해 Kubernetes RBAC 권한 부여를 사용하려는 경우 그룹에 대한 하나 이상의 이름을 지정합니다. 그룹 이름을 지정하지 않고 Amazon EKS 권한 부여를 사용하려는 경우 이후 단계에서 또는 액세스 항목을 생성한 후에 액세스 정책을 연결할 수 있습니다.
9. (선택사항) 태그에서 액세스 항목에 레이블을 할당합니다. 예를 들어, 태그가 동일한 모든 리소스를 더 쉽게 찾을 수 있습니다.
10. 다음을 선택합니다.
11. 액세스 정책 추가 페이지에서 선택한 유형이 표준이고 Amazon EKS가 IAM 위탁자에 클러스터의 Kubernetes 객체에 대한 권한을 부여하게 하려면 다음 단계를 완료합니다. 그렇지 않은 경우 다음을 선택합니다.
 - a. 정책 이름에서 액세스 정책을 선택합니다. 액세스 정책의 권한은 볼 수 없지만 Kubernetes 사용자 대면 ClusterRole 객체에서 이와 유사한 권한을 포함합니다. 자세한 내용은 Kubernetes Documentation의 [User-facing roles](#)를 참조하세요.
 - b. 다음 옵션 중 하나를 선택하세요.

- 클러스터 - Amazon EKS에서 IAM 위탁자가 클러스터의 모든 Kubernetes 객체에 대한 액세스 정책의 권한을 갖도록 권한을 부여하려면 이 옵션을 선택합니다.
 - Kubernetes 네임스페이스 - Amazon EKS에서 IAM 위탁자가 클러스터 내 특정 Kubernetes 네임스페이스의 모든 Kubernetes 객체에 대한 액세스 정책의 권한을 갖도록 권한을 부여하려면 이 옵션을 선택합니다. 네임스페이스의 경우 클러스터의 Kubernetes 네임스페이스 이름을 입력합니다. 네임스페이스를 더 추가하려면 새 네임스페이스 추가를 선택하고 네임스페이스 이름을 입력합니다.
- c. 정책을 더 추가하려면 정책 추가를 선택합니다. 각 정책의 범위를 다르게 지정할 수 있지만 각 정책은 한 번만 추가할 수 있습니다.
- d. 다음을 선택합니다.

12. 액세스 항목의 구성을 검토합니다. 문제가 있는 것 같으면 이전을 선택하여 이전 단계로 돌아가서 오류를 수정합니다. 구성이 올바르면 생성을 선택합니다.

AWS CLI

1. AWS 명령줄 인터페이스 사용 설명서의 [설치](#)에 설명된 대로 AWS CLI를 설치합니다.
2. 액세스 항목을 만들려면 다음 예제 중 하나를 사용하여 액세스 항목을 만들 수 있습니다:
 - 자체 관리형 Amazon EC2 Linux 노드 그룹에 대한 액세스 항목을 생성합니다. *my-cluster*를 클러스터 이름으로, *111122223333*을 AWS 계정 ID로, *EKS-my-cluster-self-managed-ng-1*을 [노드 IAM 역할](#)로 바꿉니다. 노드 그룹이 Windows 노드 그룹인 경우 *EC2_LINUX*를 *EC2_Windows*로 바꾸세요.

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/EKS-my-cluster-self-managed-ng-1 --type EC2_LINUX
```

STANDARD 이외의 유형을 지정할 때는 이 `--kubernetes-groups` 옵션을 사용할 수 없습니다. 유형이 STANDARD 이외의 값이므로 액세스 정책을 이 액세스 항목에 연결할 수 없습니다.

- Kubernetes에서 클러스터에 대한 액세스를 부여하려는 Amazon EC2 자체 관리형 노드 그룹에 사용되지 않는 IAM 역할을 허용하는 액세스 항목을 생성합니다. *my-cluster*를 클러스터 이름으로, *111122223333*을 AWS 계정 ID로, *my-role*을 IAM 역할 이름으로 바꿉니다. *Viewers*를 클러스터의 Kubernetes RoleBinding 또는 ClusterRoleBinding 객체에 지정한 그룹 이름으로 바꿉니다.

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/my-role --type STANDARD --user Viewers --
kubernetes-groups Viewers
```

- 클러스터에서 IAM 사용자를 인증할 수 있는 액세스 항목을 생성합니다. IAM 모범 사례에서는 장기 보안 인증 정보를 보유한 IAM 사용자 대신, 단기 보안 인증 정보를 보유한 IAM 역할을 사용하여 클러스터에 액세스하도록 권장하지만, 이 예제는 가능한 상황을 보여줍니다. 자세한 내용은 IAM 사용 설명서에서 [임시 자격 증명을 사용하여 AWS에 액세스하려면 인간 사용자가 ID 공급자와의 페더레이션을 사용하도록 요구](#)를 참조하세요.

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:user/my-user --type STANDARD --username my-user
```

이 사용자에게 Kubernetes API 검색 역할의 권한보다 더 많은 액세스 권한을 부여하려는 경우 --kubernetes-groups 옵션이 사용되지 않으므로 액세스 정책을 액세스 항목에 연결해야 합니다. 자세한 내용은 [the section called “액세스 정책 연결”](#)과 Kubernetes Documentation의 [API discovery roles](#)를 참조하세요.

액세스 항목 업데이트

AWS Management Console 또는 AWS CLI를 사용하여 액세스 항목을 업데이트할 수 있습니다.

AWS Management Console

- [Amazon EKS 콘솔](#)을 엽니다.
- 관리형 노드 그룹을 생성하려는 클러스터의 이름을 선택합니다.
- 액세스 탭을 선택합니다.
- 업데이트할 액세스 항목을 선택합니다.
- 편집을 선택합니다.
- 사용자 이름에서 기존 값을 변경할 수 있습니다.
- 그룹에서 기존 그룹 이름을 제거하거나 새 그룹 이름을 추가할 수 있습니다. system:nodes 또는 system:bootstrappers와 같은 그룹 이름이 있는 경우에는 제거하지 않습니다. 이러한 그룹을 제거하면 클러스터가 제대로 작동하지 않을 수 있습니다. 그룹 이름을 지정하지 않고 Amazon EKS 권한 부여를 사용하려면 이후 단계에서 [액세스 정책](#)을 연결합니다.
- 태그에서 액세스 항목에 레이블을 할당할 수 있습니다. 예를 들어, 태그가 동일한 모든 리소스를 더 쉽게 찾을 수 있습니다. 기존 태그를 제거할 수도 있습니다.

9. 변경 사항 저장을 선택합니다.

10. 액세스 정책을 항목에 연결하려면 [the section called “액세스 정책 연결”](#) 섹션을 참조하세요.

AWS CLI

1. AWS 명령줄 인터페이스 사용 설명서의 [설치](#)에 설명된 대로 AWS CLI를 설치합니다.
2. 액세스 항목을 업데이트하려면 *my-cluster*를 클러스터의 이름으로, *111122223333*를 AWS 계정 ID로, *EKS-my-cluster-my-namespace-Viewers*를 IAM 역할의 이름으로 바꿉니다.

```
aws eks update-access-entry --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/EKS-my-cluster-my-namespace-Viewers --kubernetes-
groups Viewers
```

액세스 항목이 STANDARD 이외의 값인 경우 --kubernetes-groups 옵션을 사용할 수 없습니다. 또한 STANDARD 이외의 유형인 액세스 항목에 액세스 정책을 연결할 수 없습니다.

액세스 항목 삭제

액세스 항목을 잘못 삭제했다면 언제든지 다시 생성할 수 있습니다. 삭제하려는 액세스 항목이 액세스 정책과 연결된 경우 연결은 자동으로 삭제됩니다. 액세스 항목을 삭제하기 전에 액세스 항목에서 액세스 정책을 연결 해제하지 않아도 됩니다.

AWS Management Console 또는 AWS CLI를 사용하여 액세스 항목을 삭제할 수 있습니다.

AWS Management Console

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 액세스 항목을 삭제하려는 클러스터의 이름을 선택합니다.
3. 액세스 탭을 선택합니다.
4. 액세스 항목 목록에서 삭제할 액세스 항목을 선택합니다.
5. 삭제를 선택합니다.
6. 확인 대화 상자에서 삭제를 선택합니다.

AWS CLI

1. AWS 명령줄 인터페이스 사용 설명서의 [설치](#)에 설명된 대로 AWS CLI를 설치합니다.

2. 액세스 항목을 삭제하려면 *my-cluster*를 클러스터의 이름으로, *111122223333*를 AWS 계정 ID로, *my-role*을 더 이상 클러스터에 액세스하지 않으려는 IAM 역할의 이름으로 바꿉니다.

```
aws eks delete-access-entry --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/my-role
```

ConfigMap을 사용하여 IAM 사용자에게 Kubernetes 액세스 권한 부여

⚠ Important

aws-auth ConfigMap는 더 이상 사용되지 않습니다. Kubernetes API에 대한 액세스를 관리하는 권장 방법은 [the section called “권한 부여”](#) 섹션을 참조하세요.

[IAM 보안 주체](#)를 사용하는 클러스터에 대한 액세스는 Amazon EKS 컨트롤 플레인에서 실행되는 [Kubernetes](#)용 AWS IAM Authenticator에 의해 사용 설정됩니다. 인증자는 aws-auth ConfigMap에서 구성 정보를 가져옵니다. 모든 aws-auth ConfigMap 설정은 GitHub의 [Full Configuration Format](#)을 참조하세요.

Amazon EKS 클러스터에 IAM 보안 주체 추가

Amazon EKS 클러스터를 생성할 경우, 클러스터를 생성하는 [IAM 보안 주체](#)에게는 Amazon EKS 제어 영역의 클러스터 역할 기반 액세스 제어(RBAC) 구성에 system:masters 권한이 자동으로 부여됩니다. 이 보안 주체는 표시되는 구성에 나타나지 않으므로 클러스터를 원래 생성한 보안 주체를 추적해야 합니다. 추가 IAM 위탁자에 클러스터와 상호 작용할 수 있는 기능을 부여하려면 Kubernetes 내에서 aws-auth ConfigMap을 편집하고 aws-auth ConfigMap에 지정하는 group의 이름으로 Kubernetes rolebinding 또는 clusterrolebinding을 생성합니다.

ℹ Note

Kubernetes 역할 기반 액세스 제어(RBAC) 구성에 대한 자세한 내용은 Kubernetes Documentation의 [Using RBAC Authorization](#)을 참조하세요.

1. kubectl이 클러스터에 액세스하는 데 사용하는 자격 증명을 확인합니다. 컴퓨터에서 다음 명령으로 kubectl이 사용하는 자격 증명을 볼 수 있습니다. 기본 경로를 사용하지 않는 경우 *~/.kube/config*를 kubeconfig 파일의 경로로 바꿉니다.

```
cat ~/.kube/config
```

예제 출력은 다음과 같습니다.

```
[...]
contexts:
- context:
  cluster: my-cluster.region-code.eksctl.io
  user: admin@my-cluster.region-code.eksctl.io
  name: admin@my-cluster.region-code.eksctl.io
current-context: admin@my-cluster.region-code.eksctl.io
[...]
```

이전 예제 출력에서는 *admin*이라는 사용자의 자격 증명이 *my-cluster*라는 클러스터에 대해 구성되었습니다. 클러스터를 생성한 사용자인 경우 클러스터에 대한 액세스 권한이 이미 있습니다. 클러스터를 생성한 사용자가 아닌 경우 다른 IAM 보안 주체에 대한 클러스터 액세스를 사용 설정하기 위해 나머지 단계를 완료해야 합니다. [IAM 모범 사례](#)에서는 되도록 사용자 대신 역할에 권한을 부여하세요. 다음 명령을 사용하여 현재 클러스터에 액세스할 수 있는 다른 보안 주체를 확인할 수 있습니다.

```
kubectl describe -n kube-system configmap/aws-auth
```

예제 출력은 다음과 같습니다.

```
Name:          aws-auth
Namespace:     kube-system
Labels:        <none>
Annotations:    <none>

Data
====
mapRoles:
----
- groups:
  - system:bootstrappers
  - system:nodes
rolearn: arn:aws:iam::111122223333:role/my-node-role
username: system:node:{{EC2PrivateDNSName}}
```

```
BinaryData
=====

Events:  <none>
```

이전 예는 기본 aws-auth ConfigMap입니다. 노드 인스턴스 역할만 클러스터에 액세스할 수 있습니다.

2. IAM 위탁자를 매핑할 수 있는 기존 Kubernetes roles 및 rolebindings 또는 clusterroles 및 clusterrolebindings가 있어야 합니다. 이런 리소스에 대한 자세한 내용을 알아보려면 Kubernetes 문서의 [RBAC 승인 사용](#)을 참조하세요.

a. 기존 Kubernetes roles 또는 clusterroles를 확인합니다. Roles는 namespace로 범위가 지정되지만 clusterroles는 클러스터로 범위가 지정됩니다.

```
kubectl get roles -A
```

```
kubectl get clusterroles
```

b. 이전 출력에서 반환된 모든 role 또는 clusterrole의 세부 정보를 보고 IAM 보안 주체가 클러스터에서 보유하려는 권한(rules)이 있는지 확인합니다.

*role-name*을 이전 명령의 출력에서 반환된 role 이름으로 바꿉니다. *kube-system*을 role의 네임스페이스로 바꿉니다.

```
kubectl describe role role-name -n kube-system
```

*cluster-role-name*을 이전 명령의 출력에서 반환된 clusterrole 이름으로 바꿉니다.

```
kubectl describe clusterrole cluster-role-name
```

c. 기존 Kubernetes rolebindings 또는 clusterrolebindings를 확인합니다. Rolebindings는 namespace로 범위가 지정되지만 clusterrolebindings는 클러스터로 범위가 지정됩니다.

```
kubectl get rolebindings -A
```

```
kubectl get clusterrolebindings
```


- d. `rolebinding` 또는 `clusterrolebinding`의 모든 세부 정보를 보고 `roleRef`로 나열된 이전 단계의 `role` 또는 `clusterrole` 그리고 `subjects`에 나열된 그룹 이름이 있는지 확인합니다.

*role-binding-name*을 이전 명령의 출력에서 반환된 `rolebinding` 이름으로 바꿉니다.
*kube-system*을 `rolebinding`의 namespace로 바꿉니다.

```
kubectl describe rolebinding role-binding-name -n kube-system
```

예제 출력은 다음과 같습니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: eks-console-dashboard-restricted-access-role-binding
  namespace: default
subjects:
- kind: Group
  name: eks-console-dashboard-restricted-access-group
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: eks-console-dashboard-restricted-access-role
  apiGroup: rbac.authorization.k8s.io
```

*cluster-role-binding-name*을 이전 명령의 출력에서 반환된 `clusterrolebinding` 이름으로 바꿉니다.

```
kubectl describe clusterrolebinding cluster-role-binding-name
```

예제 출력은 다음과 같습니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks-console-dashboard-full-access-binding
subjects:
- kind: Group
  name: eks-console-dashboard-full-access-group
  apiGroup: rbac.authorization.k8s.io
roleRef:
```

```
kind: ClusterRole
name: eks-console-dashboard-full-access-clusterrole
apiGroup: rbac.authorization.k8s.io
```

3. aws-auth ConfigMap을 편집합니다. eksctl과 같은 도구를 사용하여 ConfigMap을 업데이트하거나 이를 편집하여 수동으로 업데이트할 수 있습니다.

Important

eksctl 또는 다른 도구를 사용하여 ConfigMap을 편집하는 것이 좋습니다. 사용할 수 있는 다른 도구에 대한 자세한 내용을 알아보려면 Amazon EKS 모범 사례 가이드의 [도구를 사용하여 aws-auth ConfigMap](#)을 변경을 참조하세요. 부적절하게 형식이 지정된 aws-auth ConfigMap으로 인해 클러스터에 대한 액세스 권한을 상실할 수 있습니다.

- [eksctl을 사용하여 구성 맵을 편집](#)하는 단계를 확인합니다.
- [configmap을 수동으로 편집](#)하는 단계를 봅니다.

Eksctl을 사용하여 Configmap 편집

1. 장치에 설치된 eksctl 명령줄 도구의 버전 0.210.0 이상 또는 AWS CloudShell이 필요합니다. eksctl을 설치 또는 업그레이드하려면 eksctl 설명서에서 [설치](#)를 참조하세요.
2. ConfigMap에서 현재 매핑을 확인합니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다. *region-code*를 클러스터가 있는 AWS 리전으로 바꿉니다.

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

예제 출력은 다음과 같습니다.

ARN	USERNAME	GROUPS
ACCOUNT		
arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-		
NodeInstanceRole-1XLS7754U3ZPA	system:node:{{EC2PrivateDNSName}}	
system:bootstrappers,system:nodes		

3. 역할에 대한 매핑을 추가합니다. *my-role*을 역할 이름으로 바꿉니다. *eks-console-dashboard-full-access-group*을 Kubernetes RoleBinding 또는 ClusterRoleBinding

객체에 지정된 그룹 이름으로 바꿉니다. **111122223333**을 계정 ID로 바꿉니다. **admin**을 선택한 모든 이름으로 바꿀 수 있습니다.

```
eksctl create iamidentitymapping --cluster my-cluster --region=region-code \
    --arn arn:aws:iam::111122223333:role/my-role --username admin --group eks-
console-dashboard-full-access-group \
    --no-duplicate-arns
```

Important

역할 ARN에는 `role/my-team/developers/my-role`과 같은 경로가 포함될 수 없습니다. ARN 형식은 `arn:aws:iam::111122223333:role/my-role` 이어야 합니다. 이 예에서는 `my-team/developers/`를 제거해야 합니다.

예제 출력은 다음과 같습니다.

```
[...]
2022-05-09 14:51:20 [#] adding identity "{arn-aws}iam::111122223333:role/my-role" to
auth ConfigMap
```

4. 사용자에게 대한 매핑을 추가합니다. [IAM 모범 사례](#)에서는 되도록 사용자 대신 역할에 권한을 부여하세요. **my-user**를 사용자 이름으로 바꿉니다. **eks-console-dashboard-restricted-access-group**을 Kubernetes RoleBinding 또는 ClusterRoleBinding 객체에 지정된 그룹 이름으로 바꿉니다. **111122223333**을 계정 ID로 바꿉니다. **my-user**를 선택한 모든 이름으로 바꿀 수 있습니다.

```
eksctl create iamidentitymapping --cluster my-cluster --region=region-code \
    --arn arn:aws:iam::111122223333:user/my-user --username my-user --group eks-
console-dashboard-restricted-access-group \
    --no-duplicate-arns
```

예제 출력은 다음과 같습니다.

```
[...]
2022-05-09 14:53:48 [#] adding identity "arn:aws:iam::111122223333:user/my-user" to
auth ConfigMap
```

5. ConfigMap에서 매핑을 다시 확인합니다.

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

예제 출력은 다음과 같습니다.

ARN	USERNAME ACCOUNT	GROUPS
arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-NodeInstanceRole-1XLS7754U3ZPA	system:node:{{EC2PrivateDNSName}}	
arn:aws:iam::111122223333:role/admin	system:bootstrappers, system:nodes	
arn:aws:iam::111122223333:role/my-role	my-role	eks-console-dashboard-full-access-group
arn:aws:iam::111122223333:user/my-user	my-user	eks-console-dashboard-restricted-access-group

Configmap 수동 편집

1. 편집을 위해 ConfigMap을 엽니다.

```
kubectl edit -n kube-system configmap/aws-auth
```

Note

“Error from server (NotFound): configmaps "aws-auth" not found”라는 오류가 표시되면 [클러스터에 aws-auth ConfigMap 적용](#)의 절차를 사용하여 스텝 ConfigMap을 적용합니다.

2. IAM 보안 주체를 ConfigMap에 추가하세요. IAM 그룹은 IAM 보안 주체가 아니므로 ConfigMap에 추가할 수 없습니다.

- IAM 역할(예: [페더레이션 사용자](#))을 추가하려면: 역할 세부 정보를 data 아래 ConfigMap의 mapRoles 섹션에 추가합니다. 파일에 이미 존재하지 않는 경우 이 섹션을 추가합니다. 각 항목은 다음 파라미터를 지원합니다.
 - rolearn: 추가할 IAM 역할의 ARN. 이 값은 경로를 포함할 수 없습니다. 예를 들어, `arn:aws:iam::111122223333:role/my-team/developers/role-name` 과 같은 ARN

을 지정할 수 없습니다. 대신 ARN은 `arn:aws:iam::111122223333:role/role-name` 이어야 합니다.

- `username`: Kubernetes 내에서 IAM 역할에 매핑할 사용자 이름.
- `groups`: 역할을 매핑할 그룹 또는 Kubernetes 그룹 목록입니다. 그룹은 기본 그룹이나 `clusterrolebinding` 또는 `rolebinding`에 지정된 그룹일 수 있습니다. 자세한 내용은 Kubernetes Documentation의 [Default roles and role bindings](#)를 참조하세요.
- IAM 사용자를 추가하는 방법: [IAM 모범 사례](#)에서는 되도록 사용자 대신 역할에 권한을 부여하세요. 사용자 세부 정보를 `data` 아래 `ConfigMap`의 `mapUsers` 섹션에 추가합니다. 파일에 이미 존재하지 않는 경우 이 섹션을 추가합니다. 각 항목은 다음 파라미터를 지원합니다.
 - `userarn`: 추가할 IAM 사용자의 ARN.
 - `username`: Kubernetes 내에서 IAM 사용자에게 매핑할 사용자 이름.
 - `groups`: 사용자를 매핑할 그룹 또는 Kubernetes 그룹 목록입니다. 그룹은 기본 그룹이나 `clusterrolebinding` 또는 `rolebinding`에 지정된 그룹일 수 있습니다. 자세한 내용은 Kubernetes Documentation의 [Default roles and role bindings](#)를 참조하세요.

3. 예를 들어, 아래 YAML 블록에는 다음이 포함됩니다.

- 노드가 클러스터에 직접 등록할 수 있도록 IAM 노드 인스턴스를 Kubernetes 그룹에 매핑하는 `mapRoles` 섹션 및 모든 클러스터에 대한 모든 Kubernetes 리소스를 볼 수 있는 Kubernetes 그룹에 매핑되는 `my-console-viewer-role` IAM 역할. `my-console-viewer-role` IAM 역할에 필요한 IAM 및 Kubernetes 그룹 권한 목록은 [the section called “필수 권한”](#) 섹션을 참조하세요.
- 기본 AWS 계정의 `admin` IAM 사용자를 `system:masters` Kubernetes 그룹에 매핑하는 `mapUsers` 섹션 및 특정 네임스페이스에 대한 Kubernetes 리소스를 볼 수 있는 Kubernetes 그룹에 매핑되는 다른 AWS 계정의 `my-user` 사용자. `my-user` IAM 사용자에게 필요한 IAM 및 Kubernetes 그룹 권한 목록은 [the section called “필수 권한”](#) 섹션을 참조하세요.

필요에 따라 줄을 추가하거나 제거하고 모든 `##` `#`을 자신의 값으로 바꿉니다.

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file
# will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
```

```

rolearn: arn:aws:iam::111122223333:role/my-role
username: system:node:{{EC2PrivateDNSName}}
- groups:
  - eks-console-dashboard-full-access-group
rolearn: arn:aws:iam::111122223333:role/my-console-viewer-role
username: my-console-viewer-role
mapUsers: |
- groups:
  - system:masters
userarn: arn:aws:iam::111122223333:user/admin
username: admin
- groups:
  - eks-console-dashboard-restricted-access-group
userarn: arn:aws:iam::444455556666:user/my-user
username: my-user

```

4. 파일을 저장하고 텍스트 편집기를 종료합니다.

클러스터에 **aws-authConfigMap** 적용

aws-auth ConfigMap은 관리형 노드 그룹을 생성하거나 eksctl을 사용하여 노드 그룹을 생성할 때 자동으로 생성되어 클러스터에 적용됩니다. 이 ConfigMap은 처음에는 노드를 클러스터에 조인하기 만 들어졌으나 이 ConfigMap을 사용하여 IAM 보안 주체에 역할 기반 액세스 제어(RBAC) 액세스를 추가할 수도 있습니다. 자체 관리형 노드를 시작했고 클러스터에 aws-auth ConfigMap을(를) 적용하지 않았다면 다음 절차를 수행하면 됩니다.

1. aws-auth ConfigMap을(를) 이미 적용했는지 확인합니다.

```
kubectl describe configmap -n kube-system aws-auth
```

‘Error from server (NotFound): configmaps "aws-auth" not found’와 같은 오류가 발생할 경우 다음 단계를 수행하여 스텝 ConfigMap을 적용합니다.

2. AWS Authenticator 구성 맵을 다운로드, 편집 및 적용합니다.

a. 구성 맵을 다운로드합니다.

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
aws-auth-cm.yaml
```

- b. `aws-auth-cm.yaml` 파일에서 `rolearn`을 노드와 연결된 IAM 역할의 Amazon 리소스 이름 (ARN)으로 설정합니다. 이 작업은 텍스트 편집기를 사용하거나 `my-node-instance-role`을 대체하고 다음 명령을 실행하여 수행할 수 있습니다.

```
sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-role|' aws-auth-cm.yaml
```

이 파일에서 어떠한 행도 수정하지 마세요.

Important

역할 ARN에는 `role/my-team/developers/my-role`과 같은 경로가 포함될 수 없습니다. ARN 형식은 `arn:aws:iam::111122223333:role/my-role` 이어야 합니다. 이 예에서는 `my-team/developers/`를 제거해야 합니다.

노드 그룹에 대해 AWS CloudFormation 스택 출력을 점검하고 다음 값을 찾습니다.

- `InstanceRoleARN-eksctl`로 생성된 노드 그룹용
- `NodeInstanceRole-AWS` Management Console에서 Amazon EKS 판매 AWS CloudFormation 템플릿으로 생성된 노드 그룹용

- c. 구성을 적용합니다. 이 명령을 완료하는 데 몇 분이 걸릴 수 있습니다.

```
kubectl apply -f aws-auth-cm.yaml
```

Note

권한 부여 또는 리소스 유형 오류가 표시되는 경우 문제 해결 주제의 [the section called “권한이 없거나 액세스가 거부됨\(kubectl\)”](#) 섹션을 참조하세요.

3. 노드의 상태를 확인하고 Ready 상태가 될 때까지 대기합니다.

```
kubectl get nodes --watch
```

Ctrl+C를 입력하여 셸 프롬프트로 돌아갑니다.

외부 OIDC 제공자를 통해 사용자에게 Kubernetes에 대한 액세스 권한 부여

Amazon EKS는 클러스터에서 사용자를 인증하는 방법으로 OpenID Connect(OIDC) 자격 증명 공급자를 지원합니다. OIDC ID 공급자는 AWS ID 및 액세스 관리(IAM)와 함께 또는 그 대안으로 사용할 수 있습니다. IAM 사용에 대한 자세한 내용은 [the section called “Kubernetes API 액세스”](#)를 참조하세요. 클러스터에 인증을 구성한 후, Kubernetes roles 및 clusterroles를 생성하여 역할에 권한을 할당합니다. 다음 Kubernetes rolebindings 및 clusterrolebindings를 사용하여 역할을 ID에 바인딩할 수 있습니다. 자세한 내용은 Kubernetes 문서의 [Using RBAC Authorization\(RBAC 승인 사용\)](#)을 참조하십시오.

- 하나의 OIDC 자격 증명 공급자를 클러스터에 연결할 수 있습니다.
- Kubernetes는 OIDC ID 제공업체를 제공하지 않습니다. 기존 퍼블릭 OIDC 자격 증명 공급자를 사용하거나 자체 ID 공급자를 실행할 수 있습니다. 인증된 공급자 목록은 OpenID 사이트에서 [OpenID 인증](#)을 참조하세요.
- Amazon EKS가 서명 키를 검색할 수 있도록 OIDC 자격 증명 공급자의 발급자 URL에 공개적으로 액세스할 수 있어야 합니다. Amazon EKS는 자체 서명된 인증서를 사용하는 OIDC ID 제공업체를 지원하지 않습니다.
- 클러스터에 노드를 조인하는 데 여전히 필요하므로 클러스터에 IAM 인증을 비활성화할 수 없습니다.
- 하지만 여전히 OIDC 자격 증명 공급자 사용자가 아니라 AWS [IAM 보안 주체](#)가 Amazon EKS 클러스터를 생성해야 합니다. 이는 클러스터 생성자가 Kubernetes API가 아닌 Amazon EKS API와 상호 작용하기 때문입니다.
- 컨트롤 플레인에 대해 CloudWatch Logs가 설정되어 있는 경우 OIDC ID 제공업체 인증 사용자는 클러스터의 감사 로그에 표시됩니다. 자세한 내용은 [the section called “컨트롤 플레인 로그 활성화 또는 비활성화”](#) 단원을 참조하십시오.
- OIDC 제공자가 제공한 계정으로 AWS Management Console에 로그인할 수 없습니다. AWS ID 및 액세스 관리 계정으로 AWS Management Console에 로그인해야만 [the section called “클러스터 리소스 액세스”](#)할 수 있습니다.

OIDC 자격 증명 공급자 연결

OIDC 자격 증명 공급자를 클러스터에 연결하려면 먼저 공급자의 다음 정보가 필요합니다.

발급자 URL

API 서버가 토큰을 확인하기 위한 퍼블릭 서명 키를 검색할 수 있도록 허용하는 OIDC ID 제공자의 URL입니다. URL은 `https://`로 시작해야 하며 공급자의 OIDC ID 토큰에 있는 `iss` 클레임에 해당해야 합니다. OIDC 표준에 따라 경로 구성 요소는 허용되지만 쿼리 파라미터는 허용되지 않습니다. 일반적으로 URL은 <https://server.example.org> 또는 <https://example.com> 같은 하나의 호스트 이름으로만 구성됩니다. 이 URL은 `.well-known/openid-configuration` 아래 레벨을 가리켜야 하며 인터넷을 통해 공개적으로 액세스할 수 있어야 합니다.

클라이언트 ID(대상이라고도 함)

OIDC ID 제공자에게 인증을 요청하는 클라이언트 애플리케이션의 ID입니다.

`eksctl` 또는 AWS Management Console을 사용하여 자격 증명 공급자를 연결할 수 있습니다.

`eksctl`을 사용하여 ID 공급자 연결

1. 다음 내용을 사용하여 *`associate-identity-provider.yaml`*이라는 파일을 생성합니다. *`##`*을 사용자의 값으로 바꿉니다. `identityProviders` 섹션의 값은 OIDC 자격 증명 공급자로부터 가져옵니다. 값은 `identityProviders`의 `name`, `type`, `issuerUrl`, `clientId` 설정에만 필요합니다.

```
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: your-region-code

identityProviders:
- name: my-provider
  type: oidc
  issuerUrl: https://example.com
  clientId: kubernetes
  usernameClaim: email
  usernamePrefix: my-username-prefix
  groupsClaim: my-claim
  groupsPrefix: my-groups-prefix
  requiredClaims:
    string: string
  tags:
```

```
env: dev
```

Important

`groupsPrefix` 또는 `usernamePrefix`에 대해 `system:` 또는 해당 문자열의 어떤 부분도 지정하지 마세요.

2. 공급자를 생성합니다.

```
eksctl associate identityprovider -f associate-identity-provider.yaml
```

3. 클러스터와 OIDC ID 제공업체에 `kubectl`을 사용하려면 Kubernetes Documentation의 [Using kubectl](#)을 참조하세요.

AWS 콘솔을 사용하여 ID 공급자 연결

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 클러스터를 선택한 다음 액세스 탭을 선택합니다.
3. OIDC 자격 증명 공급자 섹션에서 *ID 제공업체 연결*을 선택합니다.
4. [OIDC 자격 증명 공급자 연결(Associate OIDC Identity Provider)] 페이지에서 다음 옵션을 입력하거나 선택한 후 [연결(Associate)]을 선택합니다.
 - [이름(Name)]에 공급자의 고유한 이름을 입력합니다.
 - 발급자 URL에 공급자의 URL을 입력합니다. 이는 인터넷을 통해 액세스할 수 있는 URL이어야 합니다.
 - 클라이언트 ID에 OIDC ID 제공업체의 클라이언트 ID(대상이라고도 함)를 입력합니다.
 - 사용자 이름 클레임에 사용자 이름으로 사용할 클레임을 입력합니다.
 - 그룹 클레임에 사용자의 그룹으로 사용할 클레임을 입력합니다.
 - (선택사항) 고급 옵션을 선택하고 다음 정보를 입력하거나 선택합니다.
 - 사용자 이름 접두사 — 사용자 이름 클레임 앞에 추가할 접두사를 입력합니다. 기존 이름과의 충돌을 방지하기 위해 접두사가 사용자 이름 클레임 앞에 추가됩니다. 값을 제공하지 않은 경우 사용자 이름이 email 이외의 값이면 접두사는 발급자 URL에 대한 기본값으로 설정됩니다. 값 -를 사용하여 모든 접두사를 사용 중지할 수 있습니다. `system:` 또는 해당 문자열의 어떤 부분도 지정하지 마세요.
 - 그룹 접두사 — 그룹 클레임 앞에 추가할 접두사를 입력합니다. 기존 이름(예: `system:groups`)과의 충돌을 방지하기 위해 접두사가 그룹 클레임 앞에 추가됩니다. 예를 들어, 값

oidc:는 oidc:engineering 및 oidc:infra와 같은 그룹 이름을 생성합니다. system: 또는 해당 문자열의 어떤 부분도 지정하지 마세요.

- 필수 클레임 — 클레임 추가를 선택하고 클라이언트 ID 토큰에 필요한 클레임을 설명하는 하나 이상의 키 값 페어를 입력합니다. 키 값 페어는 ID 토큰에 필요한 클레임을 설명합니다. 설정된 경우 각 클레임은 일치하는 값이 ID 토큰에 존재하는 것으로 확인됩니다.
- a. 클러스터와 OIDC ID 제공업체에 kubectl을 사용하려면 Kubernetes Documentation의 [Using kubectl](#)을 참조하세요.

IAM; 정책 예

OIDC 자격 증명 공급자가 클러스터에 연결되지 않도록 하려면 다음 IAM 정책을 생성하여 Amazon EKS 관리자의 IAM 계정에 연결합니다. 자세한 내용은 [IAM 정책 생성](#) 및 IAM 사용 설명서의 [IAM 자격 증명 권한 추가](#) 및 서비스 권한 부여 참조의 [작업](#)을 참조하세요.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "denyOIDC",
      "Effect": "Deny",
      "Action": [
        "eks:AssociateIdentityProviderConfig"
      ],
      "Resource": "arn:aws:eks:us-west-2.amazonaws.com:111122223333:cluster/*"
    },
    {
      "Sid": "eksAdmin",
      "Effect": "Allow",
      "Action": [
        "eks:*"
      ],
      "Resource": "*"
    }
  ]
}
```

다음 예제 정책은 clientID가 kubernetes이고 issuerUrl이 [https://cognito-idp.us-west-2amazonaws.com/](https://cognito-idp.us-west-2.amazonaws.com/) 인 경우 OIDC 자격 증명 공급자 연결을 허용합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowCognitoOnly",
      "Effect": "Deny",
      "Action": "eks:AssociateIdentityProviderConfig",
      "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-instance",
      "Condition": {
        "StringNotLikeIfExists": {
          "eks:issuerUrl": "https://cognito-idp.us-west-2.amazonaws.com/*"
        }
      }
    },
    {
      "Sid": "DenyOtherClients",
      "Effect": "Deny",
      "Action": "eks:AssociateIdentityProviderConfig",
      "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-instance",
      "Condition": {
        "StringNotEquals": {
          "eks:clientId": "kubernetes"
        }
      }
    },
    {
      "Sid": "AllowOthers",
      "Effect": "Allow",
      "Action": "eks:*",
      "Resource": "*"
    }
  ]
}
```

클러스터에서 OIDC 자격 증명 공급자 연결 해제

클러스터에서 OIDC 자격 증명 공급자의 연결을 해제하면 공급자에 포함된 사용자가 더 이상 클러스터에 액세스할 수 없습니다. 하지만 [IAM 보안 주체](#)로 계속 클러스터에 액세스할 수 있습니다.

1. [Amazon EKS 콘솔](#)을 엽니다.
2. [OIDC 자격 증명 공급자(OIDC Identity Providers)] 섹션에서 [연결 해제(Disassociate)]를 선택하고 자격 증명 공급자 이름을 입력한 다음 Disassociate를 선택합니다.

AWS Management Console에서 Kubernetes 리소스 보기

AWS Management Console을 사용하면 클러스터에 배포된 Kubernetes 리소스를 볼 수 있습니다. AWS CLI 또는 [eksctl](#)로는 Kubernetes 리소스를 볼 수 없습니다. 명령줄 도구를 사용하여 Kubernetes 리소스를 보려면 [kubect](#)를 사용합니다.

Note

AWS Management Console의 컴퓨팅 탭에서 리소스 탭과 노드 섹션을 보려면 사용 중인 [IAM 위탁자](#)에 특정 IAM 및 Kubernetes 권한이 있어야 합니다. 자세한 내용은 [the section called “필수 권한”](#) 섹션을 참조하세요.

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 클러스터(Clusters) 목록에서 보려는 Kubernetes 리소스를 포함하는 클러스터를 선택합니다.
3. 리소스(Resources) 탭을 선택합니다.
4. 보려는 리소스에 대한 리소스 유형 그룹을 선택합니다(예: 워크로드). 해당 그룹의 리소스 유형 목록이 표시됩니다.
5. 워크로드(Workloads) 그룹에서 리소스 유형을 선택합니다(예: 배포(Deployments)). 리소스 유형 설명, 리소스 유형의 자세한 내용이 있는 Kubernetes 설명서에 대한 링크, 클러스터에 배포된 해당 유형의 리소스 목록이 표시됩니다. 목록이 비어 있는 경우 클러스터에 배포된 해당 유형의 리소스가 없습니다.
6. 이에 대한 자세한 내용을 확인하려면 리소스를 선택합니다. 다음 예제를 시도해 보세요.
 - 워크로드 그룹을 선택하고, 배포 리소스 유형을 선택한 다음 coredns 리소스를 선택합니다. 리소스를 선택하면 기본적으로 구조화된 뷰에 있습니다. 일부 리소스 유형의 경우 구조화된 뷰의 포드 섹션이 표시됩니다. 이 섹션에서는 워크로드에 의해 관리되는 포드가 나열됩니다. 포드에 대한 정보를 보려면 나열된 모든 포드를 선택할 수 있습니다. 일부 리소스 유형의 정보가 구조적 뷰(Structured View)에 표시됩니다. 리소스 페이지의 오른쪽 상단 모서리에서 원시 뷰(Raw view)를 선택하는 경우 리소스에 대한 Kubernetes API의 전체 JSON 응답이 표시됩니다.
 - 클러스터(Cluster) 그룹을 선택한 다음 노드(Nodes) 리소스 유형을 선택합니다. 클러스터에 모든 노드 목록이 표시됩니다. 노드는 모든 [Amazon EKS 노드 유형](#)일 수 있습니다. 이 목록은 클러스터의 컴퓨팅 탭을 선택하는 경우 노드 섹션에 표시되는 것과 동일한 목록입니다. 목록에서 노드 리소스를 선택합니다. 구조화된 뷰에 포드 섹션도 표시됩니다. 이 섹션에서는 노드에서 실행되는 모든 포드를 보여줍니다.

필수 권한

AWS Management Console의 컴퓨팅 탭에서 리소스 탭과 노드 섹션을 보려면 사용 중인 [IAM 위탁자](#)에 특정 최소 IAM 및 Kubernetes 권한이 있어야 합니다. IAM 보안 주체에 필요한 권한을 할당하려면 다음 단계를 완료하세요.

1. `eks:AccessKubernetesApi` 및 Kubernetes 리소스를 보는 데 필요한 기타 IAM 권한이 사용 중인 IAM 위탁자에 할당되었는지 확인하세요. IAM 보안 주체의 권한을 편집하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 [보안 주체에 대한 액세스 제어](#) 섹션을 참조하세요. 역할의 권한을 편집하는 방법에 대한 자세한 내용을 알아보려면 IAM 사용 설명서의 [역할 권한 정책\(콘솔\) 수정](#)을 참조하세요.

다음 정책 예에는 계정의 모든 클러스터에 대한 Kubernetes 리소스를 보는 데 필요한 위탁자의 권한이 포함되어 있습니다. **111122223333**을 AWS 계정 ID로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks:ListFargateProfiles",
        "eks:DescribeNodegroup",
        "eks:ListNodegroups",
        "eks:ListUpdates",
        "eks:AccessKubernetesApi",
        "eks:ListAddons",
        "eks:DescribeCluster",
        "eks:DescribeAddonVersions",
        "eks:ListClusters",
        "eks:ListIdentityProviderConfigs",
        "iam:ListRoles"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "ssm:GetParameter",
      "Resource": "arn:aws:ssm:*:111122223333:parameter/*"
    }
  ]
}
```

}

[연결된 클러스터](#)에서 노드를 보려면 [Amazon EKS 커넥터 IAM 역할](#)이 클러스터에서 보안 주체를 가질 수 있어야 합니다. 그러면 [Amazon EKS Connector](#)에서 위탁자를 Kubernetes 사용자에게 매핑할 수 있습니다.

2. Kubernetes 리소스를 보는 데 필요한 권한이 있는 Kubernetes role 또는 clusterrole에 바인딩된 Kubernetes rolebinding 또는 clusterrolebinding을 생성합니다. Kubernetes 역할 및 역할 바인딩에 대한 자세한 내용을 알아보려면 Kubernetes 설명서의 [RBAC 권한 부여 사용](#)을 참조하세요. 필요한 Kubernetes 권한을 사용하여 role 및 rolebinding 또는 clusterrole 및 clusterrolebinding을 생성하는 클러스터에 다음 매니페스트 중 하나를 적용할 수 있습니다.

모든 네임스페이스의 Kubernetes 리소스 보기

- 파일의 그룹 이름은 eks-console-dashboard-full-access-group입니다. 다음 명령을 사용하여 클러스터에 매니페스트를 적용합니다.

```
kubectl apply -f https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-full-access.yaml
```

특정 네임스페이스의 Kubernetes 리소스 보기

- 이 파일의 네임스페이스는 default입니다. 파일의 그룹 이름은 eks-console-dashboard-restricted-access-group입니다. 다음 명령을 사용하여 클러스터에 매니페스트를 적용합니다.

```
kubectl apply -f https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-restricted-access.yaml
```

Kubernetes 그룹 이름, 네임스페이스, 권한 또는 파일의 다른 모든 구성을 변경해야 하는 경우 파일을 다운로드하여 편집한 후 클러스터에 적용합니다.

- a. 다음 명령 중 하나를 사용하여 파일을 다운로드합니다.

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-full-access.yaml
```

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/docs/eks-console-restricted-access.yaml
```

- b. 필요에 따라 파일을 편집합니다.

- c. 다음 명령 중 하나를 사용하여 클러스터에 매니페스트를 적용합니다.

```
kubectl apply -f eks-console-full-access.yaml
```

```
kubectl apply -f eks-console-restricted-access.yaml
```

3. [IAM 위탁자](#)를 aws-auth ConfigMap의 Kubernetes 사용자 또는 그룹에 매핑합니다. eksctl과 같은 도구를 사용하여 ConfigMap을 업데이트하거나 편집하여 수동으로 업데이트할 수 있습니다.

Important

eksctl 또는 다른 도구를 사용하여 ConfigMap을 편집하는 것이 좋습니다. 사용할 수 있는 다른 도구에 대한 자세한 내용을 알아보려면 Amazon EKS 모범 사례 가이드의 [도구를 사용하여 aws-authConfigMap](#)을 변경을 참조하세요. 부적절하게 형식이 지정된 aws-auth ConfigMap으로 인해 클러스터에 대한 액세스 권한을 상실할 수 있습니다.

eksctl로 편집

1. 장치에 설치된 eksctl 명령줄 도구의 버전 0.210.0 이상 또는 AWS CloudShell이 필요합니다. eksctl을 설치 또는 업그레이드하려면 eksctl 설명서에서 [설치](#)를 참조하세요.
2. ConfigMap에서 현재 매핑을 확인합니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다. *region-code*를 클러스터가 있는 AWS 리전으로 바꿉니다.

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

예제 출력은 다음과 같습니다.

ARN	USERNAME	GROUPS
ACCOUNT		
arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-NodeInstanceRole-1XLS7754U3ZPA	system:node:{{EC2PrivateDNSName}}	system:bootstrappers,system:nodes

3. 역할에 대한 매핑을 추가합니다. 이 예에서는 첫 번째 단계의 IAM 권한을 *my-console-viewer-role*이라는 역할에 연결했다고 가정합니다. *111122223333*을 계정 ID로 바꿉니다.

```
eksctl create iamidentitymapping \
  --cluster my-cluster \
```



```
--region=region-code \
--arn arn:aws:iam::111122223333:role/my-console-viewer-role \
--group eks-console-dashboard-full-access-group \
--no-duplicate-arns
```

Important

역할 ARN에는 `role/my-team/developers/my-role`과 같은 경로가 포함될 수 없습니다. ARN 형식은 `arn:aws:iam::111122223333:role/my-role` 이어야 합니다. 이 예에서는 `my-team/developers/`를 제거해야 합니다.

예제 출력은 다음과 같습니다.

```
[...]
2022-05-09 14:51:20 [#] adding identity "arn:aws:iam::111122223333:role/my-console-viewer-role" to auth ConfigMap
```

4. 사용자에게 대한 매핑을 추가합니다. [IAM 모범 사례](#)에서는 되도록 사용자 대신 역할에 권한을 부여하세요. 이 예에서는 첫 번째 단계의 IAM 권한을 *my-user*라는 사용자에게 연결했다고 가정합니다. *111122223333*을 계정 ID로 바꿉니다.

```
eksctl create iamidentitymapping \
--cluster my-cluster \
--region=region-code \
--arn arn:aws:iam::111122223333:user/my-user \
--group eks-console-dashboard-restricted-access-group \
--no-duplicate-arns
```

예제 출력은 다음과 같습니다.

```
[...]
2022-05-09 14:53:48 [#] adding identity "arn:aws:iam::111122223333:user/my-user" to auth ConfigMap
```

5. ConfigMap에서 매핑을 다시 확인합니다.

```
eksctl get iamidentitymapping --cluster my-cluster --region=region-code
```

예제 출력은 다음과 같습니다.

ARN	USERNAME	GROUPS
arn:aws:iam::111122223333:role/eksctl-my-cluster-my-nodegroup-NodeInstanceRole-1XLS7754U3ZPA	system:node:{{EC2PrivateDNSName}}	
arn:aws:iam::111122223333:role/my-console-viewer-role	system:bootstrappers,system:nodes	
		eks-console-dashboard-full-access-group
arn:aws:iam::111122223333:user/my-user		
		eks-console-dashboard-restricted-access-group

ConfigMap 수동 편집

aws-auth ConfigMap에 사용자 또는 역할을 추가하는 방법에 대한 자세한 내용은 [the section called “Amazon EKS 클러스터에 IAM 보안 주체 추가”](#) 섹션을 참조하세요.

1. 편집을 위해 aws-auth ConfigMap을 엽니다.

```
kubectl edit -n kube-system configmap/aws-auth
```

2. aws-auth ConfigMap에 매핑을 추가하지만 모든 기존 매핑을 대체하지는 마세요. 다음 예에서는 첫 번째 단계에서 추가된 권한과 이전 단계에서 생성된 Kubernetes 그룹을 사용하여 [IAM 위탁자](#) 간에 매핑을 추가합니다.

- *my-console-viewer-role* 역할 및 eks-console-dashboard-full-access-group.
- *my-user* 사용자 및 eks-console-dashboard-restricted-access-group.

이 예에서는 첫 번째 단계의 IAM 권한을 *my-console-viewer-role*이라는 역할 및 *my-user*라는 사용자에게 연결했다고 가정합니다. **111122223333**을 AWS 계정 ID로 바꿉니다.

```
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - eks-console-dashboard-full-access-group
      rolearn: arn:aws:iam::111122223333:role/my-console-viewer-role
      username: my-console-viewer-role
  mapUsers: |
```

```
- groups:
  - eks-console-dashboard-restricted-access-group
  userarn: arn:aws:iam::111122223333:user/my-user
  username: my-user
```

⚠ Important

역할 ARN에는 `role/my-team/developers/my-console-viewer-role`과 같은 경로가 포함될 수 없습니다. ARN 형식은 `arn:aws:iam::111122223333:role/my-console-viewer-role` 이어야 합니다. 이 예에서는 `my-team/developers/`를 제거해야 합니다.

3. 파일을 저장하고 텍스트 편집기를 종료합니다.

Kubeconfig 파일을 생성하여 kubectl을 EKS 클러스터에 연결

이 주제에서는 클러스터에 대한 kubeconfig 파일을 생성하거나 기존 파일을 업데이트합니다.

kubectl 명령줄 도구는 kubeconfig 파일의 구성 정보를 사용하여 클러스터의 API 서버와 통신합니다. 자세한 내용은 쿠버네티스 문서의 [kubeconfig 파일을 사용하여 클러스터 접근 구성하기](#)를 참조하세요.

Amazon EKS는 클러스터 인증을 위해 kubectl과 함께 `aws eks get-token` 명령을 사용합니다. 기본적으로 AWS CLI에서는 다음 명령을 사용하여 반환되는 것과 동일한 보안 인증 정보를 사용합니다.

```
aws sts get-caller-identity
```

- 기존 Amazon EKS 클러스터. 배포하려면 [시작](#) 섹션을 참조하세요.
- kubectl 명령줄 도구는 장치 또는 AWS CloudShell에 설치됩니다. 버전은 클러스터의 Kubernetes 버전과 동일하거나 최대 하나 이전 또는 이후의 마이너 버전일 수 있습니다. 예를 들어, 클러스터 버전이 1.29인 경우 kubectl 버전 1.28, 1.29 또는 1.30를 함께 사용할 수 있습니다. kubectl을 설치하거나 업그레이드하려면 [the section called “kubectl 및 eksctl 설정”](#) 부분을 참조하세요.
- 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설

치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.

- 지정한 클러스터에 대해 eks:DescribeCluster API 작업을 사용할 수 있는 권한을 보유한 IAM 사용자 또는 역할. 자세한 내용은 [the section called “ID 기반 정책”](#) 단원을 참조하십시오. 자체 OpenID Connect 제공업체의 ID를 사용하여 클러스터에 액세스하는 경우 Kubernetes Documentation의 [Using kubectl](#)을 참조하여 kube config 파일을 생성하거나 업데이트합니다.

자동으로 kubeconfig 파일 생성

- 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.
- 지정한 클러스터에 대해 eks:DescribeCluster API 작업을 사용할 수 있는 권한. 자세한 내용은 [the section called “ID 기반 정책”](#) 섹션을 참조하세요.
 - 클러스터에 대해 kubeconfig 파일을 생성 또는 업데이트합니다. `region-code`를 클러스터를 생성한 AWS 리전으로 바꾸고 `my-cluster`를 클러스터 이름으로 바꿉니다.

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

기본적으로 구성 파일은 홈 디렉터리의 기본 kubeconfig 경로(`.kube`)에 생성되거나 해당 위치의 기존 config 파일과 병합됩니다. `--kubeconfig` 옵션을 사용하여 다른 경로를 지정할 수 있습니다.

`--role-arn` 옵션으로 IAM 역할 ARN을 지정하면 kubectl 명령을 실행할 때 인증에 사용할 수 있습니다. 그렇지 않으면 기본 AWS CLI 또는 SDK 자격 증명 체인의 [IAM 보안 주체](#)가 사용됩니다. `aws sts get-caller-identity` 명령을 실행하여 기본 AWS CLI 또는 SDK ID를 확인할 수 있습니다.

사용 가능한 모든 옵션을 알아보려면 `aws eks update-kubeconfig help` 명령을 실행하거나 AWS CLI 명령 참조의 [update-kubeconfig](#)를 참조하세요.

2. 구성을 테스트합니다.

```
kubectl get svc
```

예제 출력은 다음과 같습니다.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
svc/kubernetes	ClusterIP	10.100.0.1	<none>	443/TCP	1m

권한 부여 또는 리소스 유형 오류가 표시되는 경우 문제 해결 주제의 [the section called “권한이 없거나 액세스가 거부됨\(kubectl\)”](#) 부분을 참조하세요.

Kubernetes 서비스 계정을 사용하여 Kubernetes 워크로드에 AWS에 대한 액세스 권한 부여

A Kubernetes service account provides an identity for processes that run in a Pod. For more information see [서비스 계정 관리](#) in the Kubernetes documentation. If your Pod needs access to AWS services, you can map the service account to an AWS Identity and Access Management identity to grant that access. For more information, see [the section called “IRSA를 사용한 자격 증명”](#) or [the section called “Pod Identity”](#).

서비스 계정 토큰

[BoundServiceAccountTokenVolume](#) 기능은 Kubernetes 버전에서 기본적으로 활성화됩니다. 이 기능은 Kubernetes에서 실행되는 워크로드가 대상, 시간 및 키 바인딩인 JSON 웹 토큰을 요청하도록 허용하여 서비스 계정 토큰의 보안을 향상시킵니다. 서비스 계정 토큰에는 1시간 만료 기간이 있습니다. 이전 Kubernetes 버전에는 토큰에 만료 기간이 없었습니다. 즉, 이러한 토큰에 의존하는 클라이언트는 1시간 이내에 토큰을 새로 고쳐야 합니다. 다음 [Kubernetes 클라이언트 SDK](#)는 필요한 시간 내에 토큰을 자동으로 새로 고칩니다.

- Go 버전 0.15.7 이상
- Python 버전 12.0.0 이상
- Java 버전 9.0.0 이상
- JavaScript 버전 0.10.3 이상

- Ruby master 브랜치
- Haskell 버전 0.3.0.0
- C# 버전 7.0.5 이상

워크로드가 이전 클라이언트 버전을 사용하는 경우 업데이트해야 합니다. 클라이언트가 최신 시간 바운드 서비스 계정 토큰으로 원활하게 마이그레이션되도록 Kubernetes는 기본 1시간 서비스 계정 토큰의 만료 기간을 추가로 연장합니다. Amazon EKS 클러스터의 경우 연장 만료 기간은 90일입니다. Amazon EKS 클러스터 Kubernetes API 서버는 90일이 지난 토큰을 사용한 요청을 거부합니다. 애플리케이션 및 해당 종속성을 확인하여 Kubernetes 클라이언트 SDK가 이전에 나열된 버전과 동일하거나 이후 버전인지 확인하는 것이 좋습니다.

API 서버가 1시간 이상 된 토큰으로 요청을 수신하면 API 감사 로그 이벤트에 `annotations.authentication.k8s.io/stale-token`로 주석을 추가합니다. 주석의 값은 다음 예와 유사합니다.

```
subject: system:serviceaccount:common:fluent-bit, seconds after warning threshold:
4185802.
```

클러스터에 [컨트롤 플레인 로깅](#)이 사용 설정되어 있는 경우 주석은 감사 로그에 있습니다. 다음과 같은 [CloudWatch Logs Insights](#) 쿼리를 사용하여 Amazon EKS 클러스터에서 오래된 토큰을 사용하는 모든 포드를 식별할 수 있습니다.

```
fields @timestamp
|filter @logStream like /kube-apiserver-audit/
|filter @message like /seconds after warning threshold/
|parse @message "subject: *, seconds after warning threshold:\"" as subject,
elapsedtime
```

subject는 포드가 사용한 서비스 계정을 참조합니다. elapsedtime은 최신 토큰을 읽은 후 경과 시간(초)을 표시합니다. API 서버에 대한 요청은 elapsedtime이 90일(7,776,000 초)을 초과하는 경우, 거부됩니다. 이전에 나열된 버전 중 하나를 사용하여 토큰을 자동으로 새로 고치도록 하려면 애플리케이션의 Kubernetes 클라이언트 SDK를 적극적으로 업데이트해야 합니다. 사용된 서비스 계정 토큰이 90일에 가까우며 토큰 만료 전에 클라이언트 SDK 버전을 업데이트할 시간이 충분하지 않은 경우 기존 포드를 종료하고 새 포드를 생성할 수 있습니다. 이로 인해 서비스 계정 토큰을 다시 가져와서 클라이언트 버전 SDK를 업데이트하는 데 추가로 90일이 제공됩니다.

포드가 배포의 일부인 경우 고가용성을 유지하면서 포드를 종료하라고 제안된 방법은 다음 명령으로 롤아웃을 수행하는 것입니다. `my-deployment`를 배포 이름으로 바꿉니다.

```
kubectl rollout restart deployment/my-deployment
```

클러스터 추가 기능

다음 클러스터 추가 기능은 Kubernetes 클라이언트 SDK를 사용하여 서비스 계정 토큰을 자동으로 다시 가져오도록 업데이트되었습니다. 나열된 버전 또는 이후 버전이 클러스터에 설치되었는지 확인하는 것이 좋습니다.

- Kubernetes용 Amazon VPC CNI 플러그인 및 지표 헬퍼 플러그인 버전 1.8.0 이상. 현재 버전을 확인하거나 업데이트하려면 [the section called “Amazon VPC CNI”](#) 및 [cni-metrics-helper](#) 부분을 참조하세요.
- CoreDNS 버전 1.8.4 이상. 현재 버전을 확인하거나 업데이트하려면 [the section called “CoreDNS”](#) 부분을 참조하세요.
- AWS Load Balancer Controller 버전 2.0.0 이상. 현재 버전을 확인하거나 업데이트하려면 [the section called “AWS 로드 밸런서 컨트롤러”](#) 부분을 참조하세요.
- 현재 kube-proxy 버전 현재 버전을 확인하거나 업데이트하려면 [the section called “kube-proxy”](#) 부분을 참조하세요.
- AWS for Fluent Bit 버전 2.25.0 이상. 현재 버전을 업데이트하려면 GitHub의 [릴리스](#)를 참조하세요.
- Fluentd 이미지 버전 [1.14.6-1.2](#) 이상 및 Kubernetes 메타데이터 버전 [2.11.1](#) 이상용 Fluentd 필터 플러그인.

Amazon Elastic Kubernetes Service 클러스터의 워크로드에 AWS ID 및 액세스 관리 권한 부여

Amazon EKS는 서비스 계정에 대한 IAM 역할 및 EKS Pod Identity를 통해 Amazon EKS 클러스터에서 실행되는 워크로드에 AWS ID 및 액세스 관리 권한을 부여합니다.

서비스 계정에 대한 IAM 역할

서비스 계정에 대한 IAM 역할(IRSA)은 세분화된 IAM 권한으로 AWS에서 실행되는 Kubernetes 애플리케이션을 구성하여 Amazon S3 버킷, Amazon DynamoDB 테이블 등과 같은 다양한 기타 AWS 리소스에 액세스할 수 있도록 합니다. 동일한 Amazon EKS 클러스터에서 여러 애플리케이션을 함께 실행하고 각 애플리케이션이 필요한 최소 권한 집합만 갖도록 할 수 있습니다. IRSA는 Amazon EKS, Amazon EKS Anywhere, AWS의 Red Hat OpenShift Service, Amazon EC2 인스턴스의 자체 관리형 Kubernetes 클러스터 등 AWS에서 지원하는 다양한 Kubernetes 배포 옵션을 지

원하도록 구축되었습니다. 따라서 IRSA는 IAM과 같은 기본 AWS 서비스를 사용하여 빌드되었으며 Amazon EKS 서비스 및 EKS API에 대한 직접적인 종속성을 갖지 않았습니다. 자세한 내용은 [the section called “IRSA를 사용한 자격 증명”](#) 섹션을 참조하세요.

EKS Pod Identity

EKS Pod Identity는 클러스터 관리자에게 Amazon S3 버킷, Amazon DynamoDB 테이블 등과 같은 다양한 기타 AWS 리소스에 액세스할 수 있도록 간소화된 애플리케이션 인증 워크플로를 제공합니다. EKS Pod Identity는 EKS 전용이므로 클러스터 관리자가 IAM 권한을 획득하도록 Kubernetes 애플리케이션을 구성하는 방법이 간소화됩니다. 이제 AWS Management Console, EKS API, AWS CLI를 통해 직접 몇 단계만 거치면 이러한 권한을 쉽게 구성할 수 있으며, 임의의 Kubernetes 객체에 있는 클러스터 내에서 별도의 조치를 취하지 않아도 됩니다. 클러스터 관리자는 EKS와 IAM 서비스 간에 전환하거나 권한이 있는 IAM 작업을 사용하여 애플리케이션에 필요한 권한을 구성할 필요가 없습니다. 이제 새 클러스터를 생성할 때 역할 신뢰 정책을 업데이트할 필요 없이 여러 클러스터에서 IAM 역할을 사용할 수 있습니다. EKS Pod Identity에서 제공하는 IAM 보안 인증 정보에는 클러스터 이름, 네임스페이스, 서비스 계정 이름 등의 속성이 포함된 역할 세션 태그가 포함됩니다. 관리자는 역할 세션 태그를 사용하여 일치하는 태그를 기반으로 AWS 리소스에 대한 액세스를 허용하여 여러 서비스 계정에서 작업할 수 있는 단일 역할을 작성할 수 있습니다. 자세한 내용은 [the section called “Pod Identity”](#) 섹션을 참조하세요.

EKS Pod Identity와 IRSA 비교

대략적으로 EKS Pod Identity와 IRSA 모두 Kubernetes 클러스터에서 실행되는 애플리케이션에 IAM 권한을 부여할 수 있도록 합니다. 하지만 구성 방법, 지원되는 제한 사항, 활성화된 기능 등이 근본적으로 다릅니다. 아래에서는 두 솔루션의 몇 가지 주요 측면을 비교합니다.

속성	EKS Pod Identity	IRSA
역할 확장성	새로 도입된 Amazon EKS 서비스 보안 주체 <code>pods.eks.amazonaws.com</code> 과의 신뢰를 구축하려면 각 역할을 한 번 설정해야 합니다. 이 한 번의 단계를 거친 후에는 새 클러스터에서 역할이 사용될 때마다 역할의 신뢰 정책을 업데이트할 필요가 없습니다.	새 클러스터에서 역할을 사용할 때마다 새 EKS 클러스터 OIDC 제공자 엔드포인트로 IAM 역할의 신뢰 정책을 업데이트해야 합니다.

속성	EKS Pod Identity	IRSA
클러스터 확장성	EKS Pod Identity의 경우 사용자가 IAM OIDC 공급자를 설정할 필요가 없으므로 이 제한은 적용되지 않습니다.	각 EKS 클러스터에는 OpenID Connect(OIDC) 발급자 URL이 연결되어 있습니다. IRSA를 사용하려면 IAM의 각 EKS 클러스터에 대해 고유한 OpenID Connect 제공자를 생성해야 합니다. IAM의 기본 글로벌 제한은 각 AWS 계정에 대해 100개의 OIDC 제공자입니다. IRSA가 있는 각 AWS 계정에 대해 100개를 초과하는 EKS 클러스터를 보유하려는 계획인 경우 IAM OIDC 제공자 한도에 도달합니다.
역할 확장성	EKS Pod Identity의 경우 사용자가 신뢰 정책에서 IAM 역할과 서비스 계정 간의 신뢰 관계를 정의할 필요가 없으므로 이 제한은 적용되지 않습니다.	IRSA에서는 역할의 신뢰 정책에서 IAM 역할과 서비스 계정 간의 신뢰 관계를 정의합니다. 기본적으로 신뢰 정책 크기의 길이는 2048입니다. 즉, 일반적으로 단일 신뢰 정책에서 4개의 신뢰 관계를 정의할 수 있습니다. 신뢰 정책 길이 제한을 늘릴 수 있지만 일반적으로 단일 신뢰 정책 내에서 최대 8개의 신뢰 관계로 제한됩니다.

속성	EKS Pod Identity	IRSA
역할 재사용 가능성	EKS Pod Identity에서 제공하는 AWS STS 보안 인증 정보에는 클러스터 이름, 네임스페이스, 서비스 계정 이름 등과 같은 역할 세션 태그가 포함됩니다. 역할 세션 태그를 사용하면 관리자는 연결된 태그를 기반으로 AWS 리소스에 대한 액세스를 허용함으로써 유효 권한이 다른 여러 서비스 계정에서 사용할 수 있는 단일 IAM 역할을 작성할 수 있습니다. 이를 속성 기반 액세스 제어(ABAC)라고 합니다. 자세한 내용은 the section called “포드 액세스 권한 부여” 섹션을 참조하세요.	AWS STS 세션 태그는 지원되지 않습니다. 클러스터 간에 역할을 재사용할 수 있지만 모든 포드는 해당 역할의 모든 권한을 받습니다.
지원되는 환경	EKS Pod Identity는 Amazon EKS에서만 사용할 수 있습니다.	IRSA는 Amazon EKS, Amazon EKS Anywhere, AWS의 Red Hat OpenShift Service, Amazon EC2 인스턴스의 자체 관리형 Kubernetes 클러스터 등에서 사용할 수 있습니다.
지원되는 EKS 버전	지원되는 모든 EKS 클러스터 버전. 특정 플랫폼 버전은 the section called “EKS Pod Identity 클러스터 버전” 섹션을 참조하세요.	지원되는 모든 EKS 클러스터 버전.

서비스 계정에 대한 IAM 역할

포드의 컨테이너에 있는 애플리케이션은 AWS SDK 또는 AWS CLI를 사용하여 AWS ID 및 액세스 관리(IAM) 권한을 사용하여 AWS 서비스에 API 요청을 할 수 있다. 애플리케이션은 AWS 자격 증명으

로 AWS API 요청에 서명해야 합니다. 서비스 계정에 대한 IAM 역할(IRSA)은 Amazon EC2 인스턴스 프로파일이 Amazon EC2 인스턴스에 자격 증명을 제공하는 것과 비슷한 방식으로 애플리케이션에 대한 자격 증명을 관리하는 기능을 제공합니다. AWS 자격 증명을 생성하여 컨테이너에 배포하거나 Amazon EC2 인스턴스의 역할을 사용하는 대신에 IAM 역할을 Kubernetes 서비스 계정과 연결하고 서비스 계정을 사용하도록 포드를 구성합니다. [AWS Ouposts의 Amazon EKS에 대한 로컬 클러스터](#)가 있는 서비스 계정에는 IAM 역할을 사용할 수 없습니다.

서비스 계정의 IAM 역할을 통해 다음과 같은 이점을 누릴 수 있습니다.

- 최소 권한 - IAM 권한의 범위를 서비스 계정으로 지정할 수 있습니다. 그러면 해당 서비스 계정을 사용하는 포드만 이 권한에 액세스할 수 있습니다. 또한 이 기능을 사용하면 kiam, kube2iam 같은 타사 솔루션이 필요 없습니다.
- 자격 증명 격리 - 포드의 컨테이너는 컨테이너가 사용하는 서비스 계정과 연결된 IAM 역할에 대한 자격 증명만 검색할 수 있습니다. 컨테이너는 다른 포드의 다른 컨테이너에서 사용하는 자격 증명에 액세스할 수 없습니다. 서비스 계정에 IAM 역할을 사용하는 경우 [Amazon EC2 인스턴스 메타데이터 서비스\(IMDS\)](#)에 대한 포드 액세스를 차단하지 않는 한 컨테이너에도 [Amazon EKS 노드 IAM 역할](#)에 할당된 권한이 있습니다. 자세한 내용은 [작업자 노드에 할당된 인스턴스 프로파일에 대한 액세스 제한](#) 부분을 참조하세요.
- 감사 - AWS CloudTrail을 통한 액세스 및 이벤트 로깅을 사용하여 소급적 감사를 지원합니다.

다음 절차를 완료하여 서비스 계정에 대한 IAM 역할을 활성화합니다.

1. [클러스터에 대한 IAM OIDC 제공자 만들기](#) - 이 절차는 각 클러스터에 대해 한 번만 완료합니다.

Note

EKS VPC 엔드포인트를 활성화한 경우 해당 VPC 내에서 EKS OIDC 서비스 엔드포인트에 액세스할 수 없습니다. 따라서 VPC에서 eksctl을 통해 OIDC 공급자를 생성하는 등의 작업은 작동하지 않으며 <https://oidc.eks.region.amazonaws.com> 요청을 시도할 때 시간 초과가 발생합니다. 오류 메시지의 예는 다음과 같습니다.

```
server cant find oidc.eks.region.amazonaws.com: NXDOMAIN
```

이 단계를 완료하려면 VPC 외부(예: AWS CloudShell 또는 인터넷에 연결된 컴퓨터)에서 명령을 실행할 수 있습니다. 또는 VPC에서 Route 53 Resolver와 같은 분할-수평 조건부 확인자를 생성하여 OIDC 발급자 URL에 대해 다른 확인자를 사용하고 VPC DNS를 사용하지 않

을 수 있습니다. CoreDNS의 조건부 전달의 예는 GitHub의 [Amazon EKS feature request](#)를 참조하세요.

2. [Kubernetes 서비스 계정에 IAM 역할 할당](#) - 애플리케이션이 갖길 원하는 각 고유 권한 집합에 대해 이 절차를 완료합니다.
3. [Kubernetes 서비스 계정을 사용하도록 포드 구성](#) - AWS 서비스에 액세스해야 하는 각 포드에 대해 이 절차를 완료합니다.
4. [AWS SDK와 함께 IRSA 사용](#) - 워크로드가 지원되는 버전의 AWS SDK를 사용하고 있는지, 워크로드가 기본 자격 증명 체인을 사용하는지 확인합니다.

IAM, Kubernetes 및 OpenID Connect(OIDC) 백그라운드 정보

2014년에 AWS Identity and Access Management는 OpenID Connect(OIDC)를 사용하여 페더레이션 ID에 대한 지원을 추가했습니다. 이 기능을 사용하면 지원되는 자격 증명 공급자를 이용해 AWS API 호출을 인증하고 유효한 OIDC JSON 웹 토큰(JWT)을 수신할 수 있습니다. 이 토큰을 AWS STS AssumeRoleWithWebIdentity API 작업에 전달하고 IAM 임시 역할 자격 증명을 수신할 수 있습니다. 이 자격 증명을 사용하여 Amazon S3 및 DynamoDB와 같은 AWS 서비스와 상호 작용할 수 있습니다.

각 JWT 토큰은 서명 키 페어로 서명됩니다. 키는 Amazon EKS에서 관리하는 OIDC 공급자가 제공하며 프라이빗 키는 7일마다 교체됩니다. Amazon EKS는 퍼블릭 키를 만료될 때까지 보관합니다. 외부 OIDC 클라이언트를 연결하는 경우 퍼블릭 키가 만료되기 전에 서명 키를 갱신해야 합니다. [서명 키를 가져와서 OIDC 토큰을 검증](#)하는 방법을 알아봅니다.

Kubernetes는 서비스 계정을 자체 내부 자격 증명 시스템으로 오랫동안 사용해 왔습니다. 포드는 Kubernetes API 서버에서만 검증할 수 있는 자동으로 마운트되는 토큰(OIDC JWT는 아니었음)을 사용하는 Kubernetes API 서버를 통해 인증을 할 수 있습니다. 이러한 레거시 서비스 계정 토큰은 만료되지 않으며, 서명 키를 교체하려면 까다로운 프로세스를 거쳐야 합니다. Kubernetes 버전 1.12에서 새로운 ProjectedServiceAccountToken 기능에 대한 지원이 추가되었습니다. 이 기능은 서비스 계정 ID도 포함된 OIDC JSON 웹 토큰이며 구성 가능한 대상을 지원합니다.

Amazon EKS는 ProjectedServiceAccountToken JSON 웹 토큰에 대한 서명 키가 포함된 각 클러스터에 대한 퍼블릭 OIDC 검색 엔드포인트를 호스팅하므로 IAM과 같은 외부 시스템이 Kubernetes에서 발급한 OIDC 토큰을 확인하고 수락할 수 있습니다.

클러스터에 대한 IAM OIDC 공급자 생성

클러스터에는 [OpenID Connect](#)(OIDC) 발급자 URL이 연결되어 있습니다. 서비스 계정에 AWS Identity and Access Management(IAM) 역할을 사용하려면 클러스터의 OIDC 발급자 URL에 IAM OIDC 제공자가 있어야 합니다.

- 기존 Amazon EKS 클러스터. 배포하려면 [시작](#) 섹션을 참조하세요.
- 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.
- kubectl 명령줄 도구는 장치 또는 AWS CloudShell에 설치됩니다. 버전은 클러스터의 Kubernetes 버전과 동일하거나 최대 하나 이전 또는 이후의 마이너 버전일 수 있습니다. 예를 들어, 클러스터 버전이 1.29인 경우 kubectl 버전 1.28, 1.29 또는 1.30를 함께 사용할 수 있습니다. kubectl을 설치하거나 업그레이드하려면 [the section called “kubectl 및 eksctl 설정”](#) 부분을 참조하세요.
- 클러스터 구성이 포함된 기존 kubectl config 파일입니다. kubectl config 파일을 생성하려면 [the section called “kubectl을 사용하여 클러스터에 액세스”](#) 섹션을 참조하세요.

eksctl 또는 AWS Management Console을 사용하여 클러스터에 대한 IAM OIDC 제공자를 생성할 수 있습니다.

OIDC 공급자 생성(eksctl)

1. 장치에 설치된 eksctl 명령줄 도구의 버전 0.210.0 이상 또는 AWS CloudShell이 필요합니다. eksctl을 설치 또는 업그레이드하려면 eksctl 설명서에서 [설치](#)를 참조하세요.
2. 클러스터의 OIDC 발행자 ID를 판단합니다.

클러스터의 OIDC 발행자 ID를 검색하고 변수에 저장합니다. <my-cluster>을 사용자의 고유한 값으로 교체합니다.

```
cluster_name=<my-cluster>
oidc_id=$(aws eks describe-cluster --name $cluster_name --query
"cluster.identity.oidc.issuer" --output text | cut -d '/' -f 5)
```

```
echo $oidc_id
```

3. 클러스터의 발행자 ID를 가진 IAM OIDC 제공자가 이미 계정에 있는지 판단합니다.

```
aws iam list-open-id-connect-providers | grep $oidc_id | cut -d "/" -f4
```

출력이 반환되면 클러스터에 대한 IAM OIDC 제공자가 이미 있는 것이므로 다음 단계를 건너뛸 수 있습니다. 출력이 반환되지 않은 경우 해당 클러스터에 대한 IAM OIDC 공급자를 생성해야 합니다.

4. 다음 명령을 사용하여 클러스터의 IAM OIDC ID 공급자를 생성합니다.

```
eksctl utils associate-iam-oidc-provider --cluster $cluster_name --approve
```

Note

EKS VPC 엔드포인트를 활성화한 경우 해당 VPC 내에서 EKS OIDC 서비스 엔드포인트에 액세스할 수 없습니다. 그에 따라 VPC에서 eksctl을 통해 OIDC 공급자를 생성하는 등의 작업은 작동하지 않으며 시간 초과가 발생합니다. 오류 메시지의 예는 다음과 같습니다.

```
** server cant find oidc.eks.<region-code>.amazonaws.com: NXDOMAIN
```

이 단계를 완료하려면 VPC 외부(예: AWS CloudShell 또는 인터넷에 연결된 컴퓨터)에서 명령을 실행할 수 있습니다. 또는 VPC에서 Route 53 Resolver와 같은 분할-수평 조건부 확인자를 생성하여 OIDC 발급자 URL에 대해 다른 확인자를 사용하고 VPC DNS를 사용하지 않을 수 있습니다. CoreDNS의 조건부 전달의 예는 GitHub의 [Amazon EKS feature request](#)를 참조하세요.

OIDC 공급자 생성(AWS 콘솔)

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 왼쪽 창에서 클러스터를 선택한 다음 클러스터 페이지에서 클러스터 이름을 선택합니다.
3. 개요 탭의 세부 정보 섹션에서 OpenID Connect 공급자 URL의 값을 적어 둡니다.
4. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
5. 왼쪽 탐색 창의 액세스 관리에서 자격 증명 공급자를 선택합니다. 클러스터의 URL과 일치하는 공급자가 목록에 있으면 클러스터에 대한 공급자가 이미 있는 것입니다. 클러스터의 URL과 일치하는 공급자가 나열되지 않는 경우 공급자를 생성해야 합니다.

6. 공급자를 생성하려면 공급자 추가를 선택합니다.
7. 제공업체 유형에서 OpenID Connect를 선택합니다.
8. 공급자 URL에 클러스터의 OIDC 제공자 URL을 입력합니다.
9. 대상에 sts.amazonaws.com을 입력합니다.
- 10(선택사항) 원하는 태그(예: 이 공급자의 클러스터를 식별하는 태그)를 추가합니다.
- 11.공급자 추가를 선택합니다.

다음 단계: [the section called “IAM 역할 할당”](#)

Kubernetes 서비스 계정에 IAM 역할 할당

이 주제에서는 Kubernetes 서비스 계정이 AWS Identity and Access Management(IAM) 역할을 수임하도록 구성하는 방법에 대해 설명합니다. 서비스 계정을 사용하도록 구성된 모든 포드는 해당 역할에 액세스 권한이 있는 모든 AWS 서비스에 액세스할 수 있습니다.

사전 조건

- 기존 클러스터가 있어야 합니다. 아직 없는 경우 [시작](#) 안내서 중 하나에 따라 생성할 수 있습니다.
- 클러스터에 대한 기존 IAM OpenID Connect(OIDC) 공급자. 이미 있는지 확인하거나 생성하는 방법을 알아보려면 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요.
- 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.
- kubectl 명령줄 도구는 장치 또는 AWS CloudShell에 설치됩니다. 버전은 클러스터의 Kubernetes 버전과 동일하거나 최대 하나 이전 또는 이후의 마이너 버전일 수 있습니다. 예를 들어, 클러스터 버전이 1.29인 경우 kubectl 버전 1.28, 1.29 또는 1.30를 함께 사용할 수 있습니다. kubectl을 설치하거나 업그레이드하려면 [the section called “kubectl 및 eksctl 설정”](#) 부분을 참조하세요.
- 클러스터 구성이 포함된 기존 kubectl config 파일입니다. kubectl config 파일을 생성하려면 [the section called “kubectl을 사용하여 클러스터에 액세스”](#) 섹션을 참조하세요.

1단계: IAM 정책 생성

IAM 역할에 기존 IAM 정책을 연결하려면 다음 단계로 건너뛰니다.

1. IAM 정책을 생성합니다. 자체 정책을 생성하거나 필요한 권한 중 일부를 이미 부여하는 AWS 관리형 정책을 복사하여 특정 요구 사항에 맞게 사용자 지정할 수 있습니다. 자세한 내용은 IAM 사용 설명서에서 [IAM 정책 생성](#)을 참조하세요.
2. 포드가 액세스할 AWS 서비스에 대한 권한이 포함된 파일을 생성합니다. 모든 AWS 서비스에 대한 모든 작업 목록은 [서비스 승인 참조](#)에서 확인하세요.

다음 명령을 실행하여 Amazon S3 버킷에 대한 읽기 전용 액세스를 허용하는 예제 정책 파일을 생성할 수 있습니다. 이 버킷에 구성 정보 또는 부트스트랩 스크립트를 저장할 수도 있고, 의 컨테이너는 이 버킷에서 파일을 읽어 애플리케이션으로 로드할 수 있습니다. 이 예제 정책을 생성하려면 다음 내용을 디바이스에 복사합니다. *my-pod-secrets-bucket*을 버킷 이름으로 바꾸고 명령을 실행합니다.

```
cat >my-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
    }
  ]
}
EOF
```

3. IAM 정책을 생성합니다.

```
aws iam create-policy --policy-name my-policy --policy-document file:///my-policy.json
```

2단계: IAM 역할 생성 및 연결

IAM 역할을 생성하고 Kubernetes 서비스 계정에 연결합니다. eksctl 또는 AWS CLI를 사용할 수 있습니다.

역할 생성 및 연결(eksctl)

장치에 설치된 eksctl 명령줄 도구의 버전 0.210.0 이상 또는 AWS CloudShell이 필요합니다. eksctl을 설치 또는 업그레이드하려면 eksctl 설명서에서 [설치](#)를 참조하세요.

*my-service-account*를 eksctl이 생성하고 IAM 역할과 연결할 Kubernetes 서비스 계정의 이름으로 바꿉니다. *##*을 eksctl이 서비스 계정을 생성할 네임스페이스로 바꿉니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다. *my-role*을 서비스 계정을 연결할 역할의 이름으로 바꿉니다. 역할이 아직 없는 경우 eksctl이 자동으로 역할을 생성합니다. *111122223333*을 계정 ID로 바꾸고 *my-policy*를 기존 정책의 이름으로 바꿉니다.

```
eksctl create iamserviceaccount --name my-service-account --namespace default --cluster my-cluster --role-name my-role \
    --attach-policy-arn arn:aws:iam::111122223333:policy/my-policy --approve
```

Important

역할 또는 서비스 계정이 이미 있는 경우 이전 명령이 실패할 수 있습니다. eksctl에는 이러한 상황에서 제공할 수 있는 다양한 옵션이 있습니다. 자세한 내용을 알아보려면 eksctl create iamserviceaccount --help를 실행하세요.

역할 생성 및 연결(AWS CLI)

IAM 역할을 수입하려는 기존 Kubernetes 서비스 계정이 있는 경우 이 단계를 건너뛸 수 있습니다.

1. Kubernetes 서비스 계정을 생성합니다. 다음 콘텐츠를 디바이스에 복사합니다. *my-service-account*를 원하는 이름으로 바꾸고 필요한 경우 *##*을 다른 네임스페이스로 바꿉니다. *##*을 변경하는 경우, 네임스페이스가 이미 존재해야 합니다.

```
cat >my-service-account.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-service-account
  namespace: default
EOF
kubectl apply -f my-service-account.yaml
```

2. 다음 명령을 사용해 AWS 계정 ID를 환경 변수로 설정합니다.

```
account_id=$(aws sts get-caller-identity --query "Account" --output text)
```

3. 다음 명령을 사용해 클러스터의 OIDC ID 제공업체를 환경 변수로 설정합니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
oidc_provider=$(aws eks describe-cluster --name my-cluster --region $AWS_REGION --query "cluster.identity.oidc.issuer" --output text | sed -e "s/^https://\\//")
```

4. 서비스 계정의 네임스페이스 및 이름에 대한 변수를 설정합니다. *my-service-account*를 역할을 수임할 Kubernetes 서비스 계정으로 바꿉니다. *##*을 서비스 계정의 네임스페이스로 바꿉니다.

```
export namespace=default
export service_account=my-service-account
```

5. 다음 명령을 실행하여 IAM 역할에 대한 신뢰 정책 파일을 생성합니다. 네임스페이스 내의 모든 서비스 계정이 역할을 사용하도록 허용하려면 다음 내용을 디바이스에 복사합니다. *StringEquals*를 *StringLike*로 바꾸고 *\$service_account*를 ***로 바꿉니다. 여러 서비스 계정 또는 네임스페이스가 역할을 수임할 수 있도록 *StringEquals* 또는 *StringLike* 조건에 여러 항목을 추가할 수 있습니다. 클러스터가 속한 계정과 다른 AWS 계정의 역할이 역할을 수임하도록 허용하려면 [the section called “교차 계정 S3 IAM”](#) 섹션에서 자세한 내용을 참조하세요.

```
cat >trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::$account_id:oidc-provider/$oidc_provider"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "$oidc_provider:aud": "sts.amazonaws.com",
          "$oidc_provider:sub": "system:serviceaccount:$namespace:$service_account"
        }
      }
    }
  ]
}
```

EOF

6. 역할을 생성합니다. *my-role*을 IAM 역할의 이름으로 바꾸고, *my-role-description*을 역할에 대한 설명으로 바꿉니다.

```
aws iam create-role --role-name my-role --assume-role-policy-document file:///trust-relationship.json --description "my-role-description"
```

7. 역할에 IAM 정책을 연결합니다. *my-role*을 IAM 역할의 이름으로 바꾸고 *my-policy*를 생성한 기존 정책의 이름으로 바꿉니다.

```
aws iam attach-role-policy --role-name my-role --policy-arn=arn:aws:iam::
$account_id:policy/my-policy
```

8. 서비스 계정이 수임할 IAM 역할의 Amazon 리소스 이름(ARN)을 서비스 계정에 주석으로 답니다. *my-role*을 기존 IAM 역할의 이름으로 바꿉니다. 클러스터가 속한 계정과 다른 AWS 계정의 역할이 이전 단계에서 역할을 수임하도록 허용했다고 가정합니다. 그런 다음 AWS 계정과 다른 계정의 역할을 지정해야 합니다. 자세한 내용은 [the section called “교차 계정 S3 IAM”](#) 섹션을 참조하세요.

```
kubectl annotate serviceaccount -n $namespace $service_account eks.amazonaws.com/
role-arn=arn:aws:iam::$account_id:role/my-role
```

9. (선택 사항) [서비스 계정에 대한 AWS Security Token Service 엔드포인트를 구성합니다](#). AWS는 글로벌 엔드포인트 대신 리전 AWS STS 엔드포인트를 사용할 것을 권장합니다. 이렇게 하면 지연 시간이 줄고, 중복성이 기본 제공되고, 세션 토큰 유효성이 향상됩니다.

3단계: 구성 확인

1. IAM 역할의 신뢰 정책이 올바르게 구성되었는지 확인합니다.

```
aws iam get-role --role-name my-role --query Role.AssumeRolePolicyDocument
```

예제 출력은 다음과 같습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
```

```

        "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
    },
    "Action": "sts:AssumeRoleWithWebIdentity",
    "Condition": {
        "StringEquals": {
            "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:default:my-service-
account",
            "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
        }
    }
}
]
}

```

2. 이전 단계에서 역할에 연결한 정책이 해당 역할에 연결되었는지 확인합니다.

```

aws iam list-attached-role-policies --role-name my-role --query
"AttachedPolicies[].PolicyArn" --output text

```

예제 출력은 다음과 같습니다.

```
arn:aws:iam::111122223333:policy/my-policy
```

3. 사용하려는 정책의 Amazon 리소스 이름(ARN)을 저장할 변수를 설정합니다. *my-policy*를 권한을 확인하려는 정책의 이름으로 바꿉니다.

```
export policy_arn=arn:aws:iam::111122223333:policy/my-policy
```

4. 정책의 기본 버전을 봅니다.

```
aws iam get-policy --policy-arn $policy_arn
```

예제 출력은 다음과 같습니다.

```

{
  "Policy": {
    "PolicyName": "my-policy",

```

```

    "PolicyId": "EXAMPLEBIOWGLDEXAMPLE",
    "Arn": "arn:aws:iam::111122223333:policy/my-policy",
    "Path": "/",
    "DefaultVersionId": "v1",
    [...]
  }
}

```

5. 정책 내용을 보고 포드에 필요한 모든 권한이 정책에 포함되어 있는지 확인합니다. 필요한 경우 다음 명령에서 **1**을 이전 출력에서 반환된 버전으로 바꿉니다.

```
aws iam get-policy-version --policy-arn $policy_arn --version-id v1
```

예제 출력은 다음과 같습니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
    }
  ]
}

```

이전 단계에서 예제 정책을 생성한 경우 출력은 동일합니다. 다른 정책을 생성한 경우 **##** 내용이 다릅니다.

6. Kubernetes 서비스 계정에 역할이 주석으로 달렸는지 확인합니다.

```
kubectl describe serviceaccount my-service-account -n default
```

예제 출력은 다음과 같습니다.

```

Name:                my-service-account
Namespace:           default
Annotations:         eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/my-
                    role
Image pull secrets:   <none>
Mountable secrets:    my-service-account-token-qqjfl

```

```
Tokens:          my-service-account-token-qqjfl
[...]
```

다음 단계

- [the section called “포드에 할당”](#)

Kubernetes 서비스 계정을 사용하도록 포드 구성

포드가 AWS 서비스에 액세스해야 하는 경우 Kubernetes 서비스 계정을 사용하도록 구성해야 합니다. 서비스 계정은 AWS 서비스에 액세스할 수 있는 권한이 있는 AWS ID 및 액세스 관리(IAM) 역할에 연결되어야 합니다.

- 기존 클러스터가 있어야 합니다. 아직 없는 경우 [시작](#) 가이드 중 하나를 사용하여 생성할 수 있습니다.
- 클러스터에 대한 기존 IAM OpenID Connect(OIDC) 공급자. 이미 있는지 확인하거나 생성하는 방법을 알아보려면 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요.
- IAM 역할과 연결된 기존 Kubernetes 서비스 계정. 서비스 계정에 IAM 역할의 Amazon 리소스 이름(ARN)을 주석으로 달아야 합니다. 포드가 AWS 서비스를 사용하는 데 필요한 권한을 포함하는 연결된 IAM 정책이 역할에 있어야 합니다. 서비스 계정 및 역할을 만들고 구성하는 방법에 대한 자세한 내용을 알아보려면 [the section called “IAM 역할 할당”](#) 섹션을 참조하세요.
- 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.
- kubectl 명령줄 도구는 장치 또는 AWS CloudShell에 설치됩니다. 버전은 클러스터의 Kubernetes 버전과 동일하거나 최대 하나 이전 또는 이후의 마이너 버전일 수 있습니다. 예를 들어, 클러스터 버전이 1.29인 경우 kubectl 버전 1.28, 1.29 또는 1.30를 함께 사용할 수 있습니다. kubectl을 설치하거나 업그레이드하려면 [the section called “kubectl 및 eksctl 설정”](#) 부분을 참조하세요.
- 클러스터 구성이 포함된 기존 kubectl config 파일입니다. kubectl config 파일을 생성하려면 [the section called “kubectl을 사용하여 클러스터에 액세스”](#) 섹션을 참조하세요.

1. 다음 명령을 사용하여 구성을 확인할 포드를 배포할 수 있는 배포 매니페스트를 생성합니다. **##** **#**을 사용자의 값으로 바꿉니다.

```
cat >my-deployment.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      serviceAccountName: my-service-account
      containers:
      - name: my-app
        image: public.ecr.aws/nginx/nginx:X.XX
EOF
```

2. 클러스터에 매니페스트를 배포합니다.

```
kubectl apply -f my-deployment.yaml
```

3. 포드에 필요한 환경 변수가 있는지 확인합니다.

- a. 이전 단계에서 배포와 함께 배포된 포드를 봅니다.

```
kubectl get pods | grep my-app
```

예제 출력은 다음과 같습니다.

```
my-app-6f4dfff6cb-76cv9    1/1      Running    0          3m28s
```

- b. 포드가 사용 중인 IAM 역할의 ARN을 봅니다.

```
kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep AWS_ROLE_ARN:
```

예제 출력은 다음과 같습니다.

```
AWS_ROLE_ARN: arn:aws:iam::111122223333:role/my-role
```

역할 ARN이 기존 서비스 계정에 주석으로 단 역할 ARN과 일치해야 합니다. 서비스 계정에 주석 달기에 대한 자세한 내용을 알아보려면 [the section called “IAM 역할 할당”](#) 섹션을 참조하세요.

- c. 포드에 웹 ID 토큰 파일 탑재가 있는지 확인합니다.

```
kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep AWS_WEB_IDENTITY_TOKEN_FILE:
```

예제 출력은 다음과 같습니다.

```
AWS_WEB_IDENTITY_TOKEN_FILE: /var/run/secrets/eks.amazonaws.com/serviceaccount/token
```

kubelet은 포드를 대신하여 토큰을 요청하고 저장합니다. 기본적으로 kubelet은 총 TTL(Time To Live)의 80% 또는 24시간보다 오래된 토큰을 새로 고칩니다. 포드 사양의 설정을 사용하여 기본 서비스 계정을 제외한 모든 계정의 만료 기간을 수정할 수 있습니다. 자세한 내용은 Kubernetes 문서의 [서비스 계정 토큰 볼륨 예측\(Service Account Token Volume Projection\)](#)을 참조하세요.

클러스터의 [Amazon EKS Pod Identity 웹훅](#)은 다음 주석을 이용해 서비스 계정을 사용하는 포드를 관찰합니다.

```
eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/my-role
```

웹훅은 이전 환경 변수를 해당 포드에 적용합니다. 클러스터는 환경 변수 및 토큰 파일 탑재를 구성하기 위해 웹훅을 사용할 필요가 없습니다. 이러한 환경 변수를 갖도록 포드를 수동으로 구성할 수 있습니다. [지원되는 AWS SDK 버전](#)은 자격 증명 체인 제공자에서 이 환경 변수를 먼저 찾습니다. 역할 자격 증명은 이 기준을 충족하는 포드에 사용됩니다.

4. 역할에 연결된 IAM 정책에서 할당한 권한을 사용하여 포드가 AWS 서비스와 상호 작용할 수 있는지 확인합니다.

Note

포드가 서비스 계정과 연결된 IAM 역할의 AWS 자격 증명을 사용하는 경우 해당 포드의 컨테이너에 있는 AWS CLI나 다른 SDK는 역할이 제공하는 자격 증명을 사용합니다.

[Amazon EKS 노드 IAM 역할](#)에 제공된 자격 증명에 대한 액세스를 제한하지 않으면 포드는 계속해서 해당 자격 증명에 액세스할 수 있습니다. 자세한 내용은 [작업자 노드에 할당된 인스턴스 프로필에 대한 액세스 제한](#) 부분을 참조하세요.

포드가 예상대로 서비스와 상호 작용할 수 없으면 다음 단계를 완료하여 모두 제대로 구성되었는지 확인합니다.

- 포드에서 OpenID Connect 웹 ID 토큰 파일을 통해 IAM 역할을 수임할 수 있도록 지원하는 AWS SDK 버전을 사용하는지 확인합니다. 자세한 내용은 [the section called “지원되는 SDK”](#) 단원을 참조하십시오.
- 배포에서 서비스 계정을 사용하고 있는지 확인합니다.

```
kubectl describe deployment my-app | grep "Service Account"
```

예제 출력은 다음과 같습니다.

```
Service Account: my-service-account
```

- 포드가 여전히 서비스에 액세스할 수 없는 경우 [IAM 역할을 쿠버네티스 서비스 계정에 할당하기](#)에 설명된 [단계](#)를 검토하여 역할과 서비스 계정이 올바르게 구성되었는지 확인합니다.

서비스 계정의 AWS 보안 토큰 서비스 엔드포인트 구성

[서비스 계정에 대한 IAM 역할](#)이 포함된 Kubernetes 서비스 계정을 사용하는 경우 클러스터 및 플랫폼 버전이 동일하거나 다음 표에 나열된 버전보다 이후라면 Kubernetes 서비스 계정에서 사용하는 AWS Security Token Service 엔드포인트 유형을 구성할 수 있습니다. Kubernetes 또는 플랫폼 버전이 테이블에 나열된 버전보다 이전 버전인 경우 서비스 계정은 글로벌 엔드포인트만 사용할 수 있습니다.

Kubernetes 버전	플랫폼 버전	기본 엔드포인트 유형
1.31	eks.4	리전
1.30	eks.2	리전
1.29	eks.1	리전
1.28	eks.1	리전

Kubernetes 버전	플랫폼 버전	기본 엔드포인트 유형
1.27	eks.1	리전
1.26	eks.1	리전

AWS는 글로벌 엔드포인트 대신 리전별 AWS STS 엔드포인트를 사용할 것을 권장합니다. 이렇게 하면 지연 시간이 줄고, 중복성이 기본 제공되고, 세션 토큰 유효성이 향상됩니다. AWS Security Token Service는 포드가 실행 중인 AWS 리전에서 활성 상태여야 합니다. 또한 AWS 리전에서 서비스 장애가 발생할 경우 다른 AWS 리전에 대한 기본 제공 중복성이 애플리케이션에 있어야 합니다. 자세한 내용은 IAM 사용 설명서의 [AWS 리전에서 AWS STS 관리](#)를 참조하세요.

- 기존 클러스터가 있어야 합니다. 아직 없는 경우 [시작](#) 가이드 중 하나를 사용하여 생성할 수 있습니다.
- 클러스터에 대한 기존 IAM OIDC 공급자입니다. 자세한 내용은 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요.
- [서비스 계정에 대한 Amazon EKS IAM](#) 기능에서 사용하도록 구성된 기존 Kubernetes 서비스 계정.

다음 예에서는 모두 [Amazon VPC CNI 플러그인](#)에서 사용하는 aws-node Kubernetes 서비스 계정을 사용합니다. ## #을 고유 서비스 계정, 포드, 네임스페이스 및 기타 리소스로 바꿀 수 있습니다.

1. 엔드포인트를 변경하려는 서비스 계정을 사용하는 포드를 선택합니다. 포드가 실행되는 AWS 리전을 결정합니다. *aws-node-6mfgv*를 포드 이름으로, *kube-system*을 포드의 네임스페이스로 바꿉니다.

```
kubectl describe pod aws-node-6mfgv -n kube-system |grep Node:
```

예제 출력은 다음과 같습니다.

```
ip-192-168-79-166.us-west-2/192.168.79.166
```

이전 출력에서 포드는 us-west-2 AWS 리전의 노드에서 실행됩니다.

2. 포드의 서비스 계정이 사용 중인 엔드포인트 유형을 결정합니다.

```
kubectl describe pod aws-node-6mfgv -n kube-system |grep AWS_STS_REGIONAL_ENDPOINTS
```

예제 출력은 다음과 같습니다.

```
AWS_STS_REGIONAL_ENDPOINTS: regional
```

현재 엔드포인트가 전역인 경우 `global`은 출력에서 반환됩니다. 출력이 반환되지 않는 경우 기본 엔드포인트 유형이 재정의되지 않고 사용 중에 있습니다.

- 클러스터 또는 플랫폼 버전이 표에 나열된 버전과 동일하거나 이후인 경우 다음 명령 중 하나를 사용하여 서비스 계정에서 사용하는 엔드포인트 유형을 기본 유형에서 다른 유형으로 변경할 수 있습니다. `aws-node`를 서비스 계정의 이름으로, `kube-system`을 서비스 계정의 네임스페이스로 바꿉니다.

- 기본 또는 현재 엔드포인트 유형이 전역이고 다음과 같은 리전으로 변경하려는 경우:

```
kubectl annotate serviceaccount -n kube-system aws-node eks.amazonaws.com/sts-regional-endpoints=true
```

[서비스 계정에 대한 IAM 역할](#)을 사용하여 포드 컨테이너에서 실행 중인 애플리케이션에서 사전 서명된 S3 URL을 생성하는 경우 리전 엔드포인트의 URL 형식은 다음 예제와 유사합니다.

```
https://bucket.s3.us-west-2.amazonaws.com/path?...&X-Amz-Credential=your-access-key-id/date/us-west-2/s3/aws4_request&...
```

- 기본 또는 현재 엔드포인트 유형이 리전이고 다음과 같은 전역으로 변경하려는 경우:

```
kubectl annotate serviceaccount -n kube-system aws-node eks.amazonaws.com/sts-regional-endpoints=false
```

애플리케이션이 AWS STS 글로벌 엔드포인트에 명시적으로 요청하고 Amazon EKS 클러스터에서 리전 엔드포인트를 사용하는 기본 동작을 재정의하지 않는 경우 오류로 인해 요청이 실패합니다. 자세한 내용은 [the section called “포드 컨테이너는 다음과 같은 오류가 발생합니다. An error occurred \(SignatureDoesNotMatch\) when calling the GetCallerIdentity operation: Credential should be scoped to a valid region”](#) 섹션을 참조하세요.

[서비스 계정에 대한 IAM 역할](#)을 사용하여 포드 컨테이너에서 실행 중인 애플리케이션에서 사전 서명된 S3 URL을 생성하는 경우 글로벌 엔드포인트의 URL 형식은 다음 예제와 유사합니다.

```
https://bucket.s3.amazonaws.com/path?...&X-Amz-Credential=your-access-key-id/date/us-west-2/s3/aws4_request&...
```

사전 서명된 URL을 특정 형식으로 예상하는 자동화가 있거나 사전 서명된 URL을 사용하는 애플리케이션 또는 다운스트림 종속성이 대상 지정된 AWS 리전을 예상하는 경우, 필요한 변경을 수행하여 적절한 AWS STS 엔드포인트를 사용합니다.

4. 서비스 계정에 연결된 기존 포드를 삭제하고 다시 생성하여 자격 증명 환경 변수를 적용합니다. 변형 웹후크는 이미 실행 중인 포드에 이 변수를 적용하지 않습니다. *Pods*, *kube-system* 및 *-l k8s-app=aws-node*를 주석을 설정한 포드에 대한 정보로 바꿀 수 있습니다.

```
kubectl delete Pods -n kube-system -l k8s-app=aws-node
```

5. 모든 포드가 다시 시작되었는지 확인합니다.

```
kubectl get Pods -n kube-system -l k8s-app=aws-node
```

6. 포드 중 하나에 대한 환경 변수를 봅니다. `AWS_STS_REGIONAL_ENDPOINTS` 값이 이전 단계에서 설정한 값인지 확인합니다.

```
kubectl describe pod aws-node-kzbtr -n kube-system |grep AWS_STS_REGIONAL_ENDPOINTS
```

예제 출력은 다음과 같습니다.

```
AWS_STS_REGIONAL_ENDPOINTS=regional
```

IRSA를 사용하여 다른 계정에 인증

다른 계정의 클러스터에서 ID 공급자를 생성하거나 연결된 AssumeRole 작업을 사용하여 크로스 계정 IAM 권한을 구성할 수 있습니다. 다음 예에서 계정 A는 서비스 계정에 대해 IAM 역할을 지원하는 Amazon EKS 클러스터를 소유합니다. 이 클러스터에서 실행되는 포드는 계정 B의 IAM 권한을 수입해야 합니다.

Example 다른 계정의 클러스터에서 ID 공급자 생성

Example

이 예제에서 계정 A는 계정 B에 클러스터의 OpenID Connect(OIDC) 발행자 URL을 제공합니다. 계정 B는 계정 A의 클러스터에서 OIDC 발급자 URL을 사용하여 [클러스터에 대한 IAM OIDC 제공자 만](#)

[돌기](#) 및 [the section called “IAM 역할 할당”](#)의 지침을 따릅니다. 그런 다음 클러스터 관리자는 계정 B(**444455556666**)의 역할을 사용할 계정 A의 클러스터에 있는 서비스 계정에 주석을 겁니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::444455556666:role/account-b-role
```

Example 연결된 **AssumeRole** 작업 사용

Example

이 예에서 계정 B는 계정 A의 클러스터에 있는 포드에 부여할 권한이 포함된 IAM 정책을 생성합니다. 계정 B(**444455556666**)는 계정 A(**111122223333**)에 AssumeRole 권한을 허용하는 신뢰 관계를 사용하여 이 정책을 IAM 역할에 연결합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {}
    }
  ]
}
```

계정 A는 클러스터의 OIDC 발행자 주소로 생성된 ID 공급자의 자격 증명을 가져오는 신뢰 정책을 이용해 역할을 생성합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      }
    }
  ]
}
```

```
    },
    "Action": "sts:AssumeRoleWithWebIdentity"
  }
]
}
```

계정 A는 이 역할을 계정 B가 생성한 역할을 맡을 수 있는 다음과 같은 권한이 포함된 정책에 연결합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::444455556666:role/account-b-role"
    }
  ]
}
```

포드가 계정 B의 역할을 수입할 수 있게 하는 애플리케이션 코드에서는 두 가지 프로필, 즉 `account_b_role` 및 `account_a_role`을 사용합니다. `account_b_role` 프로필에서는 `account_a_role` 프로필을 소스로 사용합니다. AWS CLI의 경우 `~/.aws/config` 파일은 다음과 유사합니다.

```
[profile account_b_role]
source_profile = account_a_role
role_arn=arn:aws:iam::444455556666:role/account-b-role

[profile account_a_role]
web_identity_token_file = /var/run/secrets/eks.amazonaws.com/serviceaccount/token
role_arn=arn:aws:iam::111122223333:role/account-a-role
```

연결된 프로필을 다른 AWS SDK에 지정하려면 사용 중인 SDK에 대한 설명서를 참조하세요. 자세한 내용은 [AWS 기반의 도구](#)를 참조하세요.

AWS SDK와 함께 IRSA 사용

보안 인증 정보 사용

서비스 계정(IRSA)에 대한 IAM 역할의 자격 증명을 사용하려면 코드에서 임의의 AWS SDK를 사용하여 SDK가 포함된 AWS 서비스에 대한 클라이언트를 만들 수 있고, 기본적으로 SDK는 일련의 위치에

서 사용할 AWS Identity and Access Management ID를 검색합니다. 클라이언트를 생성하거나 SDK를 초기화했을 때 보안 인증 정보 공급자를 지정하지 않으면 서비스 계정에 대한 IAM 역할 보안 인증 정보가 사용됩니다.

이는 서비스 계정에 대한 IAM 역할이 기본 보안 인증 정보 체인의 한 단계로 추가되었기 때문에 작동합니다. 워크로드가 현재 보안 인증 정보 체인의 이전 단계에 있는 보안 인증 정보를 사용하는 경우 동일한 워크로드에 대해 서비스 계정에 대한 IAM 역할을 구성하더라도 해당 보안 인증 정보는 계속 사용됩니다.

SDK는 AssumeRoleWithWebIdentity 작업을 사용하여 AWS Security Token Service에서 임시 자격 증명에 대한 서비스 계정 OIDC 토큰을 자동 교환합니다. Amazon EKS와 이 SDK 작업은 만료되기 전에 갱신하여 임시 보안 인증 정보를 계속 교체합니다.

[서비스 계정에 IAM 역할](#)을 사용할 때 포드에 있는 컨테이너에서는 OpenID Connect 웹 ID 토큰 파일을 통해 IAM 역할을 수입할 수 있도록 지원하는 AWS SDK 버전을 사용해야 합니다. AWS SDK에 대해 다음과 같은 버전 또는 이후 버전을 사용해야 합니다.

- Java(버전 2) - [2.10.11](#)
- Java - [1.11.704](#)
- Go - [1.23.13](#)
- Python(Boto3) - [1.9.220](#)
- Python(botocore) - [1.12.200](#)
- AWS CLI - [1.16.232](#)
- 노드 - [2.525.0](#) 및 [3.27.0](#)
- Ruby - [3.58.0](#)
- C++ - [1.7.174](#)
- .NET - [3.3.659.1](#) - AWSSDK.SecurityToken도 포함해야 합니다.
- PHP - [3.110.7](#)

[클러스터 자동 확장 처리](#), [AWS Load Balancer Controller](#)를 사용한 인터넷 트래픽 라우팅, [Kubernetes용 Amazon VPC CNI 플러그인](#) 등 많은 인기 있는 Kubernetes 추가 기능이 서비스 계정에 대한 IAM 역할을 지원합니다.

지원되는 SDK를 사용 중인지 확인하려면 컨테이너를 빌드할 때 [AWS에서의 구축을 위한 도구](#)에서 선호하는 SDK에 대한 설치 지침을 따르세요.

고려 사항

Java

Java를 사용할 때는 클래스 경로에 sts 모듈을 포함해야 합니다. 자세한 내용은 Java SDK 문서의 [WebIdentityTokenFileCredentialsProvider](#)를 참조하세요.

서명 키를 가져와서 OIDC 토큰 검증

Kubernetes는 각 Kubernetes 서비스 계정에 ProjectedServiceAccountToken을 발급합니다. 이 토큰은 JSON 웹 토큰(JWT)의 한 유형인 OIDC 토큰입니다. Amazon EKS는 외부 시스템에서 토큰을 검증할 수 있도록 토큰에 대한 서명 키가 포함된 각 클러스터에 대해 퍼블릭 OIDC 엔드포인트를 호스팅합니다.

ProjectedServiceAccountToken을 검증하려면 JSON Web Key Set(JWKS)라고도 하는 OIDC 퍼블릭 서명 키를 가져와야 합니다. 애플리케이션에서 이러한 키를 사용하여 토큰을 검증합니다. 예를 들어, [PyJWT Python 라이브러리](#)를 사용하여 이러한 키로 토큰을 검증할 수 있습니다. ProjectedServiceAccountToken에 대한 자세한 내용은 [the section called “IAM, Kubernetes 및 OpenID Connect\(OIDC\) 백그라운드 정보”](#) 섹션을 참조하세요.

사전 조건

- 클러스터에 대한 기존 AWS Identity and Access Management(IAM) OpenID Connect(OIDC) 제공자입니다. 이미 있는지 아니면 생성해야 하는지 확인하려면 [the section called “IAM OIDC 제공자”](#) 부분을 참조하세요.
- AWS CLI - Amazon EKS를 비롯한 AWS 서비스를 사용한 작업을 위한 명령줄 도구입니다. 자세한 내용은 AWS 명령줄 인터페이스 사용 설명서에서 [설치하기](#)를 참조하세요. AWS CLI 설치 후, 구성 작업도 수행하는 것이 좋습니다. 자세한 내용은 AWS 명령줄 인터페이스 사용 설명서에서 [aws config를 사용한 빠른 구성](#)을 참조하세요.

절차

1. AWS CLI를 사용하여 Amazon EKS 클러스터의 OIDC URL을 검색합니다.

```
$ aws eks describe-cluster --name my-cluster --query 'cluster.identity.oidc.issuer'
"https://oidc.eks.us-west-2.amazonaws.com/id/8EBDXXXX00BAE"
```

2. curl 또는 유사한 도구를 사용하여 공개 서명 키를 검색합니다. 그 결과, [JSON 웹 키 세트\(JWKS\)](#)가 생성됩니다.

⚠ Important

Amazon EKS는 OIDC 엔드포인트에 대한 직접 호출을 스로틀링합니다. 공개 서명 키를 캐시해야 합니다. 응답에 포함된 `cache-control` 헤더를 준수합니다.

⚠ Important

Amazon EKS는 7일마다 OIDC 서명 키를 교체합니다.

```
$ curl https://oidc.eks.us-west-2.amazonaws.com/id/8EBDXXX00BAE/keys
{"keys":
[{"kty":"RSA","kid":"2284XXX4a40","use":"sig","alg":"RS256","n":"wk1bXXXMVfQ","e":"AQAB"}]}
```

EKS Pod Identity가 포드에 AWS 서비스에 대한 액세스 권한을 부여하는 방법 알아보기

포드의 컨테이너에 있는 애플리케이션은 AWS SDK 또는 AWS CLI를 사용하여 AWS ID 및 액세스 관리(IAM) 권한을 사용하여 AWS 서비스에 API 요청을 할 수 있다. 애플리케이션은 AWS 자격 증명으로 AWS API 요청에 서명해야 합니다.

EKS Pod Identity는 Amazon EC2 인스턴스 프로필이 Amazon EC2 인스턴스에 자격 증명을 제공하는 것과 비슷한 방식으로 애플리케이션에 대한 자격 증명을 관리하는 기능을 제공합니다. AWS 자격 증명을 생성하여 컨테이너에 배포하거나 Amazon EC2 인스턴스의 역할을 사용하는 대신에 IAM 역할을 Kubernetes 서비스 계정과 연결하고 서비스 계정을 사용하도록 포드를 구성합니다.

각 EKS Pod Identity 연결은 역할을 지정된 클러스터의 네임스페이스에 있는 서비스 계정에 매핑합니다. 여러 클러스터에 동일한 애플리케이션이 있는 경우 역할의 신뢰 정책을 수정하지 않고 각 클러스터에서 동일한 연결을 생성할 수 있습니다.

포드가 연결이 있는 서비스 계정을 사용하는 경우 Amazon EKS는 포드의 컨테이너에 환경 변수를 설정합니다. 환경 변수는 AWS CLI를 포함한 AWS SDK를 구성하여 EKS Pod Identity 보안 인증 정보를 사용합니다.

EKS Pod Identity의 이점

EKS Pod Identity는 다음 이점을 제공합니다.

- **최소 권한** - IAM 권한의 범위를 서비스 계정으로 지정할 수 있습니다. 그러면 해당 서비스 계정을 사용하는 포드만 이 권한에 액세스할 수 있습니다. 또한 이 기능을 사용하면 kiam, kube2iam 같은 타사 솔루션이 필요 없습니다.
- **자격 증명 격리** - 포드의 컨테이너는 컨테이너가 사용하는 서비스 계정과 연결된 IAM 역할에 대한 자격 증명만 검색할 수 있습니다. 컨테이너는 다른 포드의 다른 컨테이너에서 사용하는 자격 증명에 액세스할 수 없습니다. Pod Identity를 사용할 때, [Amazon EC2 인스턴스 메타데이터 서비스\(IMDS\)](#)에 대한 포드 액세스를 차단하지 않는 한, 포드의 컨테이너에도 [the section called “노드 IAM 역할”](#)에 할당된 권한이 있습니다. 자세한 내용은 [작업자 노드에 할당된 인스턴스 프로필에 대한 액세스 제한](#) 부분을 참조하세요.
- **감사** - 소급적 감사를 위해 AWS CloudTrail을 통해 액세스 및 이벤트 로깅이 가능합니다.

EKS Pod Identity는 OIDC ID 제공업체를 사용하지 않으므로 [the section called “IRSA를 사용한 자격 증명”](#)에 비해 간단한 방법입니다. EKS Pod Identity의 개선 사항:

- **독립적 운영** - 많은 조직에서 OIDC ID 제공업체를 만드는 것은 Kubernetes 클러스터를 관리하는 팀이 아닌 다른 팀의 책임입니다. EKS Pod Identity는 업무가 분명히 분리되어 있어 EKS Pod Identity 연결의 모든 구성은 Amazon EKS에서, IAM 권한의 모든 구성은 IAM에서 이루어집니다.
- **재사용 가능성** - EKS Pod Identity는 서비스 계정에 대한 IAM 역할이 사용하는 각 클러스터에 대해 별도의 보안 주체 대신 단일 IAM 보안 주체를 사용합니다. IAM 관리자는 모든 역할의 신뢰 정책에 다음 보안 주체를 추가하여 EKS Pod Identity에서 사용할 수 있도록 합니다.

```
"Principal": {
  "Service": "pods.eks.amazonaws.com"
}
```

- **확장성** - 각 임시 자격 증명 세트는 각 포드에서 실행하는 각 AWS SDK 대신 EKS Pod Identity의 EKS Auth 서비스에서 수입합니다. 그러면 각 노드에서 실행되는 Amazon EKS Pod Identity 에이전트가 SDK에 보안 인증 정보를 발급합니다. 따라서 부하가 각 노드당 한 번으로 줄어들고 각 포드에 중복되지 않습니다. 프로세스의 자세한 내용은 [the section called “작동 방법”](#) 섹션을 참조하세요.

두 대안의 비교에 대한 자세한 내용은 [the section called “AWS에 대한 워크로드 액세스”](#) 섹션을 참조하세요.

EKS Pod Identity 설정 개요

다음 절차를 완료하여 EKS Pod Identity를 활성화합니다.

1. [the section called “에이전트 설정”](#) - 각 클러스터에 대해 한 번만 이 절차를 완료합니다. 클러스터에서 EKS Auto Mode가 활성화된 경우 이 단계를 완료할 필요가 없습니다.
2. [the section called “IAM 역할 할당”](#) - 애플리케이션에 부여하려는 고유한 권한 세트에 대해 이 절차를 완료합니다.
3. [the section called “포드 서비스 계정”](#) - AWS 서비스에 액세스해야 하는 각 포드에 대해 이 절차를 완료합니다.
4. [the section called “지원되는 SDK”](#) - 워크로드가 지원되는 버전의 AWS SDK를 사용하고 워크로드가 기본 보안 인증 정보 체인을 사용하는지 확인합니다.

EKS Pod Identity 고려 사항

- 각 클러스터의 각 Kubernetes 서비스 계정에 IAM 역할을 하나 연결할 수 있습니다. EKS Pod Identity 연결을 편집하여 서비스 계정에 매핑되는 역할을 변경할 수 있습니다.
- 클러스터와 동일한 AWS 계정에 있는 역할만 연결할 수 있습니다. EKS Pod Identity가 사용하도록 구성된 이 계정의 역할에 다른 계정의 액세스 권한을 위임할 수 있습니다. 액세스 권한 위임 및 AssumeRole에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 역할을 사용한 AWS 계정 간 액세스 권한 위임](#)을 참조하세요.
- EKS Pod Identity 에이전트가 필요합니다. 노드에서 Kubernetes DaemonSet로 실행되고 실행되는 노드의 포드에만 자격 증명을 제공합니다. EKS Pod Identity 에이전트 호환성에 대한 자세한 내용은 [the section called “EKS Pod Identity 제한 사항”](#) 섹션을 참조하세요.
- AWS IAM 동작과 마찬가지로, EKS Pod Identity 연결은 최종적으로 일관되고, 초기 API 직접 호출이 성공적으로 반환된 후 효력이 발생하는 데 몇 초가 걸릴 수 있습니다. 이러한 잠재적 지연을 고려하도록 애플리케이션을 설계해야 합니다. 애플리케이션의 중요 고가용성 코드 경로에 Pod Identity 연결 생성/업데이트를 포함하지 않는 것이 좋습니다. 대신 자주 실행하지 않는 별도의 초기화 루틴이나 설정 루틴에서 이를 변경하세요.
- Pod Identity Agent와 함께 포드용 보안 그룹을 사용하는 경우 AWS VPC CNI에 대한 `POD_SECURITY_GROUP_ENFORCING_MODE` 플래그를 설정해야 할 수 있습니다. 포드용 보안 그룹 고려 사항에 대한 자세한 내용은 [the section called “포드의 보안 그룹”](#) 섹션을 참조하세요.
- EKS Pod Identity 에이전트는 노드의 `hostNetwork`를 사용하고 노드의 링크-로컬 주소에 있는 포트 80 및 포트 2703을 사용합니다. 이 주소는 IPv4의 경우 169.254.170.23, IPv6 클러스터의 경우 `[fd00:ec2::23]`입니다.

IPv6 주소를 비활성화하거나 localhost IPv6 IP 주소를 차단하는 경우 에이전트를 시작할 수 없습니다. IPv6을 사용할 수 없는 노드에서 에이전트를 시작하려면 [the section called “IPv6 사용 중지”](#)의 단계에 따라 IPv6 구성을 비활성화합니다.

- 포드가 프록시를 사용하는 경우 포드에 삽입된 `no_proxy/NO_PROXY` 환경 변수에 `169.254.170.23`(IPv4) 또는 `[fd00:ec2::23]`(IPv6)을 추가해야 합니다. 그렇지 않으면 요청이 프록시로 전송되고 프록시가 IP를 라우팅할 수 없으므로 애플리케이션 포드에서 `eks-pod-identity-agent` DaemonSet으로의 요청이 실패합니다.

EKS Pod Identity 클러스터 버전

EKS Pod Identity를 사용하려면 클러스터의 플랫폼 버전이 다음 표에 나열된 버전과 동일한 버전 또는 이후 버전이거나 Kubernetes 버전이 표에 나열된 버전보다 이후 버전이어야 합니다.

Kubernetes 버전	플랫폼 버전
나열되지 않은 Kubernetes 버전	모든 플랫폼 버전 지원
1.28	eks.4
1.27	eks.8
1.26	eks.9

EKS Pod Identity 제한 사항

EKS Pod Identity는 다음에서 사용 가능합니다.

- 이전 주제 [the section called “EKS Pod Identity 클러스터 버전”](#)에 나열된 Amazon EKS 클러스터 버전.
- Linux Amazon EC2 인스턴스인 클러스터의 워커 노드.

EKS Pod Identity는 다음에서 사용할 수 없습니다.

- AWS Outposts
- Amazon EKS Anywhere.

- Amazon EC2에서 생성하고 실행하는 Kubernetes 클러스터. EKS Pod Identity 구성 요소는 Amazon EKS에서만 사용할 수 있습니다.

다음에서는 EKS Pod Identity를 사용할 수 없습니다.

- Linux Amazon EC2 인스턴스를 제외한 모든 위치에서 실행되는 포드. AWS Fargate(Fargate)에서 실행되는 Linux 및 Windows 포드는 지원되지 않습니다. Windows Amazon EC2 인스턴스에서 실행되는 포드는 지원되지 않습니다.

EKS Pod Identity 작동 방식 이해

Amazon EKS Pod Identity는 Amazon EC2 인스턴스 프로필이 Amazon EC2 인스턴스에 자격 증명을 제공하는 것과 비슷한 방식으로 애플리케이션에 대한 자격 증명을 관리하는 기능을 제공합니다.

Amazon EKS Pod Identity는 추가 EKS Auth API와 각 노드에서 실행되는 에이전트 포드를 통해 워크로드에 보안 인증 정보를 제공합니다.

Amazon EKS 추가 기능 및 자체 관리형 컨트롤러, 운영자 및 기타 추가 기능과 같은 추가 기능에서 작성자는 최신 AWS SDK를 사용하도록 소프트웨어를 업데이트해야 합니다. EKS Pod Identity와 Amazon EKS에서 만든 추가 기능 간의 호환성 목록은 이전 섹션([the section called “EKS Pod Identity 제한 사항”](#))을 참조하세요.

코드에서 EKS Pod Identity 사용

코드에서 AWS SDK를 사용하여 AWS 서비스에 액세스할 수 있습니다. 코드를 작성하여 SDK가 포함된 AWS 서비스에 대한 클라이언트를 만들고, 기본적으로 SDK는 일련의 위치에서 사용할 AWS ID 및 액세스 관리 보안 인증 정보를 검색합니다. 유효한 보안 인증 정보를 찾은 후에는 검색이 중지됩니다. 사용되는 기본 위치에 대한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 [보안 인증 정보 공급자 체인](#)을 참조하세요.

기본 보안 인증 정보 체인의 한 단계에서 검색되는 컨테이너 보안 인증 정보 공급자에 EKS Pod Identity가 추가되었습니다. 워크로드가 현재 보안 인증 정보 체인의 이전 단계에 있는 보안 인증 정보를 사용하는 경우 동일한 워크로드에 대해 EKS Pod Identity 연결을 구성하더라도 해당 보안 인증 정보는 계속 사용됩니다. 이렇게 하면 이전 보안 인증 정보를 제거하기 전에 먼저 연결을 생성하여 다른 유형의 보안 인증 정보에서 안전하게 마이그레이션할 수 있습니다.

컨테이너 보안 인증 정보 공급자는 각 노드에서 실행되는 에이전트의 임시 보안 인증 정보를 제공합니다. Amazon EKS에서 에이전트는 Amazon EKS Pod Identity 에이전트이고 Amazon Elastic Container

Service에서 에이전트는 amazon-ecs-agent입니다. SDK는 환경 변수를 사용하여 연결할 에이전트를 찾습니다.

반면 서비스 계정에 대한 IAM 역할은 AWS SDK가 AssumeRoleWithWebIdentity를 사용하여 AWS 보안 토큰 서비스와 교환해야 하는 웹 ID 토큰을 제공합니다.

EKS Pod Identity 에이전트와 포드의 작동 방식

1. Amazon EKS가 EKS Pod Identity 연결이 있는 서비스 계정을 사용하는 새 포드를 시작하면 클러스터는 포드 매니페스트에 다음 콘텐츠를 추가합니다.

```
env:
- name: AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE
  value: "/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/eks-pod-identity-token"
- name: AWS_CONTAINER_CREDENTIALS_FULL_URI
  value: "http://169.254.170.23/v1/credentials"
volumeMounts:
- mountPath: "/var/run/secrets/pods.eks.amazonaws.com/serviceaccount/"
  name: eks-pod-identity-token
volumes:
- name: eks-pod-identity-token
  projected:
    defaultMode: 420
    sources:
    - serviceAccountToken:
        audience: pods.eks.amazonaws.com
        expirationSeconds: 86400 # 24 hours
        path: eks-pod-identity-token
```

2. Kubernetes는 포드를 실행할 노드를 선택합니다. 그러면 노드의 Amazon EKS Pod Identity 에이전트는 [AssumeRoleForPodIdentity](#) 작업을 사용하여 EKS Auth API에서 임시 보안 인증 정보를 검색합니다.
3. EKS Pod Identity 에이전트는 컨테이너 내에서 실행하는 AWS SDK에 이러한 보안 인증 정보를 사용할 수 있도록 합니다.
4. 기본 보안 인증 정보 체인을 사용하도록 보안 인증 정보 공급자를 지정하지 않고도 애플리케이션에서 SDK를 사용할 수 있습니다. 또는 컨테이너 보안 인증 정보 공급자를 지정합니다. 사용되는 기본 위치에 대한 자세한 내용은 AWS SDK 및 도구 참조 가이드의 [보안 인증 정보 공급자 체인](#)을 참조하세요.
5. SDK는 환경 변수를 사용하여 EKS Pod Identity 에이전트에 연결하고 보안 인증 정보를 검색합니다.

Note

워크로드가 현재 보안 인증 정보 체인의 이전 단계에 있는 보안 인증 정보를 사용하는 경우 동일한 워크로드에 대해 EKS Pod Identity 연결을 구성하더라도 해당 보안 인증 정보는 계속 사용됩니다.

Amazon EKS Pod Identity 에이전트 설정

Amazon EKS Pod Identity는 Amazon EC2 인스턴스 프로필이 Amazon EC2 인스턴스에 자격 증명을 제공하는 것과 비슷한 방식으로 애플리케이션에 대한 자격 증명을 관리하는 기능을 제공합니다.

Amazon EKS Pod Identity는 추가 EKS Auth API와 각 노드에서 실행되는 에이전트 포드를 통해 워크로드에 보안 인증 정보를 제공합니다.

Tip

EKS Auto Mode 클러스터에 EKS Pod Identity 에이전트를 설치할 필요가 없습니다. 이 기능은 EKS Auto Mode에 내장되어 있습니다.

고려 사항

- 기본적으로 EKS Pod Identity 에이전트는 포드의 IPv4 및 IPv6 주소에서 수신 대기하여 자격 증명을 요청합니다. 에이전트는 IPv4의 경우, 루프백(localhost) IP 주소 169.254.170.23을 사용하고 IPv6의 경우, localhost IP 주소 [fd00:ec2::23]을 사용합니다.
- IPv6 주소를 비활성화하거나 localhost IPv6 IP 주소를 차단하는 경우 에이전트를 시작할 수 없습니다. IPv6을 사용할 수 없는 노드에서 에이전트를 시작하려면 [the section called “IPv6 사용 중지”](#)의 단계에 따라 IPv6 구성을 비활성화합니다.

Amazon EKS Pod Identity 에이전트 생성

에이전트 필수 조건

- 기존 Amazon EKS 클러스터. 배포하려면 [시작](#) 섹션을 참조하세요. 클러스터 버전 및 플랫폼 버전은 [EKS pod identity 클러스터 버전](#)에 나열된 버전과 동일한 버전 또는 이후 버전이어야 합니다.

- 노드 역할에는 에이전트가 EKS Auth API에서 AssumeRoleForPodIdentity 작업을 수행할 수 있는 권한이 있습니다. [AWS 관리형 정책: AmazonEKSWorkerNodePolicy](#)를 사용하거나 다음과 유사한 사용자 지정 정책을 추가할 수 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks-auth:AssumeRoleForPodIdentity"
      ],
      "Resource": "*"
    }
  ]
}
```

이 작업은 태그로 제한하여 에이전트를 사용하는 포드가 맡을 수 있는 역할을 제한할 수 있습니다.

- 노드는 Amazon ECR에 연결하여 이미지를 다운로드할 수 있습니다. 추가 기능의 컨테이너 이미지는 [Amazon EKS 추가 기능의 Amazon 컨테이너 이미지 레지스트리 보기](#)에 나열된 레지스트리에 있습니다.

AWS Management Console의 선택적 구성 설정, 그리고 AWS CLI의 `--configuration-values`에서 이미지 위치를 변경하고 EKS 추가 기능에 대해 `imagePullSecrets`를 제공할 수 있습니다.

- 노드는 Amazon EKS Auth API에 연결할 수 있습니다. 프라이빗 클러스터의 경우 AWS 프라이빗 링크의 `eks-auth` 엔드포인트가 필요합니다.

AWS 콘솔을 사용하여 에이전트 설정

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 왼쪽 탐색 창에서 클러스터를 선택한 다음 EKS Pod Identity 에이전트 추가 기능을 구성할 클러스터의 이름을 선택합니다.
3. 추가 기능 탭을 선택합니다.
4. 추가 기능 더 가져오기를 선택합니다.
5. EKS Pod Identity 에이전트 추가 기능 상자의 오른쪽 상단에 있는 상자를 선택하고 다음을 선택합니다.

- 선택한 추가 기능 설정 구성 페이지의 버전 드롭다운 목록에서 임의의 버전을 선택합니다.
- (선택사항) 선택적 구성 설정을 확장하여 추가 구성을 입력합니다. 예를 들어, 대체 컨테이너 이미지 위치 및 ImagePullSecrets를 제공할 수 있습니다. 허용된 키가 포함된 JSON Schema는 추가 기능 구성 스키마에 표시됩니다.

구성 값에 구성 키와 값을 입력합니다.

- 다음을 선택합니다.
- EKS Pod Identity 에이전트 포드가 클러스터에서 실행 중인지 확인합니다.

```
kubectl get pods -n kube-system | grep 'eks-pod-identity-agent'
```

예제 출력은 다음과 같습니다.

eks-pod-identity-agent-gmqp7	1/1	Running
1 (24h ago) 24h		
eks-pod-identity-agent-prnsh	1/1	Running
1 (24h ago) 24h		

이제 클러스터에서 EKS Pod Identity 연결을 사용할 수 있습니다. 자세한 내용은 [the section called “IAM 역할 할당”](#) 섹션을 참조하세요.

AWS CLI를 사용하여 에이전트 설정

- 다음 AWS CLI 명령을 실행합니다. my-cluster를 클러스터 이름으로 바꿉니다.

```
aws eks create-addon --cluster-name my-cluster --addon-name eks-pod-identity-agent --addon-version v1.0.0-eksbuild.1
```

Note

EKS Pod Identity 에이전트는 서비스 계정에 대한 IAM 역할에 service-account-role-arn을 사용하지 않습니다. EKS Pod Identity 에이전트에 노드 역할의 권한을 제공해야 합니다.

- EKS Pod Identity 에이전트 포드가 클러스터에서 실행 중인지 확인합니다.

```
kubectl get pods -n kube-system | grep 'eks-pod-identity-agent'
```

예제 출력은 다음과 같습니다.

eks-pod-identity-agent-gmqp7	1/1	Running
1 (24h ago) 24h		
eks-pod-identity-agent-prnsh	1/1	Running
1 (24h ago) 24h		

이제 클러스터에서 EKS Pod Identity 연결을 사용할 수 있습니다. 자세한 내용은 [the section called “IAM 역할 할당”](#) 섹션을 참조하세요.

Kubernetes 서비스 계정에 IAM 역할 할당

이 주제에서는 EKS Pod Identity 관련 AWS Identity and Access Management(IAM) 역할을 수입하도록 Kubernetes 서비스 계정을 구성하는 방법을 다룹니다. 서비스 계정을 사용하도록 구성된 모든 포드는 해당 역할에 액세스 권한이 있는 모든 AWS 서비스에 액세스할 수 있습니다.

EKS Pod Identity 연결을 생성하려면 한 단계만 거치면 됩니다. AWS Management Console, AWS CLI, AWS SDK, AWS CloudFormation 및 기타 도구를 통해 EKS에서 연결을 생성하면 됩니다. 임의 Kubernetes 객체의 클러스터 내부에 연결에 대한 데이터나 메타데이터가 없으므로 서비스 계정에 주석을 추가하지 않습니다.

사전 조건

- 기존 클러스터가 있어야 합니다. 아직 없는 경우 [시작](#) 안내서 중 하나에 따라 생성할 수 있습니다.
- 연결을 생성하는 IAM 보안 주체에 iam:PassRole이 있어야 합니다.
- AWS CLI의 최신 버전이 디바이스나 AWS CloudShell에 설치 및 구성되어 있어야 합니다. `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용하여 현재 버전을 확인할 수 있습니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.
- kubectl 명령줄 도구는 장치 또는 AWS CloudShell에 설치됩니다. 버전은 클러스터의 Kubernetes 버전과 동일하거나 최대 하나 이전 또는 이후의 마이너 버전일 수 있습니다. 예를 들어, 클러스터 버전이 1.29인 경우 kubectl 버전 1.28, 1.29 또는 1.30를 함께 사용할 수 있습니다. kubectl을 설치하거나 업그레이드하려면 [the section called “kubectl 및 eksctl 설정”](#) 부분을 참조하세요.

- 클러스터 구성이 포함된 기존 `kubectl config` 파일입니다. `kubectl config` 파일을 생성하려면 [the section called “kubectl을 사용하여 클러스터에 액세스”](#) 섹션을 참조하세요.

pod identity 연결 생성(AWS 콘솔)

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 왼쪽 탐색 창에서 클러스터를 선택한 다음 EKS Pod Identity 에이전트 추가 기능을 구성할 클러스터의 이름을 선택합니다.
3. 액세스 탭을 선택합니다.
4. Pod Identity 연결에서 생성을 선택합니다.
5. IAM 역할의 경우 워크로드에 부여하려는 권한이 있는 IAM 역할을 선택합니다.

Note

이 목록에는 EKS Pod Identity에서의 사용을 허용하는 다음 신뢰 정책이 있는 역할만 포함되어 있습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
      "Effect": "Allow",
      "Principal": {
        "Service": "pods.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

`sts:AssumeRole` - EKS Pod Identity는 `AssumeRole`을 사용하여 임시 보안 인증 정보를 포드에 전달하기 전에 IAM 역할을 말합니다.

sts:TagSession - EKS Pod Identity는 TagSession을 사용하여 세션 태그를 AWS STS 요청에 포함합니다.

신뢰 정책의 조건 키에서 이러한 태그를 사용하여 이 역할을 사용할 수 있는 서비스 계정, 네임스페이스, 클러스터를 제한할 수 있습니다.

Amazon EKS 조건 키 목록을 보려면 서비스 인증 참조의 [Amazon Elastic Kubernetes Service의 조건 키](#)를 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 [Amazon Elastic Kubernetes Service에서 정의한 작업](#)을 참조하세요.

6. Kubernetes 네임스페이스에서 서비스 계정 및 워크로드가 포함된 Kubernetes 네임스페이스를 선택합니다. 클러스터에 없는 네임스페이스를 이름으로 지정할 수 있습니다.
7. Kubernetes 서비스 계정에서 사용할 Kubernetes 서비스 계정을 선택합니다. Kubernetes 워크로드의 매니페스트에서 이 서비스 계정을 지정해야 합니다. 클러스터에 없는 서비스 계정을 이름으로 지정할 수 있습니다.
8. (선택사항) 태그의 경우, 태그 추가를 선택하여 키-값 페어에 메타데이터를 추가합니다. 이러한 태그는 연결에 적용되며 IAM 정책에서 사용할 수 있습니다.

이 단계를 반복하여 여러 개의 태그를 추가할 수 있습니다.

9. 생성을 선택합니다.

pod identity 연결 생성(AWS CLI)

1. IAM 역할에 기존 IAM 정책을 연결하려면 다음 단계로 건너뛰십시오.

IAM 정책을 생성합니다. 자체 정책을 생성하거나 필요한 권한 중 일부를 이미 부여하는 AWS 관리형 정책을 복사하여 특정 요구 사항에 맞게 사용자 지정할 수 있습니다. 자세한 내용은 IAM 사용 설명서에서 [IAM 정책 생성](#)을 참조하세요.

- a. 포드가 액세스할 AWS 서비스에 대한 권한이 포함된 파일을 생성합니다. 모든 AWS 서비스에 대한 모든 작업 목록은 [서비스 승인 참조](#)에서 확인하세요.

다음 명령을 실행하여 Amazon S3 버킷에 대한 읽기 전용 액세스를 허용하는 예제 정책 파일을 생성할 수 있습니다. 이 버킷에 구성 정보 또는 부트스트랩 스크립트를 저장할 수도 있고, 의 컨테이너는 이 버킷에서 파일을 읽어 애플리케이션으로 로드할 수 있습니다. 이 예제 정책을 생성하려면 다음 내용을 디바이스에 복사합니다. *my-pod-secrets-bucket*을 버킷 이름으로 바꾸고 명령을 실행합니다.

```
cat >my-policy.json <<EOF
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
    }
  ]
}
EOF
```

b. IAM 정책을 생성합니다.

```
aws iam create-policy --policy-name my-policy --policy-document file:///my-policy.json
```

2. IAM 역할을 생성하고 Kubernetes 서비스 계정에 연결합니다.

a. IAM 역할을 수입하려는 기존 Kubernetes 서비스 계정이 있는 경우 이 단계를 건너뛸 수 있습니다.

Kubernetes 서비스 계정을 생성합니다. 다음 콘텐츠를 디바이스에 복사합니다. *my-service-account*를 원하는 이름으로 바꾸고 필요한 경우 *##*을 다른 네임스페이스로 바꿉니다. *##*을 변경하는 경우, 네임스페이스가 이미 존재해야 합니다.

```
cat >my-service-account.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-service-account
  namespace: default
EOF
kubectl apply -f my-service-account.yaml
```

다음 명령을 실행합니다.

```
kubectl apply -f my-service-account.yaml
```

b. 다음 명령을 실행하여 IAM 역할에 대한 신뢰 정책 파일을 생성합니다.

```
cat >trust-relationship.json <<EOF
```

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
      "Effect": "Allow",
      "Principal": {
        "Service": "pods.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
EOF
```

- c. 역할을 생성합니다. *my-role*을 IAM 역할의 이름으로 바꾸고, *my-role-description*을 역할에 대한 설명으로 바꿉니다.

```
aws iam create-role --role-name my-role --assume-role-policy-document file://
trust-relationship.json --description "my-role-description"
```

- d. 역할에 IAM 정책을 연결합니다. *my-role*을 IAM 역할의 이름으로 바꾸고 *my-policy*를 생성한 기존 정책의 이름으로 바꿉니다.

```
aws iam attach-role-policy --role-name my-role --policy-
arn=arn:aws:iam::111122223333:policy/my-policy
```

Note

서비스 계정에 대한 IAM 역할과 달리 EKS Pod Identity는 서비스 계정의 주석을 사용하지 않습니다.

- e. 다음 명령을 실행하여 연결을 생성합니다. *my-cluster*를 클러스터 이름으로 바꾸고, 필요한 경우 *my-service-account*를 원하는 이름으로 바꾸고 *##*을 다른 네임스페이스로 바꿉니다.

```
aws eks create-pod-identity-association --cluster-name my-cluster --role-arn
arn:aws:iam::111122223333:role/my-role --namespace default --service-account my-
service-account
```

예제 출력은 다음과 같습니다.

```
{
  "association": {
    "clusterName": "my-cluster",
    "namespace": "default",
    "serviceAccount": "my-service-account",
    "roleArn": "arn:aws:iam::111122223333:role/my-role",
    "associationArn": "arn:aws::111122223333:podidentityassociation/my-cluster/a-abcdefghijklmnop1",
    "associationId": "a-abcdefghijklmnop1",
    "tags": {},
    "createdAt": 1700862734.922,
    "modifiedAt": 1700862734.922
  }
}
```

Note

클러스터에 없는 네임스페이스와 서비스 계정을 이름으로 지정할 수 있습니다. EKS Pod Identity 연결이 작동하려면 네임스페이스, 서비스 계정 및 서비스 계정을 사용하는 워크로드를 생성해야 합니다.

구성 확인

1. IAM 역할의 신뢰 정책이 올바르게 구성되었는지 확인합니다.

```
aws iam get-role --role-name my-role --query Role.AssumeRolePolicyDocument
```

예제 출력은 다음과 같습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow EKS Auth service to assume this role for Pod Identities",
      "Effect": "Allow",
      "Principal": {
        "Service": "pods.eks.amazonaws.com"
      }
    }
  ]
}
```

```

        },
        "Action": [
            "sts:AssumeRole",
            "sts:TagSession"
        ]
    }
]
}

```

2. 이전 단계에서 역할에 연결한 정책이 해당 역할에 연결되었는지 확인합니다.

```

aws iam list-attached-role-policies --role-name my-role --query
'AttachedPolicies[].PolicyArn' --output text

```

예제 출력은 다음과 같습니다.

```

arn:aws:iam::111122223333:policy/my-policy

```

3. 사용하려는 정책의 Amazon 리소스 이름(ARN)을 저장할 변수를 설정합니다. *my-policy*를 권한을 확인하려는 정책의 이름으로 바꿉니다.

```

export policy_arn=arn:aws:iam::111122223333:policy/my-policy

```

4. 정책의 기본 버전을 봅니다.

```

aws iam get-policy --policy-arn $policy_arn

```

예제 출력은 다음과 같습니다.

```

{
  "Policy": {
    "PolicyName": "my-policy",
    "PolicyId": "EXAMPLEBIOWGLDEXAMPLE",
    "Arn": "arn:aws:iam::111122223333:policy/my-policy",
    "Path": "/",
    "DefaultVersionId": "v1",
    [...]
  }
}

```


5. 정책 내용을 보고 포드에 필요한 모든 권한이 정책에 포함되어 있는지 확인합니다. 필요한 경우 다음 명령에서 **1**을 이전 출력에서 반환된 버전으로 바꿉니다.

```
aws iam get-policy-version --policy-arn $policy_arn --version-id v1
```

예제 출력은 다음과 같습니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-pod-secrets-bucket"
    }
  ]
}
```

이전 단계에서 예제 정책을 생성한 경우 출력은 동일합니다. 다른 정책을 생성한 경우 **##** 내용이 다릅니다.

다음 단계

[the section called “포드 서비스 계정”](#)

EKS Pod Identity 대상 IAM 역할을 사용하여 AWS 리소스에 액세스

Amazon Elastic Kubernetes Service(Amazon EKS)에서 애플리케이션을 실행할 때 동일하거나 다른 AWS 계정에 있는 AWS 리소스에 액세스해야 할 수 있습니다. 이 설명서에서는 Kubernetes 포드가 다른 AWS 리소스에 액세스할 수 있도록 EKS Pod Identity를 사용하여 이러한 계정 간에 액세스를 설정하는 방법을 보여줍니다.

사전 조건

시작하기 전에 다음 단계를 완료합니다.

- [Amazon EKS Pod Identity 에이전트 설정](#)
- [EKS Pod Identity 역할 생성](#)

작동 방식

Pod Identity를 사용하면 역할 체이닝이라는 프로세스를 통해 EKS 클러스터의 애플리케이션이 계정 전체에서 AWS 리소스에 액세스할 수 있습니다. Pod Identity 연결을 생성할 때 두 가지 IAM 역할, 즉 EKS 클러스터와 동일한 계정의 [EKS Pod Identity 역할](#)과 AWS 리소스(예: S3 버킷 또는 DynamoDB 테이블)가 포함된 계정의 대상 IAM 역할을 제공할 수 있습니다. [EKS Pod Identity 역할](#)은 [IAM PassRole](#) 요구 사항으로 인해 EKS 클러스터의 계정에 있어야 하는 반면, 대상 IAM 역할은 모든 AWS 계정에 있을 수 있습니다. PassRole을 사용하면 AWS 엔터티가 역할 수임을 다른 서비스에 위임할 수 있습니다. EKS Pod Identity는 PassRole을 사용하여 역할을 Kubernetes 서비스 계정에 연결하므로 역할을 전달하는 ID와 역할이 모두 EKS 클러스터와 동일한 AWS 계정에 있어야 합니다. 애플리케이션 포드가 AWS 리소스에 액세스해야 하는 경우 Pod Identity에서 자격 증명을 요청합니다. 그런 다음 Pod Identity는 두 가지 역할 가정을 순서대로 자동으로 수행합니다. 먼저 [EKS Pod Identity 역할](#)을 수입한 다음 해당 자격 증명을 사용하여 대상 IAM 역할을 수입합니다. 이 프로세스는 포드에 대상 역할에 정의된 권한이 있는 임시 자격 증명을 제공하여 다른 AWS 계정의 리소스에 대한 보안 액세스를 허용합니다.

캐싱 고려 사항

캐싱 메커니즘으로 인해 기존 Pod Identity 연결의 IAM 역할에 대한 업데이트는 EKS 클러스터에서 실행되는 포드에 즉시 적용되지 않을 수 있습니다. Pod Identity Agent는 자격 증명을 가져올 때 연결의 구성을 기반으로 IAM 자격 증명을 캐싱합니다. 연결에 [EKS Pod Identity 역할](#) ARN만 포함되어 있고 대상 IAM 역할이 없는 경우 캐시된 자격 증명은 6시간 동안 지속됩니다. 연결에 [EKS Pod Identity 역할](#) ARN과 대상 IAM 역할이 모두 포함된 경우 캐시된 자격 증명은 59분 동안 지속됩니다. [EKS Pod Identity 역할](#) ARN 업데이트 또는 대상 IAM 역할 추가와 같은 기존 연결을 수정해도 기존 캐시는 재설정되지 않습니다. 따라서 에이전트는 캐시된 자격 증명에 새로 고쳐질 때까지 업데이트를 인식하지 못합니다. 변경 사항을 더 빨리 적용하려면 기존 포드를 다시 생성할 수 있습니다. 그렇지 않으면 캐시가 만료될 때까지 기다려야 합니다.

1단계: 대상 IAM 역할 생성 및 연결

이 단계에서는 대상 IAM 역할을 생성하고 구성하여 보안 신뢰 체인을 설정합니다. 데모를 위해 두 AWS 계정 간에 신뢰 체인을 구축하기 위해 새로운 대상 IAM 역할을 생성합니다. EKS 클러스터의 AWS 계정에 있는 [EKS Pod Identity 역할](#)(예: eks-pod-identity-primary-role)은 대상 계정에서 대상 IAM 역할(예: eks-pod-identity-aws-resources)을 수행할 수 있는 권한을 얻어 Amazon S3 버킷과 같은 AWS 리소스에 액세스할 수 있습니다.

대상 IAM 역할 생성

1. [Amazon IAM 콘솔](#)을 엽니다.

2. 상단 탐색 모음에서 대상 IAM 역할에 대한 AWS 리소스(예: S3 버킷 또는 DynamoDB 테이블)가 포함된 계정에 로그인했는지 확인합니다.
3. 왼쪽 탐색 창에서 역할을 선택합니다.
4. 역할 생성 버튼을 선택하고 '신뢰할 수 있는 엔터티 유형' 아래에서 AWS 계정을 선택합니다.
5. 다른 AWS 계정을 선택하고 AWS 계정 번호([EKS Pod Identity 역할](#)이 있는 계정)을 입력한 후 다음을 선택합니다.
6. 역할에 연결하려는 권한 정책(예: AmazonS3FullAccess)을 추가하고 다음을 선택합니다.
7. MyCustomIAMTargetRole 등의 역할 이름을 입력하고 역할 생성을 선택합니다.

대상 IAM 역할 신뢰 정책 업데이트

1. 역할을 생성한 후 역할 목록으로 돌아갑니다. 이전 단계에서 생성한 새 역할(MyCustomIAMTargetRole)을 찾아 선택합니다.
2. 신뢰 관계 탭을 선택합니다.
3. 오른쪽에서 신뢰 정책 편집을 클릭합니다.
4. 정책 편집기에서 기본 JSON을 신뢰 정책으로 바꿉니다. IAM 역할 ARN의 역할 이름 및 111122223333에 대한 자리 표시자 값을 EKS 클러스터를 호스팅하는 AWS 계정 ID로 바꿉니다.
예시:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/eks-pod-identity-primary-role"
      },
      "Action": "sts:AssumeRole",
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/eks-pod-identity-primary-role"
      },
      "Action": "sts:TagSession"
    }
  ]
}
```

}

EKS Pod Identity 역할에 대한 권한 정책 업데이트

이 단계에서는 대상 IAM 역할 ARN을 리소스로 추가하여 Amazon EKS 클러스터와 연결된 [EKS Pod Identity 역할](#)의 권한 정책을 업데이트합니다.

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 왼쪽 탐색 창에서 클러스터를 선택하고 EKS 클러스터의 이름을 선택합니다.
3. 액세스 탭을 선택합니다.
4. Pod Identity 연결에서 [EKS Pod Identity 역할](#)을 선택합니다.
5. 권한, 권한 추가, 인라인 정책 생성을 차례로 선택합니다.
6. 오른쪽에서 JSON을 선택합니다.
7. 정책 편집기에서 기본 JSON을 권한 정책으로 바꿉니다. IAM 역할 ARN의 역할 이름 및 222233334444에 대한 자리 표시자 값을 대상 IAM 역할로 바꿉니다. 예시:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ],
      "Resource": "arn:aws:iam::222233334444:role/eks-pod-identity-aws-resources"
    }
  ]
}
```

2단계: Kubernetes 서비스 계정에 대상 IAM 역할 연결

이 단계에서는 대상 IAM 역할과 EKS 클러스터의 Kubernetes 서비스 계정 간에 연결을 생성합니다.

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 왼쪽 탐색 창에서 클러스터를 선택하고 연결을 추가할 클러스터의 이름을 선택합니다.
3. 액세스 탭을 선택합니다.
4. Pod Identity 연결에서 생성을 선택합니다.

5. 워크로드가 수임할 IAM 역할에서 [EKS Pod Identity 역할](#)을 선택합니다.
6. 대상 IAM 역할에서 [EKS Pod Identity](#) 역할이 수임할 대상 IAM 역할을 선택합니다.
7. Kubernetes 네임스페이스 필드에 연결을 생성하려는 네임스페이스의 이름(예: my-app-namespace)을 입력합니다. 이는 서비스 계정이 상주하는 위치를 정의합니다.
8. Kubernetes 서비스 계정 필드에 IAM 자격 증명을 사용할 서비스 계정의 이름(예: my-service-account)을 입력합니다. 이렇게 하면 IAM 역할이 서비스 계정에 연결됩니다.
9. 생성을 선택하여 연결을 생성합니다.

(선택 사항) 3단계: IAM 대상 역할에 외부 권한 추가

이따금 AWS 리소스에 대한 액세스를 타사에 부여해야 할 때가 있습니다(액세스 위임). 예를 들어 Example Corp라는 타사를 고용해 AWS 계정을 모니터링하고 비용을 최적화하려고 합니다. 일일 경비를 추적하기 위해 Example Corp은 AWS 리소스에 접근해야 합니다. 이 경우 [혼동된 대리자](#) 문제를 방지하려면 IAM 대상 역할의 신뢰 정책에 ExternalId를 추가하는 것이 좋습니다.

신뢰 정책을 편집합니다.

1. 역할을 생성한 후 역할 목록으로 돌아갑니다. 이전 단계에서 생성한 새 역할(예: MyCustomIAMTargetRole)을 찾아 클릭합니다.
2. 신뢰 관계 탭을 선택합니다.
3. 오른쪽에서 신뢰 정책 편집을 클릭합니다.
4. 정책 편집기에서 기본 JSON을 신뢰 정책으로 바꿉니다. aws-region/other-account/cluster-name/namespace/service-account-name의 ExternalId 자리 표시자 값을 바꿉니다. 여기서 'region'은 클러스터의 AWS 리전이고, '111122223333'은 다른 AWS 계정 ID이고, 'cluster-name'은 EKS 클러스터 이름이고, 'namespace'는 Kubernetes 네임스페이스이고, 'service-account-name'은 Kubernetes 서비스 계정 이름입니다. 예시:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/eks-pod-identity-primary-role"
      },
      "Action": "sts:AssumeRole",
```

```

    "Condition": {
      "StringEquals": {
        "sts:ExternalId": "region/111122223333/cluster-name/namespace/
service-account-name"
      }
    },
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/eks-pod-identity-primary-role"
      },
      "Action": "sts:TagSession"
    }
  ]
}

```

서비스 계정으로 AWS 서비스에 액세스하도록 포드 구성

포드가 AWS 서비스에 액세스해야 하는 경우 Kubernetes 서비스 계정을 사용하도록 구성해야 합니다. 서비스 계정은 AWS 서비스에 액세스할 수 있는 권한이 있는 AWS ID 및 액세스 관리(IAM) 역할에 연결되어야 합니다.

- 기존 클러스터가 있어야 합니다. 아직 없는 경우 [시작](#) 가이드 중 하나를 사용하여 생성할 수 있습니다.
- 기존 Kubernetes 서비스 계정 및 서비스 계정을 IAM 역할에 연결하는 EKS Pod Identity 연결 포드가 AWS 서비스를 사용하는 데 필요한 권한을 포함하는 연결된 IAM 정책이 역할에 있어야 합니다. 서비스 계정 및 역할을 만들고 구성하는 방법에 대한 자세한 내용을 알아보려면 [the section called “IAM 역할 할당”](#) 섹션을 참조하세요.
- AWS CLI의 최신 버전이 디바이스나 AWS CloudShell에 설치 및 구성되어 있어야 합니다. `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용하여 현재 버전을 확인할 수 있습니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.
- kubectl 명령줄 도구는 장치 또는 AWS CloudShell에 설치됩니다. 버전은 클러스터의 Kubernetes 버전과 동일하거나 최대 하나 이전 또는 이후의 마이너 버전일 수 있습니다. 예를 들어, 클러스터 버전이 1.29인 경우 kubectl 버전 1.28, 1.29 또는 1.30를 함께 사용할 수 있습니다. kubectl을 설치하거나 업그레이드하려면 [the section called “kubectl 및 eksctl 설정”](#) 부분을 참조하세요.

- 클러스터 구성이 포함된 기존 `kubectl config` 파일입니다. `kubectl config` 파일을 생성하려면 [the section called “kubectl을 사용하여 클러스터에 액세스”](#) 섹션을 참조하세요.

- 다음 명령을 사용하여 구성을 확인할 포드를 배포할 수 있는 배포 매니페스트를 생성합니다. `##` `#`을 사용자의 값으로 바꿉니다.

```
cat >my-deployment.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      serviceAccountName: my-service-account
      containers:
      - name: my-app
        image: public.ecr.aws/nginx/nginx:X.XX
EOF
```

- 클러스터에 매니페스트를 배포합니다.

```
kubectl apply -f my-deployment.yaml
```

- 포드에 필요한 환경 변수가 있는지 확인합니다.

- 이전 단계에서 배포와 함께 배포된 포드를 봅니다.

```
kubectl get pods | grep my-app
```

예제 출력은 다음과 같습니다.

```
my-app-6f4dfff6cb-76cv9    1/1    Running    0    3m28s
```

- 포드에 서비스 계정 토큰 파일 탑재가 있는지 확인합니다.

```
kubectl describe pod my-app-6f4dfff6cb-76cv9 | grep
AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE:
```

예제 출력은 다음과 같습니다.

```
AWS_CONTAINER_AUTHORIZATION_TOKEN_FILE: /var/run/secrets/
pods.eks.amazonaws.com/serviceaccount/eks-pod-identity-token
```

4. 역할에 연결된 IAM 정책에서 할당한 권한을 사용하여 포드가 AWS 서비스와 상호 작용할 수 있는지 확인합니다.

Note

포드가 서비스 계정과 연결된 IAM 역할의 AWS 자격 증명을 사용하는 경우 해당 포드의 컨테이너에 있는 AWS CLI나 다른 SDK는 역할이 제공하는 자격 증명을 사용합니다. [Amazon EKS 노드 IAM 역할](#)에 제공된 자격 증명에 대한 액세스를 제한하지 않으면 포드는 계속해서 해당 자격 증명에 액세스할 수 있습니다. 자세한 내용은 [작업자 노드에 할당된 인스턴스 프로필에 대한 액세스 제한](#) 부분을 참조하세요.

포드가 예상대로 서비스와 상호 작용할 수 없으면 다음 단계를 완료하여 모두 제대로 구성되었는지 확인합니다.

- a. 포드에서 EKS Pod Identity 연결을 통해 IAM 역할을 수입할 수 있도록 지원하는 AWS SDK 버전을 사용하는지 확인합니다. 자세한 내용은 [the section called “지원되는 SDK”](#) 단원을 참조하십시오.
- b. 배포에서 서비스 계정을 사용하고 있는지 확인합니다.

```
kubectl describe deployment my-app | grep "Service Account"
```

예제 출력은 다음과 같습니다.

```
Service Account: my-service-account
```


태그를 기반으로 포드에 AWS 리소스에 대한 액세스 권한 부여

속성 기반 액세스 제어(ABAC)는 속성을 함께 결합하는 정책을 통해 사용자에게 권한을 부여합니다. EKS Pod Identity는 클러스터 이름, 네임스페이스, 서비스 계정 이름 등의 속성을 사용하여 각 포드의 임시 자격 증명에 태그를 연결합니다. 관리자는 이러한 역할 세션 태그를 사용하여 일치하는 태그를 기반으로 AWS 리소스에 대한 액세스를 허용하여 여러 서비스 계정에서 작업할 수 있는 단일 역할을 작성할 수 있습니다. 역할 세션 태그에 대한 지원을 추가함으로써 동일한 IAM 역할과 IAM 정책을 재사용하면서 클러스터 간 및 클러스터 내 워크로드 간에 더 엄격한 보안 경계를 적용할 수 있습니다.

태그가 있는 샘플 정책

다음은 해당 객체에 EKS 클러스터 이름으로 태그가 지정되었을 때 `s3:GetObject` 권한을 부여하는 IAM 정책 예제입니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:GetObjectTagging"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "s3:ExistingObjectTag/eks-cluster-name": "${aws:PrincipalTag/eks-cluster-name}"
        }
      }
    }
  ]
}
```

세션 태그 활성화 또는 비활성화

EKS Pod Identity는 역할을 수임할 때 사전 정의된 세션 태그 세트를 추가합니다. 관리자는 이러한 세션 태그를 사용하여 일치하는 태그를 기반으로 AWS 리소스에 대한 액세스를 허용하여 여러 리소스에서 작업할 수 있는 단일 역할을 작성할 수 있습니다.

세션 태그 활성화

세션 태그는 EKS Pod Identity에서 자동으로 활성화되므로 사용자가 별도의 조치를 취할 필요가 없습니다. 기본적으로 EKS Pod Identity는 미리 정의된 태그 세트를 세션에 연결합니다. 정책에서 이러한 태그를 참조하려면 구문 `${aws:PrincipalTag/}`와 태그 키를 사용합니다. 예를 들어 `${aws:PrincipalTag/kubernetes-namespaces}`입니다.

- eks-cluster-arn
- eks-cluster-name
- kubernetes-namespaces
- kubernetes-service-account
- kubernetes-pod-name
- kubernetes-pod-uid

세션 태그 비활성화

AWS는 인라인 세션 정책, 관리형 정책 ARN 및 세션 태그를 별도의 제한이 있는 압축된 바이너리 형식으로 압축합니다. 압축된 바이너리 형식이 크기 제한을 초과했다는 `PackedPolicyTooLarge` 오류가 발생하면 EKS Pod Identity에서 추가한 세션 태그를 비활성화하여 크기를 줄여볼 수 있습니다. 이러한 세션 태그를 비활성화하려면 다음 단계를 따르세요.

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 왼쪽 탐색 창에서 클러스터를 선택한 다음 수정할 클러스터의 이름을 선택합니다.
3. 액세스 탭을 선택합니다.
4. Pod Identity 연결의 연결 ID에서 수정하려는 연결 ID를 선택하고 편집을 선택합니다.
5. 세션 태그에서 세션 태그 비활성화를 선택합니다.
6. 변경 사항 저장을 선택합니다.

교차 계정 태그

EKS Pod Identity에서 추가한 모든 세션 태그는 전이적이며, 워크로드가 역할을 다른 계정으로 전환하는 데 사용하는 모든 AssumeRole 작업에 태그 키와 값이 전달됩니다. 다른 계정의 정책에 이러한 태그를 사용하여 교차 계정 시나리오에서 액세스를 제한할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [세션 태그를 사용하는 역할 체인](#)을 참조하세요.

사용자 지정 태그

EKS Pod Identity는 수행하는 AssumeRole 작업에 사용자 지정 태그를 추가할 수 없습니다. 하지만 IAM 역할에 적용하는 태그는 항상 동일한 형식(`${aws:PrincipalTag/ 다음에 키 사용}`)을 통해 사용 가능합니다(예: `${aws:PrincipalTag/MyCustomTag}`).

Note

sts:AssumeRole 요청을 통해 세션에 추가된 태그가 충돌하는 경우 우선시됩니다. 예를 들어, 다음 내용을 가정해 봅니다.

- 예를 들어, EKS가 고객 역할을 수입할 때 Amazon EKS가 세션에 `eks-cluster-name` 키와 `my-cluster` 값을 추가한다고 가정하고
- IAM 역할에 `my-own-cluster` 값이 있는 `eks-cluster-name` 태그를 추가합니다.

이 경우 전자가 우선하여 `eks-cluster-name` 태그의 값은 `my-cluster`입니다.

AWS SDK와 함께 Pod Identity 사용

EKS Pod Identity 보안 인증 정보 사용

EKS Pod Identity 연결의 보안 인증 정보를 사용하려면 코드에서 임의의 AWS SDK를 사용하여 SDK가 포함된 AWS 서비스에 대한 클라이언트를 만들 수 있고, 기본적으로 SDK는 일련의 위치에서 사용할 AWS ID 및 액세스 관리 보안 인증 정보를 검색합니다. 클라이언트를 생성하거나 SDK를 초기화했을 때 보안 인증 정보 공급자를 지정하지 않으면 EKS Pod Identity 보안 인증 정보가 사용됩니다.

이는 기본 보안 인증 정보 체인의 한 단계에서 검색되는 컨테이너 보안 인증 정보 공급자에 EKS Pod Identity가 추가되기 때문에 가능합니다. 워크로드가 현재 보안 인증 정보 체인의 이전 단계에 있는 보안 인증 정보를 사용하는 경우 동일한 워크로드에 대해 EKS Pod Identity 연결을 구성하더라도 해당 보안 인증 정보는 계속 사용됩니다.

EKS Pod Identity 작동 방식에 대한 자세한 내용은 [the section called “작동 방법”](#) 섹션을 참조하세요.

[EKS Pod Identity가 포드에 AWS 서비스 액세스 권한을 부여하는 방법 알아보기](#) 사용 시 포드의 컨테이너는 EKS Pod Identity 에이전트에서 IAM 역할을 수임하는 것을 지원하는 AWS SDK 버전을 사용해야 합니다. AWS SDK에 대해 다음과 같은 버전 또는 이후 버전을 사용해야 합니다.

- Java(버전 2) – [2.21.30](#)
- Java – [1.12.746](#)
- Go v1 – [v1.47.11](#)
- Go v2 – [release-2023-11-14](#)
- Python(Boto3) – [1.34.41](#)
- Python(botocore) - [1.34.41](#)
- AWS CLI – [1.30.0](#)

AWS CLI – [2.15.0](#)

- JavaScript v2 – [2.1550.0](#)
- JavaScript v3 - [v3.458.0](#)
- Kotlin - [v1.0.1](#)
- Ruby – [3.188.0](#)
- Rust - [release-2024-03-13](#)
- C++ - [1.11.263](#)
- .NET - [3.7.734.0](#)
- PowerShell - [4.1.502](#)
- PHP – [3.287.1](#)

지원되는 SDK를 사용 중인지 확인하려면 컨테이너를 빌드할 때 [AWS에서의 구축을 위한 도구](#)에서 선호하는 SDK에 대한 설치 지침을 따르세요.

EKS Pod Identity를 지원하는 추가 기능 목록은 [the section called “Pod Identity 지원 참조”](#) 섹션을 참조하세요.

EKS Pod Identity 에이전트에서 IPv6 비활성화

AWS Management Console

1. EKS Pod Identity 에이전트에서 IPv6를 비활성화하려면 EKS 추가 기능의 선택적 구성 설정에 다음 구성을 추가합니다.

- a. [Amazon EKS 콘솔](#)을 엽니다.
- b. 왼쪽 탐색 창에서 클러스터를 선택한 다음에 추가 기능을 구성할 클러스터의 이름을 선택합니다.
- c. 추가 기능 탭을 선택합니다.
- d. EKS Pod Identity 에이전트 추가 기능 상자의 오른쪽 상단에 있는 상자를 선택하고 편집을 선택합니다.
- e. EKS Pod Identity 에이전트 구성 페이지에서
 - i. 사용할 버전을 선택합니다. 이전 단계와 동일한 버전을 유지하고 별도의 작업으로 버전과 구성을 업데이트하는 것이 좋습니다.
 - ii. 선택적 구성 설정을 확장합니다.
 - iii. 구성 값에 중첩 JSON 객체의 JSON 키 "agent": 및 값과 키 "additionalArgs":를 입력합니다. 결과 텍스트는 유효한 JSON 객체여야 합니다. 이 키와 값이 텍스트 상자의 유일한 데이터인 경우, 키와 값을 중괄호({ })로 둘러쌉니다. 다음 예시는 네트워크 정책이 활성화된 것을 보여줍니다.

```
{
  "agent": {
    "additionalArgs": {
      "-b": "169.254.170.23"
    }
  }
}
```

이 구성은 IPv4 주소를 에이전트가 사용하는 유일한 주소로 설정합니다.

- f. EKS Pod Identity 에이전트 포드를 바꿔서 새 구성을 적용하려면 변경 사항 저장을 선택합니다.

Amazon EKS는 EKS Pod Identity 에이전트용 Kubernetes DaemonSet의 롤아웃을 사용하여 EKS 추가 기능에 변경 사항을 적용합니다. AWS Management Console의 추가 기능 업데이트 기록과 `kubectl rollout status daemonset/eks-pod-identity-agent --namespace kube-system`을 사용하여 롤아웃 상태를 추적할 수 있습니다.

`kubectl rollout`에는 다음 명령이 있습니다.

```
$ kubectl rollout

history  -- View rollout history
pause   -- Mark the provided resource as paused
restart  -- Restart a resource
```

```
resume    -- Resume a paused resource
status    -- Show the status of the rollout
undo      -- Undo a previous rollout
```

롤아웃이 너무 오래 걸리면 Amazon EKS가 롤아웃을 취소하고 유형이 추가 기능 업데이트이고 상태가 실패인 메시지가 추가 기능의 업데이트 기록에 추가됩니다. 문제를 조사하려면 롤아웃 기록부터 시작하고 EKS Pod Identity 에이전트 포드에서 `kubectl logs`를 실행하여 EKS Pod Identity 에이전트의 로그를 확인합니다.

- 업데이트 기록의 새 항목 상태가 성공이면 롤아웃이 완료되었으며 추가 기능이 모든 EKS Pod Identity 에이전트 포드에서 새 구성을 사용하고 있는 것입니다.

AWS CLI

- EKS Pod Identity 에이전트에서 IPv6를 비활성화하려면 EKS 추가 기능의 구성 값에 다음 구성을 추가합니다.

다음 AWS CLI 명령을 실행합니다. `my-cluster`를 본인의 클러스터 이름으로 바꾸고 IAM 역할 ARN을 사용 중인 역할로 바꿉니다.

```
aws eks update-addon --cluster-name my-cluster --addon-name eks-pod-identity-agent \
  --resolve-conflicts PRESERVE --configuration-values '{"agent":{"additionalArgs":
  { "-b": "169.254.170.23"}}}'
```

이 구성은 IPv4 주소를 에이전트가 사용하는 유일한 주소로 설정합니다.

Amazon EKS는 EKS Pod Identity 에이전트용 Kubernetes DaemonSet의 롤아웃을 사용하여 EKS 추가 기능에 변경 사항을 적용합니다. AWS Management Console의 추가 기능 업데이트 기록과 `kubectl rollout status daemonset/eks-pod-identity-agent --namespace kube-system`을 사용하여 롤아웃 상태를 추적할 수 있습니다.

`kubectl rollout`에는 다음 명령이 있습니다.

```
kubectl rollout

history  -- View rollout history
pause    -- Mark the provided resource as paused
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
```

```
undo      -- Undo a previous rollout
```

롤아웃이 너무 오래 걸리면 Amazon EKS가 롤아웃을 취소하고 유형이 추가 기능 업데이트이고 상태가 실패인 메시지가 추가 기능의 업데이트 기록에 추가됩니다. 문제를 조사하려면 롤아웃 기록부터 시작하고 EKS Pod Identity 에이전트 포드에서 `kubectl logs`를 실행하여 EKS Pod Identity 에이전트의 로그를 확인합니다.

EKS Pod Identity에서 요구하는 신뢰 정책으로 IAM 역할 생성

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
      "Effect": "Allow",
      "Principal": {
        "Service": "pods.eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

sts:AssumeRole

EKS Pod Identity는 AssumeRole을 사용하여 임시 보안 인증 정보를 포드에 전달하기 전에 IAM 역할을 말합니다.

sts:TagSession

EKS Pod Identity는 TagSession을 사용하여 세션 태그를 AWS STS 요청에 포함합니다.

신뢰 정책의 조건 키에서 이러한 태그를 사용하여 이 역할을 사용할 수 있는 서비스 계정, 네임스페이스, 클러스터를 제한할 수 있습니다.

Amazon EKS 조건 키 목록을 보려면 서비스 인증 참조의 [Amazon Elastic Kubernetes Service의 조건 키](#)를 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 [Amazon Elastic Kubernetes Service에서 정의한 작업](#)을 참조하세요.

노드를 사용하여 컴퓨팅 리소스 관리

Kubernetes 노드는 컨테이너화된 애플리케이션을 실행하는 시스템입니다. 각 노드에는 다음과 같은 구성 요소가 있습니다.

- [컨테이너 런타임](#) – 컨테이너 실행을 담당하는 소프트웨어입니다.
- [kubelet](#) – 컨테이너가 정상 상태이고 연결된 포드 내에서 실행되고 있는지 확인합니다.
- [kube-proxy](#) – 포드에 대한 통신을 허용하는 네트워크 규칙을 유지 관리합니다.

자세한 내용은 Kubernetes 문서의 [노드](#)를 참조하세요.

Amazon EKS 클러스터는 [EKS 자동 모드 관리형 노드](#), [자체 관리형 노드](#), [Amazon EKS 관리형 노드 그룹](#), [AWS Fargate](#), [Amazon EKS Hybrid Nodes](#)의 모든 조합에서 포드를 예약할 수 있습니다. 클러스터에 배포된 노드에 대한 자세한 내용은 [the section called “클러스터 리소스 액세스”](#) 부분을 참조하세요.

Note

하이브리드 노드를 제외한 노드는 클러스터를 생성할 때 선택한 서브넷과 동일한 VPC에 있어야 합니다. 하지만 노드가 동일한 서브넷에 있을 필요는 없습니다.

컴퓨팅 옵션 비교

다음 표에는 요구 사항에 가장 적합한 옵션을 결정할 때 평가할 몇 가지 기준이 나와 있습니다. 자체 관리형 노드는 나열된 모든 기준을 지원하는 또 다른 옵션이지만 필요한 수동 유지 관리 작업이 훨씬 더 많습니다. 자세한 내용은 [the section called “자체 관리형 노드”](#) 단원을 참조하십시오.

Note

Bottlerocket은 이 표의 일반 정보와 몇 가지 구체적인 차이점이 있습니다. 자세한 내용은 GitHub의 Bottlerocket [설명서](#)를 참조하세요.

기준	EKS 관리형 노드 그룹	EKS Auto Mode	Amazon EKS Hybrid Nodes
AWS Outposts 에 배포 가능	아니요	아니요	아니요
AWS Local Zones 에 배포할 수 있습니다.	예	아니요	아니요
Windows가 필요한 컨테이너를 실행할 수 있습니다.	예	아니요	아니요
Linux가 필요한 컨테이너를 실행할 수 있습니다.	예	예	예
Inferentia 칩이 필요한 워크로드를 실행할 수 있습니다.	예 — Amazon Linux 노드만 해당	예	아니요
GPU가 필요한 워크로드를 실행할 수 있습니다.	예 — Amazon Linux 노드만 해당	예	예
Arm 프로세서가 필요한 워크로드를 실행할 수 있습니다.	예	예	예
AWS Bottlerocket 을 실행할 수 있습니다.	예	예	아니요
포드가 CPU, 메모리, 스토리지 및 네트워크 리소스를 다른 포드와 공유합니다.	예	예	예

기준	EKS 관리형 노드 그룹	EKS Auto Mode	Amazon EKS Hybrid Nodes
Amazon EC2 인스턴스를 배포하고 관리해야 합니다.	예	아니요- EC2 관리형 인스턴스 알아보기	예-온프레미스의 물리적 또는 가상 머신은 사용자가 선택한 도구로 관리합니다.
Amazon EC2 인스턴스의 운영 체제를 보호, 유지 관리 및 패치해야 합니다.	예	아니요	예-물리적 머신 또는 가상 머신에서 실행되는 운영 체제는 사용자가 선택한 도구로 관리합니다.
노드 배포 시 추가 kubelet 인수와 같은 부트스트랩 인수를 제공할 수 있습니다.	예-사용자 지정 AMI와 함께 eksctl 또는 시작 템플릿 을 사용합니다.	아니요- NodeClass 를 사용하여 노드 구성	예-nodeadm을 사용하여 부트스트랩 인수를 사용자 지정할 수 있습니다. the section called “하이브리드 노드 nodeadm” 섹션을 참조하세요.
노드에 할당된 IP 주소와 다른 CIDR 블록의 포드에 IP 주소를 할당할 수 있습니다.	예 - 사용자 지정 AMI와 함께 시작 템플릿 사용. 자세한 내용은 the section called “시작 템플릿” 단원을 참조하십시오.	아니요	예- the section called “CNI 구성” 섹션을 참조하세요.
노드에 SSH를 연결할 수 있습니다.	예	아니요- 노드 문제 해결 방법 알아보기	예
사용자 지정 AMI 노드에 배포할 수 있습니다.	예 - 시작 템플릿 사용	아니요	예

기준	EKS 관리형 노드 그룹	EKS Auto Mode	Amazon EKS Hybrid Nodes
사용자 지정 CNI 노드에 배포할 수 있습니다.	예 — 사용자 지정 AMI와 함께 시작 템플릿 사용	아니요	예
노드 AMI를 직접 업데이트해야 합니다.	예 - Amazon EKS 최적화 AMI를 배포한 경우 업데이트가 제공되면 Amazon EKS 콘솔에 알림이 표시됩니다. 콘솔에서 클릭 한 번으로 업데이트를 수행할 수 있습니다. 사용자 정의 AMI를 배포한 경우 업데이트가 제공되면 Amazon EKS 콘솔에 알림이 표시되지 않습니다. 업데이트는 직접 수행해야 합니다.	아니요	예-물리적 머신 또는 가상 머신에서 실행되는 운영 체제는 사용자가 선택한 도구로 관리합니다. the section called “운영 체제 준비” 섹션을 참조하세요.
노드 Kubernetes 버전을 직접 업데이트해야 합니다.	예 - Amazon EKS 최적화 AMI를 배포한 경우 업데이트가 제공되면 Amazon EKS 콘솔에 알림이 표시됩니다. 콘솔에서 클릭 한 번으로 업데이트를 수행할 수 있습니다. 사용자 정의 AMI를 배포한 경우 업데이트가 제공되면 Amazon EKS 콘솔에 알림이 표시되지 않습니다. 업데이트는 직접 수행해야 합니다.	아니요	예-자체 선택 도구 또는 nodeadm을 사용하여 하이브리드 노드 업그레이드를 관리합니다. the section called “하이브리드 노드 업그레이드” 섹션을 참조하세요.

기준	EKS 관리형 노드 그룹	EKS Auto Mode	Amazon EKS Hybrid Nodes
포트에 Amazon EBS 스토리지를 사용할 수 있습니다.	예	예-통합 기능으로 사용할 수 있습니다. 스토리지 클래스 생성 방법을 알아봅니다.	아니요
포드에 Amazon EFS 스토리지를 사용할 수 있습니다.	예	예	아니요
포트에 Amazon FSx for Lustre 스토리지를 사용할 수 있습니다.	예	예	아니요
서비스에 Network Load Balancer를 사용할 수 있습니다.	예	예	예-대상 유형 ip를 사용해야 합니다.
포드가 퍼블릭 서브넷에서 실행될 수 있습니다.	예	예	아니요-포드는 온프레미스 환경에서 실행됩니다.
개별 포드에 서로 다른 VPC 보안 그룹을 할당할 수 있습니다.	예 — Linux 노드만	아니요	아니요
Kubernetes DaemonSet를 실행할 수 있습니다.	예	예	예
포드 매니페스트에서 HostPort 및 HostNetwork 를 지원합니다.	예	예	예

기준	EKS 관리형 노드 그룹	EKS Auto Mode	Amazon EKS Hybrid Nodes
AWS 리전 가용성	모든 Amazon EKS 지원 리전	모든 Amazon EKS 지원 리전	AWS GovCloud(미국) 리전 및 중국 리전을 제외한 모든 Amazon EKS 지원 리전
Amazon EC2 전용 호스트에서 컨테이너를 실행할 수 있습니다.	예	아니요	아니요
요금	여러 포드를 실행하는 Amazon EC2 인스턴스의 비용입니다. 자세한 설명은 Amazon EC2 요금 을 참조하세요.	클러스터에서 EKS Auto Mode가 활성화된 경우 Auto Mode의 컴퓨팅 기능을 사용하여 시작된 인스턴스에 대해 표준 EC2 인스턴스 요금 외에 별도의 요금을 지불합니다. 금액은 시작된 인스턴스 유형과 클러스터가 위치한 AWS 리전에 따라 다릅니다. 자세한 내용은 Amazon EKS 요금 을 참조하세요.	시간당 하이브리드 노드 vCPU 비용. 자세한 내용은 Amazon EKS 요금 을 참조하세요.

관리형 노드 그룹을 사용한 노드 수명 주기 간소화

Amazon EKS 관리형 노드 그룹은 Amazon EKS Kubernetes 클러스터의 노드(Amazon EC2 인스턴스) 프로비저닝 및 수명 주기 관리를 자동화합니다.

Amazon EKS 관리형 노드 그룹을 사용하면 Kubernetes 애플리케이션을 실행하기 위해 컴퓨팅 용량을 제공하는 Amazon EC2 인스턴스를 별도로 프로비저닝하거나 등록할 필요가 없습니다. 한 번의 조작으로 클러스터에 대한 노드를 자동으로 생성, 업데이트 또는 종료할 수 있습니다. 노드 업데이트 및 종료는 자동으로 노드를 드레이닝하여 애플리케이션을 계속 사용할 수 있도록 합니다.

모든 관리형 노드는 Amazon EKS에서 관리하는 Amazon EC2 Auto Scaling 그룹의 일부로 프로비저닝됩니다. 인스턴스 및 Auto Scaling 그룹을 포함한 모든 리소스는 AWS 계정 내에서 실행됩니다. 각 노드 그룹은 정의한 여러 가용 영역에서 실행됩니다.

관리형 노드 그룹은 노드 자동 복구를 선택적으로 활용하여 노드 상태를 지속적으로 모니터링할 수도 있습니다. 감지된 문제에 자동으로 대응하고 가능한 경우 노드를 교체합니다. 이렇게 하면 수동 개입을 최소화하면서 클러스터의 전반적인 가용성을 높일 수 있습니다. 자세한 내용은 [the section called “노드 상태”](#) 단원을 참조하십시오.

Amazon EKS 콘솔, eksctl, AWS CLI, AWS API 또는 AWS CloudFormation을 포함한 코드형 인프라를 사용하여 관리형 노드 그룹을 신규 또는 기존 클러스터에 추가할 수 있습니다. 관리형 노드 그룹의 일부로 시작된 노드는 Kubernetes [Cluster Autoscaler](#)에서 자동 검색할 수 있도록 자동으로 태그가 지정됩니다. 노드 그룹을 사용하여 Kubernetes 레이블을 노드에 적용하고 언제든지 업데이트할 수 있습니다.

Amazon EKS 관리형 노드 그룹을 사용하기 위한 추가 비용은 없으며 프로비저닝한 AWS 리소스에 대해서만 비용을 지불합니다. 여기에는 Amazon EC2 인스턴스, Amazon EBS 볼륨, Amazon EKS 클러스터 시간 및 기타 AWS 인프라가 포함됩니다. 최소 요금 및 선수금은 없습니다.

새 Amazon EKS 클러스터 및 관리형 노드 그룹을 시작하려면 [the section called “클러스터 생성\(콘솔 및 CLI\)”](#) 부분을 참조하세요.

기존 클러스터에 관리형 노드 그룹을 추가하려면 [the section called “생성”](#)을 참조하세요.

관리형 노드 그룹 개념

- Amazon EKS 관리형 노드 그룹은 사용자를 위해 Amazon EC2 인스턴스를 생성하고 관리합니다.
- 모든 관리형 노드는 Amazon EKS에서 관리하는 Amazon EC2 Auto Scaling 그룹의 일부로 프로비저닝됩니다. 게다가 Amazon EC2 인스턴스 및 Auto Scaling 그룹을 포함한 모든 리소스는 AWS 계정 내에서 실행됩니다.
- 관리형 노드 그룹의 Auto Scaling 그룹은 그룹을 생성할 때 지정하는 모든 서브넷에 걸쳐 있습니다.
- Amazon EKS는 관리형 노드 그룹 리소스에 태그를 지정하여 Kubernetes [Cluster Autoscaler](#)를 사용하도록 구성합니다.

Important

Amazon EBS 볼륨에 의해 백업되고 Kubernetes [Cluster Autoscaler](#)를 사용하는 상태 기반 애플리케이션을 여러 가용 영역에서 실행하는 경우 각 단일 가용 영역으로 범위가 지정된 여

러 노드 그룹을 구성해야 합니다. 또한 `--balance-similar-node-groups` 기능을 활성화해야 합니다.

- 관리형 노드를 배포할 때 더 높은 수준의 유연성과 사용자 지정을 위해 사용자 정의 시작 템플릿을 사용할 수 있습니다. 예를 들어, 추가 kubelet 인수를 지정하고 사용자 지정 AMI를 사용할 수 있습니다. 자세한 내용은 [the section called “시작 템플릿”](#) 섹션을 참조하세요. 관리형 노드 그룹을 처음 생성할 때 사용자 정의 시작 템플릿을 사용하지 않으면 자동 생성된 시작 템플릿이 표시됩니다. 이 자동 생성된 템플릿을 수동으로 수정하지 마세요. 수정하면 오류가 발생합니다.
- Amazon EKS는 관리형 노드 그룹의 CVE 및 보안 패치에 대한 공동 책임 모델을 따릅니다. 관리형 노드가 Amazon EKS 최적화 AMI를 실행하는 경우 버그나 문제가 보고될 때 패치 버전의 AMI를 빌드해야 합니다. 수정 사항을 게시할 수 있습니다. 그러나 이렇게 패치된 AMI 버전은 사용자가 관리형 노드 그룹에 배포해야 합니다. 관리형 노드가 사용자 지정 AMI를 실행하는 경우 버그나 문제가 보고되고 AMI를 배포할 때 패치 버전의 AMI를 빌드해야 합니다. 자세한 내용은 [the section called “업데이트”](#) 섹션을 참조하세요.
- Amazon EKS 관리형 노드 그룹은 퍼블릭 서브넷과 프라이빗 서브넷 모두에서 시작할 수 있습니다. 2020년 4월 22일 또는 이후에 퍼블릭 서브넷에서 관리형 노드 그룹을 시작하는 경우 인스턴스가 클러스터에 성공적으로 조인하려면 서브넷에서 `MapPublicIpOnLaunch`를 `true`로 설정해야 합니다. 2020년 3월 26일 이후에 `eksctl` 또는 [Amazon EKS 벤딩 AWS CloudFormation 템플릿](#)을 사용하여 퍼블릭 서브넷을 생성한 경우 이 설정이 이미 `true`로 설정되어 있습니다. 2020년 3월 26일 이전에 퍼블릭 서브넷을 생성한 경우 해당 설정을 수동으로 변경해야 합니다. 자세한 내용은 [서브넷의 퍼블릭 IPv4 주소 지정 속성 수정](#) 섹션을 참조하세요.
- 프라이빗 서브넷에 관리형 노드 그룹을 배포할 때 컨테이너 이미지를 가져오기 위해 Amazon ECR에 액세스할 수 있는지 확인해야 합니다. NAT 게이트웨이를 서브넷의 라우팅 테이블에 연결하거나 다음 [AWS 프라이빗 링크 VPC 엔드포인트](#)를 추가하여 이 작업을 수행할 수 있습니다.
 - Amazon ECR API 엔드포인트 인터페이스 - `com.amazonaws.region-code.ecr.api`
 - Amazon ECR Docker 레지스트리 API 엔드포인트 인터페이스 - `com.amazonaws.region-code.ecr.dkr`
 - Amazon S3 게이트웨이 엔드포인트 - `com.amazonaws.region-code.s3`

일반적으로 사용되는 다른 서비스와 엔드포인트는 [the section called “프라이빗 클러스터”](#) 섹션을 참조하세요.

- 관리형 노드 그룹은 [AWS Outposts](#)나 [AWS Wavelength](#)에 배치할 수 없습니다. 관리형 노드 그룹은 [AWS 로컬 영역](#)에서 생성할 수 있습니다. 자세한 내용은 [the section called “AWS 로컬 영역”](#) 단원을 참조하십시오.

- 단일 클러스터 내에서 여러 관리형 노드 그룹을 생성할 수 있습니다. 예를 들어, 일부 워크로드에는 표준 Amazon EKS 최적화 Amazon Linux AMI를 사용하는 노드 그룹을 생성하고 GPU 지원이 필요한 워크로드에는 GPU 변형을 사용하는 다른 노드 그룹을 생성할 수 있습니다.
- 관리형 노드 그룹에 [Amazon EC2 인스턴스 상태 확인](#) 오류가 발생하면 Amazon EKS는 문제 진단에 도움이 되는 오류 메시지를 반환합니다. 자세한 내용은 [the section called “관리형 노드 그룹 오류”](#) 단원을 참조하십시오.
- Amazon EKS는 관리형 노드 그룹 인스턴스에 Kubernetes 레이블을 추가합니다. 이러한 Amazon EKS 제공 레이블에는 `eks.amazonaws.com` 접두사가 붙습니다.
- Amazon EKS는 종료 또는 업데이트 중에 Kubernetes API를 사용하여 노드를 자동으로 비웁니다.
- AZRebalance로 노드를 종료하거나 원하는 노드 수를 줄이는 경우 포드 중단 예산은 반영되지 않습니다. 이러한 작업은 노드에서 포드 제거를 시도합니다. 그러나 15분 넘게 걸리면 노드의 모든 포드가 종료되었는지 여부에 관계없이 노드가 종료됩니다. 노드가 종료될 때까지 시간을 연장하려면 Auto Scaling 그룹에 수명 주기 후크를 추가하세요. 자세한 내용을 알아보려면 Amazon EC2 Auto Scaling 사용 설명서의 [수명 주기 후크 추가](#)를 참조하세요.
- 스팟 중단 알림 또는 용량 재조정 알림을 수신한 후 드레인 프로세스를 올바르게 실행하려면 CapacityRebalance를 true로 설정해야 합니다.
- 관리형 노드 그룹을 업데이트하면 포드에 대해 설정한 포드 중단 예산을 준수합니다. 자세한 내용은 [the section called “동작 세부 정보 업데이트”](#) 단원을 참조하십시오.
- Amazon EKS 관리형 노드 그룹을 사용하는 데 따른 추가 비용은 없습니다. 프로비저닝하는 AWS 리소스에 대해서만 비용을 지불합니다.
- 노드에 대해 Amazon EBS 볼륨을 암호화하려는 경우 시작 템플릿을 사용하여 노드를 배포할 수 있습니다. 시작 템플릿을 사용하지 않고 암호화된 Amazon EBS 볼륨이 있는 관리형 노드를 배포하려면 계정에 생성된 모든 새 Amazon EBS 볼륨을 암호화합니다. 자세한 내용은 Amazon EC2 사용 설명서에서 [기본적으로 암호화](#)를 참조하세요.

관리형 노드 그룹 용량

관리형 노드 그룹을 생성할 때 온디맨드 또는 스팟 용량 유형을 선택할 수 있습니다. Amazon EKS는 온디맨드 또는 Amazon EC2 스팟 인스턴스만 포함하는 Amazon EC2 Auto Scaling 그룹과 함께 관리형 노드 그룹을 배포합니다. 내결함성 애플리케이션이 스팟 관리형 노드 그룹에 대해 포드를 예약하고, 단일 Kubernetes 클러스터 내의 온디맨드 노드 그룹에 내결함성 애플리케이션을 예약할 수 있습니다. 기본적으로 관리형 노드 그룹은 온디맨드 Amazon EC2 인스턴스를 배포합니다.

온디맨드

온디맨드 인스턴스를 사용하면 장기 약정 없이 초 단위로 컴퓨팅 용량을 구입할 수 있습니다.

기본적으로, 용량 유형을 지정하지 않은 경우 관리형 노드 그룹이 온디맨드 인스턴스를 사용하여 프로비저닝됩니다. 관리형 노드 그룹은 다음 설정을 적용하여 사용자를 대신하여 Amazon EC2 Auto Scaling 그룹을 구성합니다.

- 온디맨드 용량을 프로비저닝하기 위한 할당 전략이 `prioritized`로 설정됩니다. 관리형 노드 그룹은 API에 전달된 인스턴스 유형의 순서를 사용하여 온디맨드 용량을 채울 때 먼저 사용할 인스턴스 유형을 결정합니다. 예를 들어, 세 가지 인스턴스 유형을 `c5.large`, `c4.large`, `c3.large`의 순서로 지정할 수 있습니다. 온디맨드 인스턴스가 시작되면 관리형 노드 그룹은 `c5.large`, `c4.large`, `c3.large` 순으로 시작하여 온디맨드 용량을 채웁니다. 자세한 내용은 Amazon EC2 Auto Scaling 사용 설명서의 [Amazon EC2 Auto Scaling 그룹](#)을 참조하세요.
- Amazon EKS는 용량 유형을 지정하는 관리형 노드 그룹의 모든 노드에 다음 Kubernetes 레이블을 추가합니다. `eks.amazonaws.com/capacityType: ON_DEMAND`. 이 레이블을 사용하여 온디맨드 노드에서 상태 저장 또는 내결함성이 없는 애플리케이션을 예약할 수 있습니다.

스팟

Amazon EC2 스팟 인스턴스는 온디맨드 가격에서 큰 폭의 할인을 제공하는 여분의 Amazon EC2 용량입니다. EC2에서 용량을 복구하려면 2분 중단 알림으로 Amazon EC2 스팟 인스턴스를 중단시킬 수 있습니다. 자세한 내용은 Amazon EC2 사용 설명서의 [스팟 인스턴스](#)를 참조하세요. Amazon EC2 스팟 인스턴스로 관리형 노드 그룹을 구성하여 Amazon EKS 클러스터에서 실행되는 컴퓨팅 노드의 비용을 최적화할 수 있습니다.

관리되는 노드 그룹 내에서 스팟 인스턴스를 사용하려면 용량 유형을 `spot`으로 설정하여 관리형 노드 그룹을 생성합니다. 관리형 노드 그룹은 다음 스팟 모범 사례를 적용하여 사용자를 대신하여 Amazon EC2 Auto Scaling 그룹을 구성합니다.

- 스팟 노드가 최적의 스팟 용량 풀에서 프로비저닝되도록 다음 중 하나로 할당 전략을 설정합니다.
 - `price-capacity-optimized(PCO)` - Kubernetes 버전 1.28 이상의 클러스터에서 새 노드 그룹을 생성할 때 할당 전략은 `price-capacity-optimized`로 설정됩니다. 하지만 Amazon EKS 관리형 노드 그룹이 PCO 지원을 시작하기 전에 `capacity-optimized`로 이미 생성된 노드 그룹의 할당 전략은 변경되지 않습니다.
 - `capacity-optimized(CO)` - Kubernetes 버전 1.27 이하의 클러스터에서 새 노드 그룹을 생성할 때 할당 전략은 `capacity-optimized`로 설정됩니다.

용량 할당에 사용할 수 있는 스팟 용량 풀 수를 늘리려면 여러 인스턴스 유형을 사용하도록 관리형 노드 그룹을 구성합니다.

- Amazon EC2 스팟 용량 리밸런싱이 활성화되어 있으므로 Amazon EKS가 스팟 노드를 정상적으로 비우고 리밸런싱하여 스팟 노드가 중단 위험이 높을 때 애플리케이션 중단을 최소화할 수 있습니다. 자세한 내용은 Amazon EC2 Auto Scaling 사용 설명서의 [Amazon EC2 Auto Scaling 용량 재분배](#)를 참조하세요.
- 스팟 노드가 리밸런싱 권고를 수신하면 Amazon EKS는 자동으로 새로운 대체 스팟 노드를 시작하려고 시도합니다.
- 교체 스팟 노드가 Ready 상태가 되면 Amazon EKS가 재조정 권장 사항을 받은 스팟 노드를 비우기 시작합니다. Amazon EKS는 최선을 다해 노드를 드레이닝합니다. 따라서 Amazon EKS가 기존 노드를 드레이닝하기 전에 대체 노드가 클러스터에 조인할 때까지 기다린다는 보장이 없습니다.
- 대체 스팟 노드가 부트스트랩되고 Ready 상태에서 Amazon EKS 코드를 실행하고 리밸런싱 권장 사항을 받은 스팟 노드를 비웁니다. 스팟 노드를 코드화하면 서비스 컨트롤러가 이 스팟 노드로 새 요청을 보내지 않습니다. 또한 정상 활성 스팟 노드 목록에서 제거합니다. 스팟 노드를 비우면 실행 중인 포드가 정상적으로 제거됩니다.
- Amazon EKS는 용량 유형을 지정하는 관리형 노드 그룹의 모든 노드에 다음 Kubernetes 레이블을 추가합니다. `eks.amazonaws.com/capacityType: SPOT`. 이 레이블을 사용하여 스팟 노드에서 내결함성 애플리케이션을 예약할 수 있습니다.

온디맨드 또는 스팟 용량을 사용하여 노드 그룹을 배포할지 여부를 결정할 때 다음 조건을 고려해야 합니다.

- 스팟 인스턴스는 유연한 무상태 내결함성 애플리케이션에 적합합니다. 여기에는 배치 및 기계 학습 교육 워크로드, Apache Spark와 같은 빅 데이터 ETL, 대기열 처리 애플리케이션 및 무상태 API 엔드포인트가 포함됩니다. 스팟은 시간이 지남에 따라 변경될 수 있는 예비 Amazon EC2 용량이므로 중단 방지 워크로드에 스팟 용량을 사용하는 것이 좋습니다. 보다 구체적으로 말하면 스팟 용량은 필요한 용량을 사용할 수 없는 기간을 허용할 수 있는 워크로드에 적합합니다.
- 내결함성이 없는 애플리케이션의 경우 온디맨드를 사용하는 것이 좋습니다. 모니터링 및 운영 도구와 같은 클러스터 관리 도구, StatefulSets가 필요한 배포, 상태 유지 애플리케이션(예: 데이터베이스)이 여기에 포함됩니다.
- 스팟 인스턴스를 사용하는 동안 애플리케이션의 가용성을 극대화하려면 여러 인스턴스 유형을 사용하도록 스팟 관리 노드 그룹을 구성하는 것이 좋습니다. 여러 인스턴스 유형을 사용할 때는 다음 규칙을 적용하는 것이 좋습니다.

- 관리형 노드 그룹 내에서 [Cluster Autoscaler](#)를 사용하는 경우 vCPU 및 메모리 리소스의 양이 같은 유연한 인스턴스 유형 집합을 사용하는 것이 좋습니다. 이는 클러스터의 노드가 예상대로 확장되도록 하기 위한 것입니다. 예를 들어, 4개의 vCPU와 8개의 GiB 메모리가 필요한 경우 c3.xlarge, c4.xlarge, c5.xlarge, c5d.xlarge, c5a.xlarge, c5n.xlarge 또는 기타 유사한 인스턴스 유형을 사용합니다.
- 애플리케이션 가용성을 높이려면 여러 스팟 관리형 노드 그룹을 배포하는 것이 좋습니다. 이를 위해 각 그룹은 동일한 vCPU 및 메모리 리소스를 가진 유연한 인스턴스 유형 집합을 사용해야 합니다. 예를 들어, 4개의 vCPU와 8개의 GiB 메모리가 필요한 경우 c3.xlarge, c4.xlarge, c5.xlarge, c5d.xlarge, c5a.xlarge, c5n.xlarge 또는 기타 유사한 인스턴스 유형을 사용하여 관리형 노드 그룹을 하나 생성하고 m3.xlarge, m4.xlarge, m5.xlarge, m5d.xlarge, m5a.xlarge, m5n.xlarge 또는 기타 유사한 인스턴스 유형을 사용하여 두 번째 관리형 노드 그룹을 생성하는 것이 좋습니다.
- 사용자 정의 시작 템플릿을 사용하는 스팟 용량 유형으로 노드 그룹을 배포하는 경우 API를 사용하여 여러 인스턴스 유형을 전달합니다. 시작 템플릿을 통해 단일 인스턴스 유형을 전달하지 마세요. 시작 템플릿을 사용하여 노드 그룹을 배포하는 방법에 대한 자세한 내용은 [the section called “시작 템플릿”](#) 부분을 참조하세요.

클러스터에 대한 관리형 노드 그룹 생성

이 주제는 Amazon EKS 클러스터에 등록하는 노드의 Amazon EKS 관리형 노드 그룹을 시작하는 데 도움이 됩니다. 노드가 클러스터에 조인한 이후 Kubernetes 애플리케이션을 배포할 수 있습니다.

Amazon EKS 관리형 노드 그룹을 처음 시작하는 경우, [시작](#)의 가이드 중 하나를 따르는 것이 좋습니다. 이 가이드에서는 노드가 있는 Amazon EKS 클러스터를 생성하기 위한 시연을 제공합니다.

Important

- Amazon EKS 노드는 표준 Amazon EC2 인스턴스입니다. 일반 Amazon EC2 가격을 기준으로 요금이 청구됩니다. 자세한 내용은 [Amazon EC2 요금](#)을 참조하세요.
- AWS Outposts 또는 AWS Wavelength가 활성화된 AWS 리전에서는 관리형 노드를 만들 수 없습니다. 대신 자체 관리형 노드만 생성할 수 있습니다. 자세한 내용은 [the section called “Amazon Linux”](#), [the section called “Windows”](#), [the section called “Bottlerocket”](#) 섹션을 참조하세요. Outpost에서 자체 관리형 Amazon Linux 노드 그룹을 생성할 수도 있습니다. 자세한 내용은 [the section called “노드”](#) 섹션을 참조하세요.
- Amazon EKS 최적화 Linux 또는 Bottlerocket에 포함된 bootstrap.sh 파일에 [AMI ID를 지정](#)하지 않는 경우 관리형 노드 그룹은 최대 값 수 maxPods를 적용합니다. vCPU가 30개 이

하인 인스턴스의 경우 최대 수는 110 입니다. vCPU가 30개 이상인 인스턴스의 경우 최대 개수가 250 로 바뀝니다. 이 수치는 [Kubernetes 확장성 임계값](#)과 내부 Amazon EKS 확장성 팀 테스트에 따른 권장 설정을 기반으로 합니다. 자세한 내용은 [Amazon VPC CNI 플러그인을 통한 노드당 포트 제한 증가](#) 블로그 게시물을 참조하세요.

- 기존 Amazon EKS 클러스터. 배포하려면 [the section called “클러스터 생성”](#) 섹션을 참조하세요.
- 노드가 사용할 기존 IAM 역할. 파일을 만들려면 [the section called “노드 IAM 역할”](#) 섹션을 참조하세요. 이 역할에 VPC CNI에 대한 정책 중 하나도 없는 경우 VPC CNI 포트에 대해 다음과 같은 별도의 역할이 필요합니다.
- (선택 사항이지만 권장됨) 필요한 IAM 정책이 연결된 자체 IAM 역할로 구성된 Kubernetes용 Amazon VPC CNI 플러그인 추가 기능. 자세한 내용은 [the section called “IRSA에 대한 구성”](#) 섹션을 참조하세요.
- [최적의 Amazon EC2 노드 인스턴스 유형 선택](#)에 나열된 고려 사항에 익숙합니다. 선택하는 인스턴스 유형에 따라 클러스터 및 VPC에 대한 추가 전제 조건이 있을 수 있습니다.
- Windows 관리형 노드 그룹을 추가하려면 먼저 클러스터에 대한 Windows 지원을 활성화해야 합니다. 자세한 내용은 [the section called “Windows 지원 사용”](#) 섹션을 참조하세요.

다음 중 하나를 사용하여 관리형 노드 그룹을 생성할 수 있습니다.

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)

eksctl

eksctl로 관리형 노드 그룹 만들기

이 절차에는 eksctl 버전 0.210.0 이상이 필요합니다. 버전은 다음 명령을 통해 확인할 수 있습니다.

```
eksctl version
```

eksctl 설치 또는 업데이트에 대한 지침은 eksctl 설명서의 [Installation](#)를 참조하세요.

1. (선택 사항) AmazonEKS_CNI_Policy 관리형 IAM 정책이 [Amazon EKS 노드 IAM 역할](#)에 연결되어 있는 경우 대신 Kubernetes aws-node 서비스 계정에 연결한 IAM 역할에 할당하는 것이 좋습니다. 자세한 내용은 [the section called “IRSA에 대한 구성”](#) 섹션을 참조하세요.

2. 사용자 정의 시작 템플릿을 사용하거나 사용하지 않고 관리형 노드 그룹을 생성합니다. 시작 템플릿을 수동으로 지정하면 노드 그룹을 더욱 수월하게 사용자 지정할 수 있습니다. 예를 들어, Amazon EKS 최적화 AMI의 `bootstrap.sh` 스크립트에 사용자 지정 AMI를 배포하거나 인수를 제공할 수 있습니다. 사용 가능한 모든 옵션 및 기본값의 전체 목록을 보려면 다음 명령을 입력합니다.

```
eksctl create nodegroup --help
```

다음 명령에서 *my-cluster*를 클러스터 이름으로 바꾸고 *my-mng*를 노드 그룹 이름으로 바꿉니다. 노드 그룹 이름은 63자를 초과할 수 없습니다. 문자나 숫자로 시작하되, 나머지 문자의 경우 하이픈과 밑줄을 포함할 수 있습니다.

Important

관리형 노드 그룹을 처음 생성할 때 사용자 정의 시작 템플릿을 사용하지 않는 경우 나중에 노드 그룹에 대해 시작 템플릿을 사용하지 마세요. 사용자 정의 시작 템플릿을 지정하지 않은 경우 시스템에서 시작 템플릿을 자동으로 생성하므로 수동으로 수정하지 않는 것이 좋습니다. 이 자동 생성된 시작 템플릿을 수동으로 수정하면 오류가 발생할 수 있습니다.

시작 템플릿 제외

`eksctl`은 계정에 기본 Amazon EC2 시작 템플릿을 생성하고 사용자가 지정한 옵션에 따라 생성되는 시작 템플릿을 사용하여 노드 그룹을 배포합니다. `--node-type`의 값을 지정하려면 [the section called “Amazon EC2 인스턴스 유형”](#) 부분을 참조하세요.

허용된 키워드로 *ami-family*를 바꿉니다. 자세한 내용은 `eksctl` 설명서의 [노드 AMI 패밀리 설정](#)을 참조하세요. *my-key*를 Amazon EC2 키 페어 또는 퍼블릭 키 이름으로 바꿉니다. 이 키는 노드를 시작한 후 SSH로 연결하는 데 사용됩니다.

Note

Windows의 경우 이 명령에서 SSH를 활성화하지 않습니다. 대신 Amazon EC2 키 페어를 인스턴스와 연결하고 인스턴스에 RDP할 수 있습니다.

Amazon EC2 키 페어가 아직 없는 경우 AWS Management Console에서 새로 생성할 수 있습니다. Linux에 대한 자세한 내용은 Amazon EC2 사용 설명서의 [Amazon EC2 키 페어 및 Linux 인스턴스](#)를

참조하세요. Windows에 대한 자세한 내용은 Amazon EC2 사용 설명서의 [Amazon EC2 키 페어 및 Windows 인스턴스](#)를 참조하세요.

다음과 같은 조건에 해당하면 IMDS에 대한 포트 액세스를 차단하는 것이 좋습니다.

- 포트에 필요한 최소 권한만 있도록 모든 Kubernetes 서비스 계정에 IAM 역할을 할당할 계획입니다.
- 클러스터의 어떤 포트도 현재 AWS 리전 검색 등의 다른 이유로 Amazon EC2 인스턴스 메타데이터 서비스(IMDS)에 액세스할 필요가 없습니다.

자세한 내용은 [작업자 노드에 할당된 인스턴스 프로파일에 대한 액세스 제한](#) 섹션을 참조하세요.

포트가 IMDS에 액세스하지 못하게 차단하고 싶다면 `--disable-pod-imds` 옵션을 다음 명령에 추가합니다.

```
eksctl create nodegroup \
  --cluster my-cluster \
  --region region-code \
  --name my-mng \
  --node-ami-family ami-family \
  --node-type m5.large \
  --nodes 3 \
  --nodes-min 2 \
  --nodes-max 4 \
  --ssh-access \
  --ssh-public-key my-key
```

인스턴스는 필요에 따라 포트에 훨씬 더 많은 수의 IP 주소를 할당하고, 인스턴스와 다른 CIDR 블록의 포트에 IP 주소를 할당하며, 인터넷 액세스 없이 클러스터에 배포할 수 있습니다. 자세한 내용은 [the section called “IP 주소 증가”](#), [the section called “사용자 지정 네트워킹”](#) 및 [the section called “프라이빗 클러스터”](#)에서 이전 명령에 추가할 추가 옵션을 참조하세요.

관리형 노드 그룹은 인스턴스 유형에 따라 노드 그룹의 각 노드에서 실행될 수 있는 최대 포트 수에 대해 단일 값을 계산하고 적용합니다. 다른 인스턴스 유형으로 노드 그룹을 생성하는 경우 모든 인스턴스 유형에 대해 계산된 가장 작은 값이 노드 그룹의 모든 인스턴스 유형에서 실행될 수 있는 최대 포트 수로 적용됩니다. 관리형 노드 그룹은 [각 Amazon EC2 인스턴스 유형에 대해 Amazon EKS 권장 최대 포트](#)에 참조된 스크립트를 사용하여 값을 계산합니다.

시작 템플릿 사용

시작 템플릿은 이미 존재해야 하며 [시작 템플릿 구성 기본 사항](#)에 지정된 요구 사항을 충족해야 합니다. 다음과 같은 조건에 해당하면 IMDS에 대한 포트 액세스를 차단하는 것이 좋습니다.

- 포드에 필요한 최소 권한만 있도록 모든 Kubernetes 서비스 계정에 IAM 역할을 할당할 계획입니다.
- 클러스터의 어떤 포드도 현재 AWS 리전 검색 등의 다른 이유로 Amazon EC2 인스턴스 메타데이터 서비스(IMDS)에 액세스할 필요가 없습니다.

자세한 내용은 [작업자 노드에 할당된 인스턴스 프로파일에 대한 액세스 제한](#) 섹션을 참조하세요.

IMDS에 대한 포드 액세스를 차단하려면 시작 템플릿에서 필요한 설정을 지정합니다.

- 다음 콘텐츠를 디바이스에 복사합니다. **## #**을 바꾼 다음 수정된 명령을 실행하여 eks-nodegroup.yaml 파일을 생성합니다. 시작 템플릿 없이 배포할 때 지정하는 몇 가지 설정이 시작 템플릿으로 이동됩니다. version을 지정하지 않으면 템플릿의 기본 버전이 사용됩니다.

```
cat >eks-nodegroup.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-cluster
  region: region-code
managedNodeGroups:
- name: my-mng
  launchTemplate:
    id: lt-id
    version: "1"
EOF
```

eksctl config 파일 설정의 전체 목록은 eksctl 설명서에서 [Config 파일 스키마](#)를 참조하세요. 인스턴스는 필요에 따라 포드에 훨씬 더 많은 수의 IP 주소를 할당하고, 인스턴스와 다른 CIDR 블록의 포드에 IP 주소를 할당하며, 아웃바운드 인터넷 액세스 없이 클러스터에 배포할 수 있습니다. 자세한 내용은 [the section called “IP 주소 증가”](#), [the section called “사용자 지정 네트워킹”](#) 및 [the section called “프라이빗 클러스터”](#)에서 config 파일에 추가할 추가 옵션을 참조하세요.

시작 템플릿에서 AMI ID를 지정하지 않은 경우 관리형 노드 그룹은 인스턴스 유형에 따라 노드 그룹의 각 노드에서 실행될 수 있는 최대 포드 수에 대해 단일 값을 계산하고 적용합니다. 다른 인스턴스 유형으로 노드 그룹을 생성하는 경우 모든 인스턴스 유형에 대해 계산된 가장 작은 값이 노드 그룹의 모든 인스턴스 유형에서 실행될 수 있는 최대 포드 수로 적용됩니다. 관리형 노드 그룹은 [각 Amazon EC2 인스턴스 유형에 대해 Amazon EKS 권장 최대 포드](#)에 참조된 스크립트를 사용하여 값을 계산합니다.

시작 템플릿에서 AMI ID를 지정한 경우 [사용자 지정 네트워킹](#)을 사용하거나 [인스턴스에 할당된 IP 주소의 수를 늘리고 싶으면](#) 노드 그룹의 각 노드에서 실행될 수 있는 최대 포드 수를 지정합니다. 자세한 내용은 [the section called “각 Amazon EC2 인스턴스 유형의 Amazon EKS 권장 최대 포드 수”](#) 섹션을 참조하세요.

b. 다음 명령을 사용하여 노드 그룹을 배포합니다.

```
eksctl create nodegroup --config-file eks-nodegroup.yaml
```

AWS Management Console

AWS Management Console을 사용하여 관리형 노드 그룹을 생성하려면

1. 클러스터 상태가 ACTIVE가 되기를 기다립니다. 이미 ACTIVE 상태가 아닌 클러스터에 대한 관리형 노드 그룹을 생성할 수 없습니다.
2. [Amazon EKS 콘솔](#)을 엽니다.
3. 관리형 노드 그룹을 만들려는 클러스터의 이름을 선택합니다.
4. 컴퓨팅을 선택합니다.
5. 노드 그룹 추가를 선택합니다.
6. 노드 그룹 구성 페이지에서 적절히 파라미터를 입력하고 다음을 선택합니다.
 - 이름 - 관리형 노드 그룹의 고유한 이름을 입력합니다. 노드 그룹 이름은 63자를 초과할 수 없습니다. 문자나 숫자로 시작하되, 나머지 문자의 경우 하이픈과 밑줄을 포함할 수 있습니다.
 - 노드 IAM 역할 - 노드 그룹에 사용할 노드 인스턴스 역할을 선택합니다. 자세한 내용은 [the section called “노드 IAM 역할”](#) 섹션을 참조하세요.

Important

- 클러스터를 생성하는 데 사용된 것과 동일한 역할을 사용할 수 없습니다.
- 자체 관리형 노드 그룹에서 현재 사용하지 않는 역할을 사용하는 것이 좋습니다. 그렇지 않으면 새로운 자체 관리형 노드 그룹과 함께 사용할 계획입니다. 자세한 내용은 [the section called “삭제”](#) 섹션을 참조하세요.

- 시작 템플릿 사용 - (선택사항) 기존 시작 템플릿을 사용할지 여부를 선택합니다. 시작 템플릿 이름을 선택합니다. 그런 다음 시작 템플릿 버전을 선택합니다. 버전을 선택하지 않으면 Amazon

EKS가 템플릿의 기본 버전을 사용합니다. 시작 템플릿을 사용하면 사용자 지정 AMI 배포를 허용하고, 포드에 훨씬 더 많은 수의 IP 주소를 할당하고, 인스턴스와 다른 CIDR 블록의 포드에 IP 주소를 할당하고, 아웃바운드 인터넷 액세스 없이 클러스터에 노드를 배포하는 등 더욱 다양한 노드 그룹 사용자 지정이 허용됩니다. 자세한 내용은 [the section called “IP 주소 증가”](#), [the section called “사용자 지정 네트워킹”](#), [the section called “프라이빗 클러스터”](#) 섹션을 참조하세요.

시작 템플릿은 [시작 템플릿을 사용하여 관리형 노드 사용자 지정](#)의 요구 사항을 충족해야 합니다. 자체 시작 템플릿을 사용하지 않는 경우 Amazon EKS API는 계정에 기본 Amazon EC2 시작 템플릿을 생성하고 기본 시작 템플릿을 사용하여 노드 그룹을 배포합니다.

[서비스 계정에 대한 IAM 역할](#)을 구현하고 AWS 서비스에 대한 액세스 권한이 필요한 모든 포드에 직접 필요한 권한을 할당하며 클러스터의 포드가 현재 AWS 리전 검색 등 다른 이유로 IMDS에 액세스할 필요가 없는 경우 시작 템플릿에서 호스트 네트워킹을 사용하지 않는 포드에 대해 IMDS에 대한 액세스를 사용 중지할 수도 있습니다. 자세한 내용은 [작업자 노드에 할당된 인스턴스 프로파일에 대한 액세스 제한](#) 섹션을 참조하세요.

- Kubernetes 레이블 - (선택 사항) 관리형 노드 그룹의 노드에 Kubernetes 레이블을 적용하도록 선택할 수 있습니다.
- Kubernetes 테인트 - (선택 사항) 관리형 노드 그룹의 노드에 Kubernetes 테인트를 적용하도록 선택할 수 있습니다. 효과 메뉴에서 사용할 수 있는 옵션은 **NoSchedule**, **NoExecute** 및 **PreferNoSchedule** 입니다. 자세한 내용은 [the section called “테인트 GPU 노드”](#) 섹션을 참조하세요.
- 태그 - (선택사항) Amazon EKS 관리형 노드 그룹에 태그를 지정하도록 선택할 수 있습니다. 이러한 태그는 노드 그룹의 다른 리소스(예: Auto Scaling 그룹이나 인스턴스)에는 전파되지 않습니다. 자세한 내용은 [the section called “리소스에 태깅”](#) 섹션을 참조하세요.

7. 컴퓨팅 및 크기 조정 구성 설정 페이지에서 파라미터를 입력하고 다음을 선택합니다.

- AMI 유형 - AMI 유형을 선택합니다. Arm 인스턴스를 배포하는 경우 배포하기 전에 [Amazon EKS 최적화 Arm Amazon Linux AMI](#)의 고려 사항을 검토해야 합니다.

이전 페이지에서 시작 템플릿을 지정하고 시작 템플릿에 AMI를 지정한 경우 값을 선택할 수 없습니다. 템플릿의 값이 표시됩니다. 템플릿에 지정된 AMI에서 [AMI 지정하기](#)의 요구 사항을 충족해야 합니다.

- 용량 유형 - 용량 유형을 선택합니다. 용량 유형을 선택하는 방법에 대한 자세한 내용은 [the section called “관리형 노드 그룹 용량”](#) 부분을 참조하세요. 동일한 노드 그룹 내에 서로 다른 용량 유형을 혼합할 수 없습니다. 두 용량 유형을 모두 사용하려면 각각 고유한 용량 및 인스턴스 유형을 가진 별도의 노드 그룹을 생성합니다. GPU 가속 워커 노드 프로비저닝 및 조정에 대한 자세한 내용은 [the section called “MNG용 GPU 예약”](#) 섹션을 참조하세요.

- 인스턴스 유형 - 기본적으로 하나 이상의 인스턴스 유형이 지정됩니다. 기본 인스턴스 유형을 제거하려면 인스턴스 유형의 오른쪽에 있는 X를 선택합니다. 관리형 노드 그룹에 사용할 인스턴스 유형을 선택합니다. 자세한 내용은 [the section called “Amazon EC2 인스턴스 유형”](#) 섹션을 참조하세요.

콘솔에는 일반적으로 사용되는 인스턴스 유형 세트가 표시됩니다. 표시되지 않은 인스턴스 유형을 사용하여 관리형 노드 그룹을 생성해야 하는 경우 eksctl, AWS CLI, AWS CloudFormation 또는 SDK를 사용하여 노드 그룹을 생성합니다. 이전 페이지에서 시작 템플릿을 지정한 경우 인스턴스 유형이 시작 템플릿에 지정되어야 하므로 값을 선택할 수 없습니다. 시작 템플릿의 값이 표시됩니다. 용량 유형에서 스팟을 선택한 경우, 가용성을 높이기 위해 여러 인스턴스 유형을 지정하는 것이 좋습니다.

- 디스크 크기 - 노드 루트 볼륨에 사용할 디스크 크기(GiB)를 입력합니다.

이전 페이지에서 시작 템플릿을 지정한 경우 시작 템플릿에 값을 지정해야 하므로 값을 선택할 수 없습니다.

- 원하는 크기 - 시작할 때 관리형 노드 그룹에서 유지해야 하는 현재 노드 수를 지정합니다.

Note

Amazon EKS는 노드 그룹을 자동으로 확장 또는 축소하지 않습니다. 그러나 Kubernetes Cluster Autoscaler가 이 작업을 수행하도록 구성할 수 있습니다. 자세한 내용은 [AWS의 Cluster Autoscaler](#)를 참조하세요.

- 최소 크기-관리형 노드 그룹이 확장될 수 있는 최소 노드 수를 지정합니다.
- 최대 크기 - 관리형 노드 그룹이 확장될 수 있는 최대 노드 수를 지정합니다.
- 노드 그룹 업데이트 구성 - (선택사항) 병렬로 업데이트할 노드의 수 또는 백분율을 선택할 수 있습니다. 이러한 노드는 업데이트 중에 사용할 수 없습니다. 최대 사용 불가(Maximum unavailable)에서 다음 옵션 중 하나를 선택하고 값을 지정합니다.
 - 숫자 - 노드 그룹에서 병렬로 업데이트할 수 있는 노드 수를 선택하고 지정합니다.
 - 비율 - 노드 그룹에서 병렬로 업데이트할 수 있는 노드의 비율을 선택하고 지정합니다. 이 기능은 노드 그룹에 많은 수의 노드가 있는 경우에 유용합니다.
- 노드 자동 복구 구성-(선택 사항) 노드 자동 복구 활성화 확인란을 활성화하면 감지된 문제가 발생할 때 Amazon EKS가 노드를 자동으로 교체합니다. 자세한 내용은 [the section called “노드 상태”](#) 섹션을 참조하세요.

8. 네트워킹 지정 페이지에서 파라미터를 입력하고 다음을 선택합니다.

- 서브넷 - 관리형 노드를 시작할 서브넷을 선택합니다.

⚠ Important

Amazon EBS 볼륨에 의해 백업되고 Kubernetes [Cluster Autoscaler](#)를 사용하는 상태 기반 애플리케이션을 여러 가용 영역에서 실행하는 경우 각 단일 가용 영역으로 범위가 지정된 여러 노드 그룹을 구성해야 합니다. 또한 `--balance-similar-node-groups` 기능을 활성화해야 합니다.

⚠ Important

- 퍼블릭 서브넷을 선택한 경우 클러스터에 퍼블릭 API 서버 엔드포인트만 활성화되어 있으면, 인스턴스의 서브넷에 `MapPublicIPOnLaunch`가 `true`로 설정되어 있어야 클러스터에 성공적으로 조인할 수 있습니다. 2020년 3월 26일 이후에 `eksctl` 또는 [Amazon EKS 벤딩 AWS CloudFormation 템플릿](#)을 사용하여 퍼블릭 서브넷을 생성한 경우 이 설정이 이미 `true`로 설정되어 있습니다. 2020년 3월 26일 이전에 `eksctl` 또는 AWS CloudFormation 템플릿으로 서브넷을 생성한 경우 해당 설정을 수동으로 변경해야 합니다. 자세한 내용은 [서브넷의 퍼블릭 IPv4 주소 지정 속성 수정](#) 섹션을 참조하세요.
- 시작 템플릿을 사용하고 여러 네트워크 인터페이스를 지정하는 경우 `MapPublicIpOnLaunch`가 `true`로 설정되어 있더라도 Amazon EC2가 퍼블릭 IPv4 주소를 자동으로 할당하지 않습니다. 이 시나리오에서 노드가 클러스터에 조인하려면 클러스터의 프라이빗 API 서버 엔드포인트를 활성화하거나 NAT 게이트웨이와 같은 대체 방법을 통해 제공되는 아웃바운드 인터넷 액세스를 사용하여 프라이빗 서브넷에서 노드를 시작해야 합니다. 자세한 내용은 Amazon EC2 사용 설명서의 [인스턴스 IP 주소 지정](#)을 참조하세요.

- 노드에 대한 SSH 액세스 구성(선택사항). SSH를 활성화하면 문제가 있는 경우 인스턴스에 연결하여 진단 정보를 수집할 수 있습니다. 노드 그룹을 생성할 때 원격 액세스를 사용 설정하는 것이 좋습니다. 노드 그룹을 생성한 후에는 원격 액세스를 사용 설정할 수 없습니다.

시작 템플릿을 사용하도록 선택한 경우 이 옵션은 표시되지 않습니다. 노드에 대한 원격 액세스를 활성화하려면 시작 템플릿에 키 페어를 지정하고 시작 템플릿에서 지정한 보안 그룹의 노드에 적절한 포트가 열려 있는지 확인합니다. 자세한 내용은 [the section called “사용자 지정 보안 그룹 사용”](#) 섹션을 참조하세요.

Note

Windows의 경우 이 명령에서 SSH를 활성화하지 않습니다. 대신 Amazon EC2 키 페어를 인스턴스와 연결하고 인스턴스에 RDP할 수 있습니다.

- SSH 키 페어에서 사용할 Amazon EC2 SSH 키를 선택합니다. Linux에 대한 자세한 내용은 Amazon EC2 사용 설명서의 [Amazon EC2 키 페어 및 Linux 인스턴스](#)를 참조하세요. Windows에 대한 자세한 내용은 Amazon EC2 사용 설명서의 [Amazon EC2 키 페어 및 Windows 인스턴스](#)를 참조하세요. 시작 템플릿을 사용하도록 선택한 경우 시작 템플릿을 선택할 수 없습니다. Bottlerocket AMI를 사용하여 노드 그룹에 Amazon EC2 SSH 키가 제공되면 관리 컨테이너도 사용됩니다. 자세한 내용은 GitHub의 [Admin container](#)를 참조하세요.
 - 특정 인스턴스에 대한 액세스를 제한하려면 다음에서 SSH 원격 액세스 허용(Allow SSH remote access from)에서 해당 인스턴스에 연결된 보안 그룹을 선택합니다. 특정 보안 그룹을 선택하지 않는 경우, 인터넷의 어느 곳에서나(0.0.0.0/0) SSH 액세스가 허용됩니다.
9. 검토 및 생성 페이지에서 관리형 노드 그룹 구성을 검토하고 생성을 선택합니다.

노드가 클러스터에 조인하지 못한 경우 문제 해결 장의 [the section called “노드가 클러스터 조인에 실패”](#) 섹션을 참조하세요.

10. 노드의 상태를 확인하고 Ready 상태가 될 때까지 대기합니다.

```
kubectl get nodes --watch
```

11. (GPU 노드만 해당) GPU 인스턴스 유형과 Amazon EKS 최적화 가속 AMI를 선택한 경우 클러스터에 [Kubernetes용 NVIDIA 디바이스 플러그인](#)을 DaemonSet으로 적용해야 합니다. 다음 명령을 실행하기 전에 **vX.X.X**를 원하는 [NVIDIA/k8s-device-plugin](#) 버전으로 바꿉니다.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/deployments/static/nvidia-device-plugin.yml
```

Kubernetes 추가 기능 설치

정상 작동하는 Amazon EKS 클러스터와 노드가 있으므로, Kubernetes 추가 기능을 설치하고 클러스터에 애플리케이션을 배포하기 시작할 준비가 되었습니다. 다음은 클러스터의 기능을 확장하는 데 도움이 되는 설명서 주제입니다.

- 클러스터를 생성한 [IAM 위탁자](#)는 kubectl 또는 AWS Management Console을 사용하여 Kubernetes API 서버를 직접적으로 직접 호출할 수 있는 유일한 위탁자입니다. 다른 IAM 보안 주체가 클러스터에 액세스할 수 있게 하려면 해당 보안 주체를 추가해야 합니다. 자세한 내용은 [the section called “Kubernetes API 액세스”](#) 및 [the section called “필수 권한”](#)(을)를 참조하세요.
- 다음과 같은 조건에 해당하면 IMDS에 대한 포드 액세스를 차단하는 것이 좋습니다.
 - 포드에 필요한 최소 권한만 있도록 모든 Kubernetes 서비스 계정에 IAM 역할을 할당할 계획입니다.
 - 클러스터의 어떤 포드도 현재 AWS 리전 검색 등의 다른 이유로 Amazon EC2 인스턴스 메타데이터 서비스(IMDS)에 액세스할 필요가 없습니다.

자세한 내용은 [작업자 노드에 할당된 인스턴스 프로파일에 대한 액세스 제한](#) 부분을 참조하세요.

- 노드 그룹의 노드 수를 자동으로 조정하도록 Kubernetes [Cluster Autoscaler](#)를 구성합니다.
- 클러스터에 [샘플 애플리케이션](#)을 배포합니다.
- 클러스터를 관리하기 위한 중요한 도구를 사용하여 [클러스터 리소스를 구성하고 모니터링](#)합니다.

클러스터에 대한 관리형 노드 그룹 업데이트

관리형 노드 그룹 업데이트를 시작하면 Amazon EKS가 [노드 업데이트의 각 단계 이해](#)에 나열된 단계를 완료하여 자동으로 노드를 업데이트합니다. Amazon EKS 최적화 AMI를 사용하는 경우 Amazon EKS는 최신 AMI 릴리스 버전의 일부로 최신 보안 패치 및 운영 체제 업데이트를 노드에 자동으로 적용합니다.

Amazon EKS 관리형 노드 그룹의 버전 또는 구성을 업데이트하는 것이 유용한 몇 가지 시나리오가 있습니다.

- Amazon EKS 클러스터의 Kubernetes 버전을 업데이트했으며 동일한 Kubernetes 버전을 사용하도록 노드를 업데이트하려고 합니다.
- 관리형 노드 그룹에 새 AMI 릴리스 버전을 사용할 수 있습니다. AMI 버전에 대한 자세한 내용은 다음 섹션을 참조하세요.
 - [the section called “버전 정보 받기”](#)
 - [the section called “Bottlerocket”](#)
 - [the section called “버전 정보 받기”](#)
- 관리형 노드 그룹에 있는 인스턴스의 최소, 최대 또는 원하는 수를 조정하려고 합니다.
- 관리형 노드 그룹의 인스턴스에서 Kubernetes 레이블을 추가하거나 제거하려고 합니다.
- 관리형 노드 그룹에서 AWS 태그를 추가하거나 제거하려고 합니다.

- 업데이트된 사용자 지정 AMI와 같은 구성 변경 사항이 있는 시작 템플릿의 새 버전을 배포해야 합니다.
- Amazon VPC CNI 추가 기능의 버전 1.9.0 이상을 배포하고, 접두사 위임에 대해 추가 기능을 사용하도록 설정했으며, 노드 그룹의 새로운 AWS Nitro System 인스턴스가 크게 증가된 포드 수를 지원하도록 했습니다. 자세한 내용은 [the section called “IP 주소 증가”](#) 단원을 참조하십시오.
- Windows 노드에 대해 IP 접두사 위임을 활성화했으며 노드 그룹의 새 AWS Nitro System 인스턴스가 크게 늘어난 수의 포드를 지원하도록 하려고 합니다. 자세한 내용은 [the section called “IP 주소 증가”](#) 단원을 참조하십시오.

관리형 노드 그룹의 Kubernetes 버전에 대한 최신 AMI 릴리스 버전이 있는 경우 노드 그룹의 버전을 업데이트하여 최신 AMI 버전을 사용할 수 있습니다. 마찬가지로 클러스터에서 노드 그룹보다 최신 버전의 Kubernetes 버전을 실행 중인 경우 최신 AMI 릴리스 버전을 사용하여 클러스터의 Kubernetes 버전과 일치하도록 노드 그룹을 업데이트할 수 있습니다.

크기 조정 작업 또는 업데이트로 인해 관리형 노드 그룹의 노드가 종료되면 해당 노드의 포드가 먼저 드레이닝됩니다. 자세한 내용은 [the section called “동작 세부 정보 업데이트”](#) 단원을 참조하십시오.

노드 그룹 버전 업데이트

다음 중 하나를 사용하여 노드 그룹 버전을 업데이트할 수 있습니다.

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)

업데이트하는 버전은 컨트롤 플레인 버전보다 이후일 수 없습니다.

eksctl

eksctl를 사용하여 관리형 노드 그룹 업데이트

다음 명령을 사용하여 노드에 현재 배포된 동일한 Kubernetes 버전의 최신 AMI 릴리스로 관리형 노드 그룹을 업데이트합니다. 모든 **##** **#**을 자신의 값으로 바꾸세요.

```
eksctl upgrade nodegroup \  
  --name=node-group-name \  
  --cluster=my-cluster \  
  --region=region-code
```

Note

시작 템플릿과 함께 배포된 노드 그룹을 새 시작 템플릿 버전으로 업그레이드하는 경우 이전 명령에 `--launch-template-version version-number` 을 추가합니다. 시작 템플릿은 [시작 템플릿을 사용하여 관리형 노드 사용자 지정](#)에 설명된 요구 사항을 충족해야 합니다. 시작 템플릿에 사용자 지정 AMI가 포함되어 있는 경우 해당 AMI가 [AMI 지정](#)의 요구 사항을 충족해야 합니다. 노드 그룹을 최신 버전의 시작 템플릿으로 업그레이드하면 지정된 시작 템플릿 버전의 새 구성과 일치하도록 모든 노드가 재활용됩니다.

시작 템플릿 없이 배포된 노드 그룹은 새 시작 템플릿 버전으로 직접 업그레이드할 수 없습니다. 대신 시작 템플릿을 사용하여 새 노드 그룹을 배포하여 노드 그룹을 새 시작 템플릿 버전으로 업데이트해야 합니다.

노드 그룹을 컨트롤 플레인의 Kubernetes 버전과 동일한 버전으로 업그레이드할 수 있습니다. 예를 들어 Kubernetes 1.32을 실행하는 클러스터가 있는 경우 다음 명령을 사용하여 현재 Kubernetes 1.31을 실행 중인 노드를 버전 1.32으로 업그레이드할 수 있습니다.

```
eksctl upgrade nodegroup \
  --name=node-group-name \
  --cluster=my-cluster \
  --region=region-code \
  --kubernetes-version=1.32
```

AWS Management Console

AWS Management Console를 사용하여 관리형 노드 그룹 업데이트

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 업데이트할 노드 그룹이 포함된 클러스터를 선택합니다.
3. 하나 이상의 노드 그룹에 사용 가능한 업데이트가 있는 경우 페이지 상단에 사용 가능한 업데이트를 알리는 상자가 나타납니다. 컴퓨팅 탭을 선택하면 사용 가능한 업데이트가 있는 노드 그룹에 대한 노드 그룹 테이블의 AMI 릴리스 버전 열에 지금 업데이트가 표시됩니다. 노드 그룹을 업데이트하려면 지금 업데이트를 선택합니다.

사용자 지정 AMI와 함께 배포된 노드 그룹에 대한 알림이 표시되지 않습니다. 노드가 사용자 지정 AMI와 함께 배포된 경우 다음 단계를 완료하여 새로운 업데이트된 사용자 지정 AMI를 배포합니다.

- a. AMI의 새 버전을 생성합니다.

- b. 새 AMI ID를 사용하여 새 시작 템플릿 버전을 생성합니다.
 - c. 노드를 새 버전의 시작 템플릿으로 업그레이드합니다.
4. 노드 그룹 버전 업데이트 대화 상자에서 다음과 같은 옵션을 활성화하거나 비활성화합니다.
- 노드 그룹 버전 업데이트 - 사용자 지정 AMI를 배포했거나 Amazon EKS에 최적화된 AMI가 현재 클러스터의 최신 버전에 있는 경우에는 이 옵션을 사용할 수 없습니다.
 - 시작 템플릿 버전 변경 - 노드 그룹이 사용자 지정 시작 템플릿 없이 배포된 경우에는 이 옵션을 사용할 수 없습니다. 사용자 지정 시작 템플릿을 사용하여 배포된 노드 그룹의 시작 템플릿 버전만 업데이트할 수 있습니다. 노드 그룹을 업데이트할 시작 템플릿 버전을 선택합니다. 노드 그룹이 사용자 지정 AMI로 구성된 경우 선택한 버전에서도 AMI를 지정해야 합니다. 최신 버전의 시작 템플릿으로 업그레이드하면 지정된 시작 템플릿 버전의 새 구성과 일치하도록 모든 노드가 재활용 됩니다.
5. 전략 업데이트의 경우, 다음과 같은 옵션 중 하나를 선택합니다.
- 롤링 업데이트 - 이 옵션은 클러스터에 대한 포드 중단 예산을 고려합니다. 포드 중단 예산 문제로 인해 Amazon EKS에서 이 노드 그룹에서 실행 중인 포드를 적절하게 드레이닝할 수 없는 경우 업데이트에 실패합니다.
 - 강제 업데이트 - 이 옵션은 포드 중단 예산을 따르지 않습니다. 노드 재시작을 강제로 적용하여 포드 중단 예산 문제와 관계없이 업데이트가 수행됩니다.

6. 업데이트를 선택합니다.

노드 그룹 구성 편집

관리형 노드 그룹의 일부 구성을 수정할 수 있습니다.

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 편집할 노드 그룹이 포함된 클러스터를 선택합니다.
3. 컴퓨팅 탭을 선택합니다.
4. 편집할 노드 그룹을 선택한 다음에 편집을 선택합니다.
5. (선택사항) 노드 그룹 편집 페이지에서 다음을 수행합니다.
 - a. 노드 그룹 크기 조정 구성을 편집합니다.
 - 원하는 크기 - 관리형 노드 그룹에서 유지해야 하는 현재 노드 수를 지정합니다.
 - 최소 크기-관리형 노드 그룹이 확장될 수 있는 최소 노드 수를 지정합니다.
 - 최대 크기 - 관리형 노드 그룹이 확장될 수 있는 최대 노드 수를 지정합니다. 노드 그룹에서 지원되는 최대 노드 수는 [the section called “Service quotas”](#) 섹션을 참조하세요.

- b. (선택 사항) Kubernetes 레이블을 노드 그룹의 노드에 추가하거나 제거합니다. 여기에 표시된 레이블은 Amazon EKS에 적용한 레이블일 뿐입니다. 여기에 표시되지 않는 노드에 다른 레이블이 존재할 수 있습니다.
- c. (선택 사항) Kubernetes 테인트를 노드 그룹의 노드에 추가하거나 제거합니다. 추가된 테인트는 **NoSchedule**, **NoExecute** 또는 **PreferNoSchedule** 중 하나의 효과를 가질 수 있습니다. 자세한 내용은 [the section called “테인트 GPU 노드”](#) 섹션을 참조하세요.
- d. (선택사항) 태그를 추가하거나 노드 그룹 리소스에서 제거합니다. 이러한 태그는 Amazon EKS 노드 그룹에만 적용됩니다. 노드 그룹의 서브넷 또는 Amazon EC2 인스턴스 등의 다른 리소스로 전파되지 않습니다.
- e. (선택사항) 노드 그룹 업데이트 구성을 편집합니다. 숫자 또는 비율을 선택합니다.
 - 숫자 - 노드 그룹에서 병렬로 업데이트할 수 있는 노드 수를 선택하고 지정합니다. 이러한 노드는 업데이트 중에 사용할 수 없습니다.
 - 비율 - 노드 그룹에서 병렬로 업데이트할 수 있는 노드의 비율을 선택하고 지정합니다. 이러한 노드는 업데이트 중에 사용할 수 없습니다. 이 기능은 노드 그룹에 많은 노드가 있는 경우에 유용합니다.
- f. 편집을 마쳤으면 변경 사항 저장을 선택합니다.

Important

노드 그룹 구성을 업데이트할 때 [NodegroupScalingConfig](#)를 수정해도 포드 중단 예산(PDB)이 반영되지 않습니다. 업그레이드 단계에서 노드를 소모시키고 PDB를 고려하는 [노드 그룹 업데이트](#) 프로세스와 달리, 스케일링 구성을 업데이트하면 Auto Scaling 그룹(ASG) 스케일 다운 호출을 통해 노드가 즉시 종료됩니다. 이는 스케일 다운하려는 대상 크기와 상관없이 PDB를 고려하지 않고 이루어집니다. 즉, Amazon EKS 관리형 노드 그룹의 `desiredSize`를 줄이면 포드는 PDB를 적용하지 않고 노드가 종료되는 즉시 제거됩니다.

노드 업데이트의 각 단계 이해

Amazon EKS 관리형 작업자 노드 업그레이드 전략에는 다음 섹션에서 설명하는 4가지 단계가 있습니다.

설정 단계

설정 단계는 다음과 같습니다.

1. 노드 그룹과 연결된 Auto Scaling 그룹의 새 Amazon EC2 시작 템플릿 버전을 생성합니다. 새 시작 템플릿 버전은 업데이트를 위해 대상 AMI 또는 사용자 지정 시작 템플릿 버전을 사용합니다.
2. 최신 시작 템플릿 버전을 사용하도록 Auto Scaling 그룹을 업데이트합니다.
3. 노드 그룹의 `updateConfig` 속성을 사용하여 병렬로 업그레이드할 최대 노드 수를 결정합니다. 사용할 수 없는 최대 노드의 할당량은 100개입니다. 기본값은 노드 1개입니다. 자세한 내용은 Amazon EKS API 참조에서 [updateConfig](#) 속성을 참조하세요.

확장 단계

관리형 노드 그룹의 노드를 업그레이드할 때 업그레이드된 노드는 업그레이드 중인 노드와 동일한 가용 영역에서 시작됩니다. 이 배치를 보장하기 위해 Amazon EC2의 가용 영역 재분배를 사용합니다. 자세한 내용은 Amazon EC2 Auto Scaling 사용 설명서의 [가용 영역 재분배](#)를 참조하세요. 이 요구 사항이 충족되도록 관리형 노드 그룹의 가용 영역당 최대 2개의 인스턴스를 시작할 수 있습니다.

확장 단계는 다음과 같습니다.

1. Auto Scaling 그룹의 최대 크기와 원하는 크기를 다음 중 더 큰 값으로 늘립니다.

- Auto Scaling 그룹이 배포된 가용 영역 수의 최대 2배.
- 업그레이드할 수 없는 최대값입니다.

예를 들어, 노드 그룹에 5개의 가용 영역이 있고 `maxUnavailable`이 하나인 경우 업그레이드 프로세스에서 최대 10개의 노드를 시작할 수 있습니다. 하지만 `maxUnavailable`이 20이거나 10보다 크면 프로세스에서 20개의 새 노드를 시작합니다.

2. Auto Scaling 그룹 크기를 조정된 후 해당 노드 그룹에 최신 설정을 사용하는 노드가 있는지 확인합니다. 이 단계는 다음 기준을 충족하는 경우에만 성공합니다.

- 노드가 있는 모든 가용 영역에서 하나 이상의 새 노드가 시작됩니다.
- 모든 새 노드는 Ready 상태여야 합니다.
- 새 노드에 Amazon EKS 적용 레이블이 있어야 합니다.

다음은 일반 노드 그룹에 있는 작업자 노드의 Amazon EKS 적용 레이블입니다.

- `eks.amazonaws.com/nodegroup-image=$amiName`
- `eks.amazonaws.com/nodegroup=$nodeGroupName`

다음은 사용자 정의 시작 템플릿 또는 AMI 노드 그룹에 있는 작업자 노드의 Amazon EKS 적용 레이블입니다.

- `eks.amazonaws.com/nodegroup-image=$amiName`

- `eks.amazonaws.com/nodegroup=$nodeGroupName`
- `eks.amazonaws.com/sourceLaunchTemplateId=$launchTemplateId`
- `eks.amazonaws.com/sourceLaunchTemplateVersion=$launchTemplateVersion`

3. 새 포드의 예약을 피하기 위해 노드를 스케줄 불가능으로 표시합니다. 또한 노드를 종료하기 전에 로드 밸런서에서 노드를 제거하도록 `node.kubernetes.io/exclude-from-external-load-balancers=true`로 노드에 레이블을 지정합니다.

이 단계에서 `NodeCreationFailure` 오류가 발생하는 알려진 이유는 다음과 같습니다.

가용 영역의 용량 부족

요청된 인스턴스 유형의 용량이 가용 영역에 없을 가능성이 있습니다. 관리형 노드 그룹을 생성하는 동안 여러 인스턴스 유형을 구성하는 것이 좋습니다.

계정의 EC2 인스턴스 제한

Service Quotas를 사용하여 계정이 동시에 실행할 수 있는 Amazon EC2 인스턴스 수를 늘려야 할 수 있습니다. 자세한 내용은 Linux 인스턴스용 Amazon Elastic Compute Cloud 사용 설명서의 [EC2 Service Quotas](#)를 참조하세요.

사용자 정의 사용자 데이터

사용자 정의 사용자 데이터로 인해 부트스트랩 프로세스가 중단될 수 있습니다. 이 시나리오에서는 노드에서 kubelet이 시작되지 않거나 예상되는 Amazon EKS 레이블을 가져오지 못할 수 있습니다. 자세한 내용은 [the section called “AMI 지정”](#) 섹션을 참조하세요.

노드를 비정상적이거나 준비되지 않은 상태로 만드는 모든 변경 사항

노드 디스크 압력, 메모리 압력 및 이와 유사한 조건으로 인해 노드가 Ready 상태가 되지 않을 수 있습니다.

15분 내에 각 노드 대부분 부트스트랩

노드가 클러스터를 부트스트랩하고 조인하는 데 15분 이상 걸리는 경우 업그레이드 시간이 초과됩니다. 새 노드가 필요한 시점부터 클러스터에 조인할 때까지 측정된 새 노드 부트스트래핑을 위한 총 런타임입니다. 관리형 노드 그룹을 업그레이드할 때 Auto Scaling 그룹 크기가 커지는 즉시 시간 카운터가 시작됩니다.

업그레이드 단계

업그레이드 단계는 업데이트 전략에 따라 두 가지 방식으로 작동합니다. 기본과 최소의 두 가지 업데이트 전략이 있습니다.

대부분의 시나리오에서는 기본 전략을 사용하는 것이 좋습니다. 이 전략은 이전 노드를 종료하기 전에 새 노드를 생성하므로 업그레이드 단계에서 사용 가능한 용량이 유지됩니다. 최소 전략은 GPU 등의 하드웨어 액셀러레이터와 같이 리소스 또는 비용에 제약이 있는 시나리오에서 유용합니다. 이 전략은 새 노드를 생성하기 전에 이전 노드를 종료하므로 총 용량이 구성된 수량을 초과하여 증가하지 않습니다.

기본 업데이트 전략에는 다음 단계가 있습니다.

1. Auto Scaling 그룹에서 노드의 양(원하는 수)을 늘려 노드 그룹이 추가 노드를 생성하게 합니다.
2. 노드 그룹에 대해 구성된 사용 불가능한 최대값까지 업그레이드해야 하는 노드를 무작위로 선택합니다.
3. 노드에서 포드를 드레이닝합니다. 포드가 15분 이내에 노드를 떠나지 않고 force 플래그가 없으면 PodEvictionFailure 오류와 함께 업그레이드 단계가 실패합니다. 이 시나리오에서는 update-nodegroup-version 요청과 함께 force 플래그를 적용하여 포드를 삭제할 수 있습니다.
4. 모든 포드가 제거된 후 노드를 차단하고 60초 동안 기다립니다. 이 작업은 서비스 컨트롤러가 이 노드에 새 요청을 보내지 않고 활성 노드 목록에서 이 노드를 제거하도록 하기 위해 수행됩니다.
5. 코드화된 노드에 대한 Auto Scaling 그룹에 종료 요청을 보냅니다.
6. 이전 버전의 시작 템플릿으로 배포된 노드 그룹에 노드가 없을 때까지 이전 업그레이드 단계를 반복합니다.

최소 업데이트 전략에는 다음 단계가 있습니다.

1. 서비스 컨트롤러가 이러한 노드에 새 요청을 보내지 않도록 처음부터 노드 그룹의 모든 노드를 차단합니다.
2. 노드 그룹에 대해 구성된 사용 불가능한 최대값까지 업그레이드해야 하는 노드를 무작위로 선택합니다.
3. 선택한 노드에서 포드를 드레이닝합니다. 포드가 15분 이내에 노드를 떠나지 않고 force 플래그가 없으면 PodEvictionFailure 오류와 함께 업그레이드 단계가 실패합니다. 이 시나리오에서는 update-nodegroup-version 요청과 함께 force 플래그를 적용하여 포드를 삭제할 수 있습니다.

4. 모든 포드가 제거되면 60초 동안 기다린 후 선택한 노드의 Auto Scaling 그룹에 종료 요청을 전송합니다. Auto Scaling 그룹이 누락된 용량을 대체하기 위해 (선택한 노드 수와 동일하게) 새 노드를 생성합니다.
5. 이전 버전의 시작 템플릿으로 배포된 노드 그룹에 노드가 없을 때까지 이전 업그레이드 단계를 반복합니다.

업그레이드 단계 중 **PodEvictionFailure** 오류

이 단계에서 PodEvictionFailure 오류가 발생하는 알려진 이유는 다음과 같습니다.

적극적인 PDB

적극적인 PDB가 포드에 정의되어 있거나 동일한 포드를 가리키는 여러 PDB가 있습니다.

모든 테인트를 허용하는 배포

모든 포드가 제거되면 이전 단계에서 노드가 [테인트](#)되었기 때문에 노드가 비어 있어야 합니다. 그러나 배포가 모든 테인트를 허용하는 경우 노드가 비어 있지 않을 가능성이 높아져 포드 제거 실패로 이어집니다.

축소 단계

축소 단계는 Auto Scaling 그룹의 최대 크기와 원하는 크기를 하나씩 줄여 업데이트가 시작되기 전의 값으로 돌아갑니다.

업그레이드 워크플로에서 Cluster Autoscaler가 워크플로의 축소 단계에서 노드 그룹을 확장하고 있다고 판단하면 노드 그룹을 원래 크기로 되돌리지 않고 즉시 종료됩니다.

시작 템플릿을 사용한 관리형 노드 사용자 지정

최고 수준의 사용자 지정을 위해 자체 시작 템플릿과 사용자 지정 AMI를 사용하여 관리형 노드를 배포할 수 있습니다. 시작 템플릿을 사용하면 다음과 같은 기능을 사용할 수 있습니다.

- 노드 배포 시 추가 [kubelet](#) 인수와 같은 부트스트랩 인수를 제공합니다.
- 노드에 할당된 IP 주소와 다른 CIDR 블록의 포드에 IP 주소를 할당합니다.
- 사용자 지정 AMI 노드에 배포할 수 있습니다.
- 사용자 지정 CNI 노드에 배포할 수 있습니다.

관리형 노드 그룹을 처음 만들 때 자체 시작 템플릿을 제공하면 나중에 더 유연하게 사용할 수 있습니다. 자체 시작 템플릿을 사용하여 관리형 노드 그룹을 배포한 후 동일한 시작 템플릿의 다른 버전으로 업데이트할 수 있습니다. 노드 그룹을 다른 버전의 시작 템플릿으로 업데이트하면 그룹의 모든 노드가 지정된 시작 템플릿 버전의 새 구성과 일치하도록 재할용됩니다.

관리형 노드 그룹은 항상 Amazon EC2 Auto Scaling 그룹에서 사용할 시작 템플릿과 함께 배포됩니다. Amazon EKS API는 사용자가 제공한 템플릿을 복사하거나 계정의 기본값으로 자동 생성하여 이 시작 템플릿을 생성합니다. 자동 생성된 시작 템플릿을 수정하지 않는 것이 좋습니다. 사용자 정의 시작 템플릿을 사용하지 않는 기존 노드 그룹은 직접 업데이트할 수 없습니다. 대신 사용자 정의 시작 템플릿을 사용하여 새 노드 그룹을 생성해야 합니다.

시작 템플릿 기본 사항

AWS Management Console, AWS CLI 또는 AWS SDK를 사용하여 Amazon EC2 Auto Scaling 시작 템플릿을 생성할 수 있습니다. 자세한 내용을 알아보려면 Amazon EC2 Auto Scaling 사용 설명서의 [Auto Scaling 그룹을 위한 시작 템플릿 생성](#)을 참조하세요. 시작 템플릿의 일부 설정은 관리형 노드 구성에 사용되는 설정과 유사합니다. 시작 템플릿을 사용하여 노드 그룹을 배포하거나 업데이트할 때 노드 그룹 구성 또는 시작 템플릿에 일부 설정을 지정해야 합니다. 두 위치 모두에 설정을 지정하지 않습니다. 잘못된 위치에 설정이 있으면 노드 그룹 생성 또는 업데이트와 같은 작업이 실패합니다.

다음 표에는 시작 템플릿에서 금지된 설정이 나열되어 있습니다. 사용 가능한 경우 관리형 노드 그룹 구성에 필요한 유사한 설정도 나열되어 있습니다. 나열된 설정은 콘솔에 표시되는 설정입니다. AWS CLI 및 SDK에서 비슷하지만 다른 이름을 가질 수 있습니다.

시작 템플릿 - 금지됨	Amazon EKS 노드 그룹 구성
네트워크 인터페이스의 서브넷(네트워크 인터페이스 추가)	네트워킹 지정 페이지의 노드 그룹 네트워크 구성에 있는 서브넷
고급 세부 정보의 IAM 인스턴스 프로필	노드 그룹 구성 페이지의 노드 그룹 구성에 있는 노드 IAM 역할
고급 세부 정보의 종료 동작 및 중지 - 최대 절전 모드 동작. 두 설정 모두 실행 템플릿에서 기본 값인 실행 템플릿 설정에 포함하지 않음을 유지합니다.	동일한 사항 없음 Amazon EKS는 Auto Scaling 그룹이 아니라 인스턴스 수명 주기를 제어해야 합니다.

다음 표에는 관리형 노드 그룹 구성에서 금지된 설정이 나열되어 있습니다. 사용 가능한 경우 시작 템플릿에 필요한 유사한 설정도 나열되어 있습니다. 나열된 설정은 콘솔에 표시되는 설정입니다. AWS CLI 및 SDK에서 비슷한 이름을 가질 수 있습니다.

Amazon EKS 노드 그룹 구성 — 금지됨	시작 템플릿
(시작 템플릿에서 사용자 정의 AMI를 지정한 경우,에만 해당) 컴퓨팅 및 크기 조정 구성 설정 페이지의 노드 그룹 컴퓨팅 구성에 있는 AMI 유형 - 콘솔에 시작 템플릿에서 지정됨이라는 메시지와 지정된 AMI ID가 표시됩니다. 시작 템플릿에 애플리케이션 및 OS 이미지 (Amazon 머신 이미지)가 지정되지 않은 경우, 노드 그룹 구성에서 AMI를 선택할 수 있습니다.	시작 템플릿 콘텐츠의 애플리케이션 및 OS 이미지(Amazon Machine Image) - 다음 요구 사항 중 하나가 있는 경우, ID를 지정해야 합니다. • 사용자 지정 AMI 사용 AMI 지정에 나열된 요구 사항을 충족하지 않는 AMI를 지정하는 경우 노드 그룹 배포가 실패합니다. • Amazon EKS 최적화 AMI에 포함된 <code>bootstrap.sh</code> 파일에 인수를 제공하기 위해 사용자 데이터를 제공하려고 함 인스턴스가 포드에 훨씬 더 많은 수의 IP 주소를 할당하도록 하거나, 인스턴스와 다른 CIDR 블록의 포드에 IP 주소를 할당하거나, 아웃바운드 인터넷 액세스 없이 프라이빗 클러스터를 배포하도록 할 수 있습니다. 자세한 정보는 다음의 주제를 참조하세요. • 접두사를 사용하여 Amazon EKS 노드에 추가 IP 주소 할당 • 사용자 지정 네트워킹을 통해 대체 서브넷에 포드 배포 • 인터넷 액세스가 제한된 프라이빗 클러스터 배포 • AMI 지정
컴퓨팅 및 크기 조정 구성 설정 페이지의 노드 그룹 컴퓨팅 구성에 있는 디스크 크기 - 콘솔에 시작 템플릿에 지정됨이라는 메시지가 표시됩니다.	스토리지(볼륨)의 크기(새 볼륨 추가). 시작 템플릿에서 이 옵션을 지정해야 합니다.
네트워킹 지정 페이지의 노드 그룹 구성에 있는 SSH 키 페어 - 콘솔에 시작 템플릿에 지정되어	키 페어(로그인)의 키 페어 이름.

Amazon EKS 노드 그룹 구성 — 금지됨	시작 템플릿
있는 키가 표시되거나 시작 템플릿에 지정되지 않음이라는 메시지가 표시됩니다.	
시작 템플릿을 사용할 때 원격 액세스가 허용되는 소스 보안 그룹을 지정할 수 없습니다.	인스턴스의 네트워크 설정에 있는 보안 그룹 또는 네트워크 인터페이스의 보안 그룹(네트워크 인터페이스 추가), 두 가지 모두 설정할 수는 없습니다. 자세한 내용은 the section called “사용자 지정 보안 그룹 사용” 섹션을 참조하세요.

Note

- 시작 템플릿을 사용하여 노드 그룹을 배포하는 경우 시작 템플릿의 시작 템플릿 콘텐츠에서 인스턴스 유형을 0개 또는 1개 지정합니다. 또는 콘솔의 컴퓨팅 및 크기 조정 구성 설정 페이지에서 인스턴스 유형에 대해 0~20개의 인스턴스 유형을 지정할 수 있습니다. 또는 Amazon EKS API를 사용하는 다른 도구로 인스턴스 유형을 지정할 수 있습니다. 시작 템플릿에서 인스턴스 유형을 지정하고 해당 시작 템플릿을 사용하여 노드 그룹을 배포하는 경우 콘솔에서 인스턴스 유형을 지정하거나 Amazon EKS API를 사용하는 다른 도구를 사용하여 인스턴스 유형을 지정할 수 없습니다. 시작 템플릿, 콘솔 또는 Amazon EKS API를 사용하는 다른 도구를 사용하여 인스턴스 유형을 지정하지 않으면 t3.medium 인스턴스 유형이 사용됩니다. 노드 그룹에서 스팟 용량 유형을 사용하는 경우 콘솔을 사용하여 여러 인스턴스 유형을 지정하는 것이 좋습니다. 자세한 내용은 [the section called “관리형 노드 그룹 용량”](#) 섹션을 참조하세요.
- 노드 그룹에 배포하는 컨테이너가 인스턴스 메타데이터 서비스 버전 2를 사용하는 경우 시작 템플릿에 메타데이터 응답 홉 제한이 2로 설정되어 있어야 합니다. 자세한 내용은 Amazon EC2 사용 설명서의 [인스턴스 메타데이터 및 사용자 데이터](#)를 참조하세요. 사용자 지정 시작 템플릿을 사용하지 않고 관리형 노드 그룹을 배포하는 경우 이 값은 기본 시작 템플릿의 노드 그룹에 대해 자동으로 설정됩니다.
- 시작 템플릿은 유연한 인스턴스 유형 선택을 허용하는 InstanceRequirements 기능을 지원하지 않습니다.

Amazon EC2 인스턴스 태깅

시작 템플릿의 TagSpecification 파라미터를 사용하여 노드 그룹의 Amazon EC2 인스턴스에 적용할 태그를 지정할 수 있습니다. CreateNodegroup 또는 UpdateNodegroupVersionAPI를 호출하는 IAM 엔터티에는 ec2:RunInstances 및 ec2:CreateTags에 대한 권한이 있어야 하며, 시작 템플릿에 태그가 추가되어 있어야 합니다.

사용자 지정 보안 그룹 사용

시작 템플릿을 사용하여 사용자 지정 Amazon EC2 [보안 그룹](#) 지정하고 노드 그룹의 인스턴스에 적용할 수 있습니다. 이는 인스턴스 수준 보안 그룹 파라미터 또는 네트워크 인터페이스 구성 파라미터의 일부로 사용할 수 있습니다. 하지만 인스턴스 수준 및 네트워크 인터페이스 보안 그룹을 모두 지정하는 시작 템플릿을 생성할 수 없습니다. 관리형 노드 그룹과 함께 사용자 지정 보안 그룹을 사용할 때 적용되는 다음 조건을 고려하세요.

- AWS Management Console을 사용하면 Amazon EKS는 단일 네트워크 인터페이스 사양이 있는 시작 템플릿만 허용합니다.
- 기본적으로 Amazon EKS는 [클러스터 보안 그룹](#)을 노드 그룹의 인스턴스에 적용하여 노드와 제어 플레인 간의 통신을 용이하게 합니다. 앞에서 언급한 옵션 중 하나를 사용하여 시작 템플릿에 사용자 지정 보안 그룹을 지정하는 경우 Amazon EKS는 클러스터 보안 그룹을 추가하지 않습니다. 따라서 보안 그룹의 인바운드 및 아웃바운드 규칙이 클러스터 엔드포인트와의 통신을 사용 설정하는지 확인해야 합니다. 보안 그룹 규칙이 올바르지 않으면 작업자 노드가 클러스터에 참여할 수 없습니다. 보안 그룹 규칙에 대한 자세한 내용은 [the section called “보안 그룹 요구 사항”](#) 부분을 참조하세요.
- 노드 그룹의 인스턴스에 대한 SSH 액세스가 필요한 경우 해당 액세스를 허용하는 보안 그룹을 포함합니다.

Amazon EC2 사용자 데이터

시작 템플릿에는 사용자 정의 사용자 데이터에 대한 부분이 포함되어 있습니다. 개별 사용자 정의 AMI를 수동으로 생성하지 않고 이 섹션에서 노드 그룹에 대한 구성 설정을 지정할 수 있습니다. Bottlerocket에 사용할 수 있는 설정에 대한 자세한 내용은 GitHub의 [Using user data](#)을 참조하세요.

인스턴스를 시작할 때 ccloud-init를 사용하여 시작 템플릿에 Amazon EC2 사용자 데이터를 제공할 수 있습니다. 자세한 내용은 [cloud-init 설명서](#)를 참조하세요. 사용자 데이터를 사용하여 일반적인 구성 작업을 수행할 수 있습니다. 여기에는 다음 옵션이 포함됩니다.

- [사용자 또는 그룹 포함](#)
- [패키지 설치](#)

관리형 노드 그룹과 함께 사용되는 시작 템플릿의 Amazon EC2 사용자 데이터는 Amazon Linux AMI의 경우 [MIME 멀티파트 아카이브](#) 형식이어야 하고 Bottlerocket AMI의 경우 TOML 형식이어야 합니다. 이는 사용자 데이터가 노드가 클러스터에 조인하는 데 필요한 Amazon EKS 사용자 데이터와 병합되기 때문입니다. kubelet을 시작하거나 수정하는 명령을 사용자 데이터에 지정하지 마세요. 이는 Amazon EKS에 의해 병합된 사용자 데이터의 일부로 수행됩니다. 노드의 레이블 설정과 같은 특정 kubelet 파라미터는 관리형 노드 그룹 API를 통해 직접 구성될 수 있습니다.

Note

수동으로 시작하거나 사용자 정의 구성 파라미터를 전달하는 등의 고급 kubelet 사용자 지정에 대한 자세한 내용은 [the section called “AMI 지정”](#) 부분을 참조하세요. 시작 템플릿에 사용자 정의 AMI ID가 지정된 경우 Amazon EKS는 사용자 데이터를 병합하지 않습니다.

다음 세부 정보는 사용자 데이터 섹션에 대한 자세한 정보를 제공합니다.

Amazon Linux 2 사용자 데이터

여러 사용자 데이터 블록을 단일 MIME 멀티파트 파일로 결합할 수 있습니다. 예를 들어 Docker 데몬을 구성하는 클라우드 boothook를 사용자 지정 패키지를 설치하는 사용자 데이터 셸 스크립트와 결합할 수 있습니다. MIME 멀티파트 파일은 다음과 같은 구성 요소로 이루어집니다.

- 콘텐츠 유형 및 요소 경계 선언 - Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="
- MIME 버전 선언 - MIME-Version: 1.0
- 다음 구성 요소를 포함하는 하나 이상의 사용자 데이터 블록:
 - 사용자 데이터 블록의 시작을 나타내는 열린 경계 - ---==MYBOUNDARY==
 - 블록에 대한 콘텐츠 유형 선언: Content-Type: text/cloud-config; charset="us-ascii". 콘텐츠 유형에 대한 자세한 내용은 [cloud-init](#) 문서를 참조하세요.
 - 사용자 데이터 콘텐츠(예: 셸 명령 또는 cloud-init 지시어 목록)
 - MIME 멀티파트 파일의 끝을 나타내는 폐쇄 경계: ---==MYBOUNDARY===

다음은 직접 파일을 작성하는 데 사용할 수 있는 MIME 멀티파트 파일의 예입니다.

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="==MYBOUNDARY=="

---==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"
```

```
#!/bin/bash
echo "Running custom user data script"

---MYBOUNDARY---
```

Amazon Linux 2023 사용자 데이터

Amazon Linux 2023(AL2023)에 YAML 구성 스키마를 사용하는 새로운 노드 초기화 프로세스 nodeadm이 도입됩니다. 자체 관리형 노드 그룹 또는 시작 템플릿이 있는 AMI를 사용하는 경우 이제 새 노드 그룹을 생성할 때 추가 클러스터 메타데이터를 명시적으로 제공해야 합니다. 최소 필수 파라미터의 [예시](#)는 다음과 같으며, 여기에서 apiServerEndpoint, certificateAuthority 및 서비스 cidr가 필수입니다.

```
---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    cidr: 10.100.0.0/16
```

대체로 이 구성은 사용자 데이터에 있는 그대로 또는 MIME 멀티파트 문서에 포함되도록 설정합니다.

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="BOUNDARY"

--BOUNDARY
Content-Type: application/node.eks.aws

---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig spec: [...]

--BOUNDARY--
```

AL2에서 이러한 파라미터의 메타데이터는 Amazon EKS DescribeCluster API 직접 호출에서 확인되었습니다. AL2023 버전에서는 대형 노드 스케일 업 도중 추가 API 직접 호출로 인

해 제한이 발생할 위험이 있기 때문에 이러한 동작이 변경되었습니다. 시작 템플릿이 없는 관리형 노드 그룹을 사용하거나 Karpenter를 사용하는 경우 이 변경 사항이 영향을 미치지 않습니다. `certificateAuthority` 및 서비스 `cidr`에 대한 자세한 내용은 Amazon EKS API 참조의 [DescribeCluster](#)를 참조하세요.

다음은 노드를 사용자 지정하기 위한 셸 스크립트(예: 패키지 설치 또는 컨테이너 이미지 사전 캐싱)를 필수 `nodeadm` 구성과 결합하는 AL2023 사용자 데이터의 전체 예제입니다. 이 예제는 다음을 포함한 일반적인 사용자 지정을 보여줍니다. * 추가 시스템 패키지 설치 * 포드 시작 시간을 개선하기 위해 컨테이너 이미지 사전 캐싱 * HTTP 프록시 구성 설정 * 노드 레이블 지정을 위한 `kubelet` 플래그 구성

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="BOUNDARY"

--BOUNDARY
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
set -o errexit
set -o pipefail
set -o nounset

# Install additional packages
yum install -y htop jq iptables-services

# Pre-cache commonly used container images
nohup docker pull public.ecr.aws/eks-distro/kubernetes/pause:3.2 &

# Configure HTTP proxy if needed
cat > /etc/profile.d/http-proxy.sh << 'EOF'
export HTTP_PROXY="http://proxy.example.com:3128"
export HTTPS_PROXY="http://proxy.example.com:3128"
export NO_PROXY="localhost,127.0.0.1,169.254.169.254,.internal"
EOF

--BOUNDARY
Content-Type: application/node.eks.aws

apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
```

```

name: my-cluster
apiServerEndpoint: https://example.com
certificateAuthority: Y2VydG1maWNhdGVBdXRob3JpdHk=
cidr: 10.100.0.0/16
kubelet:
  config:
    clusterDNS:
      - 10.100.0.10
  flags:
    - --node-labels=app=my-app,environment=production

--BOUNDARY--

```

Bottlerocket 사용자 데이터

Bottlerocket은 사용자 데이터를 TOML 형식으로 구성합니다. Amazon EKS에서 제공하는 사용자 데이터와 병합할 사용자 데이터를 제공할 수 있습니다. 예를 들어, 추가 kubelet 설정을 제공할 수 있습니다.

```

[settings.kubernetes.system-reserved]
cpu = "10m"
memory = "100Mi"
ephemeral-storage= "1Gi"

```

지원되는 설정에 대한 자세한 내용은 [Bottlerocket 설명서](#)를 참조하세요. 사용자 데이터에서 노드 레이블과 [테인트](#)를 구성할 수 있습니다. 그러나 대신 노드 그룹 내에서 구성하는 것이 좋습니다. 그러면 Amazon EKS에서 이러한 구성을 적용합니다.

사용자 데이터가 병합되면 서식이 유지되지 않지만 내용은 동일하게 유지됩니다. 사용자 데이터에 제공하는 구성은 Amazon EKS에서 구성하는 모든 설정을 재정의합니다. 따라서 `settings.kubernetes.max-pods` 또는 `settings.kubernetes.cluster-dns-ip`를 설정하면 사용자 데이터의 값이 노드에 적용됩니다.

Amazon EKS는 모든 유효한 TOML을 지원하지 않습니다. 다음은 지원되지 않는 알려진 형식 목록입니다.

- 따옴표 붙은 키 안의 따옴표: `'quoted "value"' = "value"`
- 값의 이스케이프된 따옴표: `str = "I'm a string. \"You can quote me\""`
- 혼합 부동 소수점 및 정수: `numbers = [ 0.1, 0.2, 0.5, 1, 2, 5 ]`
- 배열의 혼합 유형: `contributors = [ "foo@example.com", { name = "Baz", email = "baz@example.com" } ]`

- 다음표 붙은 키가 있는 대괄호로 묶은 헤더: [foo."bar.baz"]

Windows 사용자 데이터

Windows 사용자 데이터는 PowerShell 명령을 사용합니다. 관리형 노드 그룹을 생성할 때 사용자 지정 사용자 데이터는 Amazon EKS 관리형 사용자 데이터와 결합됩니다. PowerShell 명령과 관리형 사용자 데이터 명령이 모두 하나의 <powershell></powershell> 태그 내에서 차례로 나타납니다.

Important

Windows 노드 그룹을 생성할 때 Amazon EKS는 Linux 기반 노드가 클러스터에 조인할 수 있도록 aws-auth ConfigMap을 업데이트합니다. 서비스는 Windows AMI에 대한 권한을 자동으로 구성하지 않습니다. Windows 노드를 사용하는 경우 액세스 항목 API를 통해 또는 aws-auth ConfigMap을 직접 업데이트하여 액세스를 관리해야 합니다. 자세한 내용은 [the section called “Windows 지원 사용”](#) 단원을 참조하십시오.

Note

시작 템플릿에 AMI ID가 지정되어 있지 않으면 사용자 데이터의 Windows Amazon EKS Bootstrap 스크립트를 사용하여 Amazon EKS를 구성하지 마세요.

예시 사용자 데이터는 다음과 같습니다.

```
<powershell>
Write-Host "Running custom user data script"
</powershell>
```

AMI 지정

다음 요구 사항 중 하나가 있는 경우 시작 템플릿의 ImageId 필드에 AMI ID를 지정합니다. 추가 정보를 보려면 요구 사항을 선택하세요.

Amazon EKS 최적화 Linux/Bottlerocket AMI에 포함된 **bootstrap.sh** 파일에 인수를 전달하기 위해 사용자 데이터를 제공합니다

부트스트래핑은 인스턴스가 시작될 때 실행할 수 있는 명령의 추가에 대해 설명하는 데 사용되는 용어입니다. 예를 들어, 부트스트랩을 사용하면 추가 [kubelet](#) 인수를 사용할 수 있습니다. 시작 템플릿을 지정하지 않고 eksctl을 사용하여 bootstrap.sh 스크립트에 인수를 전달할 수 있습니다. 이를 위해 시작 템플릿의 사용자 데이터 부분에 정보를 지정할 수도 있습니다.

시작 템플릿을 지정하지 않고 eksctl

다음 내용으로 *my-nodegroup.yaml*이라는 파일을 만듭니다. 모든 **##** **#**을 자신의 값으로 바꾸세요. --apiserver-endpoint, --b64-cluster-ca 및 --dns-cluster-ip 인수는 선택 사항입니다. 그러나 이를 정의하면 bootstrap.sh 스크립트가 describeCluster를 호출하는 것을 방지할 수 있습니다. 이는 노드를 자주 확장하고 축소하는 프라이빗 클러스터 설정이나 클러스터에서 유용합니다. bootstrap.sh 스크립트에 대한 자세한 내용은 GitHub의 [bootstrap.sh](#) 파일을 참조하세요.

- 유일한 필수 인수는 클러스터 이름(*my-cluster*)입니다.
- ami-*1234567890abcdef0*에 대해 최적화된 AMI ID를 검색하려면 다음 섹션을 참조하세요.
 - [권장 Amazon Linux AMI ID 검색](#)
 - [권장 Bottlerocket AMI ID 검색](#)
 - [권장 Microsoft Windows AMI ID 검색](#)
- 클러스터의 *certificate-authority*를 검색하려면 다음 명령을 실행합니다.

```
aws eks describe-cluster --query "cluster.certificateAuthority.data" --output text
--name my-cluster --region region-code
```

- 클러스터의 *api-server-endpoint*를 검색하려면 다음 명령을 실행합니다.

```
aws eks describe-cluster --query "cluster.endpoint" --output text --name my-
cluster --region region-code
```

- --dns-cluster-ip의 값은 .10으로 끝나는 서비스 CIDR입니다. 클러스터의 *service-cidr*를 검색하려면 다음 명령을 실행합니다. 예를 들어, 반환된 값이 ipv4 10.100.0.0/16인 경우 값은 *10.100.0.10*입니다.

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr"
--output text --name my-cluster --region region-code
```

- 이 예에서는 추가 kubelet 인수를 제공하여 Amazon EKS 최적화 AMI에 포함된 bootstrap.sh 스크립트로 사용자 지정 max-pods 값을 설정합니다. 노드 그룹 이름은 63자를 초과할 수 없습니다. 문자나 숫자로 시작하되, 나머지 문자의 경우 하이픈과 밑줄을 포함할 수 있습니다. **my-max-pods-value**를 선택하는 방법에 대한 도움말은 [the section called “각 Amazon EC2 인스턴스 유형의 Amazon EKS 권장 최대 포트 수”](#)을 참조하세요.

```
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code

managedNodeGroups:
- name: my-nodegroup
  ami: ami-1234567890abcdef0
  instanceType: m5.large
  privateNetworking: true
  disableIMDSv1: true
  labels: { x86-a12-specified-mng }
  overrideBootstrapCommand: |
    #!/bin/bash
    /etc/eks/bootstrap.sh my-cluster \
      --b64-cluster-ca certificate-authority \
      --apiserver-endpoint api-server-endpoint \
      --dns-cluster-ip service-cidr.10 \
      --kubelet-extra-args '--max-pods=my-max-pods-value' \
      --use-max-pods false
```

사용 가능한 모든 eksctl config 파일 옵션은 eksctl 설명서의 [구성 파일 스키마](#)를 참조하세요. eksctl 유틸리티는 여전히 사용자를 위해 시작 템플릿을 생성하고 해당 사용자 데이터를 config 파일에서 제공되는 데이터로 채웁니다.

다음 명령을 사용하여 노드 그룹을 생성합니다.

```
eksctl create nodegroup --config-file=my-nodegroup.yaml
```


시작 템플릿의 사용자 데이터

시작 템플릿의 사용자 데이터 부분에서 다음 정보를 지정합니다. 모든 **## #**을 자신의 값으로 바꾸세요. `--apiserver-endpoint`, `--b64-cluster-ca` 및 `--dns-cluster-ip` 인수는 선택 사항입니다. 그러나 이를 정의하면 `bootstrap.sh` 스크립트가 `describeCluster`를 호출하는 것을 방지할 수 있습니다. 이는 노드를 자주 확장하고 축소하는 프라이빗 클러스터 설정이나 클러스터에서 유용합니다. `bootstrap.sh` 스크립트에 대한 자세한 내용은 GitHub의 [bootstrap.sh](#) 파일을 참조하세요.

- 유일한 필수 인수는 클러스터 이름(*my-cluster*)입니다.
- 클러스터의 *certificate-authority*를 검색하려면 다음 명령을 실행합니다.

```
aws eks describe-cluster --query "cluster.certificateAuthority.data" --output text
--name my-cluster --region region-code
```

- 클러스터의 *api-server-endpoint*를 검색하려면 다음 명령을 실행합니다.

```
aws eks describe-cluster --query "cluster.endpoint" --output text --name my-
cluster --region region-code
```

- `--dns-cluster-ip`의 값은 .10으로 끝나는 서비스 CIDR입니다. 클러스터의 *service-cidr*를 검색하려면 다음 명령을 실행합니다. 예를 들어, 반환된 값이 ipv4 10.100.0.0/16인 경우 값은 *10.100.0.10*입니다.

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr"
--output text --name my-cluster --region region-code
```

- 이 예에서는 추가 kubelet 인수를 제공하여 Amazon EKS 최적화 AMI에 포함된 `bootstrap.sh` 스크립트로 사용자 지정 `max-pods` 값을 설정합니다. *my-max-pods-value*를 선택하는 방법에 대한 도움말은 [the section called “각 Amazon EC2 인스턴스 유형의 Amazon EKS 권장 최대 포드 수”](#)을 참조하세요.

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=="MYBOUNDARY=="

--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"

#!/bin/bash
set -ex
/etc/eks/bootstrap.sh my-cluster \
```

```
--b64-cluster-ca certificate-authority \
--apiserver-endpoint api-server-endpoint \
--dns-cluster-ip service-cidr.10 \
--kubelet-extra-args '--max-pods=my-max-pods-value' \
--use-max-pods false

---MYBOUNDARY---
```

Amazon EKS 최적화 Windows AMI에 포함된 **Start-EKSBootstrap.ps1** 파일에 인수를 전달하기 위해 사용자 데이터를 제공함

부트스트래핑은 인스턴스가 시작될 때 실행할 수 있는 명령의 추가에 대해 설명하는 데 사용되는 용어입니다. 시작 템플릿을 지정하지 않고 eksctl을 사용하여 Start-EKSBootstrap.ps1 스크립트에 인수를 전달할 수 있습니다. 이를 위해 시작 템플릿의 사용자 데이터 부분에 정보를 지정할 수도 있습니다.

사용자 지정 Windows AMI ID를 지정하려면 다음과 같은 사항을 고려하세요.

- 시작 템플릿을 사용하고 사용자 데이터 부분의 필수 부트스트랩 명령을 제공해야 합니다. 원하는 Windows ID를 검색하려면 [최적화된 Windows AMI가 있는 노드 생성](#) 테이블을 사용할 수 있습니다.
- 몇 가지 제한과 조건이 있습니다. 예를 들어, AWS IAM Authenticator 구성 맵에 eks:kube-proxy-windows를 추가해야 합니다. 자세한 내용은 [the section called “AMI ID 지정 시 제한과 조건”](#) 섹션을 참조하세요.

시작 템플릿의 사용자 데이터 부분에서 다음 정보를 지정합니다. 모든 **## #**을 자신의 값으로 바꾸세요. -APIServerEndpoint, -Base64ClusterCA 및 -DNSClusterIP 인수는 선택 사항입니다. 그러나 이를 정의하면 Start-EKSBootstrap.ps1 스크립트가 describeCluster를 호출하는 것을 방지할 수 있습니다.

- 유일한 필수 인수는 클러스터 이름(*my-cluster*)입니다.
- 클러스터의 *certificate-authority*를 검색하려면 다음 명령을 실행합니다.

```
aws eks describe-cluster --query "cluster.certificateAuthority.data" --output text --
name my-cluster --region region-code
```

- 클러스터의 *api-server-endpoint*를 검색하려면 다음 명령을 실행합니다.

```
aws eks describe-cluster --query "cluster.endpoint" --output text --name my-cluster
--region region-code
```

- --dns-cluster-ip의 값은 .10으로 끝나는 서비스 CIDR입니다. 클러스터의 *service-cidr*를 검색하려면 다음 명령을 실행합니다. 예를 들어, 반환된 값이 ipv4 10.100.0.0/16인 경우 값은 *10.100.0.10*입니다.

```
aws eks describe-cluster --query "cluster.kubernetesNetworkConfig.serviceIpv4Cidr" --
output text --name my-cluster --region region-code
```

- 추가 인수는 [the section called “부트스트랩 스크립트 구성 파라미터”](#) 부분을 참조하세요.

Note

사용자 지정 서비스 CIDR을 사용하는 경우 -ServiceCIDR 파라미터를 사용하여 CIDR을 지정해야 합니다. 그렇지 않으면 클러스터의 포드에 대한 DNS 확인이 실패합니다.

```
<powershell>
[string]$EKSBootstrapScriptFile = "$env:ProgramFiles\Amazon\EKS\Start-EKSBootstrap.ps1"
& $EKSBootstrapScriptFile -EKSClusterName my-cluster `
  -Base64ClusterCA certificate-authority `
  -APIServerEndpoint api-server-endpoint `
  -DNSClusterIP service-cidr.10
</powershell>
```

특정 보안, 규정 준수 또는 내부 정책 요구 사항으로 인해 사용자 정의 AMI 실행함

자세한 내용은 Amazon EC2 사용 설명서에서 [Amazon Machine Image\(AMI\)](#)를 참조하세요. Amazon EKS AMI 빌드 사양에는 Amazon Linux를 기반으로 사용자 지정 Amazon EKS AMI를 구축하기 위한 리소스 및 구성 스크립트가 포함되어 있습니다. 자세한 내용은 GitHub에서 [Amazon EKS AMI 빌드 사양](#) 섹션을 참조하세요. 다른 운영 체제와 함께 설치된 사용자 지정 AMI를 생성하려면 GitHub에서 [Amazon EKS 샘플 사용자 지정 AMI](#) 섹션을 참조하세요.

관리형 노드 그룹에 사용되는 시작 템플릿에서는 AMI ID에 동적 파라미터 참조를 사용할 수 없습니다.

⚠ Important

AMI를 지정할 때 Amazon EKS는 사용자 데이터를 병합하지 않습니다. 사용자가 필요한 `bootstrap` 명령을 사용하여 노드를 클러스터에 조인해야 합니다. 노드가 클러스터에 조인되지 않은 경우 Amazon EKS `CreateNodegroup` 및 `UpdateNodegroupVersion` 작업도 실패합니다.

AMI ID 지정 시 제한과 조건

다음은 관리형 노드 그룹으로 AMI ID를 지정하는 것과 관련된 제한 및 조건입니다.

- 시작 템플릿에서 AMI ID를 지정하는 것과 AMI ID를 지정하지 않는 것 사이를 전환하려면 새 노드 그룹을 생성해야 합니다.
- 최신 AMI 버전을 사용할 수 있는 경우에도 콘솔에 알림이 표시되지 않습니다. 노드 그룹을 최신 AMI 버전으로 업데이트하려면 업데이트된 AMI ID로 시작 템플릿의 새 버전을 생성해야 합니다. 그런 다음 새 시작 템플릿 버전으로 노드 그룹을 업데이트해야 합니다.
- AMI ID를 지정하는 경우 API에서 다음 필드를 설정할 수 없습니다.
 - `amiType`
 - `releaseVersion`
 - `version`
- AMI ID를 지정하는 경우 API에 설정된 모든 `taints`가 비동기적으로 적용됩니다. 노드가 클러스터에 합류하기 전에 테인트를 적용하려면 `kubelet` 명령줄 플래그를 사용하여 사용자 데이터의 `--register-with-taints`에 테인트를 전달해야 합니다. 자세한 내용은 Kubernetes 문서의 [kubelet](#)를 참조하세요.
- Windows 관리형 노드 그룹에 사용자 지정 AMI ID를 지정할 때는 AWS IAM Authenticator 구성 맵에 `eks:kube-proxy-windows`를 추가합니다. DNS가 제대로 작동하려면 필요합니다.
 - 편집을 위해 AWS IAM Authenticator 구성 맵을 엽니다.

```
kubectl edit -n kube-system cm aws-auth
```

- Windows 노드와 연결된 각 `rolearn` 아래의 `groups` 목록에 이 항목을 추가합니다. 구성 맵은 [aws-auth-cm-windows.yaml](#)과 비슷해야 합니다.

```
- eks:kube-proxy-windows
```

- 파일을 저장하고 텍스트 편집기를 종료합니다.

클러스터에서 관리형 노드 그룹 삭제

이 주제에서는 Amazon EKS 관리형 노드 그룹을 삭제하는 방법에 대해 설명합니다. 관리형 노드 그룹을 삭제하면 Amazon EKS는 먼저 Auto Scaling 그룹의 최소, 최대 및 원하는 크기를 0으로 설정합니다. 그러면 노드 그룹이 축소됩니다.

각 인스턴스가 종료되기 전에 Amazon EKS는 신호를 보내 해당 노드에서 포드를 드레이닝합니다. 몇 분 후에 포드가 드레이닝되지 않을 경우 Amazon EKS는 오토 스케일링이 인스턴스 종료를 계속 진행할 수 있도록 합니다. 모든 인스턴스가 종료되면 Auto Scaling 그룹이 삭제됩니다.

Important

클러스터의 다른 관리형 노드 그룹에서 사용하지 않는 노드 IAM 역할을 사용하는 관리형 노드 그룹을 삭제하면 역할이 aws-auth ConfigMap에서 제거됩니다. 클러스터의 자체 관리형 노드 그룹이 동일한 노드 IAM 역할을 사용하는 경우 자체 관리형 노드는 NotReady 상태로 전환됩니다. 또한 클러스터 작업도 중단됩니다. 자체 관리 노드 그룹에만 사용하는 역할에 대한 매핑을 추가하려면 클러스터의 플랫폼 버전이 [EKS 액세스 항목으로 IAM 사용자에게 Kubernetes 액세스 권한 부여](#)의 전제 조건 섹션에 나열된 최소 버전 이상인 경우 [the section called “액세스 항목 생성”](#)를 참조하세요. 플랫폼 버전이 액세스 항목에 필요한 최소 버전보다 이전인 경우 항목을 aws-auth ConfigMap에 다시 추가할 수 있습니다. 자세한 내용을 보려면 터미널에 `eksctl create iamidentitymapping --help`를 입력합니다.

관리되는 노드 그룹은 다음을 사용하여 삭제할 수 있습니다:

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)
- [the section called “AWS CLI”](#)

eksctl

eksctl로 관리되는 노드 그룹 삭제

다음 명령을 입력합니다. 모든 **##** **#**을 자신의 값으로 바꾸세요.

```
eksctl delete nodegroup \
  --cluster my-cluster \
  --name my-mng \
```

```
--region region-code
```

추가 옵션은 eksctl 설명서의 [Deleting and draining nodegroups](#)를 참조하세요.

AWS Management Console

AWS Management Console로 관리되는 노드 그룹 삭제

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 클러스터 페이지에서 삭제할 노드 그룹이 포함된 클러스터를 선택합니다.
3. 선택한 클러스터 페이지에서 컴퓨팅 탭을 선택합니다.
4. 노드 그룹 섹션에서 삭제할 노드 그룹을 선택합니다. 그런 다음 삭제를 선택합니다.
5. 노드 그룹 삭제 확인 대화 상자에서 노드 그룹의 이름을 입력합니다. 그런 다음 삭제를 선택합니다.

AWS CLI

AWS CLI을 사용하여 관리형 노드 그룹을 삭제

1. 다음 명령을 입력합니다. 모든 **##** **#**을 자신의 값으로 바꾸세요.

```
aws eks delete-nodegroup \
  --cluster-name my-cluster \
  --nodegroup-name my-mng \
  --region region-code
```

2. 키보드의 화살표 키를 사용하여 응답 출력을 스크롤합니다. 작업을 마치면 q 키를 누릅니다.

자세한 옵션은 AWS CLI 명령 참조에서 [delete-nodegroup](#) 명령을 참조하세요.

자체 관리형 노드로 노드 직접 유지

클러스터에는 포드가 예약된 하나 이상의 Amazon EC2 노드가 포함되어 있습니다. Amazon EKS 노드는 AWS 계정에서 실행되고 클러스터 API 서버 엔드포인트를 통해 클러스터의 제어 영역에 연결됩니다. Amazon EC2 가격을 기준으로 요금이 청구됩니다. 자세한 설명은 [Amazon EC2 요금](#)을 참조하세요.

클러스터에는 여러 노드 그룹이 포함될 수 있습니다. 각 노드 그룹에는 [Amazon EC2 Auto Scaling 그룹](#)에 배포된 하나 이상의 노드가 포함되어 있습니다. 그룹 내 노드의 인스턴스 유형은 [Karpenter](#)를 사

용한 [attribute-based 인스턴스 유형 선택](#)을 사용하는 경우와 같이 달라질 수 있습니다. 노드 그룹의 모든 인스턴스는 [Amazon EKS 노드 IAM 역할](#)을 사용해야 합니다.

Amazon EKS에서는 Amazon EKS 최적화 AMI라는 특별한 Amazon Machine Image(AMI)를 제공합니다. AMI는 Amazon EKS와 함께 작동하도록 구성되어 있습니다. 구성 요소에는 containerd, kubelet, AWS IAM 인증자가 포함됩니다. 또한 AMI에는 특별한 [부트스트랩 스크립트](#)도 포함되어 있어 클러스터의 제어 영역을 자동으로 찾고 연결할 수 있습니다.

CIDR 블록을 사용하여 클러스터의 퍼블릭 엔드포인트에 대한 액세스를 제한하는 경우 프라이빗 엔드포인트 액세스도 사용 설정하는 것이 좋습니다. 이는 노드가 클러스터와 통신할 수 있도록 하기 위한 것입니다. 프라이빗 엔드포인트를 활성화하지 않은 경우, 퍼블릭 액세스에 대해 지정하는 CIDR 블록에는 VPC의 송신 소스가 포함되어야 합니다. 자세한 내용은 [the section called “클러스터 엔드포인트 액세스”](#) 섹션을 참조하세요.

Amazon EKS 클러스터에 자체 관리형 노드를 추가하려면 다음 항목을 참조하세요. 자체 관리형 노드를 수동으로 시작하는 경우 <cluster-name>이 클러스터와 일치하는지 확인하면서 각 노드에 다음 태그를 추가합니다. 자세한 내용은 [개별 리소스에 대한 태그 추가 및 삭제](#)를 참조하세요. 가이드의 단계를 수행하면 필수 태그가 자동으로 노드에 추가됩니다.

키	값
kubernetes.io/cluster/<cluster-name>	owned

Important

Amazon EC2 인스턴스 메타데이터 서비스(IMDS)의 태그는 EKS 노드와 호환되지 않습니다. 인스턴스 메타데이터 태그가 활성화되면 태그 값에 슬래시(/)를 사용할 수 없습니다. 이러한 제한으로 인해 인스턴스 시작이 실패할 수 있습니다. 특히 Karpenter 또는 Cluster Autoscaler와 같은 노드 관리 도구를 사용하는 경우 적절한 기능을 위해 슬래시가 포함된 태그에 의존하기 때문입니다.

일반 Kubernetes 관점에서 노드에 대한 자세한 내용은 Kubernetes 문서의 [노드](#)를 참조하세요.

주제

- [자체 관리형 Amazon Linux 노드 생성](#)

- [자체 관리형 Bottlerocket 노드 생성](#)
- [자체 관리형 Microsoft Windows 노드 생성](#)
- [자체 관리형 Ubuntu Linux 노드 생성](#)
- [클러스터의 자체 관리형 노드 업데이트](#)

자체 관리형 Amazon Linux 노드 생성

이 주제에서는 Amazon EKS 클러스터에 등록하는 Linux 노드의 Auto Scaling 그룹을 시작하는 방법을 설명합니다. 노드가 클러스터에 조인한 이후 Kubernetes 애플리케이션을 배포할 수 있습니다. `eksctl` 또는 AWS Management Console을 사용하여 자체 관리형 Amazon Linux 노드를 시작할 수도 있습니다. AWS Outposts에서 노드를 시작해야 하는 경우 [the section called “노드”](#) 부분을 참조하세요.

- 기존 Amazon EKS 클러스터. 배포하려면 [the section called “클러스터 생성”](#) 섹션을 참조하세요. AWS Outposts, AWS Wavelength 또는 AWS Local Zones이 사용된 AWS 리전에 서브넷이 있는 경우 클러스터를 생성할 때 해당 서브넷이 전달되지 않은 상태여야 합니다.
- 노드가 사용할 기존 IAM 역할. 파일을 만들려면 [the section called “노드 IAM 역할”](#) 섹션을 참조하세요. 이 역할에 VPC CNI에 대한 정책 중 하나도 없는 경우 VPC CNI 포트에 대해 다음과 같은 별도의 역할이 필요합니다.
- (선택 사항이지만 권장됨) 필요한 IAM 정책이 연결된 자체 IAM 역할로 구성된 Kubernetes용 Amazon VPC CNI 플러그인 추가 기능. 자세한 내용은 [the section called “IRSA에 대한 구성”](#) 섹션을 참조하세요.
- [최적의 Amazon EC2 노드 인스턴스 유형 선택](#)에 나열된 고려 사항에 익숙합니다. 선택하는 인스턴스 유형에 따라 클러스터 및 VPC에 대한 추가 전제 조건이 있을 수 있습니다.

다음 중 하나를 사용하여 자체 관리형 Linux 노드를 시작할 수 있습니다.

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)

eksctl

`eksctl`을 사용하여 자체 관리형 Linux 노드 시작

1. 장치에 설치된 `eksctl` 명령줄 도구의 버전 0.210.0 이상 또는 AWS CloudShell이 필요합니다. `eksctl`을 설치 또는 업그레이드하려면 `eksctl` 설명서에서 [설치](#)를 참조하세요.

2. (선택 사항) AmazonEKS_CNI_Policy 관리형 IAM 정책이 [Amazon EKS 노드 IAM 역할](#)에 연결되어 있는 경우 대신 Kubernetes aws-node 서비스 계정에 연결한 IAM 역할에 할당하는 것이 좋습니다. 자세한 내용은 [the section called “IRSA에 대한 구성”](#) 섹션을 참조하세요.
3. 다음 명령은 기존 클러스터의 노드 그룹을 생성합니다. *al-nodes*를 노드 그룹의 이름으로 바꿉니다. 노드 그룹 이름은 63자를 초과할 수 없습니다. 문자나 숫자로 시작하되, 나머지 문자의 경우 하이픈과 밑줄을 포함할 수 있습니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다. 나머지 *##* *#*을 자신의 값으로 바꿉니다. 노드는 기본적으로 제어 영역과 동일한 Kubernetes 버전으로 작성됩니다.

--node-type에 대한 값을 선택하기 전에 [최적의 Amazon EC2 노드 인스턴스 유형 선택](#)을 검토합니다.

*my-key*를 Amazon EC2 키 페어 또는 퍼블릭 키 이름으로 바꿉니다. 이 키는 노드를 시작한 후 SSH로 연결하는 데 사용됩니다. Amazon EC2 키 페어가 아직 없는 경우 AWS Management Console에서 새로 생성할 수 있습니다. 자세한 내용을 알아보려면 Amazon EC2 key pairs의 [Amazon EC2 키 페어](#)를 참조하세요.

다음 명령을 사용하여 노드 그룹을 생성합니다.

Important

노드 그룹을 AWS Outposts, Wavelength 또는 Local Zones 서브넷에 배포하려는 경우 추가 고려 사항이 있습니다.

- 클러스터를 생성할 때 서브넷이 전달되지 않았어야 합니다.
- 서브넷과 `volumeType`: gp2을 지정하는 구성 파일로 노드 그룹을 생성해야 합니다. 자세한 내용은 eksctl 문서의 [구성 파일을 사용하여 nodegroup 생성](#) 및 [구성 파일 스키마](#) 부분을 참조하세요.

```
eksctl create nodegroup \
  --cluster my-cluster \
  --name al-nodes \
  --node-type t3.medium \
  --nodes 3 \
  --nodes-min 1 \
  --nodes-max 4 \
```

```
--ssh-access \
--managed=false \
--ssh-public-key my-key
```

다음과 같은 노드 그룹을 배포할 수 있습니다.

- 포드에 기본 구성보다 훨씬 더 많은 수의 IP 주소를 할당할 수 있는 노드 그룹을 배포하려면 [the section called “IP 주소 증가”](#) 섹션을 참조하세요.
- 인스턴스와 다른 CIDR 블록의 포드에 IPv4 주소를 할당할 수 있는 노드 그룹을 배포하려면 [the section called “사용자 지정 네트워킹”](#) 섹션을 참조하세요.
- 포드 및 서비스에 IPv6 주소를 할당할 수 있는 노드 그룹을 배포하려면 [the section called “IPv6”](#) 섹션을 참조하세요.
- 아웃바운드 인터넷 액세스가 없는 노드 그룹을 배포하려면 [the section called “프라이빗 클러스터”](#) 부분을 참조하세요.

사용 가능한 모든 옵션 및 기본값의 전체 목록을 보려면 다음 명령을 입력합니다.

```
eksctl create nodegroup --help
```

노드가 클러스터에 조인하지 못한 경우 문제 해결 장의 [the section called “노드가 클러스터 조인에 실패”](#) 섹션을 참조하세요.

예제 출력은 다음과 같습니다. 노드가 생성되는 동안 여러 줄이 출력됩니다. 출력의 마지막 줄 중 하나는 다음 예제 줄과 유사합니다.

```
[#] created 1 nodegroup(s) in cluster "my-cluster"
```

4. (선택 사항) [샘플 애플리케이션](#)을 배포하여 클러스터 및 Linux 노드를 테스트합니다.
5. 다음과 같은 조건에 해당하면 IMDS에 대한 포드 액세스를 차단하는 것이 좋습니다.
 - 포드에 필요한 최소 권한만 있도록 모든 Kubernetes 서비스 계정에 IAM 역할을 할당할 계획입니다.
 - 클러스터의 어떤 포드도 현재 AWS 리전 검색 등의 다른 이유로 Amazon EC2 인스턴스 메타데이터 서비스(IMDS)에 액세스할 필요가 없습니다.

자세한 내용은 [작업자 노드에 할당된 인스턴스 프로필에 대한 액세스 제한](#) 부분을 참조하세요.

AWS Management Console

1단계: AWS Management Console을 사용하여 자체 관리형 Linux 노드 시작

1. AWS CloudFormation 템플릿의 최신 버전 다운로드

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2022-12-23/  
amazon-eks-nodegroup.yaml
```

2. 클러스터 상태가 ACTIVE가 되기를 기다립니다. 클러스터가 활성화되기 전에 노드를 시작하면 노드가 클러스터에 등록되지 않고 노드를 다시 시작해야 합니다.
3. [AWS CloudFormation 콘솔](#)을 엽니다.
4. 스택 생성을 선택한 다음 새 리소스 사용(표준)을 선택합니다.
5. 템플릿 지정에서 템플릿 파일 업로드를 선택한 다음 파일 선택을 선택합니다.
6. 다운로드한 amazon-eks-nodegroup.yaml 파일을 선택합니다.
7. 다음을 선택합니다.
8. 스택 세부 정보 지정 페이지에서 이에 따라 다음 파라미터를 입력한 후 다음을 선택합니다.
 - 스택 이름: AWS CloudFormation 스택에 대한 스택 이름을 선택합니다. 예를 들어, **my-cluster-nodes**로 지정할 수 있습니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다.
 - ClusterName: Amazon EKS 클러스터 생성 시 사용할 이름을 입력합니다. 이 이름은 클러스터 이름과 같아야 합니다. 그렇지 않으면 노드가 클러스터에 조인할 수 없습니다.
 - ClusterControlPlaneSecurityGroup: [VPC](#)를 생성할 때 생성한 AWS CloudFormation 출력에서 SecurityGroups값을 선택합니다.

다음 단계에서는 해당 그룹을 검색하는 한 가지 작업을 보여줍니다.

- a. [Amazon EKS 콘솔](#)을 엽니다.
- b. 클러스터의 이름을 선택합니다.
- c. 네트워킹 탭을 선택합니다.
- d. ClusterControlPlaneSecurityGroup 드롭다운 목록에서 선택할 때 추가 보안 그룹 값을 참조로 사용하세요.
- NodeGroupName: 노드 그룹의 이름을 입력합니다. 이 이름은 나중에 노드에 대해 생성된 Auto Scaling 노드 그룹을 식별하는 데 사용할 수 있습니다. 노드 그룹 이름은 63자를 초과할 수 없습니다. 문자나 숫자로 시작하되, 나머지 문자의 경우 하이픈과 밑줄을 포함할 수 있습니다.

- `NodeAutoScalingGroupMinSize`: Auto Scaling 그룹이 축소할 수 있는 노드의 최소 노드 수를 입력합니다.
- `NodeAutoScalingGroupDesiredCapacity`: 스택을 생성할 때 조정할 원하는 노드 수를 입력합니다.
- `NodeAutoScalingGroupMaxSize`: Auto Scaling 그룹이 확장할 수 있는 노드의 최대 노드 수를 입력합니다.
- `NodeInstanceType`: 노드에 대한 인스턴스 유형을 선택합니다. 자세한 내용은 [the section called “Amazon EC2 인스턴스 유형”](#) 섹션을 참조하세요.
- `NodeImageIdSSMParam`: 가변 Kubernetes 버전용 최신 Amazon EKS 최적화 AMI의 Amazon EC2 Systems Manager 파라미터로 미리 채워집니다. Amazon EKS에서 지원되는 다른 Kubernetes 마이너 버전을 사용하려면 **1.XX**를 다른 [지원되는 버전](#)으로 바꿉니다. 클러스터와 동일한 Kubernetes 버전을 지정하는 것이 좋습니다.

amazon-linux-2를 다른 AMI 유형으로 바꿀 수도 있습니다. 자세한 내용은 [the section called “최신 ID 받기”](#) 섹션을 참조하세요.

Note

Amazon EKS 노드 AMI는 Amazon Linux를 기반으로 합니다. [Amazon Linux 보안 센터](#)에서 Amazon Linux 2의 보안 또는 프라이버시 이벤트를 추적하거나 관련 [RSS 피드](#)를 구독하세요. 보안 및 프라이버시 이벤트에는 문제의 개요, 영향을 받는 패키지 및 인스턴스를 업데이트하여 문제를 해결하는 방법이 포함됩니다.

- `NodeImageId`: (선택사항) Amazon EKS 최적화 AMI 대신 사용자 정의 AMI를 사용 중인 경우, 해당 AWS 리전에 노드 AMI ID를 입력합니다. 여기에서 값을 지정하면 `NodeImageIdSSMParam` 필드에 있는 모든 값이 재정의됩니다.
- `NodeVolumeSize`: 노드에 대해 루트 볼륨 크기를 GiB 단위로 지정합니다.
- `NodeVolumeType`: 노드의 루트 볼륨 유형을 지정합니다.
- `KeyName`: 시작 이후 SSH를 사용하여 노드에 연결하는 데 사용할 수 있는 Amazon EC2 SSH 키 페어 이름을 입력합니다. Amazon EC2 키 페어가 아직 없는 경우 AWS Management Console에서 새로 생성할 수 있습니다. 자세한 내용을 알아보려면 Amazon EC2 key pairs의 [Amazon EC2 키 페어](#)를 참조하세요.

Note

여기에 키 페어를 입력하지 않으면 AWS CloudFormation 스택이 생성되지 않습니다.

- **BootstrapArguments:** 별도의 kubelet 인수와 같이 노드 부트스트랩 스크립트에 전달할 선택적 인수를 지정합니다. 자세한 내용은 GitHub의 [부트스트랩 스크립트 사용 정보](#)를 참조하십시오.

다음과 같은 노드 그룹을 배포할 수 있습니다.

- 포드에 기본 구성보다 훨씬 더 많은 수의 IP 주소를 할당할 수 있는 노드 그룹을 배포하려면 [the section called “IP 주소 증가”](#) 섹션을 참조하세요.
- 인스턴스와 다른 CIDR 블록의 포드에 IPv4 주소를 할당할 수 있는 노드 그룹을 배포하려면 [the section called “사용자 지정 네트워킹”](#) 섹션을 참조하세요.
- 포드 및 서비스에 IPv6 주소를 할당할 수 있는 노드 그룹을 배포하려면 [the section called “IPv6”](#) 섹션을 참조하세요.
- 아웃바운드 인터넷 액세스가 없는 노드 그룹을 배포하려면 [the section called “프라이빗 클러스터”](#) 부분을 참조하세요.
- **DisableIMDSv1:** 기본적으로 각 노드는 인스턴스 메타데이터 서비스 버전 1(IMDSv1) 및 IMDSv2를 지원합니다. IMDSv1을 사용 중지할 수 있습니다. 향후 노드 그룹의 노드와 포드가 IMDSv1을 사용하지 못하게 하려면 `DisableIMDSv1`을 `true`로 설정합니다. IMDS에 대한 자세한 내용은 [인스턴스 메타데이터 서비스 구성](#)을 참조하세요. 노드의 IMDS 액세스 제한에 대한 자세한 내용은 [작업자 노드에 할당된 인스턴스 프로필에 대한 액세스 제한](#)을 참조하세요.
- **VpcId:** 생성한 [VPC](#)에 대한 ID를 입력합니다.
- **서브넷:** VPC용으로 생성한 서브넷을 선택합니다. [Amazon EKS 클러스터를 위한 Amazon VPC 만들기](#)에 설명된 단계를 사용하여 VPC를 만든 경우, 노드가 시작될 VPC 내의 사설 서브넷만 지정하세요. 클러스터의 Networking에서 각 서브넷 링크를 열면 어떤 서브넷이 프라이빗인지 확인할 수 있습니다.

Important

- 서브넷 중 하나가 퍼블릭 서브넷인 경우 자동 퍼블릭 IP 주소 할당 설정을 사용하도록 설정해야 합니다. 퍼블릭 서브넷에 대해 이 설정을 사용하지 않으면 해당 퍼블릭 서브넷에 배포하는 노드에는 퍼블릭 IP 주소가 할당되지 않으며 클러스터 또는 기타 AWS 서비스와 통신할 수 없습니다. 서브넷이 2020년 3월 26일 이전에 [Amazon EKS AWS CloudFormation VPC 템플릿](#) 또는 `eksctl`을 사용하여 배포된 경우 퍼블릭 서브넷에 대해 자동 퍼블릭 IP 주소 할당이 사용 중지됩니다. 서브넷에 퍼블릭 IP 주소 할당을 활성화하는 방법에 대한 자세한 내용은 [서브넷에 대한 퍼블릭 IPv4 주소 지정 속성 수정](#)을 참조하세요. 노드가 프라이빗 서브넷에 배포되면 NAT 게이트웨이를 통해 클러스터 및 다른 AWS 서비스와 통신할 수 있습니다.

- 서브넷에서 인터넷에 액세스할 수 없는 경우 [인터넷 액세스가 제한된 비공개 클러스터 배포하기](#)의 고려 사항과 추가 단계를 숙지하고 있는지 확인하세요.
- AWS Outposts, Wavelength 또는 Local Zones 서브넷을 선택하는 경우 클러스터를 생성할 때 해당 서브넷이 전달되지 않은 상태여야 합니다.

9. 스택 옵션 구성 페이지에서 원하는 선택 항목을 선택한 후 다음을 선택합니다.

10. AWS CloudFormation에서 IAM 리소스를 생성할 수 있음을 인정합니다의 왼쪽에 있는 확인란을 선택한 다음 스택 생성을 선택합니다.

11. 스택이 생성된 후 콘솔에서 이를 선택하고 출력을 선택합니다.

12. 생성된 노드 그룹에 대해 NodeInstanceRole을 기록합니다. Amazon EKS 노드를 구성할 때 필요합니다.

2단계: 노드가 클러스터에 조인하도록 하려면

Note

아웃바운드 인터넷 액세스 없이 프라이빗 VPC 내에서 노드를 시작한 경우 VPC 내에서 노드가 클러스터에 조인하도록 설정해야 합니다.

1. aws-auth ConfigMap이 이미 있는지 확인합니다.

```
kubectl describe configmap -n kube-system aws-auth
```

2. aws-auth ConfigMap이 표시되면 필요에 따라 이를 업데이트합니다.

a. 편집을 위해 ConfigMap을 엽니다.

```
kubectl edit -n kube-system configmap/aws-auth
```

b. 필요에 따라 새 mapRoles 항목을 추가합니다. rolearn 값을 이전 절차에서 기록한 NodeInstanceRole 값으로 설정합니다.

```
[...]
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
```

```
groups:
  - system:bootstrappers
  - system:nodes
[...]
```

c. 파일을 저장하고 텍스트 편집기를 종료합니다.

3. "Error from server (NotFound): configmaps "aws-auth" not found라는 오류 메시지가 표시되면 스텝 ConfigMap을 적용합니다.

a. 구성 맵을 다운로드합니다.

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
aws-auth-cm.yaml
```

b. aws-auth-cm.yaml 파일에서 rolearn 값을 이전 절차에서 기록한 NodeInstanceRole 값으로 설정합니다. 이 작업은 텍스트 편집기를 사용하거나 *my-node-instance-role*을 대체하고 다음 명령을 실행하여 수행할 수 있습니다.

```
sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-
role|' aws-auth-cm.yaml
```

c. 구성을 적용합니다. 이 명령을 완료하는 데 몇 분이 걸릴 수 있습니다.

```
kubectl apply -f aws-auth-cm.yaml
```

4. 노드의 상태를 확인하고 Ready 상태가 될 때까지 대기합니다.

```
kubectl get nodes --watch
```

Ctrl+C를 입력하여 셸 프롬프트로 돌아갑니다.

Note

권한 부여 또는 리소스 유형 오류가 표시되는 경우 문제 해결 주제의 [the section called “권한이 없거나 액세스가 거부됨\(kubectl\)”](#) 부분을 참조하세요.

노드가 클러스터에 조인하지 못한 경우 문제 해결 장의 [the section called “노드가 클러스터 조인에 실패”](#) 섹션을 참조하세요.

5. (GPU 노드만 해당) GPU 인스턴스 유형과 Amazon EKS 최적화 가속 AMI를 선택한 경우 클러스터에 [Kubernetes용 NVIDIA 디바이스 플러그인](#)을 DaemonSet으로 적용해야 합니다. 다음 명령을 실행하기 전에 `vX.X.X`를 원하는 [NVIDIA/k8s-device-plugin](#) 버전으로 바꿉니다.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/deployments/static/nvidia-device-plugin.yml
```

3단계: 추가 작업

1. (선택 사항) [샘플 애플리케이션](#)을 배포하여 클러스터 및 Linux 노드를 테스트합니다.
2. (선택 사항) AmazonEKS_CNI_Policy 관리형 IAM 정책(IPv4 클러스터를 사용하는 경우) 또는 [AmazonEKS_CNI_IPv6_Policy](#)(IPv6 클러스터를 사용하는 경우 [직접 만든 것](#))이 [Amazon EKS 노드 IAM 역할](#)에 연결되어 있는 경우 대신 Kubernetes aws-node 서비스 계정에 연결하는 IAM 역할에 할당하는 것을 권장합니다. 자세한 내용은 [the section called “IRSA에 대한 구성”](#) 섹션을 참조하세요.
3. 다음과 같은 조건에 해당하면 IMDS에 대한 포트 액세스를 차단하는 것이 좋습니다.
 - 포트에 필요한 최소 권한만 있도록 모든 Kubernetes 서비스 계정에 IAM 역할을 할당할 계획입니다.
 - 클러스터의 어떤 포트도 현재 AWS 리전 검색 등의 다른 이유로 Amazon EC2 인스턴스 메타데이터 서비스(IMDS)에 액세스할 필요가 없습니다.

자세한 내용은 [워커 노드에 할당된 인스턴스 프로필에 대한 액세스 제한](#) 섹션을 참조하세요.

자체 관리형 Bottlerocket 노드 생성

Note

관리형 노드 그룹은 사용 사례에 대한 몇 가지 이점을 제공할 수 있습니다. 자세한 내용은 [the section called “관리형 노드 그룹”](#) 섹션을 참조하세요.

이 주제에서는 Amazon EKS 클러스터에 등록하는 [Bottlerocket](#) 노드의 Auto Scaling 그룹을 시작하는 방법을 설명합니다. Bottlerocket은 가상 머신 또는 베어메탈 호스트에서 컨테이너를 실행하는데 사용할 수 있는 AWS의 Linux 기반 오픈 소스 운영 체제입니다. 노드가 클러스터에 조인한 이후 Kubernetes 애플리케이션을 배포할 수 있습니다. Bottlerocket에 대한 자세한 내용은 GitHub의 [Using a Bottlerocket AMI with Amazon EKS](#)와 eksctl 설명서의 [사용자 정의 AMI 지원](#)을 참조하세요.

현재 위치 업그레이드에 대한 자세한 내용은 GitHub의 [Bottlerocket Update Operator](#)를 참조하세요.

Important

- Amazon EKS 노드는 표준 Amazon EC2 인스턴스이고, 일반 Amazon EC2 인스턴스 가격을 기반으로 비용이 청구됩니다. 자세한 설명은 [Amazon EC2 요금](#)을 참조하세요.
- Bottlerocket 노드를 AWS Outposts의 Amazon EKS 확장 클러스터에서 시작할 수 있지만 AWS Outposts의 로컬 클러스터에서는 시작할 수 없습니다. 자세한 내용은 [AWS Outposts의 Amazon EKS](#) 섹션을 참조하세요.
- x86 또는 Arm 프로세서가 있는 Amazon EC2 인스턴스에 배포할 수 있습니다. 그러나 Inferentia 칩이 있는 인스턴스에는 배포할 수 없습니다.
- Bottlerocket은 AWS CloudFormation과 호환됩니다. 그러나 Amazon EKS용 Bottlerocket 노드를 배포하기 위해 복사할 수 있는 공식 CloudFormation 템플릿은 없습니다.
- Bottlerocket 이미지는 SSH 서버나 셸과 함께 제공되지 않습니다. 대역 외 액세스 방법을 사용하여 SSH가 관리 컨테이너를 사용 설정하고 사용자 데이터와 함께 일부 부트스트래핑 구성 단계를 전달하도록 할 수 있습니다. 자세한 내용은 GitHub의 [bottlerocket Readme.md](#)에 있는 관련 섹션을 참조하세요.
 - [탐색](#)
 - [관리자 컨테이너](#)
 - [Kubernetes 설정](#)

이 절차에는 eksctl 버전 0.210.0 이상이 필요합니다. 버전은 다음 명령을 통해 확인할 수 있습니다.

```
eksctl version
```

eksctl을 설치하거나 업그레이드하는 방법에 대한 지침은 eksctl 문서의 [설치](#)를 참조하세요. 참고: 이 절차는 eksctl로 생성한 클러스터에서만 작동합니다.

1. 다음 콘텐츠를 디바이스에 복사합니다. **my-cluster**를 해당 클러스터의 이름으로 바꿉니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다. **no-bottlerocket**을 노드 그룹의 이름으로 바꿉니다. 노드 그룹 이름은 63자를 초과할 수 없습니다. 문자나 숫자로 시작하되, 나머지 문자의 경우 하이픈과 밑줄을 포함할 수 있습니다. Arm 인스턴스에 배포하려면 **m5.large**을 Arm 인스턴스 유형으로 바꿉니다. **my-ec2-keypair-name**을 시작 이후 SSH를 사용하여 노드에 연결하는 데 사용할 수 있는 Amazon EC2 SSH 키 페어 이름으로 변경합니

다. Amazon EC2 키 페어가 아직 없는 경우 AWS Management Console에서 새로 생성할 수 있습니다. 자세한 내용을 알아보려면 Amazon EC2 key pairs의 [Amazon EC2 키 페어](#)를 참조하세요. 나머지 ## #을 자신의 값으로 바꿉니다. 다 바꾼 후 수정된 명령을 실행하여 bottlerocket.yaml 파일을 생성합니다.

Arm Amazon EC2 인스턴스 유형을 지정하는 경우 배포하기 전에 [Amazon EKS 최적화 Arm Amazon Linux AMI](#)의 고려 사항을 검토하세요. 사용자 정의 AMI를 사용하여 배포하는 방법에 대한 지침은 GitHub의 [Building Bottlerocket](#)과 eksctl 문서의 [사용자 정의 AMI 지원](#)을 참조하세요. 관리형 노드 그룹을 배포하려면 시작 템플릿을 사용하여 사용자 정의 AMI를 배포합니다. 자세한 내용은 [the section called “시작 템플릿”](#) 섹션을 참조하세요.

Important

노드 그룹을 AWS Outposts, AWS Wavelength 또는 AWS Local Zones 서브넷에 배포하려면 클러스터를 생성할 때 AWS Outposts, AWS Wavelength 또는 AWS 로컬 영역 서브넷을 전달하지 마세요. 다음 예에서는 서브넷을 지정해야 합니다. 자세한 내용은 eksctl 문서에서 [구성 파일을 사용하여 nodegroup 생성](#) 및 [구성 파일 스키마](#) 부분을 참조하세요. *region-code*를 클러스터가 있는 AWS 리전으로 바꿉니다.

```
cat >bottlerocket.yaml <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: '1.32'

iam:
  withOIDC: true

nodeGroups:
- name: ng-bottlerocket
  instanceType: m5.large
  desiredCapacity: 3
  amiFamily: Bottlerocket
  ami: auto-ssm
  iam:
```

```

attachPolicyARNs:
  - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
  - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
  - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
  - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
ssh:
  allow: true
  publicKeyName: my-ec2-keypair-name
EOF

```

2. 다음 명령을 사용하여 노드를 배포합니다.

```
eksctl create nodegroup --config-file=bottlerocket.yaml
```

예제 출력은 다음과 같습니다.

노드가 생성되는 동안 여러 줄이 출력됩니다. 출력의 마지막 줄 중 하나는 다음 예제 줄과 유사합니다.

```
[#] created 1 nodegroup(s) in cluster "my-cluster"
```

- (선택 사항) [Amazon EBS CSI 플러그 인](#)을 사용하여 Bottlerocket 노드에 Kubernetes [영구 볼륨](#)을 생성합니다. 기본 Amazon EBS 드라이버는 Bottlerocket에 포함되지 않은 파일 시스템 도구에 의존합니다. 드라이버를 사용하여 스토리지 클래스를 생성하는 방법에 대한 자세한 내용은 [the section called "Amazon EBS"](#) 부분을 참조하세요.
- (선택 사항) 기본적으로 kube-proxy는 Bottlerocket이 부팅 시에 원래 설정한 것과 다를 수 있는 기본값으로 nf_conntrack_max 커널 파라미터를 설정합니다. Bottlerocket의 [기본 설정](#)을 유지하려면 다음 명령을 사용하여 kube-proxy 구성을 편집합니다.

```
kubect1 edit -n kube-system daemonset kube-proxy
```

--conntrack-max-per-core 및 --conntrack-min을 다음 예에 있는 kube-proxy 인수에 추가합니다. 0으로 설정할 경우 변경하지 않음을 나타냅니다.

```

containers:
  - command:
    - kube-proxy
    - --v=2
    - --config=/var/lib/kube-proxy-config/config
    - --conntrack-max-per-core=0

```

```
- --conntrack-min=0
```

5. (선택 사항) [샘플 애플리케이션](#)을 배포하여 Bottlerocket 노드를 테스트합니다.
6. 다음과 같은 조건에 해당하면 IMDS에 대한 포트 액세스를 차단하는 것이 좋습니다.
 - 포트에 필요한 최소 권한만 있도록 모든 Kubernetes 서비스 계정에 IAM 역할을 할당할 계획입니다.
 - 클러스터의 어떤 포트도 현재 AWS 리전 검색 등의 다른 이유로 Amazon EC2 인스턴스 메타데이터 서비스(IMDS)에 액세스할 필요가 없습니다.

자세한 내용은 [워커 노드에 할당된 인스턴스 프로필에 대한 액세스 제한](#) 섹션을 참조하세요.

자체 관리형 Microsoft Windows 노드 생성

이 주제에서는 Amazon EKS 클러스터에 등록하는 Windows 노드의 오토 스케일링 그룹을 시작하는 방법을 설명합니다. 노드가 클러스터에 조인한 이후 Kubernetes 애플리케이션을 배포할 수 있습니다.

Important

- Amazon EKS 노드는 표준 Amazon EC2 인스턴스이고, 일반 Amazon EC2 인스턴스 가격을 기반으로 비용이 청구됩니다. 자세한 설명은 [Amazon EC2 요금](#)을 참조하세요.
- Windows 노드를 AWS Outposts의 Amazon EKS 확장 클러스터에서 시작할 수 있지만 AWS Outposts의 로컬 클러스터에서는 시작할 수 없습니다. 자세한 내용은 [AWS Outposts의 Amazon EKS](#) 섹션을 참조하세요.

클러스터에 Windows 지원 사용 Windows 노드 그룹을 시작하기 전에 중요한 고려 사항을 검토하는 것이 좋습니다. 자세한 내용은 [the section called “Windows 지원 사용”](#) 섹션을 참조하세요.

다음 중 하나를 사용하여 자체 관리형 Windows 노드를 시작할 수 있습니다.

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)

eksctl

eksctl을 사용하여 자체 관리형 Windows 노드 시작

이 절차에서는 eksctl을 설치하고 eksctl 버전 0.210.0 이상을 요구합니다. 버전은 다음 명령을 통해 확인할 수 있습니다.

```
eksctl version
```

eksctl 설치 또는 업데이트에 대한 지침은 eksctl 설명서에서 [설치](#)를 참조하세요.

Note

이 방법은 eksctl로 생성된 클러스터에만 사용할 수 있습니다.

1. (선택 사항) AmazonEKS_CNI_Policy 관리형 IAM 정책(IPv4 클러스터를 사용하는 경우) 또는 [AmazonEKS_CNI_IPv6_Policy](#)(IPv6 클러스터를 사용하는 경우 [직접 만든 것](#))이 [Amazon EKS 노드 IAM 역할](#)에 연결되어 있는 경우 대신 Kubernetes aws-node 서비스 계정에 연결하는 IAM 역할에 할당하는 것을 권장합니다. 자세한 내용은 [the section called “IRSA에 대한 구성”](#) 섹션을 참조하세요.
2. 이 절차에서는 기존 클러스터가 있다고 가정합니다. Windows 노드 그룹을 추가할 Amazon EKS 클러스터 및 Amazon Linux 노드 그룹이 아직 없는 경우에는 [the section called “클러스터 생성\(eksctl\)”](#) 가이드를 따르는 것이 좋습니다. 이 설명서에서는 Amazon Linux 노드로 Amazon EKS 클러스터를 생성하는 방법에 대한 전체 시연을 제공합니다.

다음 명령을 사용하여 노드 그룹을 생성합니다. **region-code**를 클러스터가 있는 AWS 리전으로 바꿉니다. **<cluster-name>**을 클러스터 이름으로 바꿉니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다. **ng-windows**을 노드 그룹의 이름으로 바꿉니다. 노드 그룹 이름은 63자를 초과할 수 없습니다. 문자나 숫자로 시작하되, 나머지 문자의 경우 하이픈과 밑줄을 포함할 수 있습니다. Windows Server 2022를 사용하려면 **2019**를 2022로 바꿉니다. 나머지 **##** **#**을 자신의 값으로 대체합니다.

Important

노드 그룹을 AWS Outposts, AWS Wavelength 또는 AWS Local Zones 서브넷에 배포하려면 클러스터를 생성할 때 AWS Outposts, Wavelength 또는 로컬 영역 서브넷을 전달하지 마세요. 구성 파일을 사용하여 노드 그룹을 생성합니다. 이때 AWS Outposts, Wavelength 또는 Local Zones 서브넷을 지정합니다. 자세한 내용은 eksctl 문서의 [구성 파일을 사용하여 nodegroup 생성](#) 및 [구성 파일 스키마](#) 섹션을 참조하세요.

```
eksctl create nodegroup \
  --region region-code \
  --cluster my-cluster \
  --name ng-windows \
  --node-type t2.large \
  --nodes 3 \
  --nodes-min 1 \
  --nodes-max 4 \
  --managed=false \
  --node-ami-family WindowsServer2019FullContainer
```

Note

- 노드가 클러스터에 조인하지 못한 경우 문제 해결 안내서의 [the section called “노드가 클러스터 조인에 실패”](#) 부분을 참조하세요.
- eksctl 명령에 사용 가능한 옵션을 보려면 다음 명령을 입력합니다.

```
eksctl command -help
```

예제 출력은 다음과 같습니다. 노드가 생성되는 동안 여러 줄이 출력됩니다. 출력의 마지막 줄 중 하나는 다음 예제 줄과 유사합니다.

```
[#] created 1 nodegroup(s) in cluster "my-cluster"
```

3. (선택 사항) [샘플 애플리케이션](#)을 배포하여 클러스터 및 Windows 노드를 테스트합니다.
4. 다음과 같은 조건에 해당하면 IMDS에 대한 포트 액세스를 차단하는 것이 좋습니다.
 - 포트에 필요한 최소 권한만 있도록 모든 Kubernetes 서비스 계정에 IAM 역할을 할당할 계획입니다.
 - 클러스터의 어떤 포트도 현재 AWS 리전 검색 등의 다른 이유로 Amazon EC2 인스턴스 메타데이터 서비스(IMDS)에 액세스할 필요가 없습니다.

자세한 내용은 [워커 노드에 할당된 인스턴스 프로필에 대한 액세스 제한](#) 섹션을 참조하세요.

AWS Management Console

사전 조건

- 기존 Amazon EKS 클러스터 및 Linux 노드 그룹. 이러한 리소스가 없는 경우 [시작](#)의 가이드 중 하나를 사용하여 생성하는 것이 좋습니다. 이러한 가이드에서는 Linux 노드로 Amazon EKS 클러스터를 생성하는 방법에 대해 설명합니다.
- Amazon EKS 클러스터의 요구 사항을 충족하는 기존 VPC 및 전용 보안 그룹. 자세한 내용은 [the section called “VPC 및 서브넷 요구 사항”](#) 및 [the section called “보안 그룹 요구 사항”](#) 섹션을 참조하세요. [시작](#)의 가이드에 따라 요구 사항을 충족하는 VPC를 만듭니다. 또는 [Amazon EKS 클러스터에 대한 Amazon VPC 생성](#)을 따라 수동으로 생성할 수도 있습니다.
- Amazon EKS 클러스터의 요구 사항을 충족하는 VPC 및 보안 그룹을 사용하는 기존 Amazon EKS 클러스터. 자세한 내용은 [the section called “클러스터 생성”](#) 섹션을 참조하세요. AWS Outposts, AWS Wavelength 또는 AWS Local Zones이 사용된 AWS 리전에 서브넷이 있는 경우 클러스터를 생성할 때 해당 서브넷이 전달되지 않은 상태여야 합니다.

1단계: AWS Management Console을 사용하여 자체 관리형 Windows 노드를 시작하려면

1. 클러스터 상태가 ACTIVE가 되기를 기다립니다. 클러스터가 활성화되기 전에 노드를 시작하면 노드가 클러스터에 등록되지 않고 노드를 다시 시작해야 합니다.
2. [AWS CloudFormation 콘솔](#) 열기
3. 스택 생성을 선택합니다.
4. 템플릿 지정에서 Amazon S3 URL을 선택합니다.
5. 다음과 같은 URL을 복사하여 Amazon S3 URL에 붙여 넣습니다.

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2023-02-09/amazon-eks-windows-nodegroup.yaml
```

6. 다음을 두 번 선택합니다
7. 빠른 스택 생성 페이지에서 다음 파라미터를 적절하게 입력합니다.
 - 스택 이름: AWS CloudFormation 스택에 대한 스택 이름을 선택합니다. 예를 들어 my-cluster-nodes라고 할 수 있습니다.
 - ClusterName: Amazon EKS 클러스터 생성 시 사용할 이름을 입력합니다.

⚠ Important

이 이름은 [1단계: Amazon EKS 클러스터 생성](#)에서 사용한 이름과 정확히 일치해야 합니다. 그렇지 않으면 노드가 클러스터에 가입할 수 없습니다.

- ClusterControlPlaneSecurityGroup: [VPC](#)를 만들 때 생성한 AWS CloudFormation 출력에서 보안 그룹을 선택합니다. 다음 단계에서는 해당 그룹을 검색하는 한 가지 방법을 보여줍니다.
 - a. [Amazon EKS 콘솔](#)을 엽니다.
 - b. 클러스터의 이름을 선택합니다.
 - c. 네트워킹 탭을 선택합니다.
 - d. ClusterControlPlaneSecurityGroup 드롭다운 목록에서 선택할 때 추가 보안 그룹 값을 참조로 사용하세요.
- NodeGroupName: 노드 그룹의 이름을 입력합니다. 이 이름은 나중에 노드에 대해 생성된 Auto Scaling 노드 그룹을 식별하는 데 사용할 수 있습니다. 노드 그룹 이름은 63자를 초과할 수 없습니다. 문자나 숫자로 시작하되, 나머지 문자의 경우 하이픈과 밑줄을 포함할 수 있습니다.
- NodeAutoScalingGroupMinSize: Auto Scaling 그룹이 축소할 수 있는 노드의 최소 노드 수를 입력합니다.
- NodeAutoScalingGroupDesiredCapacity: 스택을 생성할 때 조정할 원하는 노드 수를 입력합니다.
- NodeAutoScalingGroupMaxSize: Auto Scaling 그룹이 확장할 수 있는 노드의 최대 노드 수를 입력합니다.
- NodeInstanceType: 노드에 대한 인스턴스 유형을 선택합니다. 자세한 내용은 [the section called “Amazon EC2 인스턴스 유형”](#) 섹션을 참조하세요.

i Note

최신 버전의 [Kubernetes용 Amazon VPC CNI 플러그인](#)에 대해 지원되는 인스턴스 유형은 GitHub의 [vpc_ip_resource_limit.go](#)에 나열되어 있습니다. 지원되는 최신 인스턴스 유형을 사용하려면 CNI 버전을 업데이트해야 할 수 있습니다. 자세한 내용은 [the section called “Amazon VPC CNI”](#) 섹션을 참조하세요.

- NodeImageIdSSMParam: 현재 권장되는 Amazon EKS 최적화 Windows Core AMI ID의 Amazon EC2 Systems Manager 파라미터로 미리 채워집니다. 전체 버전의 Windows를 사용하려면 **Core**를 Full로 바꿉니다.

- **NodeImageId:** (선택사항) Amazon EKS 최적화 AMI 대신 사용자 정의 AMI를 사용 중인 경우, 해당 AWS 리전에 노드 AMI ID를 입력합니다. 이 필드의 값을 지정하면 NodeImageIdSSMParam 필드에 있는 모든 값이 재정의됩니다.
- **NodeVolumeSize:** 노드에 대해 루트 볼륨 크기를 GiB 단위로 지정합니다.
- **KeyName:** 시작 이후 SSH를 사용하여 노드에 연결하는 데 사용할 수 있는 Amazon EC2 SSH 키 페어 이름을 입력합니다. Amazon EC2 키 페어가 아직 없는 경우 AWS Management Console에서 새로 생성할 수 있습니다. 자세한 내용을 알아보려면 Amazon EC2 key pairs의 [Amazon EC2 키 페어](#)를 참조하세요.

 Note

여기에 키 페어를 제공하지 않으면 AWS CloudFormation 스택이 생성되지 않습니다.

- **BootstrapArguments:** -KubeletExtraArgs를 사용하여 별도의 kubelet 인수와 같이 노드 부트스트랩 스크립트에 전달할 선택적 인수를 지정합니다.
- **DisableIMDSv1:** 기본적으로 각 노드는 인스턴스 메타데이터 서비스 버전 1(IMDSv1) 및 IMDSv2를 지원합니다. IMDSv1을 사용 중지할 수 있습니다. 향후 노드 그룹의 노드와 포드가 IMDSv1을 사용하지 못하게 하려면 DisableIMDSv1을 true로 설정합니다. IMDS에 대한 자세한 내용은 [인스턴스 메타데이터 서비스 구성](#)을 참조하세요.
- **VpcId:** 생성한 [VPC](#)에 대한 ID를 선택합니다.
- **NodeSecurityGroups:** [VPC](#)를 생성할 때 Linux 노드 그룹에 대해 생성된 보안 그룹을 선택합니다. Linux 노드에 둘 이상의 보안 그룹이 연결되어 있는 경우 모든 보안 그룹을 지정합니다. 예를 들어 Linux 노드 그룹이 eksctl로 생성된 경우입니다.
- **서브넷:** 생성한 서브넷을 선택합니다. [Amazon EKS 클러스터를 위한 Amazon VPC 만들기](#)의 단계를 사용하여 VPC를 만든 경우, 노드가 실행할 VPC 내의 사설 서브넷만 지정합니다.

 Important

- 서브넷 중 하나가 퍼블릭 서브넷인 경우 자동 퍼블릭 IP 주소 할당 설정을 사용하도록 설정해야 합니다. 퍼블릭 서브넷에 대해 이 설정을 사용하지 않으면 해당 퍼블릭 서브넷에 배포하는 노드에는 퍼블릭 IP 주소가 할당되지 않으며 클러스터 또는 기타 AWS 서비스와 통신할 수 없습니다. 서브넷이 2020년 3월 26일 이전에 [Amazon EKS AWS CloudFormation VPC 템플릿](#) 또는 eksctl을 사용하여 배포된 경우 퍼블릭 서브넷에 대해 자동 퍼블릭 IP 주소 할당이 사용 중지됩니다. 서브넷에 퍼블릭 IP 주소 할당을 활성화하는 방법에 대한 자세한 내용은 [서브넷에 대한 퍼블릭 IPv4 주소 지정 속성 수](#)

[정](#)을 참조하세요. 노드가 프라이빗 서브넷에 배포되면 NAT 게이트웨이를 통해 클러스터 및 다른 AWS 서비스와 통신할 수 있습니다.

- 서브넷에서 인터넷에 액세스할 수 없는 경우 [인터넷 액세스가 제한된 비공개 클러스터 배포하기](#)의 고려 사항과 추가 단계를 숙지하고 있는지 확인하세요.
- AWS Outposts, Wavelength 또는 Local Zones 서브넷을 선택하는 경우 클러스터를 생성할 때 해당 서브넷이 전달되지 않은 상태여야 합니다.

8. 스택에서 IAM 리소스를 생성할 수 있는지 확인한 다음 스택 생성을 선택합니다.

9. 스택이 생성된 후 콘솔에서 이를 선택하고 출력을 선택합니다.

10. 생성된 노드 그룹에 대해 NodeInstanceRole을 기록합니다. Amazon EKS Windows 노드를 구성할 때 필요합니다.

2단계: 노드가 클러스터에 조인하도록 하려면

1. aws-auth ConfigMap이 이미 있는지 확인합니다.

```
kubectl describe configmap -n kube-system aws-auth
```

2. aws-auth ConfigMap이 표시되면 필요에 따라 이를 업데이트합니다.

a. 편집을 위해 ConfigMap을 엽니다.

```
kubectl edit -n kube-system configmap/aws-auth
```

b. 필요에 따라 새 mapRoles 항목을 추가합니다. rolearn 값을 이전 절차에서 기록한 NodeInstanceRole 값으로 설정합니다.

```
[...]
data:
  mapRoles: |
- rolearn: <ARN of linux instance role (not instance profile)>
  username: system:node:{{EC2PrivateDNSName}}
  groups:
    - system:bootstrappers
    - system:nodes
- rolearn: <ARN of windows instance role (not instance profile)>
  username: system:node:{{EC2PrivateDNSName}}
  groups:
    - system:bootstrappers
    - system:nodes
```

```
- eks:kube-proxy-windows
[...]
```

c. 파일을 저장하고 텍스트 편집기를 종료합니다.

3. "Error from server (NotFound): configmaps "aws-auth" not found라는 오류 메시지가 표시되면 스택 ConfigMap을 적용합니다.

a. 구성 맵을 다운로드합니다.

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
aws-auth-cm-windows.yaml
```

b. aws-auth-cm-windows.yaml 파일에서 rolearn 값을 이전 절차에서 기록한 해당 NodeInstanceRole 값으로 설정합니다. 이 작업은 텍스트 편집기를 사용하거나 `## #`을 바꾸고 다음 명령을 실행하여 수행할 수 있습니다.

```
sed -i.bak -e 's|<ARN of linux instance role (not instance profile)>|my-node-
linux-instance-role|' \
    -e 's|<ARN of windows instance role (not instance profile)>|my-node-windows-
instance-role|' aws-auth-cm-windows.yaml
```

Important

- 이 파일에서 어떠한 행도 수정하지 마세요.
- Windows 및 Linux 노드 모두에 동일한 IAM 역할을 사용하지 마세요.

c. 구성을 적용합니다. 이 명령을 완료하는 데 몇 분이 걸릴 수 있습니다.

```
kubectl apply -f aws-auth-cm-windows.yaml
```

4. 노드의 상태를 확인하고 Ready 상태가 될 때까지 대기합니다.

```
kubectl get nodes --watch
```

Ctrl+C를 입력하여 셸 프롬프트로 돌아갑니다.

Note

권한 부여 또는 리소스 유형 오류가 표시되는 경우 문제 해결 주제의 [the section called “권한이 없거나 액세스가 거부됨\(kubect1\)”](#) 부분을 참조하세요.

노드가 클러스터에 조인하지 못한 경우 문제 해결 장의 [the section called “노드가 클러스터 조인에 실패”](#) 섹션을 참조하세요.

3단계: 추가 작업

1. (선택 사항) [샘플 애플리케이션](#)을 배포하여 클러스터 및 Windows 노드를 테스트합니다.
2. (선택 사항) AmazonEKS_CNI_Policy 관리형 IAM 정책(IPv4 클러스터를 사용하는 경우) 또는 [AmazonEKS_CNI_IPv6_Policy](#)(IPv6 클러스터를 사용하는 경우 [직접 만든 것](#))이 [Amazon EKS 노드 IAM 역할](#)에 연결되어 있는 경우 대신 Kubernetes aws-node 서비스 계정에 연결하는 IAM 역할에 할당하는 것을 권장합니다. 자세한 내용은 [the section called “IRSA에 대한 구성”](#) 섹션을 참조하세요.
3. 다음과 같은 조건에 해당하면 IMDS에 대한 포트 액세스를 차단하는 것이 좋습니다.
 - 포트에 필요한 최소 권한만 있도록 모든 Kubernetes 서비스 계정에 IAM 역할을 할당할 계획입니다.
 - 클러스터의 어떤 포트도 현재 AWS 리전 검색 등의 다른 이유로 Amazon EC2 인스턴스 메타데이터 서비스(IMDS)에 액세스할 필요가 없습니다.

자세한 내용은 [워커 노드에 할당된 인스턴스 프로필에 대한 액세스 제한](#) 섹션을 참조하세요.

자체 관리형 Ubuntu Linux 노드 생성**Note**

관리형 노드 그룹은 사용 사례에 대한 몇 가지 이점을 제공할 수 있습니다. 자세한 내용은 [the section called “관리형 노드 그룹”](#) 섹션을 참조하세요.

이 주제에서는 Amazon EKS 클러스터에 등록하는 [Amazon Elastic Kubernetes Service\(EKS\)의 Ubuntu](#) 또는 [Amazon Elastic Kubernetes Service\(EKS\)의 Ubuntu Pro](#) 노드의 Auto Scaling 그룹을 시

작하는 방법을 설명합니다. Ubuntu 및 Ubuntu Pro for EKS는 공식 Ubuntu Minimal LTS를 기반으로 하며, AWS와 공동으로 개발되고 EKS용으로 특별히 구축된 사용자 지정 AWS 커널을 포함합니다. Ubuntu Pro는 EKS 확장 지원 기간, 커널 라이브패치, FIPS 규정 준수 및 무제한 Pro 컨테이너를 실행할 수 있는 기능을 지원하여 보안 범위를 추가합니다.

노드가 클러스터에 조인한 이후 컨테이너화된 애플리케이션을 배포할 수 있습니다. 자세한 내용은 eksctl 설명서의 [AWS 상의 Ubuntu](#) 및 [사용자 지정 AMI 지원](#)을 참조하세요.

Important

- Amazon EKS 노드는 표준 Amazon EC2 인스턴스이고, 일반 Amazon EC2 인스턴스 가격을 기반으로 비용이 청구됩니다. 자세한 설명은 [Amazon EC2 요금](#)을 참조하세요.
- Ubuntu 노드를 AWSOutposts의 Amazon EKS 확장 클러스터에서 시작할 수 있지만 AWS Outposts의 로컬 클러스터에서는 시작할 수 없습니다. 자세한 내용은 [AWSOutposts의 Amazon EKS](#) 섹션을 참조하세요.
- x86 또는 Arm프로세서가 있는 Amazon EC2 인스턴스에 배포할 수 있습니다. 하지만 Inferentia 칩이 있는 인스턴스의 경우 먼저 [Neuron SDK](#)를 설치해야 할 수 있습니다.

이 절차에는 eksctl 버전 0.210.0 이상이 필요합니다. 버전은 다음 명령을 통해 확인할 수 있습니다.

```
eksctl version
```

eksctl을 설치하거나 업그레이드하는 방법에 대한 지침은 eksctl 문서의 [설치](#)를 참조하세요.참고: 이 절차는 eksctl로 생성한 클러스터에서만 작동합니다.

1. 다음 콘텐츠를 디바이스에 복사합니다. my-cluster를 클러스터 이름으로 바꿉니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영문자로 시작해야 하며 100자 이하여야 합니다. ng-ubuntu을 노드 그룹의 이름으로 바꿉니다. 노드 그룹 이름은 63자를 초과할 수 없습니다. 문자나 숫자로 시작하되, 나머지 문자의 경우 하이픈과 밑줄을 포함할 수 있습니다. Arm 인스턴스에 배포하려면 m5.large을 Arm 인스턴스 유형으로 바꿉니다. my-ec2-keypair-name을 시작 이후 SSH를 사용하여 노드에 연결하는 데 사용할 수 있는 Amazon EC2 SSH 키 페어 이름으로 변경합니다. Amazon EC2 키 페어가 아직 없는 경우 AWS Management Console에서 새로 생성할 수 있습니다. 자세한 내용을 알아보려면 Amazon EC2 사용 설명서의 [Amazon EC2 키 페어](#)를 참조하세요. 나머지 ## #을 자신의 값으로 바꿉니다. 다 바꾼 후 수정된 명령을 실행하여 ubuntu.yaml 파일을 생성합니다.

⚠ Important

노드 그룹을 AWS Outposts, AWS Wavelength 또는 AWS Local Zones 서브�트에 배포하려면 클러스터를 생성할 때 AWS Outposts, AWS Wavelength 또는 AWS 로컬 영역 서브넛을 전달하지 마세요. 다음 예에서는 서브넛을 지정해야 합니다. 자세한 내용은 eksctl 문서에서 [구성 파일을 사용하여 nodegroup 생성](#) 및 [구성 파일 스키마](#) 부분을 참조하세요. *region-code*를 클러스터가 있는 AWS 리전으로 바꿉니다.

```
cat >ubuntu.yaml <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: '1.32'

iam:
  withOIDC: true

nodeGroups:
- name: ng-ubuntu
  instanceType: m5.large
  desiredCapacity: 3
  amiFamily: Ubuntu2204
  iam:
    attachPolicyARNs:
      - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
      - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
      - arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
      - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
  ssh:
    allow: true
    publicKeyName: my-ec2-keypair-name
EOF
```

Ubuntu Pro 노드 그룹을 생성하려면 amiFamily 값을 UbuntuPro2204로 변경하면 됩니다.

2. 다음 명령을 사용하여 노드를 배포합니다.

```
eksctl create nodegroup --config-file=ubuntu.yaml
```

예제 출력은 다음과 같습니다.

노드가 생성되는 동안 여러 줄이 출력됩니다. 출력의 마지막 줄 중 하나는 다음 예제 줄과 유사합니다.

```
[#] created 1 nodegroup(s) in cluster "my-cluster"
```

3. (선택 사항) [샘플 애플리케이션](#)을 배포하여 Ubuntu 노드를 테스트합니다.
4. 다음과 같은 조건에 해당하면 IMDS에 대한 포트 액세스를 차단하는 것이 좋습니다.
 - 포트에 필요한 최소 권한만 있도록 모든 Kubernetes 서비스 계정에 IAM 역할을 할당할 계획입니다.
 - 클러스터의 어떤 포트도 현재 AWS 리전 검색 등의 다른 이유로 Amazon EC2 인스턴스 메타데이터 서비스(IMDS)에 액세스할 필요가 없습니다.

자세한 내용은 [워커 노드에 할당된 인스턴스 프로필에 대한 액세스 제한](#) 섹션을 참조하세요.

클러스터의 자체 관리형 노드 업데이트

새로운 Amazon EKS 최적화 AMI가 릴리스되면 자체 관리형 노드 그룹의 노드를 새 AMI로 교체하는 것을 고려합니다. 마찬가지로, Amazon EKS 클러스터용 Kubernetes 버전을 업데이트한 경우 동일한 Kubernetes 버전이 있는 노드를 사용하도록 노드를 업데이트합니다.

Important

이 주제에서는 자체 관리형 노드 그룹에 대한 노드 업데이트에 대해 설명합니다. [관리형 노드 그룹](#)을 사용하는 경우 [the section called “업데이트”](#) 섹션을 참조하세요.

새 AMI를 사용하도록 클러스터의 자체 관리형 노드 그룹을 업데이트하는 기본 방법에는 두 가지가 있습니다.

[애플리케이션을 새 노드 그룹으로 마이그레이션](#)

새 노드 그룹을 생성하고 포드를 해당 그룹으로 마이그레이션합니다. 기존 AWS CloudFormation 스택에서 단순히 AMI ID를 업데이트하는 것보다 새 노드 그룹으로 마이그레이션하는 것이 낫습니

다. 이는 마이그레이션 프로세스가 이전 노드 그룹을 NoSchedule로 [테인트](#)하고 새 스택이 기존 포드 워크로드를 수락할 준비가 된 후 노드를 드레이닝하기 때문입니다.

[AWS CloudFormation 노드 스택 업데이트](#)

새 AMI를 사용하도록 기존 노드 그룹의 AWS CloudFormation 스택을 업데이트합니다. 이 방법은 eksctl로 생성한 노드 그룹에는 사용할 수 없습니다.

애플리케이션을 새 노드 그룹으로 마이그레이션

이 주제에서는 새로운 노드 그룹을 생성하고, 기존 애플리케이션을 새 그룹으로 안정적으로 마이그레이션한 다음, 이전 노드 그룹을 클러스터에서 제거하는 방법을 설명합니다. eksctl 또는 AWS Management Console을 사용하여 새 노드 그룹으로 마이그레이션할 수 있습니다.

- [the section called “eksctl”](#)
- [the section called “AWS Management Console 및 AWS CLI”](#)

eksctl

eksctl을 사용하여 애플리케이션을 새 노드 그룹으로 마이그레이션하려면

마이그레이션에 eksctl을 사용하는 방법에 대한 자세한 내용은 eksctl 설명서의 [비관리형 노드 그룹](#)을 참조하세요.

이 절차에는 eksctl 버전 0.210.0 이상이 필요합니다. 버전은 다음 명령을 통해 확인할 수 있습니다.

```
eksctl version
```

eksctl 설치 또는 업데이트에 대한 지침은 eksctl 설명서에서 [설치](#)를 참조하세요.

Note

이 절차는 eksctl을 사용하여 생성한 클러스터 및 노드 그룹에 대해서만 사용할 수 있습니다.

1. **my-cluster**를 클러스터 이름으로 바꿔서 기존 노드 그룹의 이름을 검색합니다.

```
eksctl get nodegroups --cluster=my-cluster
```

예제 출력은 다음과 같습니다.

CLUSTER	NODEGROUP	CREATED	MIN SIZE	MAX SIZE
DESIRED CAPACITY	INSTANCE TYPE	IMAGE ID		
default	standard-nodes t3.medium	2019-05-01T22:26:58Z ami-05a71d034119ffc12	1 4	3

2. 다음 명령어로 eksctl을 사용하여 새 노드 그룹을 시작합니다. 명령에서 모든 **##** #을 고유한 값으로 바꿉니다. 버전 번호는 컨트롤 플레인의 Kubernetes 버전보다 이후일 수 없습니다. 또한 컨트롤 플레인의 Kubernetes 버전보다 이전인 마이너 버전이 2개를 초과할 수 없습니다. 컨트롤 플레인과 동일한 버전을 사용하는 것이 좋습니다.

다음과 같은 조건에 해당하면 IMDS에 대한 포트 액세스를 차단하는 것이 좋습니다.

- 포트에 필요한 최소 권한만 있도록 모든 Kubernetes 서비스 계정에 IAM 역할을 할당할 계획입니다.
- 클러스터의 어떤 포트도 현재 AWS 리전 검색 등의 다른 이유로 Amazon EC2 인스턴스 메타데이터 서비스(IMDS)에 액세스할 필요가 없습니다.

자세한 내용은 [작업자 노드에 할당된 인스턴스 프로필에 대한 액세스 제한](#) 부분을 참조하세요.

포트가 IMDS에 액세스하지 못하게 차단하려면 `--disable-pod-imds` 옵션을 다음 명령에 추가합니다.

Note

사용 가능한 플래그 및 설명에 대한 자세한 내용은 <https://eksctl.io>를 참조하세요.

```
eksctl create nodegroup \
  --cluster my-cluster \
  --version 1.32 \
  --name standard-nodes-new \
  --node-type t3.medium \
  --nodes 3 \
  --nodes-min 1 \
  --nodes-max 4 \
  --managed=false
```

3. 이전 명령이 완료되면 다음 명령으로 모든 노드가 Ready 상태가 되었는지 확인합니다.

```
kubectl get nodes
```

- 다음 명령을 사용하여 원래 노드 그룹을 삭제합니다. 명령에서 모든 **##** **#**을 클러스터 이름과 노드 그룹 이름으로 바꿉니다.

```
eksctl delete nodegroup --cluster my-cluster --name standard-nodes-old
```

AWS Management Console 및 AWS CLI

AWS Management Console 및 AWS CLI에서 애플리케이션을 새 노드 그룹으로 마이그레이션하려면

1. [자체 관리 Amazon Linux 노드 생성](#)에 설명된 단계에 따라 새 노드 그룹을 시작합니다.
2. 스택이 생성된 후 콘솔에서 이를 선택하고 출력을 선택합니다.
3. 생성된 노드 그룹에 대해 NodeInstanceRole을 기록합니다. 새 Amazon EKS 노드를 클러스터에 추가하려면 이 정보가 필요합니다.

Note

추가 IAM 정책을 이전 노드 그룹의 IAM 역할에 연결한 경우 새 그룹에서 해당 기능을 유지하려면 동일한 정책을 새 노드 그룹의 IAM 역할에 연결해야 합니다. 이는 예를 들어, Kubernetes [Cluster Autoscaler](#)에 대한 권한을 추가한 경우에 적용됩니다.

4. 두 노드 그룹이 서로 통신할 수 있도록 두 노드 그룹의 보안 그룹을 업데이트합니다. 자세한 내용은 [the section called “보안 그룹 요구 사항”](#) 섹션을 참조하세요.
 - a. 두 노드 그룹에 대한 보안 그룹 ID를 모두 기록합니다. 이는 AWS CloudFormation 스택 출력에 NodeSecurityGroup 값으로 표시됩니다.

다음 AWS CLI 명령을 사용하여 스택 이름에서 보안 그룹 ID를 가져옵니다. 이러한 명령에서 oldNodes는 이전 노드 스택의 AWS CloudFormation 스택 이름이고 newNodes는 마이그레이션 하는 대상 스택의 이름입니다. 모든 **##** **#**을 자신의 값으로 바꾸세요.

```
oldNodes="old_node_CFN_stack_name"
newNodes="new_node_CFN_stack_name"

oldSecGroup=$(aws cloudformation describe-stack-resources --stack-name $oldNodes \
--query 'StackResources[?
ResourceType=='AWS::EC2::SecurityGroup'].PhysicalResourceId' \
--output text)
newSecGroup=$(aws cloudformation describe-stack-resources --stack-name $newNodes \
```

```
--query 'StackResources[?
ResourceType=='AWS::EC2::SecurityGroup'].PhysicalResourceId' \
--output text)
```

- b. 각 노드 보안 그룹이 서로의 트래픽을 수락하도록 노드 보안 그룹에 인그레스 규칙을 추가합니다.

다음 AWS CLI 명령은 상대 보안 그룹의 모든 프로토콜에서 모든 트래픽을 허용하는 인바운드 규칙을 각 보안 그룹에 추가합니다. 이렇게 구성하면 워크로드를 새 그룹으로 마이그레이션하는 동안 각 노드 그룹의 포드가 서로 통신할 수 있습니다.

```
aws ec2 authorize-security-group-ingress --group-id $oldSecGroup \
--source-group $newSecGroup --protocol -1
aws ec2 authorize-security-group-ingress --group-id $newSecGroup \
--source-group $oldSecGroup --protocol -1
```

5. aws-auth configmap을 편집하여 RBAC에서 새 노드 인스턴스 역할을 매핑합니다.

```
kubectl edit configmap -n kube-system aws-auth
```

새 노드 그룹에 대한 새 mapRoles 항목을 추가합니다.

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: ARN of instance role (not instance profile)
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes>
    - rolearn: arn:aws:iam::111122223333:role/nodes-1-16-NodeInstanceRole-
U11V27W93CX5
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
```

ARN of instance role (not instance profile) 조각을 [이전 절차](#)에서 기록한 NodeInstanceRole 값으로 교체하고 파일을 저장합니다. 그런 다음 파일을 저장하고 달아서 업데이트된 configmap을 적용합니다.

6. 노드의 상태를 보면서 새 노드가 클러스터에 조인하고 Ready 상태에 도달할 때까지 기다립니다.

```
kubectl get nodes --watch
```

7. (선택사항) [Kubernetes Cluster Autoscaler](#)를 사용하는 경우, 배포를 복제본 0개로 축소하여 조정 작업의 충돌을 방지합니다.

```
kubectl scale deployments/cluster-autoscaler --replicas=0 -n kube-system
```

8. 다음 명령을 사용하여 NoSchedule로 제거하려는 각 노드를 테인팅합니다. 이는 바꾸고 있는 노드에서 새 포드가 예약되거나 다시 예약되지 않도록 하기 위한 것입니다. 자세한 내용은 Kubernetes 설명서의 [테인팅과 허용 오차](#)를 참조하세요.

```
kubectl taint nodes node_name key=value:NoSchedule
```

노드를 새 Kubernetes 버전으로 업그레이드하는 경우 다음 코드 조각을 사용하여 특정 Kubernetes 버전(이 경우 1.30)의 모든 노드를 식별하고 테인팅할 수 있습니다. 버전 번호는 컨트롤 플레인의 Kubernetes 버전보다 이후일 수 없습니다. 또한 컨트롤 플레인의 Kubernetes 버전보다 이전인 마이너 버전이 2개를 초과할 수 없습니다. 컨트롤 플레인과 동일한 버전을 사용하는 것이 좋습니다.

```
K8S_VERSION=1.30
nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==\"v$K8S_VERSION\")].metadata.name}")
for node in ${nodes[@]}
do
    echo "Tainting $node"
    kubectl taint nodes $node key=value:NoSchedule
done
```

9. 클러스터의 DNS 공급자를 결정합니다.

```
kubectl get deployments -l k8s-app=kube-dns -n kube-system
```

예제 출력은 다음과 같습니다. 이 클러스터는 DNS 확인에 CoreDNS를 사용하고 있지만, 클러스터는 그 대신 kube-dns를 반환할 수 있습니다.

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
coredns	1	1	1	1	31m

10. 현재 배포가 두 개 미만의 복제본을 실행하고 있는 경우, 배포를 복제본 두 개로 확장합니다. 이전 명령 출력에서 해당 값이 대신 반환된 경우 *coredns*을 kubernetes로 교체합니다.

```
kubectl scale deployments/coredns --replicas=2 -n kube-system
```

11. 다음 명령을 사용하여 클러스터에서 제거할 각 노드를 드레인합니다.

```
kubectl drain node_name --ignore-daemonsets --delete-local-data
```

노드를 새 Kubernetes 버전으로 업그레이드하는 경우 다음 코드 조각을 사용하여 특정 Kubernetes 버전(이 경우 **1.30**)의 모든 노드를 식별하고 드레인합니다.

```
K8S_VERSION=1.30
nodes=$(kubectl get nodes -o jsonpath="{.items[?(@.status.nodeInfo.kubeletVersion==\`v${K8S_VERSION}\`)].metadata.name}")
for node in ${nodes[@]}
do
    echo "Draining $node"
    kubectl drain $node --ignore-daemonsets --delete-local-data
done
```

12. 이전 노드가 드레인을 완료한 후 앞서 권한을 부여한 보안 그룹 인바운드 규칙을 취소합니다. 그런 다음 다음 AWS CloudFormation 스택을 삭제하여 인스턴스를 종료합니다.

Note

[Kubernetes 클러스터 오토스케일러](#)에 대한 사용 권한 추가와 같이 이전 노드 그룹 IAM 역할에 추가 IAM 정책을 첨부한 경우, AWS CloudFormation 스택을 삭제하기 전에 해당 추가 정책을 역할에서 분리해야 합니다.

- a. 위에서 노드 보안 그룹에 대해 생성한 인바운드 규칙을 취소합니다. 이러한 명령에서 `oldNodes`는 이전 노드 스택의 AWS CloudFormation 스택 이름이고 `newNodes`는 마이그레이션하는 대상 스택의 이름입니다.

```
oldNodes="old_node_CFN_stack_name"
newNodes="new_node_CFN_stack_name"

oldSecGroup=$(aws cloudformation describe-stack-resources --stack-name $oldNodes \
--query 'StackResources[?
ResourceType=='AWS::EC2::SecurityGroup'].PhysicalResourceId' \
--output text)
```

```
newSecGroup=$(aws cloudformation describe-stack-resources --stack-name $newNodes \
--query 'StackResources[?
ResourceType=='AWS::EC2::SecurityGroup'].PhysicalResourceId' \
--output text)
aws ec2 revoke-security-group-ingress --group-id $oldSecGroup \
--source-group $newSecGroup --protocol -1
aws ec2 revoke-security-group-ingress --group-id $newSecGroup \
--source-group $oldSecGroup --protocol -1
```

- b. [AWS CloudFormation 콘솔](#)을 엽니다.
- c. 이전 노드 스택을 선택합니다.
- d. 삭제를 선택합니다.
- e. 스택 삭제 확인 대화 상자에서 스택 삭제를 선택합니다.

13. `aws-auth configmap`을 편집하여 RBAC에서 이전 노드 인스턴스 역할을 제거합니다.

```
kubectl edit configmap -n kube-system aws-auth
```

이전 노드 그룹에 대한 `mapRoles` 항목을 삭제합니다.

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: arn:aws:iam::111122223333:role/nodes-1-16-NodeInstanceRole-
W07025MZQFF8
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
    - rolearn: arn:aws:iam::111122223333:role/nodes-1-15-NodeInstanceRole-
U11V27W93CX5
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes>
```

파일을 저장하고 닫아서 업데이트된 `configmap`을 적용합니다.

14.(선택사항) Kubernetes [Cluster Autoscaler](#)를 사용하는 경우, 배포를 다시 복제본 한 개로 축소합니다.

Note

또한 새 Auto Scaling 그룹에 적절한 태그를 지정하고(예: k8s.io/cluster-autoscaler/enabled, k8s.io/cluster-autoscaler/my-cluster) 태그가 새로 지정된 Auto Scaling 그룹을 가리키도록 Cluster Autoscaler 배포에 대한 명령을 업데이트해야 합니다. 자세한 내용은 [AWS의 Cluster Autoscaler](#)를 참조하세요.

```
kubectl scale deployments/cluster-autoscaler --replicas=1 -n kube-system
```

15.(선택사항) 최신 버전의 [Kubernetes용 Amazon VPC CNI 플러그인](#)을 사용하고 있는지 확인합니다. 지원되는 최신 인스턴스 유형을 사용하려면 CNI 버전을 업데이트해야 할 수 있습니다. 자세한 내용은 [the section called “Amazon VPC CNI”](#) 섹션을 참조하세요.

16.클러스터가 DNS 확인에 kube-dns를 사용하는 경우,([[migrate-determine-dns-step](#)] 단계 참조) kube-dns 배포를 복제본 한 개로 축소합니다.

```
kubectl scale deployments/kube-dns --replicas=1 -n kube-system
```

AWS CloudFormation 노드 스택 업데이트

이 주제에서는 기존 AWS CloudFormation 자체 관리형 노드 스택을 새 AMI로 업데이트하는 방법을 설명합니다. 이 절차를 사용하여 클러스터 업데이트 후 새 버전의 Kubernetes로 노드를 업데이트할 수 있습니다. 그렇지 않으면 기존 Kubernetes 버전에 대해 최신 Amazon EKS 최적화 AMI로 업데이트할 수 있습니다.

Important

이 주제에서는 자체 관리형 노드 그룹에 대한 노드 업데이트에 대해 설명합니다. [관리형 노드 그룹에서 Simplify 노드 수명 주기](#)를 사용하는 방법에 대한 자세한 내용은 [the section called “업데이트”](#)을 참조하세요.

최신 기본 Amazon EKS 노드 AWS CloudFormation 템플릿은 예전 인스턴스를 제거하기 전에 한 번에 하나씩 새 AMI로 인스턴스를 시작하도록 구성되어 있습니다. 이렇게 구성하면 업데이트를 적용하는 동안 클러스터의 Auto Scaling 그룹에 항상 활성 인스턴스 수를 원하는 대로 항상 유지할 수 있습니다.

Note

이 방법은 eksctl로 생성한 노드 그룹에는 사용할 수 없습니다. eksctl을 사용하여 클러스터 또는 노드 그룹을 생성한 경우, [the section called “마이그레이션”](#) 부분을 참조하세요.

1. 클러스터의 DNS 공급자를 결정합니다.

```
kubectl get deployments -l k8s-app=kube-dns -n kube-system
```

예제 출력은 다음과 같습니다. 이 클러스터는 DNS 확인에 CoreDNS를 사용하고 있지만, 클러스터는 그 대신 kube-dns를 반환할 수 있습니다. 사용 중인 kubectl 버전에 따라 출력물이 다르게 보일 수 있습니다.

NAME	DESIRED	CURRENT	UP-TO-DATE	AVAILABLE	AGE
coredns	1	1	1	1	31m

2. 현재 배포가 두 개 미만의 복제본을 실행하고 있는 경우, 배포를 복제본 두 개로 확장합니다. 이전 명령 출력에서 해당 값이 대신 반환된 경우 **coredns**를 kube-dns로 교체합니다.

```
kubectl scale deployments/coredns --replicas=2 -n kube-system
```

3. (선택사항) [Kubernetes Cluster Autoscaler](#)를 사용하는 경우, 배포를 복제본 0개로 축소하여 조정 작업의 충돌을 방지합니다.

```
kubectl scale deployments/cluster-autoscaler --replicas=0 -n kube-system
```

4. 현재 노드 그룹의 인스턴스 유형과 원하는 인스턴스 수를 결정합니다. 나중에 그룹에 대한 AWS CloudFormation 템플릿을 업데이트할 때 이러한 값을 입력합니다.

- <https://console.aws.amazon.com/ec2/>에서 Amazon EC2 콘솔을 엽니다.
- 왼쪽 탐색 창에서 시작 구성을 선택하고 기존 노드 시작 구성에 대한 인스턴스 유형을 기록해 둡니다.
- 왼쪽 탐색 창에서 Auto Scaling 그룹을 선택하고 기존 노드 Auto Scaling 그룹에 대해 원하는 인스턴스 수를 적어둡니다.

5. [AWS CloudFormation 콘솔](#)을 엽니다.

6. 노드 그룹 스택을 선택한 다음 업데이트를 선택합니다.

7. 현재 템플릿 대치를 선택하고 Amazon S3 URL을 선택합니다.

8. Amazon S3 URL에서 다음 URL을 텍스트 영역에 붙여 넣어 최신 버전의 노드 AWS CloudFormation 템플릿을 사용하고 있는지 확인합니다. 그리고 다음을 선택합니다.

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2022-12-23/amazon-eks-nodegroup.yaml
```

9. 스택 세부 정보 지정 페이지에서 다음 파라미터를 입력하고 다음을 선택합니다.

- NodeAutoScalingGroupDesiredCapacity - [이전 단계](#)에서 기록한 원하는 인스턴스 수를 입력합니다. 또는 스택을 업데이트할 때 조정할 원하는 새 노드 수를 입력합니다.
- NodeAutoScalingGroupMaxSize - Auto Scaling 그룹이 확장할 수 있는 노드의 최대 노드 수를 입력합니다. 이 값은 원하는 용량보다 하나 이상의 노드여야 합니다. 이는 업데이트 중에 노드 수를 줄이지 않고 노드의 롤링 업데이트를 수행할 수 있도록 하기 위한 것입니다.
- NodeInstanceType - [이전 단계](#)에서 기록한 인스턴스 유형을 선택합니다. 또는 노드에 대해 다른 인스턴스 유형을 선택합니다. 다른 인스턴스 유형을 선택하기 전에 [최적의 Amazon EC2 노드 인스턴스 유형 선택](#)을 검토합니다. 각 Amazon EC2 인스턴스 유형은 탄력적 네트워크 인터페이스(네트워크 인터페이스)의 최대 수를 지원하며 각 네트워크 인터페이스는 최대 IP 주소 수를 지원합니다. 각 워커 노드와 포드에는 고유한 IP 주소가 할당되므로 각 Amazon EC2 노드에서 실행할 최대 포드 수를 지원하는 인스턴스 유형을 선택하는 것이 중요합니다. 인스턴스 유형이 지원하는 네트워크 인터페이스 및 IP 주소의 수 목록은 [인스턴스 유형별 네트워크 인터페이스당 IP 주소](#)를 참조하세요. 예를 들어, m5.large 인스턴스 유형은 워커 노드 및 포드에 대해 최대 30개의 IP 주소를 지원합니다.

Note

최신 버전의 [Kubernetes용 Amazon VPC CNI 플러그인](#)에 대해 지원되는 인스턴스 유형은 GitHub의 [vpc_ip_resource_limit.go](#)에 나와 있습니다. 지원되는 최신 인스턴스 유형을 사용하려면 Kubernetes용 Amazon VPC CNI 플러그인 버전을 업데이트해야 할 수 있습니다. 자세한 내용은 [the section called "Amazon VPC CNI"](#) 단원을 참조하십시오.

Important

AWS 리전에 따라 일부 인스턴스 유형은 사용할 수 없습니다.

- NodeImageIdSSMParam – 업데이트하려는 AMI ID의 Amazon EC2 Systems Manager 파라미터입니다. 다음 값에서는 Kubernetes 버전 1.32에 대해 최신 Amazon EKS 최적화 AMI를 사용합니다.

```
/aws/service/eks/optimized-ami/1.32/amazon-linux-2/recommended/image_id
```

1.32를 [지원되는 동일한 Kubernetes 버전](#)으로 바꿀 수 있습니다. 또는 제어 영역에서 실행 중인 Kubernetes 버전보다 최대 한 버전 이전이어야 합니다. 노드를 제어 영역과 동일한 버전으로 유지하는 것이 좋습니다. [amazon-linux-2](#)를 다른 AMI 유형으로 바꿀 수도 있습니다. 자세한 내용은 [the section called “최신 ID 받기”](#) 섹션을 참조하세요.

Note

Amazon EC2 Systems Manager 파라미터를 사용하면 AMI ID를 조회하고 지정하지 않아도 나중에 노드를 업데이트할 수 있습니다. AWS CloudFormation 스택에서 이 값을 사용 중이라면 모든 스택 업데이트는 지정된 Kubernetes 버전에 권장되는 최신 Amazon EKS 최적화 AMI를 항상 시작합니다. 템플릿에서 값을 변경하지 않은 경우에도 마찬가지입니다.

- NodeImageId – 고유한 사용자 지정 AMI를 사용하려면 사용할 AMI의 ID를 입력하세요.

Important

이 값은 NodeImageIdSSMParam에 지정된 값을 재정의합니다. NodeImageIdSSMParam 값을 사용하려면 NodeImageId의 값이 비어 있는지 확인하세요.

- DisableIMDSv1 - 기본적으로 각 노드는 인스턴스 메타데이터 서비스 버전 1(IMDSv1) 및 IMDSv2를 지원합니다. 그러나 IMDSv1을 사용 중지할 수 있습니다. 노드 그룹에서 예약된 노드 또는 포드가 IMDSv1을 사용하지 않도록 하려면 true를 선택합니다. IMDS에 대한 자세한 내용은 [인스턴스 메타데이터 서비스 구성](#)을 참조하세요. 서비스 계정에 대한 IAM 역할을 구현한 경우 AWS 서비스에 대한 액세스 권한이 필요한 모든 포드에 직접 필요한 권한을 할당합니다. 이렇게 하면 현재 AWS 리전 검색 등의 다른 이유로 클러스터의 포드가 IMDS에 액세스할 필요가 없습니다. 그런 다음 호스트 네트워킹을 사용하지 않는 포드에 대해 IMDSv2 액세스를 사용 중지할 수도 있습니다. 자세한 내용은 [작업자 노드에 할당된 인스턴스 프로필에 대한 액세스 제한](#) 부분을 참조하세요.

10(선택사항) 옵션 페이지에서 스택 리소스에 태그를 지정합니다. 다음을 선택합니다.

11.검토 페이지에서 정보를 검토하고, 스택이 IAM 리소스를 생성할 수 있는지 확인한 다음, 스택 업데이트를 선택합니다.

Note

클러스터의 각 노드를 업데이트하는 데 몇 분 정도 걸립니다. 모든 노드의 업데이트가 완료 될 때까지 기다린 후 다음 단계를 수행합니다.

12.클러스터의 DNS 공급자가 kube-dns인 경우 kube-dns 배포를 복제본 한 개로 축소합니다.

```
kubectl scale deployments/kube-dns --replicas=1 -n kube-system
```

13(선택 사항) Kubernetes [Cluster Autoscaler](#)를 사용하는 경우 배포를 다시 원하는 양의 복제본으로 조정합니다.

```
kubectl scale deployments/cluster-autoscaler --replicas=1 -n kube-system
```

14(선택사항) 최신 버전의 [Kubernetes용 Amazon VPC CNI 플러그인](#)을 사용하고 있는지 확인합니다. 지원되는 최신 인스턴스 유형을 사용하려면 Kubernetes용 Amazon VPC CNI 플러그인 버전을 업데이트해야 할 수 있습니다. 자세한 내용은 [the section called “Amazon VPC CNI”](#) 단원을 참조하십시오.

AWS Fargate를 사용한 컴퓨팅 관리 간소화

이 주제에서는 AWS Fargate에서 Amazon EKS를 사용하여 Kubernetes 포드를 실행하는 방법을 설명합니다. Fargate는 [컨테이너](#)에 대한 적정 규모의 온디맨드 컴퓨팅 용량을 제공하는 기술입니다. Fargate를 사용하면 컨테이너를 실행하려면 직접 가상 머신 그룹을 프로비저닝, 구성 또는 크기를 조정할 필요가 없습니다. 따라서 서버 유형을 선택하거나, 노드 그룹을 조정할 시점을 결정하거나, 클러스터 패킹을 최적화할 필요가 없습니다.

Fargate에서 시작하는 포드와 [Fargate 프로파일](#)에서 실행되는 방법을 제어할 수 있습니다. Fargate 프로파일은 Amazon EKS 클러스터의 일부로 정의됩니다. Amazon EKS는 Kubernetes에서 제공한 확장 가능한 업스트림 모델을 사용하여 AWS에서 빌드한 컨트롤러를 사용하여 Kubernetes를 Fargate와 통합합니다. 이러한 컨트롤러는 Amazon EKS 관리형 Kubernetes 컨트롤 플레인의 일부로 실행되며 Fargate에 기본 Kubernetes 포드를 예약합니다. Fargate 컨트롤러에는 여러 개의 변형 및 검증 승인 컨트롤러 외에도 기본 Kubernetes 스케줄러와 함께 실행되는 새 스케줄러가 포함되어 있습니다. Fargate에서 실행하기 위한 조건을 충족하는 포드를 시작하면 클러스터에서 실행되는 Fargate 컨트롤러가 해당 포드를 인식하고 업데이트하고 Fargate에 예약합니다.

이 주제에서는 Fargate에서 실행되는 포드의 여러 구성 요소에 대해 설명하고 Amazon EKS에서 Fargate를 사용하기 위한 특별한 고려 사항을 살펴봅니다.

AWS Fargate 고려 사항

Amazon EKS에서 Fargate를 사용할 때 고려해야 할 몇 가지 사항이 있습니다.

- Fargate에서 실행되는 각 포드에는 자체 격리 경계가 있습니다. 기본 커널, CPU 리소스, 메모리 리소스 또는 탄력적 네트워크 인터페이스를 다른 포드와 공유하지 않습니다.
- Network Load Balancer 및 Application Load Balancer(ALB)는 IP 대상을 통해서만 Fargate에서 사용할 수 있습니다. 자세한 내용은 [the section called “Network Load Balancer 생성”](#) 및 [the section called “애플리케이션 로드 밸런싱”](#) 섹션을 참조하세요.
- Fargate 노출 서비스는 노드 IP 모드가 아닌 대상 유형 IP 모드에서만 실행됩니다. 관리형 노드에서 실행 중인 서비스와 Fargate에서 실행 중인 서비스의 연결을 확인하는 권장 방법은 서비스 이름을 통해 연결하는 것입니다.
- 포드는 Fargate에서 실행되도록 예약된 시간에 Fargate 프로필과 일치해야 합니다. Fargate 프로필과 일치하지 않는 포드는 Pending과 같이 중단될 수 있습니다. 일치하는 Fargate 프로필이 있는 경우 사용자가 생성한 보류 중인 포드를 삭제하여 Fargate에 다시 예약할 수 있습니다.
- DaemonSets는 Fargate에서 지원되지 않습니다. 애플리케이션에 데몬이 필요한 경우 포드에서 사이드카 컨테이너로 실행되도록 해당 데몬을 다시 구성합니다.
- 권한 있는(Privileged) 컨테이너는 Fargate에서 지원되지 않습니다.
- Fargate에서 실행되는 포드는 포드 매니페스트에서 HostPort 또는 HostNetwork를 지정할 수 없습니다.
- Fargate 포드의 경우 기본 nofile 및 nproc 소프트웨어 제한은 1,024이고 하드 제한은 6만 5,535입니다.
- GPU는 현재 Fargate에서 사용할 수 없습니다.
- Fargate에서 실행되는 포드는 인터넷 게이트웨이에 대한 직접 경로가 없고 AWS 서비스에 대한 NAT 게이트웨이 액세스 권한이 있는 프라이빗 서브넷에서만 지원되므로 클러스터의 VPC에 프라이빗 서브넷이 있어야 합니다. 아웃바운드 인터넷 액세스가 없는 클러스터의 경우 [the section called “프라이빗 클러스터”](#)를 참조하세요.
- [Vertical Pod Autoscaler로 포드 리소스 조정](#)을 사용하여 Fargate 포드의 초기 올바른 CPU 및 메모리 크기를 설정한 다음 [Horizontal Pod Autoscaler로 포드 배포 확장](#)을 사용하여 해당 포드의 규모를 조정할 수 있습니다. Vertical Pod Autoscaler가 더 큰 CPU 및 메모리 조합으로 포드를 Fargate에 자동으로 다시 배포하도록 하려면 Vertical Pod Autoscaler의 모드를 Auto 또는 Recreate로 설정하여

올바로 작동하게 해야 합니다. 자세한 내용은 GitHub에서 [Vertical Pod Autoscaler](#) 문서를 참조하세요.

- VPC에 대해 DNS 확인 및 DNS 호스트 이름을 활성화해야 합니다. 자세한 내용은 [VPC에 대한 DNS 지원 보기 및 업데이트](#)를 참조하세요.
- Amazon EKS Fargate는 가상 머신(VM) 내에서 각 포드를 격리하여 Kubernetes 애플리케이션에 대한 심층 방어 기능을 추가합니다. 이 VM 경계를 통해 컨테이너 이스케이프 시 다른 포드에서 사용하는 호스트 기반 리소스에 대한 액세스를 방지하는데, 이 방법은 컨테이너화된 애플리케이션을 공격하고 컨테이너 외부의 리소스에 대한 액세스 권한을 얻는 일반적인 방법입니다.

Amazon EKS를 사용해도 [공동 책임 모델](#)에서 책임은 변경되지 않습니다. 클러스터 보안 및 거버넌스 제어의 구성을 신중하게 고려해야 합니다. 애플리케이션을 격리하는 가장 안전한 방법은 항상 별도의 클러스터에서 실행하는 것입니다.

- Fargate 프로파일은 VPC 보조 CIDR 블록의 서브넷 지정을 지원합니다. 보조 CIDR 블록을 지정할 수도 있습니다. 서브넷에서 사용 가능한 IP 주소의 수는 제한되어 있기 때문입니다. 따라서 클러스터에서 생성할 수 있는 포드의 수가 제한되어 있습니다. 포드에 다른 서브넷을 사용하여 사용 가능한 IP 주소의 수를 늘릴 수 있습니다. 자세한 내용은 [VPC에 IPv4 CIDR 블록 추가](#)를 참조하세요.
- Amazon EC2 인스턴스 메타데이터 서비스(IMDS)는 Fargate 노드에 배포된 포드에는 사용할 수 없습니다. IAM 자격 증명이 필요한 포드를 Fargate에 배포한 경우 [서비스 계정용 IAM 역할](#)을 사용하여 포드에 할당합니다. 포드가 IMDS를 통해 제공되는 다른 정보에 액세스해야 할 경우 이 정보를 포드 사양에 하드 코딩해야 합니다. 여기에는 포드가 배포된 AWS 리전 또는 가용 영역이 포함됩니다.
- Fargate 포드는 AWS Outposts, AWS Wavelength 또는 AWS 로컬 영역에 배포할 수 없습니다.
- Amazon EKS는 정기적으로 Fargate 포드를 패치하여 보안을 유지해야 합니다. 영향을 줄이는 방식으로 업데이트를 시도하지만 포드가 성공적으로 제거되지 않으면 포드를 삭제해야 하는 경우가 있습니다. 중단을 최소화하기 위해 취할 수 있는 몇 가지 조치가 있습니다. 자세한 내용은 [the section called “OS 패치 이벤트”](#) 섹션을 참조하세요.
- [Amazon EKS용 Amazon VPC CNI 플러그인](#)은 Fargate 노드에 설치됩니다. Fargate 노드가 있는 [Amazon EKS 클러스터에는 대체 CNI 플러그인](#)을 사용할 수 없습니다.
- Fargate에서 실행되는 포드는 수동 드라이버 설치 단계 없이 Amazon EFS 파일 시스템을 자동으로 탑재합니다. Fargate 노드에는 동적 영구 볼륨 프로비저닝을 사용할 수 없지만 고정적인 프로비저닝은 사용할 수 있습니다.
- Amazon EKS는 Fargate 스팟을 지원하지 않습니다.
- Amazon EBS 볼륨을 Fargate 포드에 탑재할 수 없습니다.
- Amazon EBS CSI 컨트롤러는 Fargate 노드에서 실행할 수 있지만 Amazon EBS CSI 노드 DaemonSet은 Amazon EC2 인스턴스에서만 실행할 수 있습니다.

- [Kubernetes 작업](#)이 Completed 또는 Failed로 표시된 후에도 작업이 생성하는 포드는 정상적으로 계속 존재합니다. 이 동작을 통해 로그와 결과를 볼 수 있지만 Fargate를 사용하는 경우 나중에 작업을 정리하지 않으면 비용이 발생합니다.

작업이 완료되거나 실패한 후 관련 포드를 자동으로 삭제하려면 TTL(Time-to-Live) 컨트롤러를 사용하여 기간을 지정할 수 있습니다. 다음 예제에서는 작업 매니페스트에 `.spec.ttlSecondsAfterFinished`를 지정하는 방법을 보여줍니다.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: busybox
spec:
  template:
    spec:
      containers:
      - name: busybox
        image: busybox
        command: ["/bin/sh", "-c", "sleep 10"]
        restartPolicy: Never
      ttlSecondsAfterFinished: 60 # <-- TTL controller
```

Fargate 비교 테이블

기준	AWS Fargate
AWS Outposts 에 배포 가능	아니요
AWS Local Zones 에 배포할 수 있습니다.	아니요
Windows가 필요한 컨테이너를 실행할 수 있습니다.	아니요
Linux가 필요한 컨테이너를 실행할 수 있습니다.	예
Inferentia 칩이 필요한 워크로드를 실행할 수 있습니다.	아니요
GPU가 필요한 워크로드를 실행할 수 있습니다.	아니요

기준	AWS Fargate
Arm 프로세서가 필요한 워크로드를 실행할 수 있습니다.	아니요
AWS Bottlerocket 을 실행할 수 있습니다.	아니요
포드가 커널 런타임 환경을 다른 포드와 공유합니다.	아니요 - 각 포드에는 전용 커널이 있습니다.
포드가 CPU, 메모리, 스토리지 및 네트워크 리소스를 다른 포드와 공유합니다.	아니요 - 각 포드에는 전용 리소스가 있으며 독립적으로 크기를 조정하여 리소스 활용도를 극대화할 수 있습니다.
포드가 포드 사양에서 요청한 것보다 더 많은 하드웨어 및 메모리를 사용할 수 있습니다.	아니요 - 더 큰 vCPU 및 메모리 구성을 사용하여 포드를 다시 배포할 수 있습니다.
Amazon EC2 인스턴스를 배포하고 관리해야 합니다.	아니요
Amazon EC2 인스턴스의 운영 체제를 보호, 유지 관리 및 패치해야 합니다.	아니요
노드 배포 시 추가 kubelet 인수와 같은 부트스트랩 인수를 제공할 수 있습니다.	아니요
노드에 할당된 IP 주소와 다른 CIDR 블록의 포드에 IP 주소를 할당할 수 있습니다.	아니요
노드에 SSH를 연결할 수 있습니다.	아니요 - SSH에 사용할 노드 호스트 운영 체제가 없습니다.
사용자 지정 AMI 노드에 배포할 수 있습니다.	아니요
사용자 지정 CNI 노드에 배포할 수 있습니다.	아니요
노드 AMI를 직접 업데이트해야 합니다.	아니요
노드 Kubernetes 버전을 직접 업데이트해야 합니다.	아니요 — 노드를 관리하지 않습니다.

기준	AWS Fargate
포드에 Amazon EBS 스토리지를 사용할 수 있습니다.	아니요
포드에 Amazon EFS 스토리지를 사용할 수 있습니다.	예
포드에 Amazon FSx for Lustre 스토리지를 사용할 수 있습니다.	아니요
서비스에 Network Load Balancer를 사용할 수 있습니다.	예, 네트워크 로드 밸런서 생성 사용 시
포드가 퍼블릭 서브넷에서 실행될 수 있습니다.	아니요
개별 포드에 서로 다른 VPC 보안 그룹을 할당할 수 있습니다.	예
Kubernetes DaemonSet를 실행할 수 있습니다.	아니요
포드 매니페스트에서 HostPort 및 HostNetwork 를 지원할 수 있습니다.	아니요
AWS 리전 가용성	일부 Amazon EKS 지원 리전
Amazon EC2 전용 호스트에서 컨테이너를 실행할 수 있습니다.	아니요
요금	개별 Fargate 메모리 및 CPU 구성 비용입니다. 각 포드에는 자체 비용이 있습니다. 자세한 내용은 AWS Fargate 요금 을 참조하세요.

클러스터를 위한 AWS Fargate 시작하기

이 주제에서는 Amazon EKS 클러스터를 사용하여 AWS Fargate에서 포드 실행을 시작하는 방법을 설명합니다.

CIDR 블록을 사용하여 클러스터의 퍼블릭 엔드포인트에 대한 액세스를 제한하는 경우 프라이빗 엔드포인트 액세스도 사용 설정하는 것이 좋습니다. 이렇게 하면 Fargate 포드가 클러스터와 통신할 수 있습니다. 프라이빗 엔드포인트를 활성화하지 않은 경우, 퍼블릭 액세스에 대해 지정하는 CIDR 블록에는 VPC의 아웃바운드 소스가 포함되어야 합니다. 자세한 내용은 [the section called “클러스터 엔드포인트 액세스”](#) 섹션을 참조하세요.

전제 조건

기존 클러스터가 있어야 합니다. Amazon EKS 클러스터가 아직 없는 경우 [시작](#) 부분을 참조하세요.

1단계: 기존 노드가 Fargate 포드와 통신할 수 있는지 확인

노드가 없는 새 클러스터 또는 [the section called “관리형 노드 그룹”](#) 관리형 노드 그룹만 있는 클러스터로 작업하는 경우 [the section called “2단계: Fargate 포드 실행 역할 생성”](#)로 건너뛸 수 있습니다.

이미 연결된 노드가 있는 기존 클러스터로 작업하고 있다고 가정합니다. 이러한 노드의 포드가 Fargate에서 실행되는 포드와 자유롭게 통신할 수 있는지 확인합니다. Fargate에서 실행되는 포드는 연결된 클러스터에 대해 클러스터 보안 그룹을 사용하도록 자동으로 구성됩니다. 클러스터의 기존 노드가 클러스터 보안 그룹과 트래픽을 주고 받을 수 있는지 확인합니다. 관리형 노드 그룹은 클러스터 보안 그룹도 사용하도록 자동으로 구성되므로 이 호환성을 수정하거나 확인할 필요가 없습니다([the section called “관리형 노드 그룹”](#) 참조).

eksctl 또는 Amazon EKS 관리형 AWS CloudFormation 템플릿을 사용하여 생성한 기존 노드 그룹의 경우 클러스터 보안 그룹을 노드에 수동으로 추가할 수 있습니다. 또는 노드 그룹에 대한 Auto Scaling 그룹 시작 템플릿을 수정하여 클러스터 보안 그룹을 인스턴스에 연결할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 [인스턴스의 보안 그룹 변경](#) 부분을 참조하세요.

클러스터에 대한 네트워킹 섹션의 AWS Management Console에서 해당 클러스터에 대한 보안 그룹을 확인할 수 있습니다. 또는 다음 AWS CLI 명령을 사용하여 확인할 수 있습니다. 이 명령을 사용하는 경우 <my-cluster>를 클러스터의 이름으로 바꿉니다.

```
aws eks describe-cluster --name <my-cluster> --query
cluster.resourcesVpcConfig.clusterSecurityGroupId
```

2단계: Fargate 포드 실행 역할 생성

클러스터가 AWS Fargate에서 포드를 생성하는 경우 Fargate 인프라에서 실행되는 구성 요소는 사용자를 대신하여 AWS API를 직접적으로 직접 호출해야 합니다. Amazon EKS 포드 실행 역할은 이 작업을 수행할 수 있는 IAM 권한을 제공합니다. AWS Fargate 포드 실행 역할을 생성하려면 [the section called “포드 실행 IAM 역할”](#) 섹션을 참조하세요.

Note

--fargate 옵션을 사용하여 eksctl로 클러스터를 생성한 경우 클러스터에는 이미 패턴 eksctl-my-cluster-FargatePodExecutionRole-ABCDEFGHijkl에서 IAM 콘솔을 찾을 수 있는 포드 실행 역할이 있습니다. 마찬가지로 eksctl을 사용하여 Fargate 프로필을 생성하는 경우 포드 실행 역할이 생성되지 않았으면 eksctl을 사용하여 포드 실행 역할을 생성합니다.

3단계: 클러스터에 대한 Fargate 프로필 생성

클러스터의 Fargate에서 실행되는 포드를 예약하려면 먼저 포드가 시작될 때 Fargate를 사용할 포드를 지정하는 Fargate 프로필을 정의해야 합니다. 자세한 내용은 [the section called “프로필 정의”](#) 섹션을 참조하세요.

Note

--fargate 옵션을 사용하여 eksctl로 클러스터를 생성한 경우 kube-system 및 default 네임스페이스의 모든 포드에 대한 선택기를 사용하여 클러스터에 대한 Fargate 프로필이 이미 생성되어 있습니다. Fargate에서 사용할 다른 네임스페이스에 대한 Fargate 프로필을 생성하려면 다음 절차를 따르세요.

다음 도구 중 하나를 사용하여 Fargate 프로필을 생성할 수 있습니다.

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)

eksctl

이 절차에는 eksctl 버전 0.210.0 이상이 필요합니다. 버전은 다음 명령을 통해 확인할 수 있습니다.

```
eksctl version
```

eksctl 설치 또는 업데이트에 대한 지침은 eksctl 설명서의 [Installation](#)를 참조하세요.

eksctl을 사용하여 Fargate 프로필을 생성하려면

다음 `eksctl` 명령으로 Fargate 프로파일을 생성하고 모든 `<example value>`를 고유한 값으로 바꿉니다. 네임스페이스를 지정해야 합니다. 그러나 `--labels` 옵션은 필요하지 않습니다.

```
eksctl create fargateprofile \
  --cluster <my-cluster> \
  --name <my-fargate-profile> \
  --namespace <my-kubernetes-namespace> \
  --labels <key=value>
```

`<my-kubernetes-namespace>` 및 `<key=value>` 레이블에 특정 와일드카드를 사용할 수 있습니다. 자세한 내용은 [the section called “Fargate 프로파일 와일드카드”](#) 섹션을 참조하세요.

AWS Management Console

AWS Management Console을 사용하여 Fargate 프로파일을 생성하려면

1. [Amazon EKS 콘솔](#)을 엽니다.
2. Fargate 프로파일을 생성할 클러스터를 선택합니다.
3. 컴퓨팅 탭을 선택합니다.
4. Fargate 프로파일에서 Fargate 프로파일 추가를 선택합니다.
5. Fargate 프로파일 구성 페이지에서 다음을 수행합니다.
 - a. 이름에 Fargate 프로파일의 이름을 입력합니다. 이름은 고유해야 합니다.
 - b. 포드 실행 역할에서 Fargate 프로파일과 함께 사용할 포드 실행 역할을 선택합니다. `eks-fargate-pods.amazonaws.com` 서비스 보안 주체가 있는 IAM 역할만 표시됩니다. 나열된 역할이 표시되지 않으면 역할을 만들어야 합니다. 자세한 내용은 [the section called “포드 실행 IAM 역할”](#) 섹션을 참조하세요.
 - c. 선택한 서브넷을 필요에 따라 수정합니다.

Note

Fargate에서 실행되는 포드에는 프라이빗 서브넷만 지원됩니다.

- d. 태그의 경우, 선택적으로 Fargate 프로파일에 태그를 지정할 수 있습니다. 이러한 태그는 프로파일과 연결된 다른 리소스(예: 포드)에 전파되지 않습니다.
 - e. 다음을 선택합니다.
6. 포드 선택 구성 페이지에서 다음을 수행합니다.
 - a. 네임스페이스에 포드에 대해 일치시킬 네임스페이스를 입력합니다.

- kube-system 또는 default와 같은 특정 네임스페이스를 사용하여 일치시킬 수 있습니다.
 - 특정 와일드카드(예: prod-*)를 사용하여 여러 네임스페이스(예: prod-deployment 및 prod-test)를 일치시킬 수 있습니다. 자세한 내용은 [the section called “Fargate 프로필 와일드카드”](#) 섹션을 참조하세요.
- b. (선택 사항) 선택기에 Kubernetes 레이블을 추가합니다. 특히 지정된 네임스페이스의 포드가 일치해야 하는 포드에 추가합니다.
- infrastructure: fargate 레이블도 있는 지정된 네임스페이스의 포드만 선택기와 일치하도록 infrastructure: fargate Kubernetes 레이블을 선택기에 추가할 수 있습니다.
 - 특정 와일드카드(예: key?: value?)를 사용하여 여러 네임스페이스(예: keya: valuea 및 keyb: valueb)를 일치시킬 수 있습니다. 자세한 내용은 [the section called “Fargate 프로필 와일드카드”](#) 섹션을 참조하세요.
- c. 다음을 선택합니다.
7. 검토 및 생성 페이지에서 Fargate 프로필에 대한 정보를 검토하고 생성을 선택합니다.

4단계: CoreDNS 업데이트

기본적으로 CoreDNS는 Amazon EKS 클러스터의 Amazon EC2 인프라에서 실행되도록 구성됩니다. 클러스터의 Fargate에서만 포드를 실행하려면 다음 단계를 수행하세요.

Note

--fargate 옵션을 사용하여 eksctl로 클러스터를 생성한 경우 [the section called “다음 단계”](#)로 건너뛸 수 있습니다.

1. 다음 명령을 사용하여 CoreDNS에 대한 Fargate 프로필을 삭제합니다. <my-cluster>를 클러스터 이름으로, <111122223333>을 계정 ID로, <AmazonEKSFargatePodExecutionRole>을 포드 실행 역할 이름으로, <0000000000000000a>, <0000000000000000b>, <0000000000000000c>를 프라이빗 서브넷의 ID로 바꿉니다. 포드 실행 역할이 없는 경우 먼저 생성해야 합니다([the section called “2단계: Fargate 포드 실행 역할 생성”](#) 참조).

Important

역할 ARN에 / 이외의 [경로](#)를 포함할 수 없습니다. 예를 들어, 역할 이름이 development/apps/AmazonEKSFargatePodExecutionRole인 경우 역할에 대한 ARN을 지정할 때 역할 이름을 AmazonEKSFargatePodExecutionRole로 변경해

아 합니다. 역할 ARN의 형식은 `arn:aws:iam::<111122223333>:role/<AmazonEKSFargatePodExecutionRole>`이어야 합니다.

```
aws eks create-fargate-profile \
  --fargate-profile-name coredns \
  --cluster-name <my-cluster> \
  --pod-execution-role-arn arn:aws:iam::<111122223333>:role/
  <AmazonEKSFargatePodExecutionRole> \
  --selectors namespace=kube-system,labels={k8s-app=kube-dns} \
  --subnets subnet-<0000000000000000a> subnet-<0000000000000000b> subnet-
  <0000000000000000c>
```

2. coredns 배포의 롤아웃을 트리거합니다.

```
kubectl rollout restart -n kube-system deployment coredns
```

다음 단계

- 다음 워크플로를 사용하여 Fargate에서 실행할 기존 애플리케이션 마이그레이션을 시작할 수 있습니다.
 - a. 애플리케이션의 Kubernetes 네임스페이스 및 Kubernetes 레이블과 일치하는 [the section called “Fargate 프로파일 생성”](#).
 - b. Fargate에 예약되도록 기존 포드를 삭제하고 다시 생성합니다. <namespace>와 <deployment-type>을 수정하여 특정 포드를 업데이트합니다.

```
kubectl rollout restart -n <namespace> deployment <deployment-type>
```

- [the section called “애플리케이션 로드 밸런싱”](#)를 배포하여 Fargate에서 실행되는 포드에 대해 수신 객체를 허용합니다.
- [the section called “Vertical Pod Autoscaler”](#)를 사용하여 Fargate 포드의 올바른 초기 CPU 및 메모리 크기를 설정한 다음 [the section called “Horizontal Pod Autoscaler”](#)를 사용하여 해당 포드의 규모를 조정할 수 있습니다. Vertical Pod Autoscaler가 더 높은 CPU 및 메모리 조합으로 포드를 Fargate에 자동으로 다시 배포하도록 하려면 Vertical Pod Autoscaler의 모드를 Auto 또는 Recreate로 설정합니다. 이를 통해 기능을 올바르게 사용할 수 있습니다. 자세한 내용은 GitHub에서 [Vertical Pod Autoscaler](#) 문서를 참조하세요.

- [이러한 지침](#)을 따라 애플리케이션을 모니터링하도록 [AWS Distro for OpenTelemetry\(ADOT\)](#) 컬렉터를 설정할 수 있습니다.

시작 시 AWS Fargate를 사용하는 포드 정의

클러스터의 Fargate에 포드를 예약하기 전에 먼저 포드가 시작될 때 Fargate를 사용할 포드를 지정하는 Fargate 프로필을 하나 이상 정의해야 합니다.

관리자는 Fargate 프로필을 사용하여 Fargate에서 실행되는 포드를 선언할 수 있습니다. 프로필의 선택기를 통해 이 작업을 수행할 수 있습니다. 각 프로필에 최대 5개의 셀렉터를 추가할 수 있습니다. 각 셀렉터에 네임스페이스를 포함해야 합니다. 레이블도 셀렉터에 포함될 수 있습니다. 레이블 필드는 여러 개의 선택적 키-값 페어로 구성됩니다. 셀렉터와 일치하는 포드는 Fargate에 예약되어 있습니다. 포드는 선택기에 지정된 네임스페이스와 레이블을 사용하여 일치됩니다. 네임스페이스 선택기가 레이블 없이 정의되면 Amazon EKS는 프로필을 사용하여 해당 네임스페이스에서 실행되는 모든 포드를 Fargate에 예약하려고 시도합니다. 예약할 포드가 Fargate 프로필의 선택기와 일치하는 경우 해당 포드가 Fargate에 예약됩니다.

포드가 여러 Fargate 프로필과 일치하는 경우 포드 사양에 Kubernetes 레이블 `eks.amazonaws.com/fargate-profile: my-fargate-profile`을 추가하여 포드가 사용하는 프로필을 지정할 수 있습니다. 포드는 Fargate에 예약하기 위해 해당 프로필의 선택기와 일치해야 합니다. Kubernetes 선호도/반선호도 규칙은 Amazon EKS Fargate 포드에 적용되지 않으며 불필요합니다.

Fargate 프로필을 생성할 때 포드 실행 역할을 지정해야 합니다. 이 실행 역할은 프로필을 사용하여 Fargate 인프라에서 실행되는 Amazon EKS 구성 요소를 위한 것입니다. 이 역할은 권한 부여를 위해 클러스터의 Kubernetes [역할 기반 액세스 컨트롤\(RBAC\)](#)에 추가됩니다. 이렇게 하면 Fargate 인프라에서 실행되는 kubelet이 Amazon EKS 클러스터에 등록되어 클러스터에 노드로 표시될 수 있습니다. 또한 포드 실행 역할은 Amazon ECR 이미지 리포지토리에 대한 읽기 액세스를 허용하는 Fargate 인프라에 대한 IAM 권한을 제공합니다. 자세한 내용은 [the section called “포드 실행 IAM 역할”](#) 단원을 참조하십시오.

Fargate 프로필은 변경할 수 없습니다. 그러나 업데이트된 프로필을 새로 생성하여 기존 프로필을 바꾼 다음 원본을 삭제할 수 있습니다.

Note

Fargate 프로필을 사용하여 실행 중인 모든 포드는 중지되고 프로필이 삭제되면 보류 상태로 전환됩니다.

클러스터의 Fargate 프로필이 DELETING 상태인 경우 해당 클러스터에 다른 프로필을 생성하려면 Fargate 프로필이 삭제될 때까지 기다려야 합니다.

Note

Fargate는 현재 Kubernetes [topologySpreadConstraints](#)를 지원하지 않습니다.

Amazon EKS 및 Fargate는 Fargate 프로필에 정의된 각 서브넷에 포드를 분산시킵니다. 그러나 고르게 분산되지 않을 수 있습니다. 고른 분산이 필요한 경우 Fargate 프로필을 2개 사용합니다. 복제본을 2개 배포하려고 하고 가동 중지를 원하지 않는 경우 고른 분산이 중요합니다. 각 프로필에는 서브넷이 하나만 있는 것이 좋습니다.

Fargate 프로필 구성 요소

다음 구성 요소는 Fargate 프로필에 포함되어 있습니다.

포드 실행 역할

클러스터가 AWS Fargate에서 포드를 생성하면, Fargate 인프라에서 실행 중인 kubelet가 사용자를 대신하여 AWS API를 직접적으로 호출해야 합니다. 예를 들어, Amazon ECR에서 컨테이너 이미지를 가져오기 위해 호출해야 합니다. Amazon EKS 포드 실행 역할은 이 작업을 수행할 수 있는 IAM 권한을 제공합니다.

Fargate 프로필을 생성할 때 포드와 함께 사용할 포드 실행 역할을 지정해야 합니다. 이 역할은 권한 부여를 위해 클러스터의 Kubernetes [역할 기반 액세스 제어](#)(RBAC)에 추가됩니다. 이렇게 하면 Fargate 인프라에서 실행 중인 kubelet이 Amazon EKS 클러스터에 등록되어 클러스터에 노드로 표시될 수 있습니다. 자세한 내용은 [the section called “포드 실행 IAM 역할”](#) 섹션을 참조하세요.

서브넷

이 프로필을 사용하는 포드를 시작할 서브넷의 ID입니다. 현재 Fargate에서 실행 중인 포드에는 퍼블릭 IP 주소가 할당되지 않습니다. 따라서 이 파라미터에 대해서는 인터넷 게이트웨이로 가는 직접 경로가 없는 프라이빗 서브넷만 허용됩니다.

선택기

포드가 이 Fargate 프로필을 사용하기 위해 일치시킬 선택기입니다. Fargate 프로필에 선택기를 최대 5개까지 지정할 수 있습니다. 셀렉터에는 다음 구성 요소가 있습니다.

- 네임스페이스 - 선택기에 대한 네임스페이스를 지정해야 합니다. 선택기는 이 네임스페이스에서 생성된 포드만 일치시킵니다. 그러나 여러 선택터를 생성하여 여러 네임스페이스를 대상으로 지정할 수 있습니다.
- 레이블 - 선택적으로 선택기에 대해 일치시킬 Kubernetes 레이블을 지정할 수 있습니다. 선택기는 선택기에 지정된 모든 레이블이 있는 포드만 일치시킵니다.

Fargate 프로파일 와일드카드

Kubernetes에서 허용하는 문자 외에도 네임스페이스, 레이블 키 및 레이블 값에 대한 선택기 기준에 *와 ?를 사용할 수 있습니다.

- *는 없음, 하나 또는 여러 문자를 나타냅니다. 예를 들어, prod*는 prod와 prod-metrics를 나타낼 수 있습니다.
- ?는 단일 문자를 나타냅니다. 예를 들어, value?는 valuea를 나타낼 수 있습니다. 그러나 ?는 정확히 하나의 문자만 나타낼 수 있기 때문에 value와 value-a를 나타낼 수는 없습니다.

이러한 와일드카드 문자는 모든 위치에서 조합하여 사용할 수 있습니다(예: prod*, *dev 및 frontend*?). 다른 와일드카드와 패턴 일치 형식(정규식 등)은 지원되지 않습니다.

포드 사양의 네임스페이스 및 레이블에 대해 일치하는 프로파일이 여러 개 있는 경우 Fargate는 프로파일 이름별 영숫자 정렬을 기준으로 프로파일을 선택합니다. 예를 들어, 프로파일 A(이름 beta-workload)와 프로파일 B(이름 prod-workload)에 시작할 포드에 대해 일치하는 선택기가 있는 경우 Fargate는 포드에 대해 프로파일 A(beta-workload)를 선택합니다. 포드에 프로파일 A라는 레이블이 있습니다(예: eks.amazonaws.com/fargate-profile=beta-workload).

와일드카드를 사용하는 새 프로파일로 기존 Fargate 포드를 마이그레이션하려는 경우 다음 두 가지 방법이 있습니다.

- 일치하는 선택터로 새 프로파일을 생성한 다음 이전 프로파일을 삭제합니다. 이전 프로파일로 레이블이 지정된 포드는 일치하는 새 프로파일로 다시 예약됩니다.
- 워크로드를 마이그레이션하고 싶지만 각 Fargate 포드에 어떤 Fargate 레이블이 있는지 확실하지 않은 경우 다음 방법을 사용할 수 있습니다. 동일한 클러스터의 프로파일 중에서 영숫자순으로 먼저 정렬되는 이름으로 새 프로파일을 생성합니다. 그런 다음 새 프로파일로 마이그레이션해야 하는 Fargate 포드를 재할용합니다.

Fargate 프로파일 생성

이 섹션에서는 Fargate 프로파일을 생성하는 방법을 설명합니다. 또한 Fargate 프로파일에 사용할 포드 실행 역할을 생성해야 합니다. 자세한 내용은 [the section called “포드 실행 IAM 역할”](#) 단원을 참조하십시오. Fargate에서 실행되는 포드는 인터넷 게이트웨이에 대한 직접 경로가 없고 AWS 서비스에 대한 [NAT 게이트웨이](#) 액세스 권한이 있는 프라이빗 서브넷에서만 지원됩니다. 이를 위해서는 클러스터의 VPC에 사용 가능한 프라이빗 서브넷이 있어야 합니다.

다음을 사용하여 프로파일을 만들 수 있습니다.

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)

eksctl

eksctl을 사용하여 Fargate 프로파일을 생성하려면

다음 **eksctl** 명령으로 Fargate 프로파일을 생성하고 모든 예제 값을 고유한 값으로 바꿉니다. 네임스페이스를 지정해야 합니다. 그러나 `--labels` 옵션은 필요하지 않습니다.

```
eksctl create fargateprofile \  
  --cluster my-cluster \  
  --name my-fargate-profile \  
  --namespace my-kubernetes-namespace \  
  --labels key=value
```

`my-kubernetes-namespace` 및 `key=value` 레이블에 특정 와일드카드를 사용할 수 있습니다. 자세한 내용은 [the section called “Fargate 프로파일 와일드카드”](#) 단원을 참조하십시오.

AWS Management Console

AWS Management Console을 사용하여 Fargate 프로파일을 생성하려면

1. [Amazon EKS 콘솔](#)을 엽니다.
2. Fargate 프로파일을 생성할 클러스터를 선택합니다.
3. 컴퓨팅 탭을 선택합니다.
4. Fargate 프로파일에서 Fargate 프로파일 추가를 선택합니다.
5. Fargate 프로파일 구성(Configure Fargate profile) 페이지에서 다음을 수행합니다.

- a. 이름(Name)에 Fargate 프로파일(예: my-profile)의 고유한 이름을 입력합니다.
- b. 포드 실행 역할에서 Fargate 프로파일과 함께 사용할 포드 실행 역할을 선택합니다. eks-fargate-pods.amazonaws.com 서비스 보안 주체가 있는 IAM 역할만 표시됩니다. 나열된 역할이 표시되지 않으면 역할을 만들어야 합니다. 자세한 내용은 [the section called “포드 실행 IAM 역할”](#) 섹션을 참조하세요.
- c. 선택한 서브넷을 필요에 따라 수정합니다.

 Note

Fargate에서 실행되는 포드에는 프라이빗 서브넷만 지원됩니다.

- d. 태그의 경우, 선택적으로 Fargate 프로파일에 태그를 지정할 수 있습니다. 이러한 태그는 프로파일과 연결된 다른 리소스(예: 포드)에 전파되지 않습니다.
 - e. Next(다음)를 선택합니다.
6. 포드 선택 구성 페이지에서 다음을 수행합니다.
- a. 네임스페이스에 포드에 대해 일치시킬 네임스페이스를 입력합니다.
 - kube-system 또는 default와 같은 특정 네임스페이스를 사용하여 일치시킬 수 있습니다.
 - 특정 와일드카드(예: prod-*)를 사용하여 여러 네임스페이스(예: prod-deployment 및 prod-test)를 일치시킬 수 있습니다. 자세한 내용은 [the section called “Fargate 프로파일 와일드카드”](#) 단원을 참조하십시오.
 - b. (선택 사항) 선택기에 Kubernetes 레이블을 추가합니다. 특히 지정된 네임스페이스의 포드가 일치해야 하는 포드에 추가합니다.
 - infrastructure: fargate 레이블도 있는 지정된 네임스페이스의 포드만 선택기와 일치하도록 infrastructure: fargate Kubernetes 레이블을 선택기에 추가할 수 있습니다.
 - 특정 와일드카드(예: key?: value?)를 사용하여 여러 네임스페이스(예: keya: valuea 및 keyb: valueb)를 일치시킬 수 있습니다. 자세한 내용은 [the section called “Fargate 프로파일 와일드카드”](#) 섹션을 참조하세요.
 - c. 다음을 선택합니다.

7. 검토 및 생성 페이지에서 Fargate 프로파일에 대한 정보를 검토하고 생성을 선택합니다.

Fargate 프로파일 삭제

이 주제에서는 Fargate 프로파일을 삭제하는 방법을 설명합니다. Fargate 프로파일을 삭제하면 해당 프로파일을 사용하여 Fargate에 예약된 모든 포드가 삭제됩니다. 이러한 포드가 다른 Fargate 프로파일과

일치하면 해당 프로필을 사용하여 Fargate에 예약됩니다. Fargate 프로필과 더이상 일치하지 않으면 Fargate에 예약되지 않고 보류 상태로 남아있을 수 있습니다.

클러스터의 Fargate 프로필은 한 번에 하나만 DELETING 상태가 될 수 있습니다. Fargate 프로필이 삭제될 때까지 기다려야 해당 클러스터에서 다른 프로필을 삭제할 수 있습니다.

다음 도구 중 하나를 사용하여 프로필을 삭제할 수 있습니다.

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)
- [the section called “AWS CLI”](#)

eksctl

eksctl를 사용하여 Fargate 프로필 삭제

다음 명령을 사용하여 클러스터에서 프로필을 삭제합니다. 모든 **##** **#**을 자신의 값으로 바꾸세요.

```
eksctl delete fargateprofile --name my-profile --cluster my-cluster
```

AWS Management Console

AWS Management Console를 사용하여 Fargate 프로필 삭제

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 좌측 탐색 창에서 클러스터를 선택합니다. 목록 클러스터에서 Fargate 프로필을 삭제할 클러스터를 선택합니다.
3. 컴퓨팅 탭을 선택합니다.
4. 삭제할 Fargate 프로필을 선택한 다음 삭제를 선택합니다.
5. Fargate 프로필 삭제 페이지에서 프로필 이름을 입력한 다음 삭제를 선택합니다.

AWS CLI

AWS CLI를 사용하여 Fargate 프로필 삭제

다음 명령을 사용하여 클러스터에서 프로필을 삭제합니다. 모든 **##** **#**을 자신의 값으로 바꾸세요.

```
aws eks delete-fargate-profile --fargate-profile-name my-profile --cluster-name my-cluster
```

Fargate 포드 구성 세부 정보 이해

이 섹션에서는 AWS Fargate에서 Kubernetes 포드를 실행하기 위한 몇 가지 고유한 포드 구성 세부 정보에 대해 설명합니다.

포드 CPU 및 메모리

Kubernetes를 사용하면 포드의 각 컨테이너에 할당되는 요청, 즉 최소량의 vCPU 및 메모리 리소스를 정의할 수 있습니다. 최소한 각 포드에 대해 요청된 리소스를 컴퓨팅 리소스에서 사용할 수 있도록 Kubernetes에서 포드를 예약합니다. 자세한 내용은 Kubernetes 설명서의 [컨테이너의 컴퓨팅 리소스 관리](#)를 참조하십시오.

Note

Amazon EKS Fargate는 노드당 하나의 포드만 실행하므로 리소스가 더 적은 경우 포드를 제거하는 시나리오가 발생하지 않습니다. 모든 Amazon EKS Fargate 포드는 보장된 우선순위로 실행되므로 요청된 CPU와 메모리는 모든 컨테이너의 한도와 같아야 합니다. 자세한 내용은 쿠버네티스 문서의 [Configure Quality of Service for Pods](#)를 참조하세요.

포드가 Fargate에 예약되면 포드 사양 내의 vCPU 및 메모리 예약에 따라 포드에 프로비저닝할 CPU 및 메모리 양이 결정됩니다.

- Init 컨테이너의 최대 요청은 Init 요청 vCPU 및 메모리 요구 사항을 결정하는 데 사용됩니다.
- 장기 실행 요청 vCPU 및 메모리 요구 사항을 결정하기 위해 모든 장기 실행 컨테이너에 대한 요청이 추가됩니다.
- 포드에 사용할 vCPU 및 메모리 요청에 대해서는 이전의 두 값 중 큰 값이 선택됩니다.
- Fargate는 필요한 Kubernetes 구성 요소(kubelet, kube-proxy 및 containerd)에 대한 각 포드의 메모리 예약에 256MB를 추가합니다.

Fargate는 포드가 실행해야 하는 리소스를 항상 보유하도록 vCPU 및 메모리 요청의 합계와 가장 일치하는 다음의 컴퓨팅 구성으로 올림 처리합니다.

vCPU 및 메모리 조합을 지정하지 않으면 사용 가능한 가장 작은 조합(.25 vCPU 및 0.5GB 메모리)이 사용됩니다.

아래 표에는 Fargate에서 실행되는 포드에 사용할 수 있는 vCPU 및 메모리 조합이 나와 있습니다.

vCPU 값	메모리 값
.25 vCPU	0.5GB, 1GB, 2GB
.5 vCPU	1GB, 2GB, 3GB, 4GB
1 vCPU	2GB, 3GB, 4GB, 5GB, 6GB, 7GB, 8GB
2 vCPU	4~16GB(1GB 증분)
4 vCPU	8~30GB(1GB 증분)
8 vCPU	16~60GB(4GB 단위)
16 vCPU	32~120GB(8GB 단위)

Kubernetes 구성 요소에 예약된 추가 메모리로 인해 요청된 것보다 많은 vCPU를 사용하는 Fargate 태스크가 프로비저닝될 수 있습니다. 예를 들어 1개의 vCPU 및 8GB 메모리에 대한 요청의 경우 해당 메모리 요청에 256MB가 추가되며, vCPU 1개와 9GB 메모리를 사용하는 태스크가 없으므로 2개의 vCPU 및 9GB 메모리로 Fargate 태스크를 프로비저닝합니다.

Fargate에서 실행 중인 포드의 크기와 `kubectl get nodes`를 사용하여 Kubernetes가 보고하는 노드 크기 사이에는 상관 관계가 없습니다. 보고된 노드 크기는 대개 포드의 용량보다 큼니다. 다음 명령을 사용하여 포드 용량을 확인할 수 있습니다. `default`를 포드의 네임스페이스로, `pod-name`을 포드의 이름으로 바꿉니다.

```
kubectl describe pod --namespace default pod-name
```

예제 출력은 다음과 같습니다.

```
[...]
annotations:
  CapacityProvisioned: 0.25vCPU 0.5GB
[...]
```

CapacityProvisioned 주석은 적용된 포드 용량을 나타내며 Fargate에서 실행되는 포드의 비용을 결정합니다. 이러한 컴퓨팅 구성에 대한 요금 정보는 [AWS Fargate 요금](#)을 참조하세요.

Fargate 스토리지

Fargate에서 실행되는 포드는 수동 드라이버 설치 단계 없이 Amazon EFS 파일 시스템을 자동으로 탑재합니다. Fargate 노드에는 동적 영구 볼륨 프로비저닝을 사용할 수 없지만 고정적인 프로비저닝은 사용할 수 있습니다. 자세한 내용은 GitHub에서 [Amazon EFS CSI 드라이버](#)를 참조하세요.

프로비저닝이 완료되면 Fargate에서 실행되는 각 포드는 기본적으로 20GiB의 임시 스토리지를 받습니다. 이 유형의 스토리지는 포드가 중지된 후에 삭제됩니다. Fargate에서 시작된 새 포드에는 임시 스토리지 볼륨의 암호화가 기본적으로 활성화되어 있습니다. 임시 포드 스토리지는 AWS Fargate 관리형 키를 사용하여 AES-256 암호화 알고리즘으로 암호화됩니다.

Note

Fargate에서 실행되는 Amazon EKS 포드의 기본 가용 스토리지는 20GiB 미만입니다. 이는 일부 공간을 kubelet 및 포드 내부에 로드되는 기타 Kubernetes 모듈에서 사용하기 때문입니다.

임시 스토리지의 총량은 최대 175GiB까지 높일 수 있습니다. Kubernetes를 사용하여 크기를 구성하려면 포드의 각 컨테이너에 대한 ephemeral-storage 리소스 요청을 지정합니다. Kubernetes가 포드를 예약할 때 이는 각 포드에 대한 리소스 요청의 합계가 Fargate 태스크의 용량보다 작도록 보장합니다. 자세한 내용은 쿠버네티스 문서의 [포드 및 컨테이너 리소스 관리](#)를 참조하세요.

Amazon EKS Fargate는 시스템 사용 목적으로 요청한 것보다 더 많은 임시 스토리지를 프로비저닝합니다. 예를 들어, 100GiB를 요청하면 Fargate 작업에 115GiB의 임시 스토리지가 프로비저닝됩니다.

AWS Fargate OS 패치 이벤트에 대한 작업 설정

Amazon EKS는 AWSFargate 노드의 OS를 주기적으로 패치하여 보안을 유지합니다. 패치 적용 프로세스의 일부로 노드를 재활용하여 OS 패치를 설치합니다. 서비스에 미치는 영향이 가장 적은 방식으로 업데이트가 시도됩니다. 그러나 포드가 성공적으로 제거되지 않으면 포드를 삭제해야 할 때가 있습니다. 다음 작업은 중단 가능성을 최소화하기 위해 수행할 수 있는 작업입니다.

- 적절한 포드 중단 예산(PDB)을 설정하여 동시에 중단되는 포드의 수를 제어한다.
- 포드가 삭제되기 전에 실패한 제거를 처리하는 Amazon EventBridge 규칙을 생성합니다.
- 수신한 알림에 게시된 제거 날짜 이전에 영향을 받는 포드를 수동으로 다시 시작합니다.
- AWS 사용자 알림에서 알림 구성을 생성합니다.

Amazon EKS는 Kubernetes 커뮤니티와 밀접하게 협력하여 버그 수정 및 보안 패치를 최대한 빨리 사용할 수 있도록 합니다. 모든 Fargate 포드는 Kubernetes 패치의 최신 버전에서 시작됩니다. 이 버전은 클러스터의 Kubernetes 버전에 대한 Amazon EKS에서 사용할 수 있습니다. 이전 패치 버전의 포드가 있는 경우 Amazon EKS는 해당 포드를 재활용하여 최신 버전으로 업데이트할 수 있습니다. 이렇게 하면 포드에 최신 보안 업데이트가 설치됩니다. 그렇게 하면 중요한 [일반 취약성 및 노출\(CVE\)](#) 문제가 있는 경우, 최신 상태로 유지되어 보안 위험을 줄여줍니다.

AWS Fargate OS가 업데이트되면 Amazon EKS가 영향을 받는 리소스와 향후 포드 제거 날짜가 포함된 알림을 보냅니다. 제공된 제거 날짜가 불편한 경우 알림에 게시된 제거 날짜 이전에 영향을 받는 포드를 수동으로 다시 시작할 수 있습니다. 알림을 수신하기 전에 생성한 모든 포드는 제거 대상입니다. 포드를 수동으로 다시 시작하는 방법에 대한 자세한 지침은 [Kubernetes 설명서](#)를 참조하세요.

포드를 재활용할 때 한 번에 중단되는 포드 수를 제한하려면 포드 중단 예산(PDB)을 설정할 수 있습니다. PDB를 사용하여 각 애플리케이션의 요구 사항에 따라 최소 가용성을 정의하면서 계속 업데이트도 할 수 있습니다. PDB의 최소 가용성은 100% 미만이어야 합니다. 자세한 내용은 Kubernetes Documentation의 [Specifying a Disruption Budget for your Application](#)을 참조하세요.

Amazon EKS가 [제거 API](#)를 사용하여 애플리케이션에 대해 설정한 PDB를 준수하면서 포드를 안전하게 드레이닝합니다. 가용 영역에서 포드를 제거하여 영향을 최소화합니다. 제거가 성공하면 새 포드가 최신 패치를 받아 추가 작업이 필요하지 않습니다.

포드의 제거가 실패하면 Amazon EKS는 제거에 실패한 포드에 대한 세부 정보와 함께 이벤트를 계정에 전송합니다. 예약된 종료 시간 전에는 메시지에서 작업할 수 있습니다. 특정 시간은 패치의 긴급성에 따라 다릅니다. 시간이 되면 Amazon EKS는 포드를 다시 제거하려고 시도합니다. 그러나 이번에는 제거가 실패하면 새 이벤트가 전송되지 않습니다. 제거가 다시 실패하면 새 포드가 최신 패치를 받을 수 있도록 기존 포드가 정기적으로 삭제됩니다.

다음 이벤트는 포드 제거가 실패할 때 수신되는 샘플 이벤트입니다. 여기에는 클러스터, 포드 이름, 포드 네임스페이스, Fargate 프로파일 및 예약된 종료 시간에 대한 세부 정보가 포함되어 있습니다.

```
{
  "version": "0",
  "id": "12345678-90ab-cdef-0123-4567890abcde",
  "detail-type": "EKS Fargate Pod Scheduled Termination",
  "source": "aws.eks",
  "account": "111122223333",
  "time": "2021-06-27T12:52:44Z",
  "region": "region-code",
  "resources": [
    "default/my-database-deployment"
```

```

    ],
    "detail": {
      "clusterName": "my-cluster",
      "fargateProfileName": "my-fargate-profile",
      "podName": "my-pod-name",
      "podNamespace": "default",
      "evictErrorMessage": "Cannot evict pod as it would violate the pod's disruption budget",
      "scheduledTerminationTime": "2021-06-30T12:52:44.832Z[UTC]"
    }
  }
}

```

또한 여러 PDB가 포드에 연결되어 있으면 제거 실패 이벤트가 발생할 수 있습니다. 이 이벤트는 다음과 같은 오류 메시지를 반환합니다.

```

"evictErrorMessage": "This pod has multiple PodDisruptionBudget, which the eviction subresource does not support",

```

이 이벤트를 기반으로 원하는 작업을 생성할 수 있습니다. 예를 들어 포드 중단 예산(PDB)을 조정하여 포드가 제거되는 방식을 제어할 수 있습니다. 조금 더 구체적으로, 사용 가능한 포드의 대상 백분율을 지정하는 PDB로 시작한다고 가정합니다. 업그레이드 중에 포드가 강제 종료되기 전에 PDB를 다른 백분율의 포드로 조정할 수 있습니다. 이 이벤트를 수신하려면 클러스터가 속한 AWS 계정 및 AWS 리전에서 Amazon EventBridge 규칙을 생성해야 합니다. 규칙은 다음과 같은 사용자 지정 패턴을 사용해야 합니다. 자세한 내용을 알아보려면 Amazon EventBridge 사용 설명서의 [이벤트에 응답하는 Amazon EventBridge 규칙 생성](#)을 참조하세요.

```

{
  "source": ["aws.eks"],
  "detail-type": ["EKS Fargate Pod Scheduled Termination"]
}

```

이벤트를 캡처하기 위해 적합한 대상을 설정할 수 있습니다. 사용 가능한 대상의 전체 목록은 Amazon EventBridge 사용 설명서의 [Amazon EventBridge 대상](#)을 참조하세요. AWS 사용자 알림에서 알림 구성을 생성할 수도 있습니다. AWS Management Console를 사용하여 알림을 생성할 때, 이벤트 규칙에서 AWS 서비스 이름에 Elastic Kubernetes Service(EKS)를 선택하고 이벤트 유형에 EKS Fargate 포드 예약 종료를 선택합니다. 자세한 내용은 AWS User Notifications User Guide의 [Getting started with AWS User Notifications](#)를 참조하세요.

EKS 포드 제거와 관련하여 자주 묻는 질문은 AWS re:Post의 [FAQ: Fargate 포드 제거 공지](#)를 참조하세요.

AWS Fargate 앱 및 사용량 지표 수집

시스템 지표 및 AWS Fargate의 CloudWatch 사용량 지표를 수집할 수 있습니다.

애플리케이션 지표

Amazon EKS 및 AWS Fargate에서 실행되는 애플리케이션의 경우 AWS Distro for OpenTelemetry(ADOT)를 사용할 수 있습니다. ADOT를 사용하면 시스템 지표를 수집하여 CloudWatch 컨테이너 인사이트 대시보드로 전송할 수 있습니다. Fargate에서 실행되는 애플리케이션용 ADOT로 시작하려면 ADOT 설명서의 [AWS Distro for OpenTelemetry를 통한 CloudWatch 컨테이너 인사이트 사용](#)을 참조하세요.

사용량 지표

CloudWatch 사용량 지표를 사용하여 계정의 리소스 사용량을 확인할 수 있습니다. 이러한 지표를 사용하여 CloudWatch 그래프 및 대시보드에서 현재 서비스 사용량을 시각화합니다.

AWS Fargate 사용량 지표는 AWS 서비스 할당량에 해당합니다. 사용량이 서비스 할당량에 가까워지면 경고하는 경보를 구성할 수 있습니다. Fargate 서비스 할당량에 대한 자세한 정보는 [the section called “Service quotas”](#) 섹션을 참조하세요.

AWS Fargate는 AWS/Usage 네임스페이스에 다음 지표를 게시합니다.

지표	설명
ResourceCount	계정에서 실행 중인 지정된 리소스의 총 수입니다. 리소스는 지표와 연결된 차원으로 정의됩니다.

다음 차원은 AWS Fargate에 의해 게시되는 사용량 지표를 구체화하는 데 사용됩니다.

차원	설명
Service	리소스가 포함된 AWS 서비스의 이름 AWS Fargate 사용량 지표의 경우 이 차원 값은 Fargate입니다.

차원	설명
Type	보고되는 엔터티의 유형입니다. 현재 AWS Fargate 사용량 지표에 대한 유일한 유효 값은 Resource입니다.
Resource	실행 중인 리소스의 유형입니다. 현재 AWS Fargate 는 Fargate 온디맨드 사용량에 대한 정보를 반환합니다. Fargate 온디맨드 사용량에 대한 리소스 값은 OnDemand입니다. [참고] ===== Fargate 온디맨드 사용량은 Fargate를 사용하는 Amazon EKS 포드, Fargate 시작 유형을 사용하는 Amazon ECS 태스크 및 FARGATE 용량 제공 업체를 사용하는 Amazon ECS가 합쳐진 값입니다. =====
Class	추적 중인 리소스의 클래스입니다. 현재 AWS Fargate에서는 클래스 차원을 사용하지 않습니다.

Fargate 리소스 사용량 지표 모니터링을 위한 CloudWatch 경보 생성

AWS Fargate는 Fargate 온디맨드 리소스 사용량에 대한 AWS 서비스 할당량에 해당하는 CloudWatch 사용량 지표를 제공합니다. Service Quotas 콘솔에서 사용량을 그래프로 시각화할 수 있습니다. 사용량이 서비스 할당량에 가까워지면 경고하는 경보를 구성할 수도 있습니다. 자세한 내용은 [the section called “지표 수집”](#) 섹션을 참조하세요.

다음 단계를 사용하여 Fargate 리소스 사용량 지표에 기반하여 CloudWatch 경보를 생성합니다.

1. <https://console.aws.amazon.com/servicequotas/>에서 Service Quotas 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 AWS 서비스를 선택합니다.
3. AWS 서비스 목록에서 AWS Fargate를 검색하여 선택합니다.

4. Service quotas 목록에서 경보를 만들려는 Fargate 사용량 할당량을 선택합니다.
5. Amazon CloudWatch 경보 섹션에서 생성을 선택합니다.
6. 경보 임계값에서 경보 값으로 설정할 적용된 할당량 값의 백분율을 선택합니다.
7. 경보 이름에서 경보 이름을 입력한 다음 생성을 선택합니다.

클러스터에 대한 AWS Fargate 로깅 시작하기

Amazon EKS on Fargate는 Fluent Bit를 기반으로 내장된 로그 라우터를 제공합니다. 즉, 사용자가 Fluent Bit 컨테이너를 사이드카로 명시적으로 실행하지는 않지만 Amazon이 자동으로 실행합니다. 로그 라우터를 구성하기만 하면 됩니다. 이 구성은 다음 기준을 충족해야 하는 전용 ConfigMap을 통해 이루어집니다.

- aws-logging으로 이름 지정
- aws-observability라는 전용 네임스페이스에서 생성
- 5300자를 초과할 수 없습니다.

ConfigMap을 생성하고 나면 Fargate의 Amazon EKS가 자동으로 감지하고 로그 라우터를 구성합니다. Fargate는 AWS for Fluent Bit의 한 버전인 AWS 관리형 Fluent Bit의 업스트림 호환 배포판을 사용합니다. 자세한 내용은 GitHub에서 [AWS for Fluent Bit](#)를 참조하세요.

로그 라우터를 사용하면 AWS에서 로그 분석 및 저장을 위한 다양한 서비스를 사용할 수 있습니다. Fargate에서 Amazon CloudWatch, Amazon OpenSearch Service로 로그를 직접 스트리밍할 수 있습니다. [Amazon S3](#), [Amazon Kinesis Data Streams](#), [Amazon Data Firehose](#)를 통한 파트너 도구 등의 대상으로 로그를 스트리밍할 수도 있습니다.

- Fargate 포드를 배포할 기존 Kubernetes 네임스페이스를 지정하는 기존 Fargate 프로필. 자세한 내용은 [the section called “3단계: 클러스터에 대한 Fargate 프로필 생성”](#) 섹션을 참조하세요.
- 기존 Fargate 포드 실행 역할. 자세한 내용은 [the section called “2단계: Fargate 포드 실행 역할 생성”](#) 섹션을 참조하세요.

로그 라우터 구성

Important

로그가 성공적으로 게시되려면 클러스터가 있는 VPC에서 로그 대상으로 네트워크 액세스가 있어야 합니다. 이는 주로 VPC에 대한 송신 규칙을 사용자 지정하는 사용자와 관련이 있습니다.

니다. CloudWatch를 사용하는 예제는 Amazon CloudWatch Logs 사용 설명서의 [인터페이스 VPC 엔드포인트에서 CloudWatch Logs 사용](#)을 참조하세요.

다음 단계에서 모든 **##** **#**을 고유한 값으로 바꿉니다.

1. aws-observability라는 전용 Kubernetes 네임스페이스를 생성합니다.

- a. 다음 콘텐츠를 컴퓨터에 aws-observability-namespace.yaml이라는 파일에 저장합니다. name의 값은 aws-observability여야 하며 aws-observability: enabled 레이블이 필요합니다.

```
kind: Namespace
apiVersion: v1
metadata:
  name: aws-observability
  labels:
    aws-observability: enabled
```

- b. 네임스페이스를 생성합니다.

```
kubectl apply -f aws-observability-namespace.yaml
```

2. Fluent Conf 데이터 값을 사용하여 ConfigMap을 생성함으로써 컨테이너 로그를 대상에 전달합니다. Fluent Conf는 Fluent Bit입니다. 이는 원하는 로그 대상에 컨테이너 로그를 라우팅하는 데 사용되는 빠르고 가벼운 로그 프로세서 구성 언어입니다. 자세한 내용은 Fluent Bit 문서의 [구성 파일 \(Configuration File\)](#)을 참조하세요.

Important

일반적인 Fluent Conf에 포함된 주요 단원은 Service, Input, Filter, Output입니다. 하지만 Fargate 로그 라우터는 다음 부분만 수락합니다.

- Filter 및 Output 부분입니다.
- Parser 부분.

기타 부분을 제공하는 경우 해당 부분은 거부됩니다.

Fargate 로그 라우터에서는 Service 및 Input 부분을 관리합니다. 여기에는 수정할 수 없고 ConfigMap에서 필요하지 않은 다음과 같은 Input 부분이 있습니다. 그러나 메모리 버퍼 제한과 로그에 적용된 태그와 같은 인사이트를 얻을 수 있습니다.

```
[INPUT]
  Name tail
  Buffer_Max_Size 66KB
  DB /var/log/flb_kube.db
  Mem_Buf_Limit 45MB
  Path /var/log/containers/*.log
  Read_From_Head On
  Refresh_Interval 10
  Rotate_Wait 30
  Skip_Long_Lines On
  Tag kube.*
```

ConfigMap 생성 시에는 Fargate가 필드를 검증하는 데 사용하는 다음 규칙을 고려하세요.

- [FILTER], [OUTPUT], [PARSER]는 각 해당 키 아래에 지정되어야 합니다. 예를 들어, `filters.conf`은 [FILTER]에 있어야 합니다. `filters.conf`에 하나 이상의 [FILTER]가 있을 수 있습니다. [OUTPUT] 및 [PARSER] 부분도 해당 키 아래에 있어야 합니다. 여러 개의 [OUTPUT] 부분을 지정하여 로그를 여러 대상으로 동시에 라우팅할 수 있습니다.
- Fargate는 각 섹션에 필요한 키를 검증합니다 Name 및 match는 [FILTER] 및 [OUTPUT]에 각각 필요합니다. Name 및 format은 [PARSER]에 각각 필요합니다. 태그는 대/소문자를 구분합니다.
- `${ENV_VAR}`과 같은 환경 변수는 ConfigMap에서 허용되지 않습니다.
- 들여쓰기는 각 `filters.conf`, `output.conf`, `parsers.conf` 내의 지시문 또는 키 값 페어에 대해 동일해야 합니다. 키 값 페어는 지시문보다 들여 써야 합니다.
- Fargate는 `grep`, `parser`, `record_modifier`, `rewrite_tag`, `throttle`, `nest`, `modify`, `kubernetes` 등의 지원 필터에 대해 검증을 실시합니다.
- Fargate는 다음과 같은 지원되는 출력에 대해 검증합니다. `es`, `firehose`, `kinesis_firehose`, `cloudwatch`, `cloudwatch_logs` 및 `kinesis`.
- 로깅을 사용 설정하려면 ConfigMap에서 지원되는 Output 플러그 인이 1개 이상 제공되어야 합니다. Filter 및 Parser는 로깅을 사용 설정하는 데 필요하지 않습니다.

검증에서 발생하는 문제를 해결하기 위해 원하는 구성을 사용하여 Amazon EC2 Fluent Bit를 실행할 수 있습니다. 다음 예 중 하나를 사용하여 ConfigMap을 생성합니다.

⚠ Important

Amazon EKS Fargate 로깅은 ConfigMap의 동적 구성을 지원하지 않습니다. ConfigMap에 대한 모든 변경 사항은 새 포드에만 적용됩니다. 기존 포드에는 변경 사항이 적용되지 않습니다.

원하는 로그 대상에 대한 예제를 사용하여 ConfigMap을 생성합니다.

ℹ Note

로그 대상에 Amazon Kinesis Data Streams를 사용할 수도 있습니다. Kinesis Data Streams를 사용하는 경우 포드 실행 역할에 `kinesis:PutRecords` 권한이 부여되었는지 확인하세요. 자세한 내용은 [Fluent Bit: 공식 설명서의 Amazon Kinesis Data Streams 권한](#)을 참조하세요.

Example**CloudWatch**

CloudWatch를 사용할 때는 다음 두 가지 출력 옵션이 있습니다.

- [C로 작성된 출력 플러그인](#)
- [Golang으로 작성된 출력 플러그인](#)

다음 예에서는 `cloudwatch_logs` 플러그인을 사용하여 CloudWatch로 로그를 전송하는 방법을 보여줍니다.

- 다음 콘텐츠를 `aws-logging-cloudwatch-configmap.yaml`이라는 파일에 저장합니다. ***region-code***를 클러스터가 있는 AWS 리전으로 바꿉니다. [OUTPUT] 아래의 파라미터는 필수입니다.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
data:
  flb_log_cw: "false" # Set to true to ship Fluent Bit process logs to
                    CloudWatch.
```

```

filters.conf: |
  [FILTER]
    Name parser
    Match *
    Key_name log
    Parser crio
  [FILTER]
    Name kubernetes
    Match kube.*
    Merge_Log On
    Keep_Log Off
    Buffer_Size 0
    Kube_Meta_Cache_TTL 300s
output.conf: |
  [OUTPUT]
    Name cloudwatch_logs
    Match kube.*
    region region-code
    log_group_name my-logs
    log_stream_prefix from-fluent-bit-
    log_retention_days 60
    auto_create_group true
parsers.conf: |
  [PARSER]
    Name crio
    Format Regex
    Regex ^(?<time>[^\ ]+) (?<stream>stdout|stderr) (?<logtag>P|F) (?
<log>.*)$
    Time_Key time
    Time_Format %Y-%m-%dT%H:%M:%S.%L%z

```

b. 매니페스트를 클러스터에 적용합니다.

```
kubectl apply -f aws-logging-cloudwatch-configmap.yaml
```

Amazon OpenSearch Service

Amazon OpenSearch Service에 로그를 전송하려면 C로 작성된 플러그인인 [es](#) 출력을 사용할 수 있습니다. 다음 예제에서는 이 플러그인을 사용하여 OpenSearch로 로그를 전송하는 방법을 보여줍니다.

a. 다음 콘텐츠를 `aws-logging-opensearch-configmap.yaml`이라는 파일에 저장합니다. 모든 `##` `#`을 자신의 값으로 바꾸세요.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
data:
  output.conf: |
    [OUTPUT]
      Name es
      Match *
      Host search-example-gjxdcilagiprbqlqn42jsty66y.region-code.es.amazonaws.com
      Port 443
      Index example
      Type example_type
      AWS_Auth On
      AWS_Region region-code
      tls On
```

b. 매니페스트를 클러스터에 적용합니다.

```
kubectl apply -f aws-logging-opensearch-configmap.yaml
```

Firehose

Firehose로 로그를 전송할 때는 다음과 같은 두 가지 출력 옵션이 있습니다.

- [kinesis_firehose](#) – C로 작성된 출력 플러그인입니다.
- [firehose](#) - Golang으로 작성된 출력 플러그인

다음 예에서는 kinesis_firehose 플러그인을 사용하여 Firehose로 로그를 전송하는 방법을 보여줍니다.

a. 다음 콘텐츠를 aws-logging-firehose-configmap.yaml이라는 파일에 저장합니다.
*region-code*를 클러스터가 있는 AWS 리전으로 바꿉니다.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
data:
```



```
output.conf: |
  [OUTPUT]
  Name  kinesis_firehose
  Match *
  region region-code
  delivery_stream my-stream-firehose
```

b. 매니페스트를 클러스터에 적용합니다.

```
kubectl apply -f aws-logging-firehose-configmap.yaml
```

3. Fargate Pod 실행 역할에 대한 권한을 설정하여 대상에 로그를 전송합니다.

a. 대상의 IAM 정책을 컴퓨터에 다운로드합니다.

Example

CloudWatch

CloudWatch IAM 정책을 컴퓨터에 다운로드합니다. GitHub에서 [정책](#)을 볼 수도 있습니다.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-eks-fluent-logging-examples/mainline/examples/fargate/cloudwatchlogs/permissions.json
```

Amazon OpenSearch Service

OpenSearch IAM 정책을 컴퓨터에 다운로드합니다. GitHub에서 [정책](#)을 볼 수도 있습니다.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-eks-fluent-logging-examples/mainline/examples/fargate/amazon-elasticsearch/permissions.json
```

OpenSearch Dashboard의 액세스 컨트롤이 제대로 구성되어 있는지 확인합니다.

OpenSearch Dashboard의 `all_access` role에는 Fargate 포드 실행 역할과 IAM 역할이 매핑되어야 합니다. `security_manager` 역할에 대해서도 동일한 매핑이 수행되어야 합니다. 이전 매핑을 추가하려면 Menu, Security, Roles를 차례로 선택한 다음 해당 역할을 선택합니다. 자세한 내용은 [CloudWatch Logs가 내 Amazon ES 도메인으로 스트리밍되도록 문제를 해결하려면 어떻게 해야 하나요?](#) 부분을 참조하세요.

Firehose

Firehose IAM 정책을 컴퓨터에 다운로드합니다. GitHub에서 [정책](#)을 볼 수도 있습니다.

```
curl -O https://raw.githubusercontent.com/aws-samples/amazon-eks-fluent-logging-examples/mainline/examples/fargate/kinesis-firehose/permissions.json
```

b. 다운로드한 정책 파일에서 IAM 정책을 생성합니다.

```
aws iam create-policy --policy-name eks-fargate-logging-policy --policy-document file://permissions.json
```

c. 다음 명령을 사용하여 Fargate 프로필에 대해 지정된 포드 실행 역할에 IAM 정책을 연결합니다. **111122223333**을 계정 ID로 바꿉니다. **AmazonEKSFargatePodExecutionRole**을 포드 실행 역할로 바꿉니다(자세한 내용은 [the section called “2단계: Fargate 포드 실행 역할 생성”](#) 참조).

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::111122223333:policy/eks-fargate-logging-policy \
  --role-name AmazonEKSFargatePodExecutionRole
```

Kubernetes 필터 지원

Fluent Bit Kubernetes 필터를 사용하면 로그 파일에 Kubernetes 메타데이터를 추가할 수 있습니다. 필터에 대한 자세한 내용은 Fluent Bit 설명서의 [Kubernetes](#)를 참조하세요. API 서버 엔드포인트를 사용하여 필터를 적용할 수 있습니다.

```
filters.conf: |
  [FILTER]
    Name          kubernetes
    Match          kube.*
    Merge_Log      On
    Buffer_Size     0
    Kube_Meta_Cache_TTL 300s
```

Important

- Kube_URL, Kube_CA_File, Kube_Token_Command 및 Kube_Token_File은 서비스 소스 구성 파라미터이므로 지정하면 안 됩니다. Amazon EKS Fargate가 이러한 값을 채웁니다.

- Kube_Meta_Cache_TTL은 Fluent Bit가 최신 메타데이터에 대해 API 서버와 통신할 때까지 대기하는 시간입니다. Kube_Meta_Cache_TTL을 지정하지 않으면 Amazon EKS Fargate는 API 서버의 로드를 줄이기 위해 기본값 30분을 추가합니다.

계정에 Fluent-bit 프로세스 로그 전송

원하는 경우 다음 ConfigMap을 사용하여 Fluent Bit 프로세스 로그를 Amazon CloudWatch로 전송할 수 있습니다. Fluent Bit 프로세스 로그를 CloudWatch로 전송하려면 추가 로그 수집 및 스토리지 비용이 필요합니다. *region-code*를 클러스터가 있는 AWS 리전으로 바꿉니다.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: aws-logging
  namespace: aws-observability
  labels:
data:
  # Configuration files: server, input, filters and output
  # =====
  flb_log_cw: "true" # Ships Fluent Bit process logs to CloudWatch.

output.conf: |
  [OUTPUT]
    Name cloudwatch
    Match kube.*
    region region-code
    log_group_name fluent-bit-cloudwatch
    log_stream_prefix from-fluent-bit-
    auto_create_group true
```

로그는 클러스터와 동일한 AWS 리전의 CloudWatch에 있습니다. 로그 그룹 이름은 *my-cluster-fluent-bit-logs*이고 Fluent Bit 로그스트림 이름은 *fluent-bit-podname-pod-namespace*입니다.

Note

- 프로세스 로그는 Fluent Bit 프로세스가 성공적으로 시작된 경우에만 전송됩니다. Fluent Bit를 시작하는 동안 오류가 발생하면 프로세스 로그가 누락됩니다. CloudWatch에는 프로세스 로그만 전송할 수 있습니다.

- 계정으로 프로세스 로그 전송을 디버깅하려면 이전 ConfigMap을 적용하여 프로세스 로그를 얻습니다. Fluent Bit 시작에 실패하는 이유는 일반적으로 시작하는 동안 Fluent Bit에서 ConfigMap을 구문 분석하거나 수락하지 않기 때문입니다.

Fluent Bit 프로세스 로그 전송 중지

Fluent Bit 프로세스 로그를 CloudWatch로 전송하려면 추가 로그 수집 및 스토리지 비용이 필요합니다. 기존 ConfigMap 설정에서 프로세스 로그를 제외하려면 다음 단계를 수행하세요.

1. Fargate 로깅을 활성화한 후 Amazon EKS 클러스터의 Fluent Bit 프로세스 로그에 대해 자동으로 생성된 CloudWatch 로그 그룹을 찾습니다. 형식 *my-cluster*-fluent-bit-logs를 따릅니다.
2. CloudWatch 로그 그룹에서 각 포드의 프로세스 로그에 대해 생성된 기존 CloudWatch 로그 스트림을 삭제합니다.
3. ConfigMap을 편집하고 `flb_log_cw: "false"`를 설정합니다.
4. 클러스터의 모든 포드를 클러스터의 포드에 대한 재시작합니다.

애플리케이션 테스트

1. 샘플 포드를 배포합니다.
 - a. 다음 콘텐츠를 컴퓨터에 `sample-app.yaml`이라는 파일에 저장합니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sample-app
  namespace: same-namespace-as-your-fargate-profile
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
```

```
image: nginx:latest
ports:
  - name: http
    containerPort: 80
```

b. 매니페스트를 클러스터에 적용합니다.

```
kubectl apply -f sample-app.yaml
```

2. ConfigMap에서 구성한 대상을 사용하여 NGINX 로그를 봅니다.

크기 조정 고려 사항

로그 라우터에 대해 최대 50MB의 메모리를 계획하는 것이 좋습니다. 애플리케이션에서 매우 높은 처리량으로 로그를 생성할 것으로 예상되는 경우 최대 100MB까지 계획해야 합니다.

문제 해결

잘못된 ConfigMap과 같은 원인으로 로깅 기능이 사용 또는 사용 중지되었는지 여부와 잘못된 이유를 확인하려면 `kubectl describe pod pod-name` 을 사용하여 포드 이벤트를 살펴봅니다. 출력에는 다음 예제 출력과 같이 로깅의 사용 여부를 명확히 하는 포드 이벤트가 포함될 수 있습니다.

```
[...]
Annotations:      CapacityProvisioned: 0.25vCPU 0.5GB
                  Logging: LoggingDisabled: LOGGING_CONFIGMAP_NOT_FOUND
[...]
Events:
  Type            Reason              Age             From
              Message
  ----            -
Warning LoggingDisabled <unknown> fargate-scheduler
                  Disabled logging because aws-logging configmap was not found. configmap
"aws-logging" not found
```

포드 이벤트는 설정에 따라 기간이 정해진 일시적인 이벤트입니다. 또한 `kubectl describe pod pod-name` 을 사용하여 포드의 주석을 볼 수 있습니다. 포드 주석에는 로깅 기능이 사용 또는 사용 중지되었는지 여부와 이유에 대한 정보가 있습니다.

최적의 Amazon EC2 노드 인스턴스 유형 선택

Amazon EC2에서는 워커 노드에 대한 다양한 인스턴스 유형을 제공합니다. 인스턴스 유형마다 서로 다른 컴퓨팅, 메모리, 스토리지 및 네트워크 기능을 제공합니다. 또한 각 인스턴스는 이러한 기능에 따라 한 인스턴스 패밀리로 그룹화됩니다. 목록은 Amazon EC2 사용 설명서의 [사용 가능한 인스턴스 유형](#)을 참조하세요. Amazon EKS에서는 Amazon EC2 AMI의 여러 변형을 출시하여 지원을 사용 설정합니다. 선택한 인스턴스 유형이 Amazon EKS와 호환되는지 확인하려면 다음 기준을 고려하세요.

- 일부 Amazon EKS AMI에서는 현재 mac 패밀리를 지원하지 않습니다.
- Arm 및 가속화되지 않은 Amazon EKS AMI에서는 g3, g4, inf 및 p 패밀리를 지원하지 않습니다.
- 가속화된 Amazon EKS AMI에서는 a, c, hpc, m 및 t 패밀리를 지원하지 않습니다.
- ARM 기반 인스턴스에서 Amazon Linux 2023(AL2023)은 Graviton2 이상 프로세서를 사용하는 인스턴스 유형만 지원합니다. AL2023은 A1 인스턴스를 지원하지 않습니다.

Amazon EKS에서 지원하는 인스턴스 유형 중에서 선택하는 경우 각 유형의 다음 기능을 고려하세요.

노드 그룹의 인스턴스 수

특히 DaemonSet이 많은 경우 일반적으로 수는 적고 크기는 큰 인스턴스가 더 좋습니다. 각 인스턴스에는 API 서버에 대한 API 직접 호출이 필요하므로 인스턴스가 많을수록 API 서버에 더 많은 부하가 발생합니다.

운영 체제

[Linux](#), [Windows](#) 및 [Bottlerocket](#)에 대해 지원되는 인스턴스 유형을 검토합니다. Windows 인스턴스를 생성하기 전에 [EKS 클러스터에 Windows 노드 배포](#)를 검토합니다.

하드웨어 아키텍처

x86 또는 Arm이 필요하나요? Arm 인스턴스를 배포하기 전에 [Amazon EKS 최적화 Arm Amazon Linux AMI](#)를 검토합니다. Nitro 시스템([Linux](#) 또는 [Windows](#))에 구축된 인스턴스나 [가속](#) 기능이 있는 인스턴스가 필요하신가요? 가속화된 기능이 필요한 경우 Amazon EKS에서 Linux만 사용할 수 있습니다.

최대 포트 수

각 포트에 고유한 IP 주소가 할당되므로 인스턴스 유형에서 지원하는 IP 주소 수는 인스턴스에서 실행할 수 있는 포트 수를 결정하는 요소입니다. 인스턴스 유형이 지원하는 포트의 수를 수동으로 확인하려면 [the section called “각 Amazon EC2 인스턴스 유형의 Amazon EKS 권장 최대 포트 수”](#) 섹션을 참조하세요.

Note

Amazon EKS에 최적화된 v20220406 이상의 Amazon Linux 2 AMI를 사용하는 경우 최신 AMI로 업그레이드하지 않고도 새로운 인스턴스 유형을 사용할 수 있습니다. 이러한 AMI의 경우 AMI는 [eni-max-pods.txt](#) 파일에 나열되지 않으면 필요한 max-pods 값을 자동으로 계산합니다. 현재 미리 보기 중인 인스턴스 유형은 기본적으로 Amazon EKS에서 지원되지 않을 수 있습니다. 이러한 유형에 대한 max-pods의 값은 여전히 AMI의 eni-max-pods.txt에 추가되어야 합니다.

[AWS Nitro 시스템\(Nitro System\)](#) 인스턴스 유형은 Nitro 시스템 이외의 인스턴스 유형보다 훨씬 많은 IP 주소를 선택적으로 지원합니다. 그러나 인스턴스에 할당된 일부 IP 주소만 포트에 사용할 수 있습니다. 훨씬 더 많은 수의 IP 주소를 인스턴스에 할당하려면 클러스터에 버전 1.9.0 이상의 Amazon VPC CNI 추가 기능이 설치되어 있고 적절하게 구성되어 있어야 합니다. 자세한 내용은 [the section called “IP 주소 증가”](#) 섹션을 참조하세요. 인스턴스에 가장 많은 수의 IP 주소를 할당하려면 클러스터에 버전 1.10.1 이상의 Amazon VPC CNI 추가 기능이 설치되어 있어야 하며 IPv6 패밀리를 사용하여 클러스터를 배포해야 합니다.

IP 패밀리

클러스터가 포트와 서비스에 프라이빗 IPv4 주소를 할당하도록 허용하는 클러스터용 IPv4 패밀리를 사용하는 경우 지원되는 모든 인스턴스 유형을 사용할 수 있습니다. 하지만 클러스터에 대한 IPv6 패밀리를 사용하려는 경우 [AWS Nitro 시스템\(Nitro System\)](#) 인스턴스 유형 또는 베어 메탈 인스턴스 유형을 사용해야 합니다. Windows 인스턴스에는 IPv4만 지원됩니다. 클러스터에서 버전 1.10.1 이상의 Amazon VPC CNI 추가 기능을 실행하고 있어야 합니다. IPv6 사용에 관한 자세한 내용은 [the section called “IPv6”](#) 부분을 참조하세요.

실행 중인 Amazon VPC CNI 추가 기능의 버전

최신 버전의 [Kubernetes용 Amazon VPC CNI 플러그인](#)은 [다음 인스턴스 유형](#)을 지원합니다. 지원되는 최신 인스턴스 유형을 활용하려면 Amazon VPC CNI 추가 기능 버전을 업데이트해야 할 수 있습니다. 자세한 내용은 [the section called “Amazon VPC CNI”](#) 섹션을 참조하세요. 최신 버전은 Amazon EKS에서 사용할 최신 기능을 지원합니다. 이전 버전에서는 일부 기능을 지원하지 않습니다. GitHub의 [Changelog](#)에 있는 다양한 버전에서 지원하는 기능을 볼 수 있습니다.

노드를 생성하는 AWS 리전

AWS 리전에 따라 일부 인스턴스 유형은 사용할 수 없습니다.

포드에 보안 그룹 사용 여부

포드에 보안 그룹을 사용하는 경우 특정 인스턴스 유형만 지원됩니다. 자세한 내용은 [the section called “포드의 보안 그룹”](#) 섹션을 참조하세요.

각 Amazon EC2 인스턴스 유형의 Amazon EKS 권장 최대 포드 수

각 포드에 고유한 IP 주소가 할당되므로 인스턴스 유형에서 지원하는 IP 주소 수는 인스턴스에서 실행할 수 있는 포드 수를 결정하는 요소입니다. Amazon EKS는 각 인스턴스 유형에서 실행할 Amazon EKS 권장 최대 포드 수를 결정하기 위해 다운로드하여 실행할 수 있는 스크립트를 제공합니다. 스크립트는 각 인스턴스의 하드웨어 속성 및 구성 옵션을 사용하여 최대 포드 수를 결정합니다. 이 단계에서 반환된 수를 사용하여 [인스턴스의 서브넷과 다른 서브넷의 포드에 IP 주소 할당](#) 및 [인스턴스에 대한 IP 주소 수 크게 증가](#)와 같은 기능을 활성화할 수 있습니다. 여러 인스턴스 유형을 통해 관리형 노드 그룹을 사용하는 경우 모든 인스턴스 유형에 작동하는 값을 사용합니다.

1. 각 인스턴스 유형에 대한 최대 포드 수를 계산하는 데 사용할 수 있는 스크립트를 다운로드합니다.

```
curl -O https://raw.githubusercontent.com/awslabs/amazon-eks-ami/master/templates/al2/runtime/max-pods-calculator.sh
```

2. 컴퓨터에서 스크립트를 실행 파일로 표시합니다.

```
chmod +x max-pods-calculator.sh
```

3. 스크립트를 실행하여 **m5.large**를 배포하려는 인스턴스 유형으로 바꾸고, **1.9.0-eksbuild.1**을 Amazon VPC CNI 추가 기능 버전으로 바꿉니다. 추가 기능 버전을 확인하려면 [Amazon VPC CNI를 사용하여 포드에 IP 할당](#)의 업데이트 절차를 참조하세요.

```
./max-pods-calculator.sh --instance-type m5.large --cni-version 1.9.0-eksbuild.1
```

예제 출력은 다음과 같습니다.

```
29
```

스크립트에 다음 옵션을 추가하여 선택적 기능을 사용할 때 지원되는 최대 포드 수를 확인할 수 있습니다.

- **--cni-custom-networking-enabled** - 인스턴스의 서브넷과 다른 서브넷의 IP 주소를 할당하려는 경우 이 옵션을 사용합니다. 자세한 내용은 [the section called “사용자 지정 네트워킹”](#) 섹션

을 참조하세요. 동일한 예제 값을 사용하여 이전 스크립트에 이 옵션을 추가하면 20이 표시됩니다.

- `--cni-prefix-delegation-enabled` - 각 Elastic 네트워크 인터페이스에 훨씬 더 많은 IP 주소를 할당하려는 경우 이 옵션을 사용합니다. 이 기능을 사용하려면 Nitro 시스템과 Amazon VPC CNI 추가 기능 버전 1.9.0 이상에서 실행되는 Amazon Linux 인스턴스가 필요합니다. 자세한 내용은 [the section called “IP 주소 증가”](#) 섹션을 참조하세요. 동일한 예제 값을 사용하여 이전 스크립트에 이 옵션을 추가하면 110이 표시됩니다.

`--help` 옵션을 사용하여 스크립트를 실행하면 사용 가능한 모든 옵션을 볼 수 있습니다.

Note

최대 포드 계산기 스크립트는 [Kubernetes 확장성 임계값](#)과 권장 설정을 기반으로 반환 값을 110으로 제한합니다. 인스턴스 유형의 vCPU가 30개 이상인 경우 한도는 내부 Amazon EKS 확장성 팀 테스트를 기반으로 한 수치로 올라갑니다.250 자세한 내용은 [Amazon VPC CNI 플러그인을 통한 노드당 포드 제한 증가](#) 블로그 게시물을 참조하세요.

EKS Auto Mode 고려 사항

EKS Auto Mode는 노드의 포드 수를 다음 중 더 낮은 값으로 제한합니다.

- 110 포드 하드 캡
- 위에서 설명한 최대 포드 계산 결과입니다.

사전 빌드된 최적화 이미지를 사용한 노드 생성

관리형 노드 그룹이나 자체 관리형 노드를 사용하는 경우 사전 구축된 Amazon EKS 최적화 [Amazon Machine Image](#)(AMI) 또는 사용자 지정 AMI를 사용하여 노드를 배포할 수 있습니다. 하이브리드 노드를 실행하는 경우 [the section called “운영 체제 준비”](#) 섹션을 참조하세요. Amazon EKS 최적화 AMI의 각 유형에 대한 자세한 내용은 다음 주제 중 하나를 참조하세요. 고유한 사용자 지정 AMI를 생성하는 방법에 대한 설명은 [the section called “사용자 지정 빌드”](#) 섹션을 참조하세요.

Amazon EKS Auto Mode를 사용하면 EKS가 AMI 선택 및 업데이트를 포함하여 EC2 인스턴스를 관리합니다.

주제

- [EKS AL2 및 AL2 가속 AMI 전환 기능 가이드](#)
- [최적화된 Amazon Linux AMI를 사용한 노드 생성](#)
- [최적화된 Bottlerocket AMI를 사용한 노드 생성](#)
- [최적화된 Ubuntu Linux AMI를 사용한 노드 생성](#)
- [최적화된 Windows AMI를 사용한 노드 생성](#)

EKS AL2 및 AL2 가속 AMI 전환 기능 가이드

AWS는 AL2 최적화 및 AL2 가속 AMI에 대한 지원을 2025년 11월 26일부로 종료합니다. 지원 종료(EOS) 날짜(2025년 11월 26일) 이후에도 EKS AL2 AMI를 계속 사용할 수 있지만 EKS는 이 날짜 이후에 마이너 릴리스, 패치 및 버그 수정을 포함하여 AL2 AMI에 대한 새로운 Kubernetes 버전 또는 업데이트를 더 이상 릴리스하지 않습니다. Amazon Linux 2023(AL2023) 또는 Bottlerocket AMI로 업그레이드하는 것이 좋습니다.

- AL2023은 사전 구성된 보안 정책, 허용 모드의 SELinux, 기본적으로 활성화된 IMDSv2 전용 모드, 최적화된 부팅 시간, 강화된 보안 및 성능을 위한 향상된 패키지 관리를 통해 secure-by-default 접근 방식을 지원하며, 직접 OS 수준 액세스 또는 광범위한 노드 변경과 같은 중요한 사용자 지정이 필요한 인프라에 적합합니다.
- Bottlerocket을 사용하면 특별히 구축된 컨테이너 최적화 설계로 보안 강화, 부팅 시간 단축, 공격 표면 축소를 통해 효율성을 개선할 수 있으며, 노드 사용자 지정이 최소한으로 필요해 컨테이너 네이티브 접근 방식에 적합합니다.

또는 EOS 날짜(2025년 11월 26일)까지 [the section called “사용자 지정 빌드”](#)하거나 Amazon Linux 2 EOS 날짜(2026년 6월 30일)까지 Amazon Linux 2 기본 인스턴스로 사용자 지정 AMI를 빌드할 수 있습니다. [AL2023 FAQ](#) 또는 [Bottlerocket FAQ](#)를 방문하여 자세한 내용을 알아보거나 [the section called “AL2023으로 업그레이드”](#) 또는 [the section called “Bottlerocket”](#) 설명서에서 자세한 마이그레이션 지침을 참조하세요.

마이그레이션 및 지원 FAQ

AL2에서 AL2023 AMI로 마이그레이션하려면 어떻게 해야 하나요?

철저한 애플리케이션 워크로드 테스트 및 문서화된 롤백 절차를 포함하는 마이그레이션 계획을 작성 및 구현한 다음 EKS 공식 설명서의 [Amazon Linux 2에서 Amazon Linux 2023으로 업그레이드](#)의 단계별 지침을 따르는 것이 좋습니다.

EKS 최적화 AL2 AMI의 EKS 지원 종료(EOS) 날짜 이후에 사용자 지정 AL2 AMI를 빌드할 수 있나요?

공식적으로 지원 및 게시되는 AL2023 또는 Bottlerocket용 EKS 최적화 AMI로 전환하는 것이 좋지만, AL2 AMI EOS 날짜(2025년 11월 26일)까지는 사용자 지정 AL2 최적화 및 AL2 가속 AMI를 구축할 수 있습니다. 또는 Amazon Linux 2 EOS 날짜(2026년 6월 30일)까지 Amazon Linux 2 기본 인스턴스로 사용자 지정 AMI를 빌드할 수 있습니다. 사용자 지정 EKS AL2 최적화 및 AL2 가속 AMI를 빌드하기 위한 단계별 지침은 EKS 공식 설명서의 [사용자 지정 Amazon Linux AMI 빌드](#)를 참조하세요.

EKS Kubernetes 버전 지원 정책은 Amazon Linux 배포에 적용되나요?

아니요. EKS AL2 최적화 및 AL2 가속 AMI의 EOS 날짜는 EKS의 Kubernetes 버전에 대한 표준 및 확장 지원 타임라인과 독립적입니다. EKS 확장 지원을 사용하는 경우에도 AL2023 또는 Bottlerocket으로 마이그레이션해야 합니다.

cgroupv1에서 cgroupv2로의 전환은 마이그레이션에 어떤 영향을 미치나요?

[Kubernetes 커뮤니티](#)는 cgroupv1 지원(AL2에서 사용)을 유지 관리 모드로 전환했습니다. 즉, 새로운 기능이 추가되지 않고 중요한 보안 및 주요 버그 수정만 제공됩니다. Kubernetes에서 cgroupv2를 채택하려면 OS, 커널, 컨테이너 런타임, Kubernetes 구성 요소 간의 호환성을 보장해야 합니다. 여기에는 AL2023, Bottlerocket, Red Hat Enterprise Linux(RHEL) 9 이상, Ubuntu 22.04 이상 또는 Debian 11 이상과 같이 기본적으로 cgroupv2를 활성화하는 Linux 배포가 필요합니다. 이러한 배포에는 커널 버전 5.8 이상이 함께 제공되며, 이는 Kubernetes의 cgroupv2 지원 최소 요구 사항입니다. 자세한 내용은 [cgroup v2 정보](#)를 참조하세요.

사용자 지정 AL2 AMI에 Neuron이 필요한 경우 어떻게 해야 하나요?

AL2 기반 AMI에서는 기본적으로 전체 Neuron 기반 애플리케이션을 실행할 수 없습니다. AL2 AMI에서 AWS Neuron을 활용하려면 비 AL2 Linux 배포판(예: Ubuntu 22.04, Amazon Linux 2023 등)을 실행하는 Neuron 지원 컨테이너를 사용하여 애플리케이션을 컨테이너화한 다음 Neuron 드라이버(aws-neuronx-dkms)가 설치된 AL2 기반 AMI에 해당 컨테이너를 배포해야 합니다.

호환성 및 버전

AL2 AMI에 지원되는 Kubernetes 버전

Kubernetes 버전 1.32는 Amazon EKS가 AL2(Amazon Linux 2) AMI를 릴리스하는 마지막 버전입니다. [지원되는](#) Kubernetes 버전 1.32 이하의 경우 EKS는 2025년 11월 26일까지 AL2 AMI(AL2_ARM_64, AL2_x86_64) 및 AL2 가속 AMI(AL2_x86_64_GPU)를 계속 릴리스합니다. 이 날짜 이후에는 EKS가 모든 Kubernetes 버전에 대해 AL2 최적화 및 AL2 가속 AMI 릴리스를 중지합니다. EKS AL2 최적화 및

AL2 가속 AMI의 EOS 날짜는 EKS의 Kubernetes 버전에 대한 표준 및 확장 지원 타임라인과 독립적입니다.

AL2, AL2023 및 Bottlerocket AMI용 NVIDIA 드라이버 비교

드라이버 브랜치	Amazon Linux 2 AMI	Amazon Linux 2023 AMI	Bottlerocket AMI	수명 종료 날짜
R535	지원되지 않음	지원되지 않음	지원	2027년 9월
R550	지원	지원	지원되지 않음	2025년 4월
R560	지원되지 않음	지원	지원되지 않음	2025년 3월
R570	지원되지 않음	지원	곧 출시 예정	2026년 2월

자세한 내용은 [Nvidia 릴리스 설명서](#)를 참조하세요.

AL2, AL2023 및 Bottlerocket AMI용 NVIDIA CUDA 버전 비교

CUDA 버전	AL2 지원	AL2023 지원	Bottlerocket 지원
10.1	지원	지원되지 않음	지원되지 않음
11.8	지원	지원	지원
12.0	지원되지 않음	지원	지원
12.5	지원되지 않음	지원	지원

자세한 내용은 [CUDA 릴리스 설명서](#)를 참조하세요.

AL2, AL2023 및 Bottlerocket AMI에 지원되는 드라이버 및 Linux 커널 버전 비교

구성 요소	AL2 AMI 소스	AL2023 AMI 소스	Bottlerocket AMI 소스
기본 OS 호환성	RHEL7/CentOS 7	Fedora/CentOS 9	N/A

구성 요소	AL2 AMI 소스	AL2023 AMI 소스	Bottlerocket AMI 소스
CUDA 도구 키트	CUDA 11.x~12.x	CUDA 12.5+	CUDA 11.x(12.5 출시 예정)
NVIDIA GPU 드라이버	R550	R565 (R570 출시 예정)	R535 (R570 출시 예정)
AWS Neuron 드라이버	2.19	2.19+	2.20
Linux 커널	5.10	6.1, 6.12	5.15, 6.1(6.12 출시 예정)

AWS Neuron과 AL2 AMI의 호환성

[AWS Neuron 릴리스 2.20](#)부터 EKS AL 기반 AMI에서 사용하는 Neuron 런타임(`aws-neuronx-runtime-lib`)은 더 이상 Amazon Linux 2(AL2)를 지원하지 않습니다. Neuron 드라이버(`aws-neuronx-dkms`)는 이제 Amazon Linux 2를 지원하는 유일한 AWS Neuron 패키지입니다. 즉, AL2 기반 AMI에서는 기본적으로 Neuron 기반 애플리케이션을 실행할 수 없습니다. AL2023 AMI에서 Neuron을 설정하려면 [AWS Neuron 설정](#) 가이드를 참조하세요.

Kubernetes와 AL2 AMI의 호환성

Kubernetes 커뮤니티는 `cgroupv1` 지원(AL2에서 사용)을 유지 관리 모드로 전환했습니다. 즉, 새로운 기능이 추가되지 않으며 중요한 보안 및 주요 버그 수정만 제공됩니다. MemoryQoS 및 향상된 리소스 격리와 같이 `cgroupv2`에 의존하는 모든 Kubernetes 기능은 AL2에서 사용할 수 없습니다. 또한 Amazon EKS Kubernetes 버전 1.32는 AL2 AMI를 지원하는 마지막 버전입니다. 최신 Kubernetes 버전과의 호환성을 유지하려면 `cgroupv2`가 기본적으로 활성화되는 AL2023 또는 Bottlerocket으로 마이그레이션하는 것이 좋습니다.

Linux 버전과 AL2 AMI의 호환성

Amazon Linux 2(AL2)는 AWS에서 지원 종료(EOS) 날짜인 2026년 6월 30일까지 지원합니다. 그러나 AL2가 노후화됨에 따라 새로운 애플리케이션 및 기능에 대한 보다 광범위한 Linux 커뮤니티의 지원은 더욱 제한되고 있습니다. AL2 AMI는 [Linux 커널 5.10](#)을 기반으로 하는 반면, AL2023은 [Linux 커널 6.10](#)을 사용합니다. AL2023과 달리, AL2는 보다 광범위한 Linux 커뮤니티의 지원이 제한적입니다. 즉, AL2의 이전 커널 버전에서 작동하려면 많은 업스트림 Linux 패키지 및 도구를 백포트해야 하고, 일부

최신 Linux 기능 및 보안 개선 사항은 이전 커널로 인해 사용할 수 없으며, 많은 오픈 소스 프로젝트는 5.10과 같은 이전 커널 버전에 대한 지원을 중단 또는 제한했습니다.

AL2023에 포함되지 않은 더 이상 사용되지 않는 패키지

AL2023에서 포함되지 않았거나 변경된 몇 가지 가장 일반적인 패키지는 다음과 같습니다.

- [Amazon Linux 2의 일부 소스 바이너리 패키지](#)는 Amazon Linux 2023에서 더 이상 사용할 수 없습니다.
- Amazon Linux가 AL2023에서 다양한 버전의 패키지(예: [amazon-linux-extras 시스템](#))를 지원하는 방법이 변경되었습니다.
- [Extra Packages for Enterprise Linux\(EPEL\)](#)은 AL2023에서 지원되지 않습니다.
- [32비트 애플리케이션](#)은 AL2023에서 지원되지 않습니다.
- AL2023은 [mate-desktop](#) 패키지를 생성하지 않습니다.

자세한 내용은 [AL2 및 AL2023 비교](#)를 참조하세요.

AL2, AL2023, Bottlerocket 간 FIPS 검증 비교

Amazon Linux 2(AL2), Amazon Linux 2023(AL2023) 및 Bottlerocket은 FIPS(연방 정보 처리 표준) 규정 준수를 지원합니다.

- AL2는 FIPS 140-2 인증을 획득했고 AL2023은 FIPS 140-3 인증을 획득했습니다. AL2023에서 FIPS 모드를 활성화하려면 Amazon EC2 인스턴스에 필요한 패키지를 설치하고 [AL2023에서 FIPS 모드 활성화](#)의 지침을 사용하여 구성 단계를 수행합니다. 자세한 내용은 [AL2023 FAQ](#)를 참조하세요.
- Bottlerocket은 커널 및 사용자 공간 구성 요소가 FIPS 140-3 암호화 모듈 검증 프로그램에 제출된 암호화 모듈만 사용하도록 제한하는 FIPS 전용 변형을 제공합니다.

EKS AMI 드라이버 및 버전 변경 로그

모든 EKS AMI 구성 요소 및 해당 버전의 전체 목록은 GitHub의 [Amazon EKS AMI Release Notes](#)를 참조하세요.

최적화된 Amazon Linux AMI를 사용한 노드 생성

Amazon EKS 최적화 Amazon Linux AMI는 Amazon Linux 2(AL2) 및 Amazon Linux 2023(AL2023) 기반으로 빌드됩니다. Amazon EKS 노드의 기본 이미지 역할을 하도록 구성되었습니다. AMI는 Amazon EKS와 연동하도록 구성되며 다음과 같은 구성 요소가 포함됩니다.

- kubelet
- AWS IAM 인증자
- containerd

Note

- 원하는 버전에 대한 탭을 선택하여 [Amazon Linux 보안 센터](#)에서 Amazon Linux의 보안 또는 개인정보 보호 이벤트를 추적할 수 있습니다. 해당 RSS 피드를 구독할 수도 있습니다. 보안 및 프라이버시 이벤트에는 문제의 개요, 영향을 받는 패키지 및 인스턴스를 업데이트하여 문제를 해결하는 방법이 포함됩니다.
- 가속 또는 Arm AMI를 배포하기 전에 [Amazon EKS 최적화 가속 Amazon Linux AMI](#) 및 [the section called “Amazon EKS 최적화 Arm Amazon Linux AMI”](#)의 정보를 검토합니다.
- Amazon EC2 P2 인스턴스는 NVIDIA 드라이버 버전 470 이하가 필요하기 때문에 Amazon EKS에서 지원되지 않습니다.
- 버전 1.30 이상의 클러스터부터 새로 생성되는 모든 관리형 노드 그룹은 자동으로 AL2023을 기본 노드 운영 체제로 사용합니다. 이전에는 새 노드 그룹이 AL2를 기본 노드 운영 체제로 사용했습니다. 새 노드 그룹을 생성할 때 AL2를 AMI 유형으로 선택하여 계속 사용할 수 있습니다.
- Amazon EKS는 2025년 11월 26일 이후 더 이상 EKS 최적화 Amazon Linux 2(AL2) AMI를 게시하지 않습니다. 추가로 Kubernetes 버전 1.32는 Amazon EKS가 AL2 AMI를 출시하는 마지막 버전입니다. 버전 1.33부터 Amazon EKS는 AL2023 및 Bottlerocket 기반 AMI를 계속 출시합니다.

Amazon EKS 최적화 가속 Amazon Linux AMI

Amazon EKS 최적화 가속 Amazon Linux AMI는 표준 Amazon EKS 최적화 Amazon Linux AMI를 기반으로 빌드됩니다. 이는 GPU, [Inferentia](#) 및 [Trainium](#) 기반 워크로드를 지원하기 위해 Amazon EKS 노드에 대한 선택적 이미지 역할을 하도록 구성되었습니다.

표준 Amazon EKS 최적화 AMI 구성 외에도 가속 AMI에는 다음이 포함됩니다.

- NVIDIA 드라이버
- nvidia-container-toolkit
- AWS Neuron 드라이버

가속 AMI에 포함된 최신 구성 요소 목록은 GitHub의 [amazon-eks-ami Releases](#)를 참조하세요.

Note

- 노드 AWS CloudFormation 템플릿에서 적용 가능한 인스턴스 유형을 지정해야 합니다. Amazon EKS에 최적화된 가속 AMI를 사용하면 귀하는 [NVIDIA의 클라우드 최종 사용자 라이선스 계약\(EULA\)](#)에 동의하게 됩니다.
- 이전에는 Amazon EKS 최적화 가속 AMI를 GPU를 지원하는 Amazon EKS 최적화 AMI라고 불렀습니다.
- Amazon EKS 최적화 가속 AMI의 이전 버전에서는 nvidia-docker 리포지토리를 설치했습니다. Amazon EKS AMI 버전 v20200529 이상에는 더 이상 이 리포지토리가 포함되지 않습니다.

Amazon EKS 최적화 가속 Amazon Linux AMI에서 워크로드를 실행하는 방법에 대한 자세한 내용은 [the section called “Linux GPU AMI 실행”](#) 섹션을 참조하세요.

Amazon EKS 최적화 Arm Amazon Linux AMI

Arm 인스턴스는 웹 서버, 컨테이너식 마이크로서비스, 캐싱 플릿 및 분산 데이터 스토어와 같은 스케일 아웃 및 Arm 기반 애플리케이션에 상당한 비용 절감 효과를 제공합니다. 클러스터에 Arm 노드를 추가할 때 다음 고려 사항을 검토하세요.

- 클러스터가 2020년 8월 17일 이전에 배포된 경우 중요한 클러스터 추가 기능 매니페스트의 일회성 업그레이드를 수행해야 합니다. 이는 Kubernetes가 클러스터에서 사용 중인 각 하드웨어 아키텍처에 대해 올바른 이미지를 가져올 수 있도록 하기 위한 것입니다. 클러스터 추가 기능을 업데이트하는 방법에 대한 자세한 내용은 [the section called “1단계: 업그레이드 준비”](#) 부분을 참조하세요. 2020년 8월 17일 또는 이후에 클러스터를 배포한 경우는 이미 CoreDNS, kube-proxy 및 Kubernetes용 Amazon VPC CNI 플러그인이 다중 아키텍처를 지원합니다.
- Arm 노드에 배포된 애플리케이션은 Arm용으로 컴파일되어야 합니다.
- 기존 클러스터에 배포된 DaemonSet이 있거나, Arm 노드도 배포할 새 클러스터에 DaemonSet을 배포하려는 경우 DaemonSet이 클러스터의 모든 하드웨어 아키텍처에서 실행될 수 있는지 확인합니다.
- 동일한 클러스터에서 Arm 노드 그룹과 x86 노드 그룹을 실행할 수 있습니다. 이 경우에 다중 아키텍처 컨테이너 이미지를 Amazon Elastic Container Registry와 같은 컨테이너 리포지토리에 배포한 다음, 매니페스트에 노드 선택기를 추가하여 어떤 하드웨어에 포드를 배포할 수 있는지 Kubernetes가

알 수 있도록 해야 합니다. 자세한 내용은 Amazon ECR 사용 설명서의 [다중 아키텍처 이미지 푸시](#) 및 [Amazon ECR용 다중 아키텍처 컨테이너 이미지 소개](#) 블로그 게시물을 참조하세요.

추가 정보

Amazon EKS 최적화 Amazon Linux AMI 사용에 대한 자세한 내용은 다음 섹션을 참조하세요.

- 관리형 노드 그룹과 함께 Amazon Linux를 사용하려면 [the section called “관리형 노드 그룹”](#) 섹션을 참조하세요.
- 자체 관리형 Amazon Linux 노드를 시작하려면 [the section called “최신 ID 받기”](#) 섹션을 참조하세요.
- 버전 정보는 [the section called “버전 정보 받기”](#)를 참조하세요.
- Amazon EKS 최적화 Amazon Linux AMI의 최신 ID를 검색하려면 [the section called “최신 ID 받기”](#) 섹션을 참조하세요.
- Amazon EKS 최적화 AMI를 빌드하는 데 사용되는 오픈 소스 스크립트는 [the section called “사용자 지정 빌드”](#) 섹션을 참조하세요.

Amazon Linux 2에서 Amazon Linux 2023으로 업그레이드

Amazon EKS 최적화 AMI는 AL2와 AL2023 기반으로 제공됩니다. AL2023은 클라우드 애플리케이션에 안전하고 안정적이면서 고성능의 환경을 제공하도록 설계된 새로운 Linux 기반 운영 체제입니다. Amazon Web Services의 차세대 Amazon Linux로, 지원되는 모든 Amazon EKS 버전에서 사용할 수 있습니다.

AL2023은 AL2에 비해 몇 가지 향상된 기능을 제공합니다. 전체 비교는 Amazon Linux 2023 사용 설명서의 [AL2와 Amazon Linux 2023 비교](#)를 참조하세요. AL2에서 여러 패키지가 추가, 업그레이드 및 제거되었습니다. 업그레이드하기 전에 AL2023 버전으로 애플리케이션을 테스트하는 것이 좋습니다. AL2023 패키지의 모든 변경 사항 목록은 Amazon Linux 2023 릴리스 정보의 [Amazon Linux 2023의 패키지 변경 사항](#)을 참조하세요.

이러한 변경 사항 외에도 다음 사항을 숙지해야 합니다.

- AL2023에 YAML 구성 스키마를 사용하는 새로운 노드 초기화 프로세스 nodeadm이 도입됩니다. 자체 관리형 노드 그룹 또는 시작 템플릿이 있는 AMI를 사용하는 경우 이제 새 노드 그룹을 생성할 때 추가 클러스터 메타데이터를 명시적으로 제공해야 합니다. 최소 필수 파라미터의 [예시](#)는 다음과 같으며, 여기에서 apiServerEndpoint, certificateAuthority 및 서비스 cidr가 필수입니다.

```
---
apiVersion: node.eks.aws/v1alpha1
```

```
kind: NodeConfig
spec:
  cluster:
    name: my-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    cidr: 10.100.0.0/16
```

AL2에서 이러한 파라미터의 메타데이터는 Amazon EKS DescribeCluster API 직접 호출에서 확인되었습니다. AL2023 버전에서는 대형 노드 스케일 업 도중 추가 API 직접 호출로 인해 제한이 발생할 위험이 있기 때문에 이러한 동작이 변경되었습니다. 시작 템플릿이 없는 관리형 노드 그룹을 사용하거나 Karpenter를 사용하는 경우 이 변경 사항이 영향을 미치지 않습니다. certificateAuthority 및 서비스 cidr에 대한 자세한 내용은 Amazon EKS API 참조의 [DescribeCluster](#)를 참조하세요.

- AL2023의 경우 nodeadm은 각 노드에 대해 [NodeConfigSpec](#)을 사용하여 파라미터를 kubelet에 적용하도록 형식도 변경합니다. AL2에서는 --kubelet-extra-args 파라미터를 사용하여 이 작업을 수행했습니다. 이는 일반적으로 노드에 레이블과 테인트를 추가하는 데 사용됩니다. 아래 예제는 노드에 maxPods 및 --node-labels를 적용하는 방법을 보여줍니다.

```
---
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: test-cluster
    apiServerEndpoint: https://example.com
    certificateAuthority: Y2VydGlmaWNhdGVBdXRob3JpdHk=
    cidr: 10.100.0.0/16
  kubelet:
    config:
      maxPods: 110
    flags:
      - --node-labels=karpenter.sh/capacity-type=on-demand,karpenter.sh/nodepool=test
```

- AL2023의 경우 Amazon VPC CNI 버전 1.16.2 이상이 필요합니다.
- AL2023의 필수 기본값은 IMDSv2입니다. IMDSv2에는 보안 태세 개선에 도움이 되는 몇 가지 이점이 있습니다. 세션을 시작하려면 간단한 HTTP PUT 요청으로 비밀 토큰을 생성해야 하는 세션 지향 인증 방법을 사용합니다. 세션의 토큰은 1초에서 6시간 사이로 유효할 수 있습니다. IMDSv1에서 IMDSv2로 전환하는 방법에 대한 자세한 내용은 [인스턴스 메타데이터 서비스 버전 2 사용으로 전환](#) 및 [Get the full benefits of IMDSv2 and disable IMDSv1 across your AWS infrastructure](#)를 참조하세요.

요. IMDSv1을 사용하려는 경우 인스턴스 메타데이터 옵션 시작 속성을 사용하여 설정을 수동으로 재정의하면 계속 사용할 수 있습니다.

Note

IMDSv2의 경우 관리형 노드 그룹의 기본 홉 수는 1로 설정되어 있습니다. 따라서 컨테이너는 IMDS를 사용하여 노드의 자격 증명에 액세스할 수 없습니다. 노드의 자격 증명에 대한 컨테이너 액세스가 필요한 경우, [사용자 정의 Amazon EC2 시작 템플릿](#)에서 `HttpPutResponseHopLimit`를 수동으로 재정의하여 2로 늘릴 수 있으며, 또는 [Amazon EKS pod identity](#)를 사용하여 IMDSv2 대신 자격 증명을 제공할 수 있습니다.

- AL2023에는 차세대 통합 제어 그룹 계층 구조(cgroupv2)가 있습니다. cgroupv2는 컨테이너 런타임을 구현하는 데 사용되며, systemd에 따라 사용됩니다. AL2023에 시스템이 cgroupv1을 사용하여 실행하도록 할 수 있는 코드가 포함되어 있지만 이 구성은 권장되거나 지원되지 않습니다. 이 구성은 Amazon Linux의 향후 메이저 릴리스에서 완전히 제거될 예정입니다.
- eksctl이 AL2023을 지원하려면 eksctl 버전 0.176.0 이상이 필요합니다.

기존 관리형 노드 그룹의 경우 시작 템플릿을 사용하는 방식에 따라 인플레이스 업그레이드 또는 블루/그린 업그레이드를 수행할 수 있습니다.

- 관리형 노드 그룹에서 사용자 지정 AMI를 사용하는 경우 시작 템플릿에서 AMI ID를 교체하여 인플레이스 업그레이드를 수행할 수 있습니다. 이 업그레이드 전략을 수행하기 전에 먼저 애플리케이션과 사용자 데이터가 AL2023으로 전송되는지 확인해야 합니다.
- 표준 시작 템플릿 또는 AMI ID를 지정하지 않은 사용자 지정 시작 템플릿에서 관리형 노드 그룹을 사용하는 경우 블루/그린 전략을 사용하여 업그레이드해야 합니다. 블루/그린 업그레이드는 대체로 더 복잡하고 AMI 유형으로 AL2023을 지정하는 완전히 새로운 노드 그룹을 생성해야 합니다. 이후 AL2 노드 그룹의 모든 사용자 지정 데이터가 새 OS와 호환되도록 새 노드 그룹을 신중하게 구성해야 합니다. 애플리케이션에서 새 노드 그룹을 테스트하고 검증한 후에는 이전 노드 그룹에서 새 노드 그룹으로 포드를 마이그레이션할 수 있습니다. 마이그레이션이 완료되면 이전 노드 그룹을 삭제할 수 있습니다.

Karpenter를 사용 중이고 AL2023을 사용하려는 경우 `EC2NodeClass` `amiFamily` 필드를 AL2023으로 수정해야 합니다. 기본적으로 드리프트는 Karpenter에서 활성화됩니다. 따라서 `amiFamily` 필드가 변경되면 Karpenter에서 사용 가능할 때 워커 노드를 최신 AMI로 자동으로 업데이트합니다.

Amazon Linux AMI 버전 정보 검색

Amazon EKS 최적화 Amazon Linux AMI는 Kubernetes 버전과 AMI의 출시 날짜에 따라 다음 형식으로 버전이 지정됩니다.

```
k8s_major_version.k8s_minor_version.k8s_patch_version-release_date
```

각 AMI 릴리스에는 다양한 버전의 [kubelet](#), Linux 커널 및 [containerd](#)가 포함되어 있습니다. 가속 AMI에는 다양한 버전의 NVIDIA 드라이버도 포함되어 있습니다. 이 버전 정보는 GitHub의 [Changelog](#)에서 확인할 수 있습니다.

권장 Amazon Linux AMI ID 검색

노드를 배포할 때 사전 구축된 Amazon EKS 최적화 Amazon Machine Image(AMI)의 ID를 지정할 수 있습니다. 원하는 구성에 적합한 AMI ID를 검색하려면 AWS Systems Manager Parameter Store API를 쿼리합니다. 이 API를 사용하면 Amazon EKS 최적화 AMI ID를 수동으로 조회할 필요가 없습니다. 자세한 내용은 [GetParameter](#)를 참조하세요. Amazon EKS 최적화 AMI 메타데이터를 검색하려면 [IAM 보안 주체](#)에 `ssm:GetParameter` IAM 권한이 있어야 합니다.

하위 파라미터 `image_id`를 사용하는 다음 명령을 사용하여 최신 권장 Amazon EKS에 최적화된 Amazon Linux AMI의 이미지 ID를 검색할 수 있습니다. 필요에 따라 명령을 다음과 같이 수정한 다음에 수정한 명령을 실행합니다.

- `<kubernetes-version>`을 지원되는 [Amazon EKS 버전](#)으로 변경합니다.
- `ami-type`를 다음 옵션 중 하나로 변경합니다. Amazon EC2 인스턴스 유형에 대한 자세한 내용은 [Amazon EC2 인스턴스 유형](#)을 참조하세요.
 - Amazon Linux 2023(AL2023) x86 기반 인스턴스에는 `amazon-linux-2023/x86_64/standard`를 사용합니다.
 - [AWS Graviton](#) 기반 인스턴스와 같은 AL2023 ARM 인스턴스에는 `amazon-linux-2023/arm64/standard`를 사용합니다.
 - 승인된 최신 AL2023 NVIDIA x86 기반 인스턴스에 `amazon-linux-2023/x86_64/nvidia`를 사용합니다.
 - 승인된 최신 AL2023 NVIDIA arm64 기반 인스턴스에 `amazon-linux-2023/arm64/nvidia`를 사용합니다.
 - 최신 AL2023 [AWS Neuron](#) 인스턴스에는 `amazon-linux-2023/x86_64/neuron`을 사용하세요.
 - `amazon-linux-2`를 Amazon Linux 2(AL2) x86 기반 인스턴스에 사용합니다.

- [AWS Graviton](#) 기반 인스턴스와 같은 AL2 ARM 인스턴스에는 *amazon-linux-2-arm64*를 사용합니다.
- NVIDIA GPU, [Inferentia](#) 및 [Trainium](#) 기반 워크로드에 대한 AL2 [하드웨어 가속](#) x86 기반 인스턴스에는 *amazon-linux-2-gpu*를 사용하세요.
- <region-code>를 AMI ID를 원하는 [Amazon EKS 지원 AWS 리전](#)으로 변경합니다.

```
aws ssm get-parameter --name /aws/service/eks/optimized-ami/<kubernetes-version>/<ami-type>/recommended/image_id \
  --region <region-code> --query "Parameter.Value" --output text
```

다음은 자리 표시자를 대체한 후의 명령 예제입니다.

```
aws ssm get-parameter --name /aws/service/eks/optimized-ami/1.31/amazon-linux-2023/x86_64/standard/recommended/image_id \
  --region us-west-2 --query "Parameter.Value" --output text
```

예제 출력은 다음과 같습니다.

```
ami-1234567890abcdef0
```

사용자 지정 Amazon Linux AMI 빌드

Important

Amazon EKS는 2025년 11월 26일 이후 더 이상 EKS 최적화 Amazon Linux 2(AL2) AMI를 게시하지 않습니다. 추가로 Kubernetes 버전 1.32는 Amazon EKS가 AL2 AMI를 출시하는 마지막 버전입니다. 버전 1.33부터 Amazon EKS는 AL2023 및 Bottlerocket 기반 AMI를 계속 출시합니다.

Amazon EKS 최적화 Amazon Linux(AL) AMI는 특히 Amazon EKS 클러스터에서 노드로 사용하기 위해 AL2 및 AL2023 기반으로 빌드됩니다. Amazon EKS는 다음과 같은 방법으로 사용할 수 있는 오픈 소스 빌드 스크립트를 [Amazon EKS AMI 빌드 사양](#) 리포지토리에 제공합니다. * kubelet, 런타임 및 Kubernetes용 AWS IAM 인증자에 대한 구성을 확인합니다. * 처음부터 AL 기반 AMI를 직접 빌드합니다.

이 리포지토리에는 부팅 시 실행되는 특수 [부트스트랩 스크립트](#)와 [nodeadm 스크립트](#)가 포함되어 있습니다. 이러한 스크립트는 인스턴스의 인증서 데이터, 컨트롤 플레인 엔드포인트, 클러스터 이름 등을 구성합니다. 스크립트는 Amazon EKS 최적화 AMI 빌드의 신뢰할 수 있는 소스로 간주되므로 GitHub 리포지토리를 따라 AMI에 대한 변경을 모니터링할 수 있습니다.

사전 조건

- [AWS CLI 설치](#)
- [HashiCorp Packer v1.9.4+ 설치](#)
- [GNU Make 설치](#)

빠른 시작

이 섹션에서는 AWS 계정에서 사용자 지정 AMI를 생성하는 명령을 보여줍니다. AMI를 사용자 지정하는 데 사용할 수 있는 구성에 대한 자세한 내용은 [Amazon Linux 2023](#) 페이지의 템플릿 변수를 참조하세요.

1단계. 환경 설정

공식 Amazon EKS AMI 리포지토리를 복제하거나 포크합니다. 예시:

```
git clone https://github.com/aws-labs/amazon-eks-ami.git
cd amazon-eks-ami
```

Packer가 설치되어 있는지 확인합니다.

```
packer --version
```

2단계. 사용자 지정 AMI를 생성

다음은 다양한 사용자 지정 AMI에 대한 명령의 예입니다.

기본 NVIDIA AL2 AMI:

```
make k8s=1.31 os_distro=al2 \
  enable_accelerator=nvidia \
  nvidia_driver_major_version=560 \
  enable_efa=true
```

기본 NVIDIA AL2023 AMI:

```
make k8s=1.31 os_distro=al2023 \
  enable_accelerator=nvidia \
  nvidia_driver_major_version=560 \
  enable_efa=true
```

STIG 준수 Neuron AL2023 AMI:

```
make k8s=1.31 os_distro=al2023 \
  enable_accelerator=neuron \
  enable_fips=true \
  source_ami_id=ami-0abcd1234efgh5678 \
  kms_key_id=alias/aws-stig
```

이러한 명령을 실행하면 Packer가 다음을 수행합니다. * 임시 Amazon EC2 인스턴스를 시작합니다. * Kubernetes 구성 요소, 드라이버 및 구성을 설치합니다. * AWS 계정에서 AMI를 생성합니다.

3단계. 기본값 보기

기본값과 추가 옵션을 보려면 다음 명령을 실행합니다.

```
make help
```

최적화된 Bottlerocket AMI를 사용한 노드 생성

[Bottlerocket](#)은 AWS에서 후원하고 지원하는 오픈 소스 Linux 배포판입니다. Bottlerocket은 컨테이너 워크로드 호스팅용으로 특별히 설계되었습니다. Bottlerocket을 사용하면 컨테이너 인프라 업데이트를 자동화하여 컨테이너화된 배포의 가용성을 개선하고 운영 비용을 절감할 수 있습니다. Bottlerocket에는 컨테이너를 실행하는 데 필수적인 소프트웨어만 포함되어 있어 리소스 사용량이 개선되고 보안 위협과 관리 오버헤드가 감소합니다. Bottlerocket AMI에는 containerd, kubelet 및 AWS IAM 인증자가 포함됩니다. Bottlerocket은 관리형 노드 그룹과 자체 관리형 노드 외에도 [Karpenter](#)에서 지원됩니다.

장점

Amazon EKS 클러스터와 함께 Bottlerocket을 사용하면 다음과 같은 이점이 있습니다.

- 낮은 운영 비용과 낮은 관리 복잡성으로 가동 시간 향상 - Bottlerocket은 다른 Linux 배포판보다 리소스 사용량이 적고 부팅 시간이 짧으며 보안 위협에 덜 취약합니다. Bottlerocket은 사용량이 적어 스토리지, 컴퓨팅 및 네트워킹 리소스를 덜 사용하므로 비용을 절감하는 데 도움이 됩니다.

- 자동 OS 업데이트로 보안 강화 – Bottlerocket 업데이트는 필요한 경우 롤백할 수 있는 단일 단위로 적용됩니다. 이렇게 하면 시스템을 사용 불가 상태로 둘 수 있는 손상되거나 실패한 업데이트의 위험이 제거됩니다. Bottlerocket을 사용하면 보안 업데이트를 사용할 수 있는 즉시 종단을 최소화하는 방식으로 자동 적용하고 장애 발생 시 롤백할 수 있습니다.
- 프리미엄 지원 – AWS에서 제공하는 Amazon EC2 기반 Bottlerocket 빌드에는 Amazon EC2, Amazon EKS, Amazon ECR 등의 AWS 서비스에도 적용되는 동일한 AWS Support 플랜이 적용됩니다.

고려 사항

AMI 유형에 Bottlerocket을 사용할 때 다음 사항을 고려하세요.

- Bottlerocket은 x86_64 및 arm64 프로세서가 있는 Amazon EC2 인스턴스를 지원합니다. Bottlerocket AMI를 Inferentia 칩이 있는 Amazon EC2 인스턴스와 함께 사용하는 것은 권장되지 않습니다.
- Bottlerocket 이미지에는 SSH 서버 또는 셸이 포함되지 않습니다. 대역 외 액세스 방법을 사용하여 SSH를 허용할 수 있습니다. 이 접근 방식을 통해 관리 컨테이너를 활성화하고 사용자 데이터와 함께 일부 부트스트래핑 구성 단계를 전달할 수 있습니다. 자세한 내용은 GitHub의 [Bottlerocket OS](#)에서 다음 섹션을 참조하세요.
 - [탐색](#)
 - [관리자 컨테이너](#)
 - [Kubernetes 설정](#)
- Bottlerocket은 다음과 같은 여러 컨테이너 유형을 사용합니다.
 - 기본적으로 [제어 컨테이너](#)는 활성화되어 있습니다. 이 컨테이너는 Amazon EC2 Bottlerocket 인스턴스에서 명령을 실행하거나 셸 세션을 시작하는 데 사용할 수 있는 [AWS Systems Manager 에이전트](#)를 실행합니다. 자세한 내용은 AWS 사용 설명서의 [세션 관리자 설정](#)을 참조하세요.
 - 노드 그룹을 생성할 때 SSH 키가 제공되는 경우 관리 컨테이너가 활성화됩니다. 개발 및 테스트 시나리오에만 관리 컨테이너를 사용하는 것이 좋습니다. 프로덕션 환경에서는 이를 사용하지 않는 것이 좋습니다. 자세한 내용은 GitHub의 [Admin container](#)를 참조하세요.

추가 정보

Amazon EKS 최적화 Bottlerocket AMI 사용에 대한 자세한 내용은 다음 섹션을 참조하세요.

- Bottlerocket에 대한 자세한 내용은 [Bottlerocket 설명서](#)를 참조하세요.

- 버전 정보 리소스는 [the section called “버전 정보 받기”](#) 섹션을 참조하세요.
- 관리형 노드 그룹과 함께 Bottlerocket을 사용하려면 [the section called “관리형 노드 그룹”](#) 섹션을 참조하세요.
- 자체 관리형 Bottlerocket 노드를 시작하려면 [the section called “Bottlerocket”](#) 섹션을 참조하세요.
- Amazon EKS 최적화 Bottlerocket AMI의 최신 ID를 검색하려면 [the section called “최신 ID 받기”](#) 섹션을 참조하세요.
- 규정 준수 지원에 대한 자세한 내용은 [the section called “규정 준수 지원”](#) 섹션을 참조하세요.

Bottlerocket AMI 버전 정보 검색

각 Bottlerocket AMI 릴리스에는 다양한 버전의 [kubelet](#), Bottlerocket 커널, [containerd](#)가 포함되어 있습니다. 가속화 AMI 변형에는 다양한 버전의 NVIDIA 드라이버가 포함되어 있습니다. 이 버전 정보는 Bottlerocket 설명서의 [OS](#) 주제에서 확인할 수 있습니다. 이 페이지에서 해당 버전 정보 하위 주제로 이동합니다.

Bottlerocket 설명서가 GitHub에서 제공되는 버전보다 뒤처지는 경우가 있습니다. 최신 버전에 대한 변경 사항 목록은 GitHub의 [releases](#)에서 확인할 수 있습니다.

권장 Bottlerocket AMI ID 검색

노드를 배포할 때 사전 구축된 Amazon EKS 최적화 Amazon Machine Image(AMI)의 ID를 지정할 수 있습니다. 원하는 구성에 적합한 AMI ID를 검색하려면 AWS Systems Manager Parameter Store API를 쿼리합니다. 이 API를 사용하면 Amazon EKS 최적화 AMI ID를 수동으로 조회할 필요가 없습니다. 자세한 내용은 [GetParameter](#)를 참조하세요. Amazon EKS 최적화 AMI 메타데이터를 검색하려면 [IAM 보안 주체](#)에 `ssm:GetParameter` IAM 권한이 있어야 합니다.

하위 파라미터 `image_id`를 사용하는 다음 AWS CLI 명령을 사용하여 최신 권장 Amazon EKS 최적화 Bottlerocket AMI의 이미지 ID를 검색할 수 있습니다. 필요에 따라 명령을 다음과 같이 수정한 다음에 수정한 명령을 실행합니다.

- [kubernetes 버전](#)을 지원되는 *Amazon EKS ##*으로 교체합니다.
- *-flavor*를 다음 옵션 중 하나로 변경합니다.
 - GPU가 없는 변형의 경우 *-flavor*를 제거합니다.
 - GPU 지원 변형에 *-nvidia*를 사용합니다.
 - FIPS 지원 변형에 *-fips*를 사용합니다.
- *####*를 다음 옵션 중 하나로 변경합니다.

- `x86_64`를 x86 기반 인스턴스에 사용합니다.
- `arm64`를 ARM 인스턴스에 사용합니다.
- `region-code`를 AMI ID를 원하는 [Amazon EKS 지원 AWS 리전](#)으로 바꿉니다.

```
aws ssm get-parameter --name /aws/service/bottlerocket/aws-k8s-kubernetes-version-flavor/architecture/latest/image_id \
  --region region-code --query "Parameter.Value" --output text
```

다음은 자리 표시자를 대체한 후의 명령 예제입니다.

```
aws ssm get-parameter --name /aws/service/bottlerocket/aws-k8s-1.31/x86_64/latest/
image_id \
  --region us-west-2 --query "Parameter.Value" --output text
```

예제 출력은 다음과 같습니다.

```
ami-1234567890abcdef0
```

Bottlerocket으로 규정 준수 요구 사항 충족

Bottlerocket은 다양한 조직에서 정의한 권장 사항을 준수합니다.

- Bottlerocket에 대해 정의된 [CIS Benchmark](#)가 있습니다. 기본 구성에서 Bottlerocket 이미지는 CIS 레벨 1 구성 프로필에 필요한 대부분의 컨트롤이 있습니다. CIS 레벨 2 구성 프로필에 필요한 컨트롤을 구현할 수 있습니다. 자세한 내용은 AWS 블로그의 [CIS Benchmark에 대해 Amazon EKS 최적화 Bottlerocket AMI 검증](#)을 참조하세요.
- 최적화된 기능 세트와 감소된 공격 표면은 Bottlerocket 인스턴스가 PCI DSS 요구 사항을 충족하는데 필요한 구성이 더 적다는 것을 의미합니다. [Bottlerocket용 CIS Benchmark](#)는 지침 강화를 위한 훌륭한 리소스이며 PCI DSS 요구 사항 2.2에 따른 보안 구성 표준에 대한 요구 사항을 지원합니다. 또한 [Fluent Bit](#)를 활용하여 PCI DSS 요구 사항 10.2에 따라 운영 체제 수준 감사 로깅에 대한 요구 사항을 지원할 수 있습니다. AWS는 PCI DSS 요구 사항 6.2(v3.2.1의 경우) 및 요구 사항 6.3.3(v4.0의 경우)을 충족할 수 있도록 새로운(패치된) Bottlerocket 인스턴스를 주기적으로 게시합니다.
- Bottlerocket은 Amazon EC2 및 Amazon EKS 모두에 대해 규제된 워크로드와 함께 사용하도록 승인된 HIPAA 적격 기능입니다. 자세한 내용은 [HIPAA 적격 서비스 참조](#)에서 확인하세요.

- FIPS 140-3 검증 암호화 모듈을 사용하도록 사전 구성된 Bottlerocket AMI를 사용할 수 있습니다. 여기에는 Amazon Linux 2023 Kernel Crypto API 암호화 모듈과 AWS-LC 암호화 모듈이 포함됩니다. 자세한 내용은 [the section called “Bottlerocket FIPS AMI”](#) 섹션을 참조하세요.

Bottlerocket FIPS AMI로 워커 노드의 FIPS 지원 제공

FIPS(Federal Information Processing Standard) Publication 140-3은 미국 및 캐나다 정부 보안 표준으로, 기밀 정보를 보호하는 암호 모듈의 보안 요건을 규정하고 있습니다. Bottlerocket은 FIPS 커널이 포함된 AMI를 제공하여 FIPS를 보다 쉽게 준수할 수 있도록 합니다.

이러한 AMI는 FIPS 140-3 검증 암호화 모듈을 사용하도록 사전 구성되어 있습니다. 여기에는 Amazon Linux 2023 Kernel Crypto API 암호화 모듈과 AWS-LC 암호화 모듈이 포함됩니다.

Bottlerocket FIPS AMI를 사용하면 워커 노드가 'FIPS 지원' 상태가 되지만 자동으로 'FIPS 준수' 상태가 되는 것은 아닙니다. 자세한 내용은 [FIPS\(Federal Information Processing Standard\) 140-3](#)을 참조하세요.

고려 사항

- 클러스터에서 격리된 서브넷을 사용하는 경우 Amazon ECR FIPS 엔드포인트에 액세스하지 못해 노드 부트스트랩이 실패할 수 있습니다. 네트워크 구성이 필요한 FIPS 엔드포인트에 대한 액세스를 허용하는지 확인합니다. 자세한 내용은 AWS PrivateLink 가이드의 [Access a resource through a resource VPC endpoint](#)를 참조하세요.
- 클러스터에서 [PrivateLink](#)가 있는 서브넷을 사용하는 경우 PrivateLink를 통해 Amazon ECR FIPS 엔드포인트를 사용할 수 없으므로 이미지 풀이 실패합니다.

Bottlerocket FIPS AMI로 관리형 노드 그룹 생성

Bottlerocket FIPS AMI는 워크로드를 지원하기 위해 두 가지 변형으로 제공됩니다.

- BOTTLEROCKET_x86_64_FIPS
- BOTTLEROCKET_ARM_64_FIPS

Bottlerocket FIPS AMI로 관리형 노드 그룹을 생성하려면 생성 프로세스 중 해당 AMI 유형을 선택합니다. 자세한 내용은 [the section called “생성”](#) 섹션을 참조하세요.

FIPS 지원 변형 선택에 대한 자세한 내용은 [the section called “최신 ID 받기”](#) 섹션을 참조하세요.

지원되지 않는 AWS 리전에 대해 FIPS 엔드포인트 비활성화

Bottlerocket FIPS AMI는 AWS GovCloud(미국) 리전을 포함하여 미국에서 직접 지원됩니다. AMI를 사용할 수 있지만 직접 지원되지 않는 AWS 리전의 경우 시작 템플릿으로 관리형 노드 그룹을 생성하여 AMI를 계속 사용할 수 있습니다.

Bottlerocket FIPS AMI는 부트스트랩 중에 일반적으로 미국 이외 리전에서는 사용할 수 없는 Amazon ECR FIPS 엔드포인트를 사용합니다. Amazon ECR FIPS 엔드포인트를 사용할 수 없는 AWS 리전에서 FIPS 커널에 AMI를 사용하려면 다음 단계를 수행하여 FIPS 엔드포인트를 비활성화하세요.

1. 다음 콘텐츠로 새 구성 파일을 생성하거나 기존 구성 파일에 콘텐츠를 통합합니다.

```
[default]
use_fips_endpoint=false
```

1. 파일 콘텐츠를 Base64 형식으로 인코딩합니다.
2. 시작 템플릿의 UserData에 TOML 형식을 사용하여 인코딩된 다음 문자열을 추가합니다.

```
[settings.aws]
config = "<your-base64-encoded-string>"
```

기타 설정은 GitHub의 Bottlerocket [설정 설명](#)을 참조하세요.

다음은 시작 템플릿의 UserData 예입니다.

```
[settings]
motd = "Hello from eksctl!"
[settings.aws]
config = "W2RlZmF1bHRdCnVzZV9maXBzX2VuZHBvaW50PWZhbHNlCg==" # Base64-encoded string.
[settings.kubernetes]
api-server = "<api-server-endpoint>"
cluster-certificate = "<cluster-certificate-authority>"
cluster-name = "<cluster-name>"
...<other-settings>
```

사용자 데이터로 시작 템플릿을 만드는 방법에 대한 자세한 내용은 [the section called “시작 템플릿”](#) 섹션을 참조하세요.

최적화된 Ubuntu Linux AMI를 사용한 노드 생성

Canonical은 Amazon EKS와 협력하여 클러스터에서 사용할 수 있는 노드 AMI를 만들었습니다.

[Canonical](#)은 특별히 구축된 Kubernetes Node OS 이미지를 제공합니다. 이 최소화된 Ubuntu 이미지는 Amazon EKS에 최적화되어 있으며 AWS와 공동 개발한 사용자 지정 AWS 커널을 포함합니다. 자세한 내용은 [Amazon Elastic Kubernetes Service\(EKS\)의 Ubuntu](#) 및 [the section called “Ubuntu Linux”](#) 섹션을 참조하세요. 지원에 대한 자세한 내용은 AWS 프리미엄 서포트 FAQ의 [서드 파티 소프트웨어](#) 섹션을 참조하세요.

최적화된 Windows AMI를 사용한 노드 생성

Windows Amazon EKS 최적화 AMI는 Windows Server 2019 및 Windows Server 2022를 기반으로 빌드되었습니다. Amazon EKS 노드의 기본 이미지 역할을 하도록 구성되었습니다. 기본적으로 AMI에는 다음 구성 요소가 포함됩니다.

- [kubelet](#)
- [kube-proxy](#)
- [AWS Kubernetes용 IAM 인증자](#)
- [csi-proxy](#)
- [containerd](#)

Note

[Microsoft 보안 업데이트 설명서](#)를 사용하여 Windows Server의 보안 또는 개인 정보 보호 이벤트를 추적할 수 있습니다.

Amazon ECS는 Windows 컨테이너에 최적화된 AMI를 다음 변형된 형태로 제공합니다.

- Amazon EKS 최적화 Windows Server 2019 Core AMI
- Amazon EKS 최적화 Windows Server 2019 Full AMI
- Amazon EKS 최적화 Windows Server 2022 Core AMI
- Amazon EKS 최적화 Windows Server 2022 Full AMI

⚠ Important

- Amazon ECS 최적화 Windows Server 20H2 Core AMI는 사용되지 않습니다. 이 AMI의 새 버전은 릴리스되지 않습니다.
- 항상 최신 보안 업데이트를 기본으로 유지할 수 있도록 Amazon EKS는 지난 4개월 동안 최적화 Windows AMI를 유지합니다. 각각의 새 AMI는 최초 릴리스 시점부터 4개월 동안 사용할 수 있습니다. 이 기간이 지나면 오래된 AMI는 프라이빗으로 전환되어 더 이상 액세스할 수 없습니다. 보안 취약성을 방지하고 지원 수명이 다한 오래된 AMI에 대한 액세스를 상실하지 않도록 최신 AMI를 사용하는 것이 좋습니다. 프라이빗으로 설정된 AMI에 대한 액세스 제공을 보장할 수는 없지만 AWS Support에 티켓을 제출하여 액세스를 요청할 수 있습니다.

출시 일정

다음 표에는 Amazon EKS의 Windows 버전 릴리스 및 지원 종료 날짜가 나와 있습니다. 종료 날짜가 비어 있으면 버전이 계속 지원되는 것입니다.

Windows 버전	Amazon EKS 릴리스	Amazon EKS 지원 종료
Windows Server 2022 Core	10/17/2022	
Windows Server 2022 Full	10/17/2022	
Windows Server 20H2 Core	8/12/2021	8/9/2022
Windows Server 2004 Core	8/19/2020	12/14/2021
Windows Server 2019 Core	10/7/2019	
Windows Server 2019 Full	10/7/2019	
Windows Server 1909 Core	10/7/2019	12/8/2020

부트스트랩 스크립트 구성 파라미터

Windows 노드를 생성할 때 노드에 다양한 파라미터를 구성할 수 있는 스크립트가 있습니다. 설정에 따라 이 스크립트는 C:\Program Files\Amazon\EKS\Start-EKSBootstrap.ps1과 유사한 위치의 노드에서 찾을 수 있습니다. 사용자 지정 파라미터 값을 부트스트랩 스크립트의 인수로 지정하여 설

정할 수 있습니다. 예를 들어, 시작 템플릿에서 사용자 데이터를 업데이트할 수 있습니다. 자세한 내용은 [the section called “Amazon EC2 사용자 데이터”](#) 섹션을 참조하세요.

이 스크립트에는 다음 명령줄 파라미터가 포함되어 있습니다.

- `-EKSClusterName` - 조인할 이 작업자 노드의 Amazon EKS 클러스터 이름을 지정합니다.
- `-KubeletExtraArgs` - kubelet에 대한 추가 인수를 지정합니다(선택사항).
- `-KubeProxyExtraArgs` - kube-proxy에 대한 추가 인수를 지정합니다(선택사항).
- `-APIServerEndpoint` - Amazon EKS 클러스터 API 서버 엔드포인트를 지정합니다(선택사항). `-Base64ClusterCA`와 함께 사용하는 경우에만 유효합니다. `Get-EKSCluster`.
- `-Base64ClusterCA` - base64로 인코딩된 클러스터 CA 콘텐츠를 지정합니다(선택사항). `-APIServerEndpoint`와 함께 사용하는 경우에만 유효합니다. `Get-EKSCluster` 호출을 무시합니다.
- `-DNSClusterIP` - 클러스터 내의 DNS 쿼리에 사용할 IP 주소를 재정의합니다(선택사항). 기본 인터페이스의 IP 주소에 따라 기본값은 10.100.0.10 또는 172.20.0.10입니다.
- `-ServiceCIDR` - 클러스터 서비스가 처리되는 Kubernetes 서비스 IP 주소 범위를 재정의합니다. 기본 인터페이스의 IP 주소에 따라 기본값은 172.20.0.0/16 또는 10.100.0.0/16입니다.
- `-ExcludedSnatCIDRs` - SNAT(Source Network Address Translation)에서 제외할 IPv4 CIDR 목록입니다. 즉, VPC 주소 지정이 가능한 포트 프라이빗 IP는 아웃바운드 트래픽에 대한 인스턴스 ENI의 기본 IPv4 주소의 IP 주소로 변환되지 않습니다. 기본적으로 Amazon EKS Windows 노드에 대한 VPC의 IPv4 CIDR이 추가됩니다. 이 파라미터에 CIDR을 지정하면 지정된 CIDR도 추가로 제외됩니다. 자세한 내용은 [the section called “아웃바운드 트래픽”](#) 섹션을 참조하세요.

명령줄 파라미터 외에도 일부 환경 변수 파라미터를 지정할 수도 있습니다. 명령줄 파라미터를 지정하는 경우 해당 파라미터가 해당 환경 변수보다 우선시됩니다. 부트스트랩 스크립트는 컴퓨터 범위 변수만 읽으므로 환경 변수는 컴퓨터(또는 시스템) 범위로 정의해야 합니다.

이 스크립트는 다음과 같은 환경 변수를 고려합니다.

- `SERVICE_IPV4_CIDR`— 정의는 `ServiceCIDR` 명령줄 파라미터를 참조하세요.
- `EXCLUDED_SNAT_CIDRS`— 쉼표로 구분된 문자열이어야 합니다. 정의는 `ExcludedSnatCIDRs` 명령줄 파라미터를 참조하세요.

gMSA 인증 지원

Amazon EKS Windows 포드는 다양한 유형의 그룹 관리형 서비스 계정(gMSA) 인증을 허용합니다.

- Amazon EKS는 인증을 위해 Active Directory 도메인 ID를 지원합니다. 도메인 조인 gMSA에 대한 자세한 내용은 AWS 블로그에서 [Amazon EKS Windows pods의 Windows 인증](#)을 참조하세요.
- Amazon EKS는 도메인에 가입되지 않은 Windows 노드가 이식 가능한 사용자 ID로 gMSA ID를 검색할 수 있도록 하는 플러그인을 제공합니다. 도메인이 없는 gMSA에 대한 자세한 내용은 AWS 블로그에서 [Amazon EKS Windows pod용 도메인이 없는 Windows 인증](#)을 참조하세요.

캐시된 컨테이너 이미지

Amazon EKS Windows 최적화 AMI에는 containerd 런타임에 대해 캐시된 특정 컨테이너 이미지가 있습니다. 컨테이너 이미지는 Amazon 관리형 빌드 구성 요소를 사용하여 사용자 지정 AMI를 빌드할 때 캐시됩니다. 자세한 내용은 [the section called “Amazon 관리형 빌드 구성 요소 사용”](#) 섹션을 참조하세요.

다음 캐시된 컨테이너 이미지는 containerd 런타임용입니다.

- amazonaws.com/eks/pause-windows
- mcr.microsoft.com/windows/nanoserver
- mcr.microsoft.com/windows/servercore

추가 정보

Amazon EKS 최적화 Windows AMI 사용에 대한 자세한 내용은 다음 섹션을 참조하세요.

- Amazon EKS 최적화 가속 Windows AMI에서 워크로드를 실행하는 방법에 대한 자세한 내용은 [the section called “Windows GPU AMI 실행”](#) 섹션을 참조하세요.
- 관리형 노드 그룹과 함께 Windows를 사용하려면 [the section called “관리형 노드 그룹”](#) 섹션을 참조하세요.
- 자체 관리형 Windows 노드를 시작하려면 [the section called “Windows”](#) 섹션을 참조하세요.
- 버전 정보는 [the section called “버전 정보 받기”](#)를 참조하세요.
- Amazon EKS 최적화 Windows AMI의 최신 ID를 검색하려면 [the section called “최신 ID 받기”](#) 섹션을 참조하세요.
- Amazon EC2 Image Builder를 사용하여 사용자 지정 Amazon EKS 최적화 Windows AMI를 생성하려면 [the section called “사용자 지정 빌드”](#) 섹션을 참조하세요.
- 모범 사례는 EKS 모범 사례 가이드에서 [Amazon EKS에 최적화된 Windows AMI 관리](#)를 참조하세요.

eksctl을 사용한 자체 관리형 Windows Server 2022 노드 생성

자체 관리형 Windows Server 2022 노드를 생성하기 위한 참조로 다음 test-windows-2022.yaml을 사용할 수 있습니다. 모든 ## #을 자신의 값으로 바꾸세요.

Note

자체 관리형 Windows Server 2022 노드를 실행하려면 eksctl 버전 [0.116.0](#) 이상을 사용해야 합니다.

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: windows-2022-cluster
  region: region-code
  version: '1.32'

nodeGroups:
  - name: windows-ng
    instanceType: m5.2xlarge
    amiFamily: WindowsServer2022FullContainer
    volumeSize: 100
    minSize: 2
    maxSize: 3
  - name: linux-ng
    amiFamily: AmazonLinux2
    minSize: 2
    maxSize: 3
```

그러면 다음 명령을 사용하여 노드 그룹을 생성할 수 있습니다.

```
eksctl create cluster -f test-windows-2022.yaml
```

Windows AMI 버전 정보 검색

이 주제에서는 Amazon EKS 최적화 Windows AMI의 버전과 해당하는 [kubelet](#), [containerd](#) 및 [csi-proxy](#)의 버전을 보여줍니다.

AMI ID를 포함하여 각 변형에 대한 Amazon EKS 최적화 AMI 메타데이터를 프로그래밍 방식으로 가져올 수 있습니다. 자세한 내용은 [the section called “최신 ID 받기”](#) 섹션을 참조하세요.

AMI는 Kubernetes 버전과 AMI의 출시 날짜에 따라 다음 형식으로 버전이 지정됩니다.

```
k8s_major_version.k8s_minor_version-release_date
```

Note

Amazon EKS 관리형 노드 그룹에서는 Windows AMI의 2022년 11월 및 이후 릴리스를 지원합니다.

이 특정 설명서 페이지의 모든 소스 파일 변경 사항에 대한 알림을 받으려면 RSS 리더를 사용하여 다음 URL을 구독하세요.

```
https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/nodes/eks-ami-versions-windows.adoc.atom
```

Amazon EKS 최적화 Windows Server 2022 Core AMI

다음 표에는 Amazon EKS 최적화 Windows Server 2022 Core AMI의 현재 및 이전 버전이 나와 있습니다.

Example

Kubernetes 버전 1.33

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.33-2025.05.17	1.33.1	1.7.27	1.2.1	

Kubernetes 버전 1.32

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.32-2025.05.17	1.32.5	1.7.20	1.2.1	
1.32-2025.04.14	1.32.1	1.7.20	1.1.3	
1.32-2025.03.14	1.32.1	1.7.20	1.1.3	
1.32-2025.02.18	1.32.1	1.7.20	1.1.3	
1.32-2025.01.15	1.32.0	1.7.20	1.1.3	

Kubernetes 버전 1.31

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.31-2025.05.17	1.31.9	1.7.20	1.2.1	
1.31-2025.04.14	1.31.5	1.7.20	1.1.3	
1.31-2025.03.14	1.31.5	1.7.20	1.1.3	
1.31-2025.02.15	1.31.5	1.7.20	1.1.3	
1.31-2025.01.15	1.31.4	1.7.20	1.1.3	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.31-2025.01.01	1.31.4	1.7.20	1.1.3	CVE-2024-9042 에 대한 패치 포함.
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes 버전 1.30

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.30-2025.05.17	1.30.13	1.7.20	1.2.1	
1.30-2025.04.14	1.30.9	1.7.20	1.1.3	
1.30-2025.03.14	1.30.9	1.7.20	1.1.3	container d 를 1.7.20로 업그레이드했습니다.
1.30-2025.02.15	1.30.9	1.7.14	1.1.3	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.30-2025.01.15	1.30.8	1.7.14	1.1.3	
1.30-2025.01.01	1.30.8	1.7.14	1.1.3	CVE-2024-9042 에 대한 패치 포함.
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	CVE-2024-5321 에 대한 패치 포함.
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	containerd 를 1.7.14로 업그레이드했습니다.
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes 버전 1.29

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.29-2025 .05.17	1.29.15	1.7.20	1.2.1	
1.29-2025 .04.14	1.29.13	1.7.20	1.1.3	
1.29-2025 .03.14	1.29.13	1.7.20	1.1.3	container d 를 1.7.20로 업그레이드했습 니다.
1.29-2025 .02.15	1.29.13	1.7.14	1.1.3	
1.29-2025 .01.15	1.29.12	1.7.14	1.1.3	
1.29-2025 .01.01	1.29.12	1.7.14	1.1.3	CVE-2024- 9042 에 대한 패치 포함.
1.29-2024 .12.11	1.29.10	1.7.14	1.1.3	
1.29-2024 .11.12	1.29.8	1.7.14	1.1.3	
1.29-2024 .10.08	1.29.8	1.7.14	1.1.3	
1.29-2024 .09.10	1.29.6	1.7.14	1.1.3	
1.29-2024 .08.13	1.29.6	1.7.14	1.1.3	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	CVE-2024-5321 에 대한 패치 포함.
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	container d 를 1.7.11로 업그레이드했습니다. kubelet를 1.29.3로 업그레이드했습니다.
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	container d 를 1.6.28로 업그레이드했습니다. golang 1.22.1을 사용하여 CNI 및 csi-proxy 를 재구축했습니다.
1.29-2024.03.12	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	kubelet 가비지 수집 프로세스에서 일시 중지 이미지가 잘못 삭제되는 버그를 수정했습니다.
1.29-2024.01.11	1.29.0	1.6.18	1.1.2	Windows Server 2022 Core AMI에서 독립 실행형 Windows 업데이트 KB5034439 가 제외되었습니다. KB는 Amazon EKS 최적화 Windows AMI에 포함되지 않은 별도의 WinRE 파티션이 있는 Windows 설치에만 적용됩니다.

Kubernetes 버전 1.28

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.28-2025.05.17	1.28.15	1.7.20	1.2.1	
1.28-2025.04.14	1.28.15	1.7.20	1.1.3	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.28-2025 .03.14	1.28.15	1.7.20	1.1.3	container d 를 1.7.20로 업그레이드했습 니다.
1.28-2025 .02.15	1.28.15	1.7.14	1.1.3	
1.28-2025 .01.15	1.28.15	1.7.14	1.1.3	
1.28-2025 -01-01	1.28.15	1.7.14	1.1.3	CVE-2024- 9042 에 대한 패치 포함.
1.28-2024 .12.11	1.28.15	1.7.14	1.1.3	
1.28-2024 .11.12	1.28.13	1.7.14	1.1.3	
1.28-2024 .10.08	1.28.13	1.7.14	1.1.3	
1.28-2024 .09.10	1.28.11	1.7.14	1.1.3	
1.28-2024 .08.13	1.28.11	1.7.14	1.1.3	
1.28-2024 .07.10	1.28.11	1.7.11	1.1.2	CVE-2024- 5321 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	containerd 를 1.7.11로 업그레이드했습니다.
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	containerd 를 1.6.28로 업그레이드했습니다. kubelet를 1.28.8로 업그레이드했습니다.
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	containerd 를 1.6.25로 업그레이드했습니다. golang 1.22.1을 사용하여 CNI 및 csi-proxy 를 재구축했습니다.
1.28-2024.03.12	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.28-2024 .01.11	1.28.5	1.6.18	1.1.2	Windows Server 2022 Core AMI에서 독립 실행형 Windows 업데이트 KB5034439 가 제외되었습니다. KB는 Amazon EKS 최적화 Windows AMI에 포함되지 않은 별도의 WinRE 파티션이 있는 Windows 설치에만 적용됩니다.
1.28-2023 .12.12	1.28.3	1.6.18	1.1.2	
1.28-2023 .11.14	1.28.3	1.6.18	1.1.2	CVE-2023-5528에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	containerd 를 1.6.18로 업그레이드했습니다. 새 부트스트랩 스크립트 환경 변수 (SERVICE_IPV4_CIDR 및 EXCLUDED_SNAT_CIDRS)가 추가되었습니다.
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	kubelet에서 보안 권고 를 수정했습니다.
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Kubernetes 버전 1.27

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2025.05.17	1.27.16	1.7.20	1.2.1	
1.27-2025.04.14	1.27.16	1.7.20	1.1.3	
1.27-2025.03.14	1.27.16	1.7.20	1.1.3	containerd 를 1.7.20로 업그레이드했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2025 .02.15	1.27.16	1.7.14	1.1.3	
1.27-2025 .01.15	1.27.16	1.7.14	1.1.3	
1.27-2025 .01.01	1.27.16	1.7.14	1.1.3	CVE-2024-9042 에 대한 패치 포함.
1.27-2024 .12.11	1.27.16	1.7.14	1.1.3	
1.27-2024 .11.12	1.27.16	1.7.14	1.1.3	
1.27-2024 .10.08	1.27.16	1.7.14	1.1.3	
1.27-2024 .09.10	1.27.15	1.7.14	1.1.3	
1.27-2024 .08.13	1.27.15	1.7.14	1.1.3	
1.27-2024 .07.10	1.27.15	1.7.11	1.1.2	CVE-2024-5321 에 대한 패치 포함.
1.27-2024 .06.17	1.27.12	1.7.11	1.1.2	containerd 를 1.7.11로 업그레이드했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2024.05.14	1.27.12	1.6.28	1.1.2	containerd 를 1.6.28로 업그레이드했습니다. kubelet를 1.27.12로 업그레이드했습니다.
1.27-2024.04.09	1.27.9	1.6.25	1.1.2	containerd 를 1.6.25로 업그레이드했습니다. golang 1.22.1을 사용하여 CNI 및 csi-proxy 를 재구축했습니다.
1.27-2024.03.12	1.27.9	1.6.18	1.1.2	
1.27-2024.02.13	1.27.9	1.6.18	1.1.2	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2024 .01.11	1.27.9	1.6.18	1.1.2	Windows Server 2022 Core AMI에서 독립 실행형 Windows 업데이트 KB5034439 가 제외되었습니다. KB는 Amazon EKS 최적화 Windows AMI에 포함되지 않은 별도의 WinRE 파티션이 있는 Windows 설치에만 적용됩니다.
1.27-2023 .12.12	1.27.7	1.6.18	1.1.2	
1.27-2023 .11.14	1.27.7	1.6.18	1.1.2	CVE-2023-5528에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2023.10.19	1.27.6	1.6.18	1.1.2	containerd 를 1.6.18로 업그레이드했습니다. 새 부트스트랩 스크립트 환경 변수(SERVICE_IPV4_CIDR 및 EXCLUDED_SNAT_CIDRS) 가 추가되었습니다.
1.27-2023-09.27	1.27.6	1.6.6	1.1.2	kubelet에서 보안 권고 를 수정했습니다.
1.27-2023.09.12	1.27.4	1.6.6	1.1.2	Kubernetes API 서버에서 포트 IP 주소를 가져오는 Kubernetes 커넥터 바이너리를 사용하도록 Amazon VPC CNI 플러그인을 업그레이드했습니다. 풀 요청 #100 을 병합했습니다.
1.27-2023.08.17	1.27.4	1.6.6	1.1.2	CVE-2023-3676 , CVE-2023-3893 및 CVE-2023-3955 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2023.08.08	1.27.3	1.6.6	1.1.1	
1.27-2023.07.11	1.27.3	1.6.6	1.1.1	
1.27-2023.06.20	1.27.1	1.6.6	1.1.1	DNS 접미사 검색 목록이 잘못 채워지는 문제가 해결되었습니다.
1.27-2023.06.14	1.27.1	1.6.6	1.1.1	CNI에서 호스트 포트 매핑에 대한 지원이 추가되었습니다. 풀 요청 #93 을 병합했습니다.
1.27-2023.06.06	1.27.1	1.6.6	1.1.1	노드가 프라이빗 Amazon ECR 이미지를 풀링하지 못하게 하는 container s-roadmap 문제 #2042 를 수정했습니다.
1.27-2023.05.17	1.27.1	1.6.6	1.1.1	

Kubernetes 버전 1.26

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2025 .05.17	1.26.15	1.7.20	1.2.1	
1.26-2025 .04.14	1.26.15	1.7.20	1.1.3	
1.26-2025 .03.14	1.26.15	1.7.20	1.1.3	container d 를 1.7.20로 업그레이드했습 니다.
1.26-2025 .02.15	1.26.15	1.7.14	1.1.3	
1.26-2025 .01.15	1.26.15	1.7.14	1.1.3	
1.26-2024 .12.11	1.26.15	1.7.14	1.1.3	
1.26-2024 .11.12	1.26.15	1.7.14	1.1.3	
1.26-2024 .10.08	1.26.15	1.7.14	1.1.3	
1.26-2024 .09.10	1.26.15	1.7.14	1.1.3	
1.26-2024 .08.13	1.26.15	1.7.14	1.1.3	
1.26-2024 .07.10	1.26.15	1.7.11	1.1.2	CVE-2024- 5321 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2024.06.17	1.26.15	1.7.11	1.1.2	containerd 를 1.7.11로 업그레이드했습니다.
1.26-2024.05.14	1.26.15	1.6.28	1.1.2	containerd 를 1.6.28로 업그레이드했습니다. kubelet를 1.26.15로 업그레이드했습니다.
1.26-2024.04.09	1.26.12	1.6.25	1.1.2	containerd 를 1.6.25로 업그레이드했습니다. golang 1.22.1을 사용하여 CNI 및 csi-proxy 를 재구축했습니다.
1.26-2024.03.12	1.26.12	1.6.18	1.1.2	
1.26-2024.02.13	1.26.12	1.6.18	1.1.2	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2024 .01.11	1.26.12	1.6.18	1.1.2	Windows Server 2022 Core AMI에서 독립 실행형 Windows 업데이트 KB5034439 가 제외되었습니다. KB는 Amazon EKS 최적화 Windows AMI에 포함되지 않은 별도의 WinRE 파티션이 있는 Windows 설치에만 적용됩니다.
1.26-2023 .12.12	1.26.10	1.6.18	1.1.2	
1.26-2023 .11.14	1.26.10	1.6.18	1.1.2	CVE-2023-5528에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2023.10.19	1.26.9	1.6.18	1.1.2	containerd 를 1.6.18로 업그레이드했습니다. kubelet를 1.26.9로 업그레이드했습니다. 새 부트스트랩 스크립트 환경 변수(SERVICE_IPV4_CIDR 및 EXCLUDED_SNAT_CIDRS) 가 추가되었습니다.
1.26-2023.09.12	1.26.7	1.6.6	1.1.2	Kubernetes API 서버에서 포트 IP 주소를 가져오는 Kubernetes 커넥터 바이너리를 사용하도록 Amazon VPC CNI 플러그인을 업그레이드했습니다. 풀 요청 #100 을 병합했습니다.
1.26-2023.08.17	1.26.7	1.6.6	1.1.2	CVE-2023-3676 , CVE-2023-3893 및 CVE-2023-3955 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2023.08.08	1.26.6	1.6.6	1.1.1	
1.26-2023.07.11	1.26.6	1.6.6	1.1.1	
1.26-2023.06.20	1.26.4	1.6.6	1.1.1	DNS 접미사 검색 목록이 잘못 채워지는 문제가 해결되었습니다.
1.26-2023.06.14	1.26.4	1.6.6	1.1.1	Kubernetes를 1.26.4으로 업그레이드했습니다. CNI에서 호스트 포트 매핑에 대한 지원이 추가되었습니다. 풀 요청 #93 을 병합했습니다.
1.26-2023.05.09	1.26.2	1.6.6	1.1.1	노드 재시작 후 포드에서 네트워크 연결 문제 #1126 을 유발하는 버그를 수정했습니다. 새 부트스트랩 스크립트 구성 파라미터 (ExcludedSnatCIDs)를 도입했습니다.
1.26-2023.04.26	1.26.2	1.6.6	1.1.1	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2023 .04.11	1.26.2	1.6.6	1.1.1	서비스 충돌 시 kubelet 및 kube-proxy에 대한 복구 메커니즘을 추가 했습니다.
1.26-2023 .03.24	1.26.2	1.6.6	1.1.1	

Amazon EKS 최적화 Windows Server 2022 Full AMI

다음 표에는 Amazon EKS 최적화 Windows Server 2022 Full AMI의 현재 및 이전 버전이 나와 있습니다.

Example

Kubernetes 버전 1.33

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.33-2025 .05.17	1.33.1	1.7.27	1.2.1	

Kubernetes 버전 1.32

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.32-2025 .05.17	1.32.5	1.7.20	1.2.1	
1.32-2025 .04.14	1.32.1	1.7.20	1.1.3	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.32-2025.03.14	1.32.1	1.7.20	1.1.3	
1.32-2025.02.18	1.32.1	1.7.20	1.1.3	
1.32-2025.01.01	1.32.0	1.7.20	1.1.3	

Kubernetes 버전 1.31

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.31-2025.05.17	1.31.9	1.7.20	1.2.1	
1.31-2025.04.14	1.31.5	1.7.20	1.1.3	
1.31-2025.03.14	1.31.5	1.7.20	1.1.3	
1.31-2025.02.15	1.31.5	1.7.20	1.1.3	
1.31-2025.01.15	1.31.4	1.7.20	1.1.3	
1.31-2025.01.01	1.31.4	1.7.20	1.1.3	CVE-2024-9042 에 대한 패치 포함.
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes 버전 1.30

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.30-2025.05.17	1.30.13	1.7.20	1.2.1	
1.30-2025.04.14	1.30.9	1.7.20	1.1.3	
1.30-2025.03.14	1.30.9	1.7.20	1.1.3	containerd 를 1.7.20로 업그레이드했습니다.
1.30-2025.02.15	1.30.9	1.7.14	1.1.3	
1.30-2025.01.15	1.30.8	1.7.14	1.1.3	
1.30-2025.01.01	1.30.8	1.7.14	1.1.3	CVE-2024-9042 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	CVE-2024-5321 에 대한 패치 포함.
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	containerd 를 1.7.14로 업그레이드했습니다.
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes 버전 1.29

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.29-2025.05.17	1.29.15	1.7.20	1.2.1	
1.29-2025.04.14	1.29.13	1.7.20	1.1.3	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.29-2025 .03.14	1.29.13	1.7.20	1.1.3	container d 를 1.7.20로 업그레이드했습 니다.
1.29-2025 .02.15	1.29.13	1.7.14	1.1.3	
1.29-2025 .01.15	1.29.12	1.7.14	1.1.3	
1.29-2025 .01.01	1.29.12	1.7.14	1.1.3	CVE-2024- 9042 에 대한 패치 포함.
1.29-2024 .12.11	1.29.10	1.7.14	1.1.3	
1.29-2024 .11.12	1.29.8	1.7.14	1.1.3	
1.29-2024 .10.08	1.29.8	1.7.14	1.1.3	
1.29-2024 .09.10	1.29.6	1.7.14	1.1.3	
1.29-2024 .08.13	1.29.6	1.7.14	1.1.3	
1.29-2024 .07.10	1.29.6	1.7.11	1.1.2	CVE-2024- 5321 에 대한 패치 포함.
1.29-2024 .06.17	1.29.3	1.7.11	1.1.2	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	containerd 를 1.7.11로 업그레이드했습니다. kubelet를 1.29.3로 업그레이드했습니다.
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	containerd 를 1.6.28로 업그레이드했습니다. golang 1.22.1을 사용하여 CNI 및 csi-proxy 를 재구축했습니다.
1.29-2024.03.12	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	kubelet 가비지 수집 프로세스에서 일시 중지 이미지가 잘못 삭제되는 버그를 수정했습니다.
1.29-2024.01.09	1.29.0	1.6.18	1.1.2	

Kubernetes 버전 1.28

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.28-2025.05.17	1.28.15	1.7.20	1.2.1	
1.28-2025.04.14	1.28.15	1.7.20	1.1.3	
1.28-2025.03.14	1.28.15	1.7.20	1.1.3	containerd 를 1.7.20로 업그레이드했습니다.
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	
1.28-2025.01.15	1.28.15	1.7.14	1.1.3	
1.28-2025.01.01	1.28.15	1.7.14	1.1.3	CVE-2024-9042 에 대한 패치 포함.
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	CVE-2024-5321 에 대한 패치 포함.
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	containerd 를 1.7.11로 업그레이드했습니다.
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	containerd 를 1.6.28로 업그레이드했습니다. kubelet를 1.28.8로 업그레이드했습니다.
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	containerd 를 1.6.25로 업그레이드했습니다. golang 1.22.1을 사용하여 CNI 및 csi-proxy 를 재구축했습니다.
1.28-2024.03.12	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.09	1.28.5	1.6.18	1.1.2	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	CVE-2023-5528 에 대한 패치 포함.
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	containerd 를 1.6.18로 업그레이드했습니다. 새 부트스트랩 스크립트 환경 변수 (SERVICE_IPV4_CIDR 및 EXCLUDED_SNAT_CIDRS)가 추가되었습니다.
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	kubelet에서 보안 권고 를 수정했습니다.
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Kubernetes 버전 1.27

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2025.05.17	1.27.16	1.7.20	1.2.1	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2025 .04.14	1.27.16	1.7.20	1.1.3	
1.27-2025 .03.14	1.27.16	1.7.20	1.1.3	container d 를 1.7.20로 업그레이드했습 니다.
1.27-2025 .02.15	1.27.16	1.7.14	1.1.3	
1.27-2025 .01.15	1.27.16	1.7.14	1.1.3	
1.27-2025 .01.01	1.27.16	1.7.14	1.1.3	CVE-2024- 9042 에 대한 패치 포함.
1.27-2024 .12.11	1.27.16	1.7.14	1.1.3	
1.27-2024 .11.12	1.27.16	1.7.14	1.1.3	
1.27-2024 .10.08	1.27.16	1.7.14	1.1.3	
1.27-2024 .09.10	1.27.15	1.7.14	1.1.3	
1.27-2024 .08.13	1.27.15	1.7.14	1.1.3	
1.27-2024 .07.10	1.27.15	1.7.11	1.1.2	CVE-2024- 5321 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2024.06.17	1.27.12	1.7.11	1.1.2	containerd 를 1.7.11로 업그레이드했습니다.
1.27-2024.05.14	1.27.12	1.6.28	1.1.2	containerd 를 1.6.28로 업그레이드했습니다. kubelet를 1.27.12로 업그레이드했습니다.
1.27-2024.04.09	1.27.9	1.6.25	1.1.2	containerd 를 1.6.25로 업그레이드했습니다. golang 1.22.1을 사용하여 CNI 및 csi-proxy 를 재구축했습니다.
1.27-2024.03.12	1.27.9	1.6.18	1.1.2	
1.27-2024.02.13	1.27.9	1.6.18	1.1.2	
1.27-2024.01.09	1.27.9	1.6.18	1.1.2	
1.27-2023.12.12	1.27.7	1.6.18	1.1.2	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2023.11.14	1.27.7	1.6.18	1.1.2	CVE-2023-5528 에 대한 패치 포함.
1.27-2023.10.19	1.27.6	1.6.18	1.1.2	containerd 를 1.6.18로 업그레이드했습니다. 새 부트스트랩 스크립트 환경 변수 (SERVICE_IPV4_CIDR 및 EXCLUDED_SNAT_CIDRS)가 추가되었습니다.
1.27-2023-09.27	1.27.6	1.6.6	1.1.2	kubelet에서 보안 권고 를 수정했습니다.
1.27-2023.09.12	1.27.4	1.6.6	1.1.2	Kubernetes API 서버에서 포트 IP 주소를 가져오는 Kubernetes 커넥터 바이너리를 사용하도록 Amazon VPC CNI 플러그인을 업그레이드했습니다. 풀 요청 #100 을 병합했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2023.08.17	1.27.4	1.6.6	1.1.2	CVE-2023-3676 , CVE-2023-3893 및 CVE-2023-3955 에 대한 패치 포함.
1.27-2023.08.08	1.27.3	1.6.6	1.1.1	
1.27-2023.07.11	1.27.3	1.6.6	1.1.1	
1.27-2023.06.20	1.27.1	1.6.6	1.1.1	DNS 접미사 검색 목록이 잘못 채워지는 문제가 해결되었습니다.
1.27-2023.06.14	1.27.1	1.6.6	1.1.1	CNI에서 호스트 포트 매핑에 대한 지원이 추가되었습니다. 폴 요청 #93 을 병합했습니다.
1.27-2023.06.06	1.27.1	1.6.6	1.1.1	노드가 프라이빗 Amazon ECR 이미지를 풀링하지 못하게 하는 container s-roadmap 문제 #2042 를 수정했습니다.
1.27-2023.05.18	1.27.1	1.6.6	1.1.1	

Kubernetes 버전 1.26

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2025 .05.17	1.26.15	1.7.20	1.2.1	
1.26-2025 .04.14	1.26.15	1.7.20	1.1.3	
1.26-2025 .03.14	1.26.15	1.7.20	1.1.3	container d 를 1.7.20로 업그레이드했습 니다.
1.26-2025 .02.15	1.26.15	1.7.14	1.1.3	
1.26-2025 .01.15	1.26.15	1.7.14	1.1.3	
1.26-2024 .12.11	1.26.15	1.7.14	1.1.3	
1.26-2024 .11.12	1.26.15	1.7.14	1.1.3	
1.26-2024 .10.08	1.26.15	1.7.14	1.1.3	
1.26-2024 .09.10	1.26.15	1.7.14	1.1.3	
1.26-2024 .08.13	1.26.15	1.7.14	1.1.3	
1.26-2024 .07.10	1.26.15	1.7.11	1.1.2	CVE-2024- 5321 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2024.06.17	1.26.15	1.7.11	1.1.2	containerd 를 1.7.11로 업그레이드했습니다.
1.26-2024.05.14	1.26.15	1.6.28	1.1.2	containerd 를 1.6.28로 업그레이드했습니다. kubelet를 1.26.15로 업그레이드했습니다.
1.26-2024.04.09	1.26.12	1.6.25	1.1.2	containerd 를 1.6.25로 업그레이드했습니다. golang 1.22.1을 사용하여 CNI 및 csi-proxy 를 재구축했습니다.
1.26-2024.03.12	1.26.12	1.6.18	1.1.2	
1.26-2024.02.13	1.26.12	1.6.18	1.1.2	
1.26-2024.01.09	1.26.12	1.6.18	1.1.2	
1.26-2023.12.12	1.26.10	1.6.18	1.1.2	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2023.11.14	1.26.10	1.6.18	1.1.2	CVE-2023-5528 에 대한 패치 포함.
1.26-2023.10.19	1.26.9	1.6.18	1.1.2	containerd 를 1.6.18로 업그레이드했습니다. kubelet를 1.26.9로 업그레이드했습니다. 새 부트스트랩 스크립트 환경 변수(SERVICE_IP, PV4_CIDR 및 EXCLUDED_SNAT_CIDRS) 가 추가되었습니다.
1.26-2023.09.12	1.26.7	1.6.6	1.1.2	Kubernetes API 서버에서 포트 IP 주소를 가져오는 Kubernetes 커넥터 바이너리를 사용하도록 Amazon VPC CNI 플러그인을 업그레이드했습니다. 풀 요청 #100 을 병합했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2023.08.17	1.26.7	1.6.6	1.1.2	CVE-2023-3676 , CVE-2023-3893 및 CVE-2023-3955 에 대한 패치 포함.
1.26-2023.08.08	1.26.6	1.6.6	1.1.1	
1.26-2023.07.11	1.26.6	1.6.6	1.1.1	
1.26-2023.06.20	1.26.4	1.6.6	1.1.1	DNS 접미사 검색 목록이 잘못 채워지는 문제가 해결되었습니다.
1.26-2023.06.14	1.26.4	1.6.6	1.1.1	Kubernetes를 1.26.4으로 업그레이드했습니다. CNI에서 호스트 포트 매핑에 대한 지원이 추가되었습니다. 풀 요청 #93 을 병합했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2023.05.09	1.26.2	1.6.6	1.1.1	노드 재시작 후 포드에서 네트워크 연결 문제 #1126 을 유발하는 버그를 수정했습니다. 새 부트스트랩 스크립트 구성 파라미터(ExcludedS natCIDRs)를 도입했습니다.
1.26-2023.04.26	1.26.2	1.6.6	1.1.1	
1.26-2023.04.11	1.26.2	1.6.6	1.1.1	서비스 충돌 시 kubelet 및 kube-proxy 에 대한 복구 메커니즘을 추가했습니다.
1.26-2023.03.24	1.26.2	1.6.6	1.1.1	

Amazon EKS 최적화 Windows Server 2019 Core AMI

다음 표에는 Amazon EKS 최적화 Windows Server 2019 Core AMI의 현재 및 이전 버전이 나와 있습니다.

Example

Kubernetes 버전 1.33

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.33-2025.05.17	1.33.1	1.7.27	1.2.1	

Kubernetes 버전 1.32

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.32-2025.05.17	1.32.5	1.7.20	1.2.1	
1.32-2025.04.14	1.32.1	1.7.20	1.1.3	
1.32-2025.03.14	1.32.1	1.7.20	1.1.3	
1.32-2025.02.18	1.32.1	1.7.20	1.1.3	
1.32-2025.01.15	1.32.4	1.7.20	1.1.3	

Kubernetes 버전 1.31

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.31-2025.05.17	1.31.9	1.7.20	1.2.1	
1.31-2025.04.14	1.31.5	1.7.20	1.1.3	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.31-2025.03.14	1.31.5	1.7.20	1.1.3	
1.31-2025.02.15	1.31.5	1.7.20	1.1.3	
1.31-2025.01.15	1.31.4	1.7.20	1.1.3	
1.31-2025.01.01	1.31.4	1.7.20	1.1.3	CVE-2024-9042 에 대한 패치 포함.
1.31-2024.12.13	1.31.3	1.7.20	1.1.3	
1.31-2024.11.12	1.31.1	1.7.20	1.1.3	
1.31-2024.10.08	1.31.1	1.7.20	1.1.3	
1.31-2024.10.01	1.31.1	1.7.20	1.1.3	
1.31-2024.09.10	1.31.0	1.7.20	1.1.3	

Kubernetes 버전 1.30

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.30-2025.05.17	1.30.13	1.7.20	1.2.1	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.30-2025.04.14	1.30.9	1.7.20	1.1.3	
1.30-2025.03.14	1.30.9	1.7.20	1.1.3	containerd 를 1.7.20로 업그레이드했습니다.
1.30-2025-02-15	1.30.9	1.7.14	1.1.3	
1.30-2025.01.15	1.30.8	1.7.14	1.1.3	
1.30-2025.01.01	1.30.8	1.7.14	1.1.3	CVE-2024-9042 에 대한 패치 포함.
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	CVE-2024-5321 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	containerd 를 1.7.14로 업그레이드했습니다.
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes 버전 1.29

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.29-2025.05.17	1.29.15	1.7.20	1.2.1	
1.29-2025.04.14	1.29.13	1.7.20	1.1.3	
1.29-2025.03.14	1.29.13	1.7.20	1.1.3	containerd 를 1.7.20로 업그레이드했습니다.
1.29-2025.02.15	1.29.13	1.7.14	1.1.3	
1.29-2025.01.15	1.29.12	1.7.14	1.1.3	
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	CVE-2024-9042 에 대한 패치 포함.
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	CVE-2024-5321 에 대한 패치 포함.
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	container d 를 1.7.11로 업그레이드했습니다. kubelet를 1.29.3로 업그레이드했습니다.
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	container d 를 1.6.28로 업그레이드했습니다. golang 1.22.1을 사용하여 CNI 및 csi-proxy 를 재구축했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.29-2024.03.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	kubelet 가비지 수집 프로세스에서 일시 중지 이미지가 잘못 삭제되는 버그를 수정했습니다.
1.29-2024.01.09	1.29.0	1.6.18	1.1.2	

Kubernetes 버전 1.28

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.28-2025.05.17	1.28.15	1.7.20	1.2.1	
1.28-2025.04.14	1.28.15	1.7.20	1.1.3	
1.28-2025.03.14	1.28.15	1.7.20	1.1.3	containerd 를 1.7.20로 업그레이드했습니다.
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.28-2025-01-15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-01	1.28.15	1.7.14	1.1.3	CVE-2024-9042 에 대한 패치 포함.
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	CVE-2024-5321 에 대한 패치 포함.
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	containerd 를 1.7.11로 업그레이드했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	containerd 를 1.6.28로 업그레이드했습니다. kubelet를 1.28.8로 업그레이드했습니다.
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	containerd 를 1.6.25로 업그레이드했습니다. golang 1.22.1을 사용하여 CNI 및 csi-proxy 를 재구축했습니다.
1.28-2024.03.13	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.09	1.28.5	1.6.18	1.1.2	
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	CVE-2023-5528 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	containerd 를 1.6.18로 업그레이드했습니다. 새 부트스트랩 스크립트 환경 변수 (SERVICE_IPV4_CIDR 및 EXCLUDED_SNAT_CIDRS)가 추가되었습니다.
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	kubelet에서 보안 권고 를 수정했습니다.
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Kubernetes 버전 1.27

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2025.05.17	1.27.16	1.7.20	1.2.1	
1.27-2025.04.14	1.27.16	1.7.20	1.1.3	
1.27-2025.03.14	1.27.16	1.7.20	1.1.3	containerd 를 1.7.20로 업그레이드했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2025 .02.15	1.27.16	1.7.14	1.1.3	
1.27-2025 .01.15	1.27.16	1.7.14	1.1.3	
1.27-2025 .01.01	1.27.16	1.7.14	1.1.3	CVE-2024-9042 에 대한 패치 포함.
1.27-2024 .12.11	1.27.16	1.7.14	1.1.3	
1.27-2024 .11.12	1.27.16	1.7.14	1.1.3	
1.27-2024 .10.08	1.27.16	1.7.14	1.1.3	
1.27-2024 .09.10	1.27.15	1.7.14	1.1.3	
1.27-2024 .08.13	1.27.15	1.7.14	1.1.3	
1.27-2024 .07.10	1.27.15	1.7.11	1.1.2	CVE-2024-5321 에 대한 패치 포함.
1.27-2024 .06.17	1.27.12	1.7.11	1.1.2	containerd 를 1.7.11로 업그레이드했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2024.05.14	1.27.12	1.6.28	1.1.2	containerd 를 1.6.28로 업그레이드했습니다. kubelet를 1.27.12로 업그레이드했습니다.
1.27-2024.04.09	1.27.9	1.6.25	1.1.2	containerd 를 1.6.25로 업그레이드했습니다. golang 1.22.1을 사용하여 CNI 및 csi-proxy 를 재구축했습니다.
1.27-2024.03.13	1.27.9	1.6.18	1.1.2	
1.27-2024.02.13	1.27.9	1.6.18	1.1.2	
1.27-2024.01.09	1.27.9	1.6.18	1.1.2	
1.27-2023.12.12	1.27.7	1.6.18	1.1.2	
1.27-2023.11.14	1.27.7	1.6.18	1.1.2	CVE-2023-5528 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2023.10.19	1.27.6	1.6.18	1.1.2	containerd 를 1.6.18로 업그레이드했습니다. 새 부트스트랩 스크립트 환경 변수(SERVICE_IPV4_CIDR 및 EXCLUDED_SNAT_CIDRS) 가 추가되었습니다.
1.27-2023-09.27	1.27.6	1.6.6	1.1.2	kubelet에서 보안 권고 를 수정했습니다.
1.27-2023.09.12	1.27.4	1.6.6	1.1.2	Kubernetes API 서버에서 포트 IP 주소를 가져오는 Kubernetes 커넥터 바이너리를 사용하도록 Amazon VPC CNI 플러그인을 업그레이드했습니다. 풀 요청 #100 을 병합했습니다.
1.27-2023.08.17	1.27.4	1.6.6	1.1.2	CVE-2023-3676 , CVE-2023-3893 및 CVE-2023-3955 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2023 .08.08	1.27.3	1.6.6	1.1.1	
1.27-2023 .07.11	1.27.3	1.6.6	1.1.1	
1.27-2023 .06.20	1.27.1	1.6.6	1.1.1	DNS 접미사 검색 목록이 잘못 채워지는 문제가 해결되었습니다.
1.27-2023 .06.14	1.27.1	1.6.6	1.1.1	CNI에서 호스트 포트 매핑에 대한 지원이 추가되었습니다. 풀 요청 #93 을 병합했습니다.
1.27-2023 .06.06	1.27.1	1.6.6	1.1.1	노드가 프라이빗 Amazon ECR 이미지를 풀링하지 못하게 하는 container s-roadmap 문제 #2042 를 수정했습니다.
1.27-2023 .05.18	1.27.1	1.6.6	1.1.1	

Kubernetes 버전 1.26

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2025 .05.17	1.26.15	1.7.20	1.2.1	
1.26-2025 .04.14	1.26.15	1.7.20	1.1.3	
1.26-2025 .03.14	1.26.15	1.7.20	1.1.3	container d 를 1.7.20로 업그레이드했습 니다.
1.26-2025 .02.15	1.26.15	1.7.14	1.1.3	
1.26-2025 .01.15	1.26.15	1.7.14	1.1.3	
1.26-2024 .12.11	1.26.15	1.7.14	1.1.3	
1.26-2024 .11.12	1.26.15	1.7.14	1.1.3	
1.26-2024 .10.09	1.26.15	1.7.14	1.1.3	
1.26-2024 .09.10	1.26.15	1.7.14	1.1.3	
1.26-2024 .08.13	1.26.15	1.7.14	1.1.3	
1.26-2024 .07.10	1.26.15	1.7.11	1.1.2	CVE-2024- 5321 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2024.06.17	1.26.15	1.7.11	1.1.2	containerd 를 1.7.11로 업그레이드했습니다.
1.26-2024.05.14	1.26.15	1.6.28	1.1.2	containerd 를 1.6.28로 업그레이드했습니다. kubelet를 1.26.15로 업그레이드했습니다.
1.26-2024.04.09	1.26.12	1.6.25	1.1.2	containerd 를 1.6.25로 업그레이드했습니다. golang 1.22.1을 사용하여 CNI 및 csi-proxy 를 재구축했습니다.
1.26-2024.03.13	1.26.12	1.6.18	1.1.2	
1.26-2024.02.13	1.26.12	1.6.18	1.1.2	
1.26-2024.01.09	1.26.12	1.6.18	1.1.2	
1.26-2023.12.12	1.26.10	1.6.18	1.1.2	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2023.11.14	1.26.10	1.6.18	1.1.2	CVE-2023-5528 에 대한 패치 포함.
1.26-2023.10.19	1.26.9	1.6.18	1.1.2	containerd 를 1.6.18로 업그레이드했습니다. kubelet를 1.26.9로 업그레이드했습니다. 새 부트스트랩 스크립트 환경 변수(SERVICE_IP, PV4_CIDR 및 EXCLUDED_SNAT_CIDRS) 가 추가되었습니다.
1.26-2023.09.12	1.26.7	1.6.6	1.1.2	Kubernetes API 서버에서 포트 IP 주소를 가져오는 Kubernetes 커넥터 바이너리를 사용하도록 Amazon VPC CNI 플러그인을 업그레이드했습니다. 풀 요청 #100 을 병합했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2023.08.17	1.26.7	1.6.6	1.1.2	CVE-2023-3676 , CVE-2023-3893 및 CVE-2023-3955 에 대한 패치 포함.
1.26-2023.08.08	1.26.6	1.6.6	1.1.1	
1.26-2023.07.11	1.26.6	1.6.6	1.1.1	
1.26-2023.06.20	1.26.4	1.6.6	1.1.1	DNS 접미사 검색 목록이 잘못 채워지는 문제가 해결되었습니다.
1.26-2023.06.14	1.26.4	1.6.6	1.1.1	Kubernetes를 1.26.4으로 업그레이드했습니다. CNI에서 호스트 포트 매핑에 대한 지원이 추가되었습니다. 풀 요청 #93 을 병합했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2023 .05.09	1.26.2	1.6.6	1.1.1	노드 재시작 후 포드에서 네트 워크 연결 문제 #1126 을 유발하 는 버그를 수정 했습니다. 새 부 트스트랩 스크립 트 구성 파라미 터(ExcludedS natCIDRs) 를 도입했습니다.
1.26-2023 .04.26	1.26.2	1.6.6	1.1.1	
1.26-2023 .04.11	1.26.2	1.6.6	1.1.1	서비스 충돌 시 kubelet 및 kube-prox y 에 대한 복구 메커니즘을 추가 했습니다.
1.26-2023 .03.24	1.26.2	1.6.6	1.1.1	

Amazon EKS 최적화 Windows Server 2019 Full AMI

다음 표에는 Amazon EKS 최적화 Windows Server 2019 Full AMI의 현재 및 이전 버전이 나와 있습니다.

Example

Kubernetes 버전 1.33

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.33-2025 .05.17	1.33.1	1.7.27	1.2.1	

Kubernetes 버전 1.32

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.32-2025 .05.17	1.32.5	1.7.20	1.2.1	
1.32-2025 .04.14	1.32.1	1.7.20	1.1.3	
1.32-2025 .03.14	1.32.1	1.7.20	1.1.3	
1.32-2025 .02.18	1.32.1	1.7.20	1.1.3	
1.32-2025 .01.15	1.32.0	1.7.20	1.1.3	

Kubernetes 버전 1.31

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.31-2025 .05.17	1.31.9	1.7.20	1.2.1	
1.31-2025 .04.14	1.31.5	1.7.20	1.1.3	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.31-2025 .03.14	1.31.5	1.7.20	1.1.3	
1.31-2025 .02.15	1.31.5	1.7.20	1.1.3	
1.31-2025 .01.15	1.31.4	1.7.20	1.1.3	
1.31-2025 .01.01	1.31.4	1.7.20	1.1.3	CVE-2024-9042 에 대한 패치 포함.
1.31-2024 .12.13	1.31.3	1.7.20	1.1.3	
1.31-2024 .11.12	1.31.1	1.7.20	1.1.3	
1.31-2024 .10.08	1.31.1	1.7.20	1.1.3	
1.31-2024 .10.01	1.31.1	1.7.20	1.1.3	
1.31-2024 .09.10	1.31.0	1.7.20	1.1.3	

Kubernetes 버전 1.30

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.30-2025 .05.17	1.30.13	1.7.20	1.2.1	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.30-2025.04.14	1.30.9	1.7.20	1.1.3	
1.30-2025.03.14	1.30.9	1.7.20	1.1.3	containerd 를 1.7.20로 업그레이드했습니다.
1.30-2025.02.15	1.30.9	1.7.14	1.1.3	
1.30-2025.01.15	1.30.8	1.7.14	1.1.3	
1.30-2025.01.01	1.30.8	1.7.14	1.1.3	CVE-2024-9042 에 대한 패치 포함.
1.30-2024.12.11	1.30.7	1.7.14	1.1.3	
1.30-2024.11.12	1.30.4	1.7.14	1.1.3	
1.30-2024.10.08	1.30.4	1.7.14	1.1.3	
1.30-2024.09.10	1.30.2	1.7.14	1.1.3	
1.30-2024.08.13	1.30.2	1.7.14	1.1.3	
1.30-2024.07.10	1.30.2	1.7.14	1.1.2	CVE-2024-5321 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.30-2024.06.17	1.30.0	1.7.14	1.1.2	containerd 를 1.7.14로 업그레이드했습니다.
1.30-2024.05.15	1.30.0	1.6.28	1.1.2	

Kubernetes 버전 1.29

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.29-2025.05.17	1.29.15	1.7.20	1.2.1	
1.29-2025.04.14	1.29.13	1.7.20	1.1.3	
1.29-2025.03.14	1.29.13	1.7.20	1.1.3	containerd 를 1.7.20로 업그레이드했습니다.
1.29-2025.02.15	1.29.13	1.7.14	1.1.3	
1.29-2025.01.15	1.29.12	1.7.14	1.1.3	
1.29-2025.01.01	1.29.12	1.7.14	1.1.3	CVE-2024-9042 에 대한 패치 포함.
1.29-2024.12.11	1.29.10	1.7.14	1.1.3	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.29-2024.11.12	1.29.8	1.7.14	1.1.3	
1.29-2024.10.08	1.29.8	1.7.14	1.1.3	
1.29-2024.09.10	1.29.6	1.7.14	1.1.3	
1.29-2024.08.13	1.29.6	1.7.14	1.1.3	
1.29-2024.07.10	1.29.6	1.7.11	1.1.2	CVE-2024-5321 에 대한 패치 포함.
1.29-2024.06.17	1.29.3	1.7.11	1.1.2	
1.29-2024.05.15	1.29.3	1.7.11	1.1.2	container d 를 1.7.11로 업그레이드했습니다. kubelet를 1.29.3로 업그레이드했습니다.
1.29-2024.04.09	1.29.0	1.6.28	1.1.2	container d 를 1.6.28로 업그레이드했습니다. golang 1.22.1을 사용하여 CNI 및 csi-proxy 를 재구축했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.29-2024.03.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.13	1.29.0	1.6.25	1.1.2	
1.29-2024.02.06	1.29.0	1.6.25	1.1.2	kubelet 가비지 수집 프로세스에서 일시 중지 이미지가 잘못 삭제되는 버그를 수정했습니다.
1.29-2024.01.09	1.29.0	1.6.18	1.1.2	

Kubernetes 버전 1.28

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.28-2025.05.17	1.28.15	1.7.20	1.2.1	
1.28-2025.04.14	1.28.15	1.7.20	1.1.3	
1.28-2025.03.14	1.28.15	1.7.20	1.1.3	containerd 를 1.7.20로 업그레이드했습니다.
1.28-2025.02.15	1.28.15	1.7.14	1.1.3	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.28-2025-01-15	1.28.15	1.7.14	1.1.3	
1.28-2025-01-01	1.28.15	1.7.14	1.1.3	CVE-2024-9042 에 대한 패치 포함.
1.28-2024.12.11	1.28.15	1.7.14	1.1.3	
1.28-2024.11.12	1.28.13	1.7.14	1.1.3	
1.28-2024.10.08	1.28.13	1.7.14	1.1.3	
1.28-2024.09.10	1.28.11	1.7.14	1.1.3	
1.28-2024.08.13	1.28.11	1.7.14	1.1.3	
1.28-2024.07.10	1.28.11	1.7.11	1.1.2	CVE-2024-5321 에 대한 패치 포함.
1.28-2024.06.17	1.28.8	1.7.11	1.1.2	containerd 를 1.7.11로 업그레이드했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.28-2024.05.14	1.28.8	1.6.28	1.1.2	containerd 를 1.6.28로 업그레이드했습니다. kubelet를 1.28.8로 업그레이드했습니다.
1.28-2024.04.09	1.28.5	1.6.25	1.1.2	containerd 를 1.6.25로 업그레이드했습니다. golang 1.22.1을 사용하여 CNI 및 csi-proxy 를 재구축했습니다.
1.28-2024.03.13	1.28.5	1.6.18	1.1.2	
1.28-2024.02.13	1.28.5	1.6.18	1.1.2	
1.28-2024.01.09	1.28.5	1.6.18	1.1.2	
1.28-2023.12.12	1.28.3	1.6.18	1.1.2	
1.28-2023.11.14	1.28.3	1.6.18	1.1.2	CVE-2023-5528 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.28-2023.10.19	1.28.2	1.6.18	1.1.2	containerd 를 1.6.18로 업그레이드했습니다. 새 부트스트랩 스크립트 환경 변수 (SERVICE_IPV4_CIDR 및 EXCLUDED_SNAT_CIDRS)가 추가되었습니다.
1.28-2023-09.27	1.28.2	1.6.6	1.1.2	kubelet에서 보안 권고 를 수정했습니다.
1.28-2023.09.12	1.28.1	1.6.6	1.1.2	

Kubernetes 버전 1.27

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2025.05.17	1.27.16	1.7.20	1.2.1	
1.27-2025.04.14	1.27.16	1.7.20	1.1.3	
1.27-2025.03.14	1.27.16	1.7.20	1.1.3	containerd 를 1.7.20로 업그레이드했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2025 .02.15	1.27.16	1.7.14	1.1.3	
1.27-2025 .01.15	1.27.16	1.7.14	1.1.3	
1.27-2025 .01.01	1.27.16	1.7.14	1.1.3	CVE-2024-9042 에 대한 패치 포함.
1.27-2024 .12.11	1.27.16	1.7.14	1.1.3	
1.27-2024 .11.12	1.27.16	1.7.14	1.1.3	
1.27-2024 .10.08	1.27.16	1.7.14	1.1.3	
1.27-2024 .09.10	1.27.15	1.7.14	1.1.3	
1.27-2024 .08.13	1.27.15	1.7.14	1.1.3	
1.27-2024 .07.10	1.27.15	1.7.11	1.1.2	CVE-2024-5321 에 대한 패치 포함.
1.27-2024 .06.17	1.27.12	1.7.11	1.1.2	containerd 를 1.7.11로 업그레이드했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2024.05.14	1.27.12	1.6.28	1.1.2	containerd 를 1.6.28로 업그레이드했습니다. kubelet를 1.27.12로 업그레이드했습니다.
1.27-2024.04.09	1.27.9	1.6.25	1.1.2	containerd 를 1.6.25로 업그레이드했습니다. golang 1.22.1을 사용하여 CNI 및 csi-proxy 를 재구축했습니다.
1.27-2024.03.13	1.27.9	1.6.18	1.1.2	
1.27-2024.02.13	1.27.9	1.6.18	1.1.2	
1.27-2024.01.09	1.27.9	1.6.18	1.1.2	
1.27-2023.12.12	1.27.7	1.6.18	1.1.2	
1.27-2023.11.14	1.27.7	1.6.18	1.1.2	CVE-2023-5528 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2023.10.19	1.27.6	1.6.18	1.1.2	containerd 를 1.6.18로 업그레이드했습니다. 새 부트스트랩 스크립트 환경 변수(SERVICE_IPV4_CIDR 및 EXCLUDED_SNAT_CIDRS) 가 추가되었습니다.
1.27-2023-09.27	1.27.6	1.6.6	1.1.2	kubelet에서 보안 권고 를 수정했습니다.
1.27-2023.09.12	1.27.4	1.6.6	1.1.2	Kubernetes API 서버에서 포트 IP 주소를 가져오는 Kubernetes 커넥터 바이너리를 사용하도록 Amazon VPC CNI 플러그인을 업그레이드했습니다. 풀 요청 #100 을 병합했습니다.
1.27-2023.08.17	1.27.4	1.6.6	1.1.2	CVE-2023-3676 , CVE-2023-3893 및 CVE-2023-3955 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.27-2023.08.08	1.27.3	1.6.6	1.1.1	
1.27-2023.07.11	1.27.3	1.6.6	1.1.1	
1.27-2023.06.20	1.27.1	1.6.6	1.1.1	DNS 접미사 검색 목록이 잘못 채워지는 문제가 해결되었습니다.
1.27-2023.06.14	1.27.1	1.6.6	1.1.1	CNI에서 호스트 포트 매핑에 대한 지원이 추가되었습니다. 풀 요청 #93 을 병합했습니다.
1.27-2023.06.06	1.27.1	1.6.6	1.1.1	노드가 프라이빗 Amazon ECR 이미지를 풀링하지 못하게 하는 container s-roadmap 문제 #2042 를 수정했습니다.
1.27-2023.05.17	1.27.1	1.6.6	1.1.1	

Kubernetes 버전 1.26

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2025 .05.17	1.26.15	1.7.20	1.2.1	
1.26-2025 .04.14	1.26.15	1.7.20	1.1.3	
1.26-2025 .03.14	1.26.15	1.7.20	1.1.3	container d 를 1.7.20로 업그레이드했습 니다.
1.26-2025 .02.15	1.26.15	1.7.14	1.1.3	
1.26-2025 .01.15	1.26.15	1.7.14	1.1.3	
1.26-2024 .12.11	1.26.15	1.7.14	1.1.3	
1.26-2024 .11.12	1.26.15	1.7.14	1.1.3	
1.26-2024 .10.08	1.26.15	1.7.14	1.1.3	
1.26-2024 .09.10	1.26.15	1.7.14	1.1.3	
1.26-2024 .08.13	1.26.15	1.7.14	1.1.3	
1.26-2024 .07.10	1.26.15	1.7.11	1.1.2	CVE-2024- 5321 에 대한 패치 포함.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2024.06.17	1.26.15	1.7.11	1.1.2	containerd 를 1.7.11로 업그레이드했습니다.
1.26-2024.05.14	1.26.15	1.6.28	1.1.2	containerd 를 1.6.28로 업그레이드했습니다. kubelet를 1.26.15로 업그레이드했습니다.
1.26-2024.04.09	1.26.12	1.6.25	1.1.2	containerd 를 1.6.25로 업그레이드했습니다. golang 1.22.1을 사용하여 CNI 및 csi-proxy 를 재구축했습니다.
1.26-2024.03.13	1.26.12	1.6.18	1.1.2	
1.26-2024.02.13	1.26.12	1.6.18	1.1.2	
1.26-2024.01.09	1.26.12	1.6.18	1.1.2	
1.26-2023.12.12	1.26.10	1.6.18	1.1.2	

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2023.11.14	1.26.10	1.6.18	1.1.2	CVE-2023-5528 에 대한 패치 포함.
1.26-2023.10.19	1.26.9	1.6.18	1.1.2	containerd 를 1.6.18로 업그레이드했습니다. kubelet를 1.26.9로 업그레이드했습니다. 새 부트스트랩 스크립트 환경 변수(SERVICE_IP, PV4_CIDR 및 EXCLUDED_SNAT_CIDRS) 가 추가되었습니다.
1.26-2023.09.12	1.26.7	1.6.6	1.1.2	Kubernetes API 서버에서 포트 IP 주소를 가져오는 Kubernetes 커넥터 바이너리를 사용하도록 Amazon VPC CNI 플러그인을 업그레이드했습니다. 풀 요청 #100 을 병합했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2023.08.17	1.26.7	1.6.6	1.1.2	CVE-2023-3676 , CVE-2023-3893 및 CVE-2023-3955 에 대한 패치 포함.
1.26-2023.08.08	1.26.6	1.6.6	1.1.1	
1.26-2023.07.11	1.26.6	1.6.6	1.1.1	
1.26-2023.06.20	1.26.4	1.6.6	1.1.1	DNS 접미사 검색 목록이 잘못 채워지는 문제가 해결되었습니다.
1.26-2023.06.14	1.26.4	1.6.6	1.1.1	Kubernetes를 1.26.4으로 업그레이드했습니다. CNI에서 호스트 포트 매핑에 대한 지원이 추가되었습니다. 풀 요청 #93 을 병합했습니다.

AMI 버전	kubelet 버전	컨테이너 버전	csi-프록시 버전	릴리스 정보
1.26-2023 .05.09	1.26.2	1.6.6	1.1.1	노드 재시작 후 포드에서 네트 워크 연결 문제 #1126 을 유발하 는 버그를 수정 했습니다. 새 부 트스트랩 스크립 트 구성 파라미 터(ExcludedS natCIDRs) 를 도입했습니다.
1.26-2023 .04.26	1.26.2	1.6.6	1.1.1	
1.26-2023 .04.11	1.26.2	1.6.6	1.1.1	서비스 충돌 시 kubelet 및 kube-prox y 에 대한 복구 메커니즘을 추가 했습니다.
1.26-2023 .03.24	1.26.2	1.6.6	1.1.1	

권장 Microsoft Windows AMI ID 검색

노드를 배포할 때 사전 구축된 Amazon EKS 최적화 Amazon Machine Image(AMI)의 ID를 지정할 수 있습니다. 원하는 구성에 적합한 AMI ID를 검색하려면 AWS Systems Manager Parameter Store API를 쿼리합니다. 이 API를 사용하면 Amazon EKS 최적화 AMI ID를 수동으로 조회할 필요가 없습니다. 자세한 내용은 [GetParameter](#)를 참조하세요. Amazon EKS 최적화 AMI 메타데이터를 검색하려면 [IAM 보안 주체](#)에 `ssm:GetParameter` IAM 권한이 있어야 합니다.

하위 파라미터 `image_id`를 사용하는 다음 명령을 사용하여 최신 권장 Amazon EKS 최적화 Windows AMI의 이미지 ID를 검색할 수 있습니다. 필요에 따라 명령을 다음과 같이 수정한 다음에 수정한 명령을 실행합니다.

- `release`를 다음 옵션 중 하나로 변경합니다.
 - Windows Server 2022의 경우 `2022`를 사용합니다.
 - Windows Server 2019의 경우 `2019`를 사용합니다.
- `## ##`을 다음 옵션 중 하나로 바꿉니다. 자세한 내용은 [Windows Server의 Server Core 설치 옵션이란 무엇인가요](#)를 참조하세요.
 - `Core`를 공격 표면이 더 작은 최소한의 설치에 사용합니다.
 - Windows 데스크톱 환경을 포함하려면 `##`를 사용합니다.
- [kubernetes 버전](#)을 지원되는 `Amazon EKS ##`으로 교체합니다.
- `region-code`를 AMI ID를 원하는 [Amazon EKS 지원 AWS 리전](#)으로 바꿉니다.

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-release-English-installation-option-EKS_Optimized-kubernetes-version/image_id \
  --region region-code --query "Parameter.Value" --output text
```

다음은 자리 표시자를 대체한 후의 명령 예제입니다.

```
aws ssm get-parameter --name /aws/service/ami-windows-latest/Windows_Server-2022-English-Core-EKS_Optimized-k8s-n-2/image_id \
  --region us-west-2 --query "Parameter.Value" --output text
```

예제 출력은 다음과 같습니다.

```
ami-1234567890abcdef0
```

Image Builder를 사용한 사용자 지정 Windows AMI 구축

EC2 Image Builder를 사용하여 다음 옵션 중 하나로 사용자 지정 Amazon EKS 최적화 Windows AMI를 생성할 수 있습니다.

- [Amazon EKS 최적화 Windows AMI를 기반으로 사용](#)
- [Amazon 관리형 빌드 구성 요소 사용](#)

두 방법 모두 고유한 Image Builder 레시피를 만들어야 합니다. 자세한 내용은 Image Builder 사용 설명서의 [이미지 레시피의 새 버전 생성](#)을 참조하세요.

Important

eks에 대한 다음 Amazon 관리형 구성 요소에는 CVE-2024-5321에 대한 패치가 포함됩니다.

- 1.26.4 이상
- 1.27.2 이상
- 1.28.2 이상
- 1.29.2 이상
- 1.30.1 이상

Amazon EKS 최적화 Windows AMI를 기반으로 사용

이 옵션은 사용자 지정 Windows AMI를 구축하는 데 권장되는 방법입니다. AWS에서 제공하는 Amazon EKS 최적화 Windows AMI는 Amazon 관리형 빌드 구성 요소보다 더 자주 업데이트됩니다.

1. 새 Image Builder 레시피를 시작합니다.

- a. <https://console.aws.amazon.com/imagebuilder>에서 EC2 Image Builder 콘솔을 엽니다.
- b. 왼쪽 탐색 창에서 이미지 레시피를 선택합니다.
- c. 이미지 레시피 생성을 선택합니다.

2. 레시피 세부 정보 섹션에서 이름과 버전을 입력합니다.

3. 기본 이미지 섹션에서 Amazon EKS 최적화 Windows AMI의 ID를 지정합니다.

- a. 사용자 지정 AMI ID 입력을 선택합니다.
- b. 필요한 Windows OS 버전의 AMI ID를 검색합니다. 자세한 내용은 [the section called “최신 ID 받기”](#) 섹션을 참조하세요.
- c. 사용자 지정 AMI ID를 입력합니다. AMI ID를 찾을 수 없는 경우 AMI ID의 AWS 리전이 콘솔의 오른쪽 상단에 표시된 AWS 리전과 일치하는지 확인합니다.

4. (선택사항) 최신 보안 업데이트를 받으려면 빌드 구성 요소 - 섹션에서 update-windows 구성 요소를 추가합니다.

- a. 이름으로 구성 요소 찾기 검색 상자 오른쪽의 드롭다운 목록에서 Amazon 관리형을 선택합니다.
- b. 이름으로 구성 요소 찾기 상자에 update-windows를 입력합니다.

- c. **update-windows** 검색 결과의 확인란을 선택합니다. 이 구성 요소에는 운영 체제의 최신 Windows 패치가 포함되어 있습니다.
5. 나머지 이미지 레시피 입력을 필요한 구성으로 완료합니다. 자세한 내용은 Image Builder 사용 설명서의 [새 이미지 레시피 버전 생성\(콘솔\)](#)을 참조하세요.
6. 레시피 생성을 선택합니다.
7. 신규 또는 기존 이미지 파이프라인에서 새 이미지 레시피를 사용합니다. 이미지 파이프라인이 성공적으로 실행되면 사용자 지정 AMI가 출력 이미지로 나열되고 사용할 준비가 됩니다. 자세한 내용은 [EC2 Image Builder 콘솔 마법사를 사용하여 이미지 파이프라인 생성](#)을 참조하세요.

Amazon 관리형 빌드 구성 요소 사용

Amazon EKS 최적화 Windows AMI를 기반으로 사용할 수 없는 경우 Amazon 관리형 빌드 구성 요소를 대신 사용할 수 있습니다. 이 옵션은 지원되는 최신 Kubernetes 버전보다 뒤쳐질 수 있습니다.

1. 새 Image Builder 레시피를 시작합니다.
 - a. <https://console.aws.amazon.com/imagebuilder>에서 EC2 Image Builder 콘솔을 엽니다.
 - b. 왼쪽 탐색 창에서 이미지 레시피를 선택합니다.
 - c. 이미지 레시피 생성을 선택합니다.
2. 레시피 세부 정보 섹션에서 이름과 버전을 입력합니다.
3. 기본 이미지 섹션에서 사용자 지정 AMI를 생성하는 데 사용할 옵션을 결정합니다.
 - 관리형 이미지 선택 - 이미지 운영 체제(OS)에서 Windows를 선택합니다. 그리고 나서 이미지 오리진에 대해 다음 옵션 중 하나를 선택합니다.
 - 빠른 시작(Amazon 관리형) - 이미지 이름 드롭다운에서 Amazon EKS 지원 Windows Server 버전을 선택합니다. 자세한 내용은 [the section called "Windows"](#) 섹션을 참조하세요.
 - 내가 소유한 이미지 - 이미지 이름으로 자체 라이선스가 있는 이미지의 ARN을 선택합니다. 제공하는 이미지에 Amazon EKS 구성 요소가 이미 설치되어 있을 수 없습니다.
 - 사용자 지정 AMI ID 입력 - AMI ID에 자체 라이선스가 있는 AMI의 ID를 입력합니다. 제공하는 이미지에 Amazon EKS 구성 요소가 이미 설치되어 있을 수 없습니다.
4. 빌드 구성 요소 - Windows 섹션에서 다음을 수행합니다.
 - a. 이름으로 구성 요소 찾기 검색 상자 오른쪽의 드롭다운 목록에서 Amazon 관리형을 선택합니다.
 - b. 이름으로 구성 요소 찾기 상자에 eks를 입력합니다.
 - c. 반환되는 결과가 원하는 버전이 아닐 수도 있지만, **eks-optimized-ami-windows** 검색 결과의 확인란을 선택합니다.

- d. 이름으로 구성 요소 찾기 상자에 update-windows를 입력합니다.
 - e. update-windows 검색 결과의 확인란을 선택합니다. 이 구성 요소에는 운영 체제의 최신 Windows 패치가 포함되어 있습니다.
5. 선택한 구성 요소 섹션에서 다음을 수행합니다.
- a. **eks-optimized-ami-windows**의 버전 관리 옵션을 선택합니다.
 - b. 구성 요소 버전 지정을 선택합니다.
 - c. 구성 요소 버전 필드에 **version.x**를 입력합니다. 이때 **##**을 지원되는 Kubernetes 버전으로 바꿉니다. 버전 번호의 일부로 **x**를 입력하면 사용자가 명시적으로 정의하는 버전 부분과도 일치하는 최신 구성 요소 버전을 사용한다는 의미입니다. 콘솔 출력에서 원하는 버전을 관리형 구성 요소로 사용할 수 있는지 여부를 알려주므로 유의하세요. 최신 Kubernetes 버전은 빌드 구성 요소에 사용하지 못할 수 있습니다. 사용 가능한 버전에 대한 자세한 내용은 [the section called “eks-optimized-ami-windows 구성 요소 버전에 대한 정보 검색”](#) 섹션을 참조하세요.
6. 나머지 이미지 레시피 입력을 필요한 구성으로 완료합니다. 자세한 내용은 Image Builder 사용 설명서의 [새 이미지 레시피 버전 생성\(콘솔\)](#)을 참조하세요.
7. 레시피 생성을 선택합니다.
8. 신규 또는 기존 이미지 파이프라인에서 새 이미지 레시피를 사용합니다. 이미지 파이프라인이 성공적으로 실행되면 사용자 지정 AMI가 출력 이미지로 나열되고 사용할 준비가 됩니다. 자세한 내용은 [EC2 Image Builder 콘솔 마법사를 사용하여 이미지 파이프라인 생성](#)을 참조하세요.

eks-optimized-ami-windows 구성 요소 버전에 대한 정보 검색

각 구성 요소와 함께 설치되는 항목에 대한 특정 정보를 검색할 수 있습니다. 예를 들어, 설치된 kubelet 버전을 확인할 수 있습니다. 구성 요소는 Amazon EKS 지원 Windows 운영 체제 버전에 대한 기능 테스트를 거칩니다. 자세한 내용은 [the section called “출시 일정”](#) 섹션을 참조하세요. 지원되는 것으로 나열되지 않았거나 지원이 종료된 다른 Windows OS 버전은 구성 요소와 호환되지 않을 수 있습니다.

1. <https://console.aws.amazon.com/imagebuilder>에서 EC2 Image Builder 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 구성 요소를 선택합니다.
3. 이름으로 구성 요소 찾기 검색 상자 오른쪽의 드롭다운 목록에서 내 소유를 빠른 시작(Amazon 관리형)으로 변경합니다.
4. 이름으로 구성 요소 찾기 상자에 eks를 입력합니다.
5. (선택사항) 최신 버전을 사용하는 경우, 버전 열을 두 번 선택하여 내림차순으로 정렬합니다.
6. 원하는 버전의 **eks-optimized-ami-windows** 링크를 선택합니다.

결과 페이지의 설명에 특정 정보가 표시됩니다.

노드 자동 복구 활성화 및 노드 상태 문제 조사

노드 상태는 워크로드를 효과적으로 실행하는 노드의 운영 상태 및 기능을 나타냅니다. 정상 노드는 예상 연결을 유지하고, 리소스가 충분하며, 중단 없이 포드를 성공적으로 실행할 수 있습니다. 노드에 대한 자세한 내용은 [the section called “노드 상태 보기”](#) 및 [the section called “노드 로그 가져오기”](#) 섹션을 참조하세요.

Amazon EKS는 노드 모니터링 에이전트 및 노드 자동 복구를 제공하여 정상 노드를 유지하는 데 도움이 됩니다.

Important

노드 모니터링 에이전트 및 노드 자동 복구는 Linux에서만 사용할 수 있습니다. Windows에서는 이러한 기능을 사용할 수 없습니다.

노드 모니터링 에이전트

노드 모니터링 에이전트는 노드 로그를 자동으로 읽어 특정 상태 문제를 감지합니다. 노드 로그를 통해 구문 분석하여 장애를 감지하고 워커 노드에 대한 다양한 상태 정보를 표시합니다. 스토리지 및 네트워킹 문제와 같이 감지된 각 문제 범주에 대해 워커 노드에 전용 NodeCondition이 적용됩니다. 감지된 상태 문제에 대한 설명은 관찰성 대시보드에서 확인할 수 있습니다. 자세한 내용은 [the section called “노드 상태 문제”](#) 섹션을 참조하세요.

노드 모니터링 에이전트는 모든 Amazon EKS Auto Mode 클러스터의 기능으로 포함됩니다. 다른 클러스터 유형의 경우 모니터링 에이전트를 Amazon EKS 추가 기능으로 추가할 수 있습니다. 자세한 내용은 [the section called “추가 기능 생성”](#) 섹션을 참조하세요.

노드 자동 복구

노드 자동 복구는 노드 상태를 지속적으로 모니터링하여 감지된 문제에 자동으로 대응하고 가능한 경우 노드를 교체하는 추가 기능입니다. 이렇게 하면 수동 개입을 최소화하면서 클러스터의 전반적인 가용성을 높일 수 있습니다. 상태 확인이 실패하면 노드에 새 포드가 예약되지 않도록 노드가 자동으로 차단됩니다.

노드 자동 복구는 자체적으로 kubelet 및 수동으로 삭제된 노드 객체의 Ready 조건에 대응할 수 있습니다. 노드 모니터링 에이전트와 페어링하면 노드 자동 복구가 감지되지 않는 더 많은 조건에 대응할

수 있습니다. 이러한 추가 조건에는 KernelReady, NetworkingReady, StorageReady가 포함됩니다.

이 자동 노드 복구는 클러스터 조인 실패, 응답하지 않는 kubelet, 액셀러레이터(디바이스) 오류 증가 등 간헐적인 노드 문제를 자동으로 해결합니다. 안정성이 향상되면 애플리케이션 가동 중지 시간을 줄이고 클러스터 작업을 개선할 수 있습니다. 노드 자동 복구는 DiskPressure, MemoryPressure, PIDPressure와 같이 보고되는 특정 문제를 처리할 수 없습니다. Amazon EKS는 AcceleratedHardwareReady NodeConditions에 대한 작업을 수행하기 전에 10분, 다른 모든 조건에 대해서는 30분을 기다립니다.

또한 관리형 노드 그룹은 두 가지 시나리오에서 안전상의 이유로 노드 복구를 자동으로 비활성화합니다. 이전에 진행 중인 모든 복구 작업은 두 상황 모두에서 계속됩니다.

- Application Recovery Controller(ARC)를 통해 클러스터의 영역 전환이 트리거된 경우 이후의 모든 복구 작업이 중지됩니다.
- 노드 그룹에 5개가 넘는 노드가 있고 노드 그룹에서 비정상 상태의 노드가 20%를 초과하는 경우 복구 작업이 중지됩니다.

관리형 노드 그룹을 생성하거나 편집할 때 노드 자동 복구를 활성화할 수 있습니다.

- Amazon EKS 콘솔을 사용하는 경우 관리형 노드 그룹에 대해 노드 자동 복구 활성화 확인란을 활성화합니다. 자세한 내용은 [the section called “생성”](#) 섹션을 참조하세요.
- AWS CLI를 사용하는 경우 `eks create nodegroup` 또는 `eks update-nodegroup-config` 명령에 `--node-repair-config enabled=true`를 추가합니다.
- 노드 자동 복구와 함께 관리형 노드 그룹을 사용하는 eksctl ClusterConfig 예제는 GitHub의 [44-node-repair.yaml](#)을 참조하세요.

노드 상태 문제

다음 표에서는 노드 모니터링 에이전트가 감지할 수 있는 노드 상태 문제를 설명합니다. 두 가지 문제가 있습니다.

- 조건-인스턴스 교체 또는 재부팅과 같은 문제 해결 작업이 필요한 터미널 문제입니다. 자동 복구가 활성화되면 Amazon EKS는 노드 교체 또는 재부팅으로 복구 작업을 수행합니다. 자세한 내용은 [the section called “노드 조건”](#) 섹션을 참조하세요.
- 이벤트-일시적인 문제 또는 최적이지 아닌 노드 구성입니다. 자동 복구 작업은 수행되지 않습니다. 자세한 내용은 [the section called “노드 이벤트”](#) 섹션을 참조하세요.

커널 노드 상태 문제

명칭	심각도	설명
ForkFailedOutOfPID	Condition	시스템이 프로세스 ID 또는 메모리를 벗어났기 때문에 포크 또는 실행 직접 호출이 실패했습니다. 이는 좀비 프로세스 또는 물리적 메모리 소진으로 인해 발생할 수 있습니다.
AppBlocked	Event	일반적으로 입력 또는 출력에서 차단되어 일정 예약에서 장시간 동안 작업이 차단되었습니다.
AppCrash	Event	노드의 애플리케이션이 충돌했습니다.
ApproachingKernelPidMax	Event	프로세스 수가 현재 kernel.pid_max 설정당 사용 가능한 최대 PID 수에 근접하고 있으며, 그 이후에는 더 이상 프로세스를 시작할 수 없습니다.
ApproachingMaxOpenFiles	Event	현재 커널 설정을 고려할 때 열려 있는 파일의 수가 가능한 최대 수에 근접하고 있으며, 그 이후에는 새 파일을 열지 못합니다.
ConntrackExceededKernel	Event	연결 추적이 커널의 최대값을 초과했으며 새 연결을 설정할 수 없어 패킷 손실이 발생할 수 있습니다.
ExcessiveZombieProcesses	Event	완전히 회수할 수 없는 프로세스가 많은 수로 누적되고 이는

명칭	심각도	설명
		애플리케이션 문제를 나타내며 시스템 프로세스 제한에 도달할 수 있습니다.
KernelBug	Event	Linux 커널 자체에서 커널 버그가 감지 및 보고되었지만, 이는 CPU 또는 메모리 사용량이 높은 노드로 인해 발생할 수 있으며 이벤트 처리가 지연될 수 있습니다.
LargeEnvironment	Event	이 프로세스의 환경 변수 개수는 예상보다 많으며, 이는 enableServiceLinks가 true로 설정된 많은 서비스에서 발생할 수 있습니다. 이로 인해 성능 문제가 발생할 수 있습니다.
RapidCron	Event	cron 작업이 이 노드에서 5분 간격보다 빠르게 실행되고 있어 작업이 상당한 리소스를 소비하면 성능에 영향을 줄 수 있습니다.
SoftLockup	Event	CPU가 지정된 시간 동안 중지되었습니다.

네트워킹 노드 상태 문제

명칭	심각도	설명
InterfaceNotRunning	Condition	이 인터페이스가 실행 중이 아니거나 네트워크 문제가 있는 것 같습니다.

명칭	심각도	설명
InterfaceNotUp	Condition	이 인터페이스가 작동하지 않거나 네트워크 문제가 있는 것 같습니다.
IPAMDNSNotReady	Condition	IPAMD가 API 서버에 연결되지 않습니다.
IPAMDNSNotRunning	Condition	aws-k8s-agent 프로세스가 실행 중인 것으로 확인되지 않았습니다.
MissingLoopbackInterface	Condition	루프백 인터페이스가 이 인스턴스에서 누락되어 로컬 연결에 따라 서비스가 실패하게 됩니다.
BandwidthInExceeded	Event	인바운드 집계 대역폭이 인스턴스의 최대값을 초과하여 패킷이 대기열에 추가되거나 손실되었습니다.
BandwidthOutExceeded	Event	아웃바운드 집계 대역폭이 인스턴스의 최대값을 초과하여 패킷이 대기열에 추가되거나 손실되었습니다.
ConntrackExceeded	Event	연결 추적이 인스턴스의 최대값을 초과했으며 새 연결을 설정할 수 없어 패킷 손실이 발생할 수 있습니다.
IPAMDNSNoIPs	Event	IPAM-D에 IP 주소가 없습니다.
IPAMDNSRepeatedlyRestart	Event	IPAMD 서비스에서 여러 번 다시 시작되었습니다.

명칭	심각도	설명
KubeProxyNotReady	Event	Kube-proxy가 리소스를 감시하거나 나열하지 못했습니다.
LinkLocalExceeded	Event	로컬 프록시 서비스에 대한 트래픽의 PPS가 네트워크 인터페이스의 최대값을 초과하여 패킷이 손실되었습니다.
MissingDefaultRoutes	Event	기본 라우팅 규칙이 누락되었습니다.
MissingIPRules, MissingIP Routes	Event	라우팅 테이블에서 다음 포트 IP에 대한 라우팅 규칙이 누락되었습니다.
NetworkSysctl	Event	이 노드의 네트워크 sysctl 설정이 잘못되었을 수도 있습니다.
PortConflict	Event	포드가 hostPort를 사용하는 경우 호스트의 이미 바인딩된 포트를 재정의하는 iptables 규칙을 작성할 수 있으므로 API 서버가 kubelet에 액세스하지 못할 수 있습니다.
PPSExceeded	Event	양방향 PPS가 인스턴스의 최대값을 초과하여 패킷이 대기열에 추가되거나 손실되었습니다.
UnexpectedRejectRule	Event	예상치 못한 REJECT 또는 DROP 규칙이 iptables에서 발견되어 예상 트래픽을 차단할 수 있습니다.

Neuron 노드 상태 문제

명칭	심각도	설명
NeuronDMAError	Condition	DMA 엔진에서 복구할 수 없는 오류가 발생했습니다.
NeuronHBMUncorrectableError	Condition	HBM에서 수정할 수 없는 오류가 발생하여 잘못된 결과가 발생했습니다.
NeuronNCUncorrectableError	Condition	Neuron Core의 수정 불가능한 메모리 오류가 감지되었습니다.
NeuronSRAMUncorrectableError	Condition	온칩 SRAM에 패리티 오류가 발생하여 잘못된 결과가 발생했습니다.

NVIDIA 노드 상태 문제

자동 복구가 활성화된 경우 나열된 복구 작업은 문제가 감지되고 10분 후에 시작됩니다. XID 오류에 대한 자세한 내용은 NVIDIA GPU 배포 및 관리 설명서의 [Xid Errors](#)를 참조하세요. 개별 XID 메시지에 대한 자세한 내용은 NVIDIA GPU 배포 및 관리 설명서의 [Understanding Xid Messages](#)를 참조하세요.

명칭	심각도	설명	복구 작업
NvidiaDoubleBitError	Condition	GPU 드라이버에서 더블 비트 오류가 발생했습니다.	Replace
NvidiaNVLinkError	Condition	GPU 드라이버에서 NVLink 오류가 보고되었습니다.	Replace
NvidiaXID13Error	Condition	그래픽 엔진 예외가 있습니다.	재부팅

명칭	심각도	설명	복구 작업
NvidiaXID31Error	Condition	의심되는 하드웨어 문제가 있습니다.	재부팅
NvidiaXID48Error	Condition	드라이버에서 더블 비트 ECC 오류가 보고되었습니다.	재부팅
NvidiaXID63Error	Condition	페이지 사용 중지 또는 행 다시 매핑이 있습니다.	재부팅
NvidiaXID64Error	Condition	페이지 사용 중지 또는 노드 다시 매핑 시도에서 실패가 발생했습니다.	재부팅
NvidiaXID74Error	Condition	GPU에서 다른 GPU 또는 NVLink를 통한 NVSwitch로 연결하는데 문제가 있습니다. 이는 링크 자체의 하드웨어 장애 또는 링크의 원격 끝에 있는 디바이스에 문제가 있음을 나타낼 수 있습니다.	Replace
NvidiaXID79Error	Condition	GPU 드라이버가 PCI Express 연결을 통해 GPU에 액세스하려고 시도했지만 GPU에 액세스할 수 없는 것을 확인했습니다.	Replace
NvidiaXID94Error	Condition	ECC 메모리 오류가 있습니다.	재부팅

명칭	심각도	설명	복구 작업
NvidiaXID95Error	Condition	ECC 메모리 오류가 있습니다.	재부팅
NvidiaXID119Error	Condition	GSP가 드라이버의 다른 비트에서 RPC 요청에 응답하는 시간을 초과했습니다.	Replace
NvidiaXID120Error	Condition	GSP가 제시간에 응답했지만 오류가 발생했습니다.	Replace
NvidiaXID121Error	Condition	C2C는 칩 상호 연결입니다. CPU, CPU 액셀러레이터 등 간에 메모리를 공유할 수 있습니다.	Replace
NvidiaXID140Error	Condition	GPU 드라이버는 동적 페이지 오프라이닝 또는 행 대시 매핑을 위해 페이지를 표시하는 GPU 드라이버의 기능을 방해하는 방식으로 GPU 메모리에서 수정 불가능한 오류를 관찰했을 수 있습니다.	Replace

명칭	심각도	설명	복구 작업
NvidiaPageRetirement	Event	GPU 드라이버가 사용 중지를 위해 메모리 페이지를 표시했습니다. 동일한 주소에 단일 더블 비트 오류가 있거나 두 개의 단일 비트 오류가 발생하는 경우 이 문제가 발생할 수 있습니다.	없음
NvidiaXID[Code]Warning	Event	이 목록에 정의된 것 이외의 XID가 발생하면 이 이벤트가 발생합니다.	없음
DCGMError	Condition	데이터 센터 GPU 관리자(DCGM) 호스트 프로세스에 대한 연결이 끊어졌거나 설정할 수 없습니다.	없음
DCGMDiagnosticError	Condition	DCGM 활성 진단을 실행하는 동안 문제가 발생했습니다.	없음
DCGMDiagnosticFailure	Condition	DCGM 활성 진단 테스트 제품군의 테스트 사례가 실패했습니다.	없음

런타임 노드 상태 문제

명칭	심각도	설명
PodStuckTerminating	Condition	포드가 과도한 시간 동안 종료되지 않거나 중단되었습니다.

명칭	심각도	설명
		이는 포드 상태 진행을 방지하는 CRI 오류로 인해 발생할 수 있습니다.
%sRepeatedRestart	Event	노드에서 모든 systemd 서비스를 다시 시작합니다(제목 대 소문자를 사용한 단위 이름 형식).
ContainerRuntimeFailed	Event	컨테이너 런타임이 컨테이너를 생성하지 못했습니다. 반복적으로 발생하는 경우 보고된 문제와 관련이 있을 수 있습니다.
KubeletFailed	Event	kubelet이 실패 상태가 되었습니다.
LivenessProbeFailures	Event	활성 프로브 장애가 감지되었습니다. 반복해서 발생하는 경우 애플리케이션 코드 문제 또는 불충분한 제한 시간 값을 나타낼 수 있습니다.
ReadinessProbeFailures	Event	준비 상태 프로브 장애가 감지되었습니다. 반복적으로 발생하는 경우 애플리케이션 코드 문제 또는 불충분한 제한 시간 값을 나타낼 수 있습니다.
ServiceFailedToStart	Event	systemd 단위를 시작하지 못했습니다.

스토리지 노드 상태 문제

명칭	심각도	설명
XFSSmallAverageClusterSize	Condition	XFS Average Cluster 크기는 작으며, 이는 사용 가능한 inode 또는 여유 공간에도 불구하고 파일 생성을 방지할 수 있는 과도한 여유 공간 조각화를 나타냅니다.
EtcHostsMountFailed	Event	kubelet-container 작업 중 사용자 데이터가 /var/lib/kubelet/pods 를 다시 탑재하여 kubelet이 생성한 /etc/hosts 를 탑재하지 못했습니다.
IODelays	Event	프로세스에서 입력 또는 출력 지연이 감지되었으며, 이는 과도한 경우 입력-출력 프로비저닝이 부족할 수 있음을 나타냅니다.
KubeletDiskUsageSlow	Event	Kubelet이 파일 시스템에 액세스하려고 할 때 느린 디스크 사용 속도를 보고하며, 이는 디스크 입력-출력이 부족하거나 파일 시스템 문제가 있음을 나타낼 수 있습니다.

노드의 상태 보기

이 주제에서는 Amazon EKS 클러스터에서 노드 상태 모니터링에 사용할 수 있는 도구와 방법을 설명합니다. 관련 정보에는 노드 수준 문제를 식별하고 진단하는 데 도움이 되는 노드 조건, 이벤트, 감지 사례가 포함됩니다. 여기에 설명된 명령과 패턴을 사용하여 노드 상태 리소스를 검사하고, 상태 조건을 해석하고, 운영 문제 해결을 위해 노드 이벤트를 분석할 수 있습니다.

모든 노드에 대한 Kubernetes 명령을 사용하여 일부 노드 상태 정보를 가져올 수 있습니다. 또한 Amazon EKS Auto Mode 또는 Amazon EKS 관리형 추가 기능을 통해 노드 모니터링 에이전트를 사용하는 경우 문제 해결에 도움이 되는 더욱 다양한 노드 신호를 얻을 수 있습니다. 노드 모니터링 에이전트에서 감지된 상태 문제에 대한 설명은 관찰성 대시보드에서도 확인할 수 있습니다. 자세한 내용은 [the section called “노드 상태”](#) 단원을 참조하십시오.

노드 조건

노드 조건은 인스턴스 교체 또는 재부팅과 같은 문제 해결 작업이 필요한 터미널 문제를 나타냅니다.

모든 노드에 대한 조건을 가져오려면:

```
kubectl get nodes -o 'custom-
columns=NAME:.metadata.name,CONDITIONS:.status.conditions[*].type,STATUS:.status.conditions[*].
```

특정 노드에 대한 세부 조건을 가져오려면

```
kubectl describe node node-name
```

정상 노드의 조건 출력 예제:

```
- lastHeartbeatTime: "2024-11-21T19:07:40Z"
  lastTransitionTime: "2024-11-08T03:57:40Z"
  message: Monitoring for the Networking system is active
  reason: NetworkingIsReady
  status: "True"
  type: NetworkingReady
```

네트워킹 문제가 있는 비정상 노드의 조건 예제:

```
- lastHeartbeatTime: "2024-11-21T19:12:29Z"
  lastTransitionTime: "2024-11-08T17:04:17Z"
  message: IPAM-D has failed to connect to API Server which could be an issue with
    IPTable rules or any other network configuration.
  reason: IPAMNotReady
  status: "False"
  type: NetworkingReady
```

노드 이벤트

노드 이벤트는 일시적인 문제 또는 최적이지 않은 구성을 나타냅니다.

노드 모니터링 에이전트가 보고한 모든 이벤트를 가져오려면

노드 모니터링 에이전트를 사용할 수 있는 경우 다음 명령을 실행할 수 있습니다.

```
kubectl get events --field-selector=reportingComponent=eks-node-monitoring-agent
```

샘플 출력:

LAST SEEN	TYPE	REASON	OBJECT	MESSAGE
4s	Warning	SoftLockup	node/ip-192-168-71-251.us-west-2.compute.internal	CPU stuck for 23s

모든 노드에 대한 이벤트를 가져오려면

```
kubectl get events --field-selector involvedObject.kind=Node
```

특정 노드에 대한 이벤트를 가져오려면

```
kubectl get events --field-selector involvedObject.kind=Node,involvedObject.name=node-name
```

이벤트를 실시간으로 보려면

```
kubectl get events -w --field-selector involvedObject.kind=Node
```

이벤트 출력 예제:

LAST SEEN	TYPE	REASON	OBJECT	MESSAGE
2m	Warning	MemoryPressure	Node/node-1	Node experiencing memory pressure
5m	Normal	NodeReady	Node/node-1	Node became ready

일반적인 문제 해결 명령

```
# Get comprehensive node status
```

```
kubectl get node node-name -o yaml

# Watch node status changes
kubectl get nodes -w

# Get node metrics
kubectl top node
```

kubectl 및 S3를 사용하여 관리형 노드에 대한 노드 로그 검색

노드 모니터링 에이전트가 있는 Amazon EKS 관리형 노드에 대한 노드 로그를 검색하는 방법을 알아 봅니다.

사전 조건

다음 항목이 준비되어 있는지 확인합니다.

- 기존 Amazon EKS 클러스터에 노드 모니터링 에이전트가 있습니다. 자세한 내용은 [the section called “노드 상태”](#) 단원을 참조하십시오.
- 클러스터와의 통신을 위해 kubectl 명령줄 도구가 설치 및 구성되어 있습니다.
- S3 버킷 및 객체를 생성할 수 있는 충분한 권한을 가지고 AWS CLI를 설치하고 로그인했습니다.
- 최신 버전의 Python 3가 설치되어 있습니다.
- AWS SDK for Python 3, Boto 3가 설치되어 있습니다.

1단계: S3 버킷 대상 생성(선택 사항)

아직 로그를 저장할 S3 버킷이 없는 경우 한 개를 생성합니다. 다음 AWS CLI 명령을 사용합니다. 버킷은 기본적으로 private 액세스 제어 목록으로 설정됩니다. *bucket-name*을 선택한 고유의 버킷 이름으로 바꿉니다.

```
aws s3api create-bucket --bucket bucket-name
```

2단계: HTTP Put에 대해 미리 서명된 S3 URL 생성

Amazon EKS는 HTTP PUT 작업을 수행하여 노드 로그를 지정한 URL로 반환합니다. 이 자습서에서는 미리 서명된 S3 HTTP PUT URL을 생성합니다.

로그는 .tar.gz 확장자와 함께 gzip tarball로 반환됩니다.

Note

AWS API 또는 SDK를 사용하여 EKS에 대해 미리 서명된 S3 업로드 URL을 생성하고 로그 파일을 업로드해야 합니다. AWS CLI를 사용하여 미리 서명된 S3 업로드 URL을 생성할 수 없습니다.

1. 버킷에서 로그를 저장할 위치를 결정합니다. 예를 들어 2024-11-12/logs1.tar.gz를 키로 사용할 수 있습니다.
2. 다음 Python 코드를 presign-upload.py 파일에 저장합니다. <bucket-name> 및 <key>를 교체합니다. 키는 .tar.gz로 끝나야 합니다.

```
import boto3; print(boto3.client('s3').generate_presigned_url(
    ClientMethod='put_object',
    Params={'Bucket': '<bucket-name>', 'Key': '<key>'},
    ExpiresIn=1000
))
```

3. 스크립트를 실행합니다.

```
python presign-upload.py
```

4. URL 출력을 기록해 둡니다. 다음 단계에서 이 값을 http-put-destination으로 사용합니다.

자세한 내용은 AWS Boto3 SDK for Python 설명서에서 [Generate a presigned URL to upload a file](#)을 참조하세요.

3단계: NodeDiagnostic 리소스 생성

로그를 수집할 노드의 이름을 식별합니다.

노드 이름을 리소스 이름으로 사용하고 HTTP PUT URL 대상을 제공하는 NodeDiagnostic 매니페스트를 생성합니다.

```
apiVersion: eks.amazonaws.com/v1alpha1
kind: NodeDiagnostic
metadata:
  name: node-name
spec:
  logCapture:
```



```
destination: http-put-destination
```

매니페스트를 클러스터에 적용합니다.

```
kubectl apply -f nodediagnostic.yaml
```

NodeDiagnostic 리소스를 설명하여 컬렉션 상태를 확인할 수 있습니다.

- Success 또는 SuccessWithErrors 상태는 작업이 완료되고 로그가 제공된 대상으로 업로드되었음을 나타냅니다(SuccessWithErrors는 일부 로그가 누락되었을 수 있음을 나타냄).
- 상태가 실패인 경우 업로드 URL이 잘 구성되고 완료되지 않았는지 확인합니다.

```
kubectl describe nodediagnostics.eks.amazonaws.com/node-name
```

4단계: S3에서 로그 다운로드

로그 다운로드를 시도하기 전에 1분 정도 기다립니다. 그런 다음 S3 CLI를 사용하여 로그를 다운로드합니다.

```
# Once NodeDiagnostic shows Success status, download the logs  
aws s3 cp s3://bucket-name/key ./node-logs.tar.gz
```

5단계: NodeDiagnostic 리소스 정리

- NodeDiagnostic 리소스는 자동으로 삭제되지 않습니다. 로그 아티팩트를 가져온 후에 이를 직접 정리해야 합니다.

```
# Delete the NodeDiagnostic resource  
kubectl delete nodediagnostics.eks.amazonaws.com/node-name
```

Amazon EKS Hybrid Nodes 개요

Amazon EKS Hybrid Nodes를 사용하면 온프레미스 및 엣지 인프라를 Amazon EKS 클러스터의 노드로 사용할 수 있습니다. AWS는 Amazon EKS 클러스터의 AWS 호스팅 Kubernetes 컨트롤 플레인을

관리하고 온프레미스 또는 엣지 환경에서 실행되는 하이브리드 노드를 관리합니다. 이를 통해 환경 전반의 Kubernetes 관리를 통합하고 온프레미스 및 엣지 애플리케이션을 위해 Kubernetes 컨트롤 플레인 관리를 AWS로 오프로드합니다.

Amazon EKS Hybrid Nodes는 모든 온프레미스 하드웨어 또는 가상 머신과 함께 작동하여 애플리케이션이 실행되어야 하는 모든 곳에 Amazon EKS의 효율성, 확장성, 가용성을 제공합니다. Amazon EKS 추가 기능, Amazon EKS Pod Identity, 클러스터 액세스 항목, 클러스터 인사이트, 확장된 Kubernetes 버전 지원을 포함하여 Amazon EKS Hybrid Nodes와 함께 다양한 Amazon EKS 기능을 사용할 수 있습니다. Amazon EKS Hybrid Nodes는 중앙 집중식 모니터링, 로깅, ID 관리를 위해 AWS Systems Manager, AWS IAM Roles Anywhere, Amazon Managed Service for Prometheus, Amazon CloudWatch, Amazon GuardDuty를 포함한 AWS 서비스와 기본적으로 통합됩니다.

Amazon EKS Hybrid Nodes를 사용하면 선결제 약정이나 최소 요금이 없으며, Amazon EKS 클러스터에 연결될 때 하이브리드 노드의 vCPU 리소스에 대해 시간당 요금이 부과됩니다. 가격에 대한 자세한 내용은 [Amazon EKS Pricing](#)을 참조하세요.

Features

EKS Hybrid Nodes에는 다음과 같은 상위 수준 기능이 있습니다.

- 관리형 Kubernetes 컨트롤 플레인: AWS는 EKS 클러스터의 AWS 호스팅 Kubernetes 컨트롤 플레인을 관리하고 사용자는 온프레미스 또는 엣지 환경에서 실행되는 하이브리드 노드를 관리합니다. 이를 통해 환경 전반의 Kubernetes 관리를 통합하고 온프레미스 및 엣지 애플리케이션을 위해 Kubernetes 컨트롤 플레인 관리를 AWS로 오프로드합니다. Kubernetes 컨트롤 플레인을 AWS로 이동하면 애플리케이션의 온프레미스 용량을 절약하고 Kubernetes 컨트롤 플레인이 워크로드에 따라 확장된다고 신뢰할 수 있습니다.
- 일관된 EKS 경험: 온프레미스 및 클라우드 환경에서 일관된 EKS 경험을 위해 EKS 추가 기능, EKS Pod Identity, 클러스터 액세스 항목, 클러스터 인사이트, 확장된 Kubernetes 버전 지원 등 대부분의 EKS 기능이 EKS Hybrid Nodes에서 지원됩니다. EKS Hybrid Nodes에서 지원되는 EKS 추가 기능에 대한 자세한 내용은 [the section called “추가 기능 구성”](#) 섹션을 참조하세요.
- 중앙 집중식 관찰성 및 ID 관리: EKS Hybrid Nodes는 중앙 집중식 모니터링, 로깅, ID 관리를 위해 AWS Systems Manager, AWS IAM Roles Anywhere, Amazon Managed Service for Prometheus, Amazon CloudWatch, Amazon GuardDuty를 포함한 AWS 서비스와 기본적으로 통합됩니다.
- 클라우드로 버스트 또는 온프레미스 용량 추가: 단일 EKS 클러스터를 사용하여 AWS 리전, AWS 로컬 영역 또는 AWS Outposts에서 하이브리드 노드 및 노드를 실행하여 클라우드로 버스트하거나 EKS 클러스터에 온프레미스 용량을 추가할 수 있습니다. 자세한 내용은 [혼합 모드 클러스터 고려 사항](#)을 참조하세요.

- **유연한 인프라:** EKS Hybrid Nodes는 자체 인프라 사용 접근 방식을 따르며 하이브리드 노드에 사용하는 인프라에 구애받지 않습니다. 물리적 또는 가상 머신과 x86 및 ARM 아키텍처에서 하이브리드 노드를 실행할 수 있으므로 다양한 인프라 유형의 하이브리드 노드에서 실행되는 온프레미스 워크로드를 마이그레이션할 수 있습니다.
- **유연한 네트워킹:** EKS Hybrid Nodes에서는 EKS 컨트롤 플레인과 하이브리드 노드 간의 통신이 클러스터 생성 중에 전달하는 VPC 및 서브넷을 통해 라우팅되며, 이는 컨트롤 플레인-노드 네트워킹을 위한 EKS의 [기존 메커니즘](#)을 기반으로 합니다. 이는 온프레미스 네트워크를 AWS의 VPC에 연결하는 기본 방법에 유연성을 제공합니다. AWS Site-to-Site VPN, AWS Direct Connect, 자체 VPN 솔루션을 포함하여 몇 가지 [문서화된 옵션](#)을 사용하고 사용 사례에 가장 적합한 방법을 선택할 수 있습니다.

일반적인 고려 사항

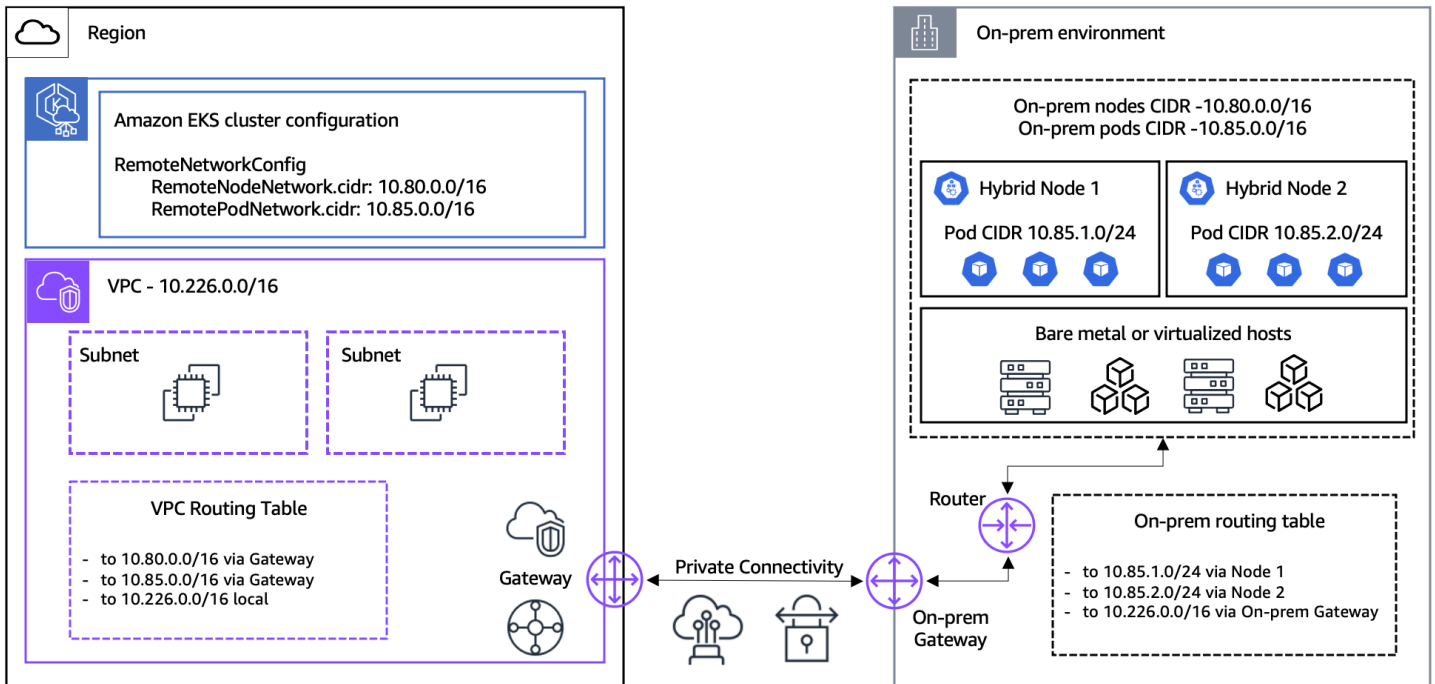
- EKS Hybrid Nodes는 신규 또는 기존 EKS 클러스터와 함께 사용할 수 있습니다.
- EKS Hybrid Nodes는 AWS GovCloud(미국) 리전 및 AWS 중국 리전을 제외한 모든 AWS 리전에서 사용할 수 있습니다.
- EKS Hybrid Nodes에는 온프레미스 환경과 AWS 간에 신뢰할 수 있는 연결이 있어야 합니다. EKS Hybrid Nodes는 연결 해제, 중단, 간헐적 또는 제한된(DDIL) 환경에 적합하지 않습니다. DDIL 환경에서 실행하는 경우 [Amazon EKS Anywhere](#)를 고려하세요.
- AWS 리전, AWS 로컬 영역, AWS Outposts를 포함한 클라우드 인프라 또는 다른 클라우드에서 EKS Hybrid Nodes를 실행하는 것은 지원되지 않습니다. Amazon EC2 인스턴스에서 하이브리드 노드를 실행하는 경우 하이브리드 노드 요금이 부과됩니다.
- 하이브리드 노드에 대한 결제는 노드가 EKS 클러스터에 조인할 때 시작되고 노드가 클러스터에서 제거될 때 중지됩니다. 하이브리드 노드를 사용하지 않는 경우 EKS 클러스터에서 하이브리드 노드를 제거해야 합니다.

하이브리드 노드에 대한 사전 조건 설정

Amazon EKS Hybrid Nodes를 사용하려면 온프레미스 환경에서 지원되는 운영 체제가 있는 AWS, 베어 메탈 서버 또는 가상 머신으로의 프라이빗 연결과 AWS IAM Roles Anywhere 또는 AWS Systems Manager(SSM) 하이브리드 활성화가 구성되어 있어야 합니다. 하이브리드 노드 수명 주기 전반에 걸쳐 이러한 사전 조건을 관리할 책임은 사용자에게 있습니다.

- 온프레미스 환경과 AWS 간의 하이브리드 네트워크 연결
- 물리적 또는 가상 머신 형태의 인프라

- 하이브리드 노드와 호환되는 운영 체제
- 구성된 온프레미스 IAM 자격 증명 공급자



하이브리드 네트워크 연결

Amazon EKS 컨트롤 플레인과 하이브리드 노드 간의 통신은 클러스터 생성 중에 전달하는 VPC와 서브넷을 통해 라우팅되며, 이는 컨트롤 플레인에서 노드 네트워킹을 위한 Amazon EKS의 [기존 메커니즘](#)을 기반으로 합니다. AWS Site-to-Site VPN, AWS Direct Connect 또는 자체 VPN 연결을 포함하여 온프레미스 환경을 VPC와 연결하는 데 사용할 수 있는 몇 가지 [문서화된 옵션](#)이 있습니다. 하이브리드 네트워크 연결에 이러한 솔루션을 사용하는 방법에 대한 자세한 내용은 [AWS Site-to-Site VPN](#) 및 [AWS Direct Connect](#) 사용 설명서를 참조하세요.

최적의 경험을 위해 AWS는 AWS 리전의 하이브리드 노드 연결에 최소 100Mbps 및 최대 200ms의 왕복 지연 시간을 제공하는 안정적인 네트워크 연결을 권장합니다. 대역폭 및 지연 시간 요구 사항은 하이브리드 노드 수와 애플리케이션 이미지 크기, 애플리케이션 탄력성, 모니터링 및 로깅 구성, 다른 AWS 서비스에 저장된 데이터에 대한 애플리케이션 종속성과 같은 워크로드 특성에 따라 달라질 수 있습니다. 네트워킹 설정이 워크로드의 요구 사항을 충족하는지 확인하려면 프로덕션에 배포하기 전에 자체 애플리케이션 및 환경을 사용하여 테스트하는 것이 좋습니다.

온프레미스 네트워크 구성

Amazon EKS 컨트롤 플레인에서 온프레미스 환경으로의 인바운드 네트워크 액세스를 활성화해야 Amazon EKS 컨트롤 플레인이 하이브리드 노드에서 실행 중인 kubelet와 통신하고, 선택적으로 하이브리드 노드에서 실행 중인 웹훅과 통신할 수 있습니다. 또한 Amazon EKS 컨트롤 플레인과 통신하려면 하이브리드 노드 및 해당 노드에서 실행되는 구성 요소에 대해 아웃바운드 네트워크 액세스를 활성화해야 합니다. AWS Direct Connect, AWS Site-to-Site VPN 또는 자체 VPN 연결에 대해 완전히 비공개로 유지되도록 이 통신을 구성할 수 있습니다. 방화벽 및 온프레미스 환경에서 활성화해야 하는 필수 포트 및 프로토콜의 전체 목록은 [the section called “네트워킹 준비”](#) 섹션을 참조하세요.

온프레미스 노드 및 포트 네트워크에 사용하는 CIDR(Classless Inter-Domain Routing) 범위는 IPv4 RFC1918 주소 범위를 사용해야 합니다. 하이브리드 노드 지원 Amazon EKS 클러스터를 생성할 때 Amazon EKS 컨트롤 플레인에서 하이브리드 노드 및 해당 노드에서 실행 중인 리소스로의 통신을 가능하게 하려면 온프레미스 노드와 선택적으로 포트 CIDR을 전달합니다. 온프레미스 라우터는 온프레미스 노드와 선택적으로 포트에 대한 경로로 구성되어야 합니다. Border Gateway Protocol(BGP) 또는 정적 구성을 사용하여 포트 IP를 라우터에 알릴 수 있습니다.

EKS 클러스터 구성

지연 시간을 최소화하려면 온프레미스 또는 엣지 환경에 가장 가까운 AWS 리전에서 Amazon EKS 클러스터를 생성하는 것이 좋습니다. Amazon EKS 클러스터를 생성하는 동안 온프레미스 노드 및 포트 CIDR을 두 개의 API 필드인 RemoteNodeNetwork 및 RemotePodNetwork를 통해 전달합니다. 온프레미스 네트워크 팀과 상의하여 온프레미스 노드 및 포트 CIDR을 식별해야 할 수 있습니다. 노드 CIDR은 온프레미스 네트워크에서 할당되고 포트 CIDR은 CNI에 오버레이 네트워크를 사용하는 경우, 사용하는 컨테이너 네트워크 인터페이스(CNI)에서 할당됩니다.

온프레미스 노드 및 포트 CIDR은 VPC를 통해 하이브리드 노드에서 실행되는 kubelet 및 포드로 트래픽을 라우팅하도록 Amazon EKS 컨트롤 플레인을 구성하는 데 사용됩니다. 온프레미스 노드 및 포트 CIDR은 서로, 클러스터 생성 중에 전달하는 VPC CIDR 또는 Amazon EKS 클러스터의 서비스 IPv4 구성과 중첩될 수 없습니다. 포트 CIDR은 선택 사항입니다. 포트 트래픽이 온프레미스 호스트에서 나갈 때 CNI가 포트 IP 주소에 네트워크 주소 변환(NAT) 또는 매스커레이딩을 사용하지 않는 경우 포트 CIDR을 구성해야 합니다. 하이브리드 노드에서 Kubernetes 웹훅을 실행하는 경우 포트 CIDR을 추가로 구성해야 합니다. 예를 들어 AWS Distro for Open Telemetry(ADOT)는 웹훅을 사용합니다.

Amazon EKS Kubernetes API 서버 엔드포인트에는 퍼블릭 또는 프라이빗 엔드포인트 액세스를 사용하는 것이 좋습니다. '퍼블릭 및 프라이빗'을 선택하면 Amazon EKS Kubernetes API 서버 엔드포인트가 항상 VPC 외부에서 실행되는 하이브리드 노드의 퍼블릭 IP로 확인되므로 하이브리드 노드가 클러스터에 조인하지 못할 수 있습니다. Amazon EKS Kubernetes API 서버 엔드포인트에 퍼블릭 또는 프

라이빗 엔드포인트 액세스를 사용할 수 있습니다. '퍼블릭 및 프라이빗'을 선택할 수 없습니다. 퍼블릭 엔드포인트 액세스를 사용하면 Kubernetes API 서버 엔드포인트가 퍼블릭 IP로 확인되고 하이브리드 노드에서 Amazon EKS 컨트롤 플레인으로의 통신이 인터넷을 통해 라우팅됩니다. 프라이빗 엔드포인트 액세스를 선택하면 Kubernetes API 서버 엔드포인트가 프라이빗 IP로 확인되고 하이브리드 노드에서 Amazon EKS 컨트롤 플레인으로의 통신이 프라이빗 연결 링크를 통해 라우팅되며, 대부분의 경우 AWS Direct Connect 또는 AWS Site-to-Site VPN으로 라우팅됩니다.

VPC 구성

온프레미스 노드의 라우팅 테이블에 있는 경로와 선택적으로 가상 프라이빗 게이트웨이(VGW) 또는 전송 게이트웨이(TGW)를 대상으로 하는 포드 네트워크로 Amazon EKS 클러스터 생성 중에 전달하는 VPC를 구성해야 합니다. 예를 들면 다음과 같습니다. REMOTE_NODE_CIDR 및 REMOTE_POD_CIDR을 온프레미스 네트워크의 값으로 바꿉니다.

대상 주소	대상	설명
10.226.0.0/16	로컬	VPC 내의 VPC 경로에 대한 로컬 트래픽
REMOTE_NODE_CIDR	tgw-abcdef123456	온프레미스 노드 CIDR, 트래픽을 TGW로 라우팅
REMOTE_POD_CIDR	tgw-abcdef123456	온프레미스 포드 CIDR, 트래픽을 TGW로 라우팅

보안 그룹 구성

클러스터를 생성할 때 Amazon EKS에서 eks-cluster-sg-<cluster-name>-<uniqueID>라는 보안 그룹을 만듭니다. 이 클러스터 보안 그룹의 인바운드 규칙은 변경할 수 없지만 아웃바운드 규칙을 제한할 수 있습니다. 하이브리드 노드에서 실행되는 kubelet와 선택적으로 웹후크가 Amazon EKS 컨트롤 플레인에 연결할 수 있게 하려면 클러스터에 보안 그룹을 추가해야 합니다. 이 추가 보안 그룹에 필요한 인바운드 규칙은 다음과 같습니다. REMOTE_NODE_CIDR 및 REMOTE_POD_CIDR을 온프레미스 네트워크의 값으로 바꿉니다.

명칭	보안 그룹 규칙 ID	IP 버전	유형	프로토콜	포트 범위	소스
온프레미스 노드 인바 운드	sgr-abcde f123456	IPv4	HTTPS	TCP	443	REMOTE_NO DE_CIDR
온프레미스 포드 인바 운드	sgr-abcde f654321	IPv4	HTTPS	TCP	443	REMOTE_PO D_CIDR

인프라

하이브리드 노드로 사용할 수 있는 베어 메탈 서버 또는 가상 머신이 있어야 합니다. 하이브리드 노드는 기본 인프라에 구매받지 않으며 x86 및 ARM 아키텍처를 지원합니다. Amazon EKS Hybrid Nodes는 '자체 인프라 구축' 접근 방식을 따르며, 이 방식에서 하이브리드 노드에 사용하는 베어 메탈 서버 또는 가상 머신의 프로비저닝 및 관리는 사용자의 책임입니다. 엄격한 최소 리소스 요구 사항은 없지만 하이브리드 노드에 대해 최소 1개의 vCPU 및 1GiB RAM이 있는 호스트를 사용하는 것이 좋습니다.

운영 체제

Amazon Linux 2023(AL2023), Ubuntu, RHEL은 하이브리드 노드의 노드 운영 체제로 사용하도록 지속적으로 검증되었습니다. AWS는 이러한 운영 체제와의 하이브리드 노드 통합을 지원하지만 운영 체제 자체는 지원하지 않습니다. AL2023은 Amazon EC2 외부에서 실행되는 경우 AWS 지원 플랜의 적용을 받지 않습니다. AL2023은 온프레미스 가상화 환경에서만 사용할 수 있습니다. 자세한 내용은 [Amazon Linux 2023 User Guide](#)를 참조하세요.

운영 체제 프로비저닝 및 관리는 사용자의 책임입니다. 하이브리드 노드를 처음 테스트하는 경우 이미 프로비저닝된 호스트에서 Amazon EKS Hybrid Nodes CLI(nodeadm)를 실행하는 것이 가장 쉽습니다. 프로덕션 배포의 경우 호스트 시작 시 호스트를 Amazon EKS 클러스터에 자동으로 조인하기 위해 systemd 서비스로 실행하도록 구성된 nodeadm을 골든 운영 체제 이미지에 포함하는 것이 좋습니다.

온프레미스 IAM 자격 증명 공급자

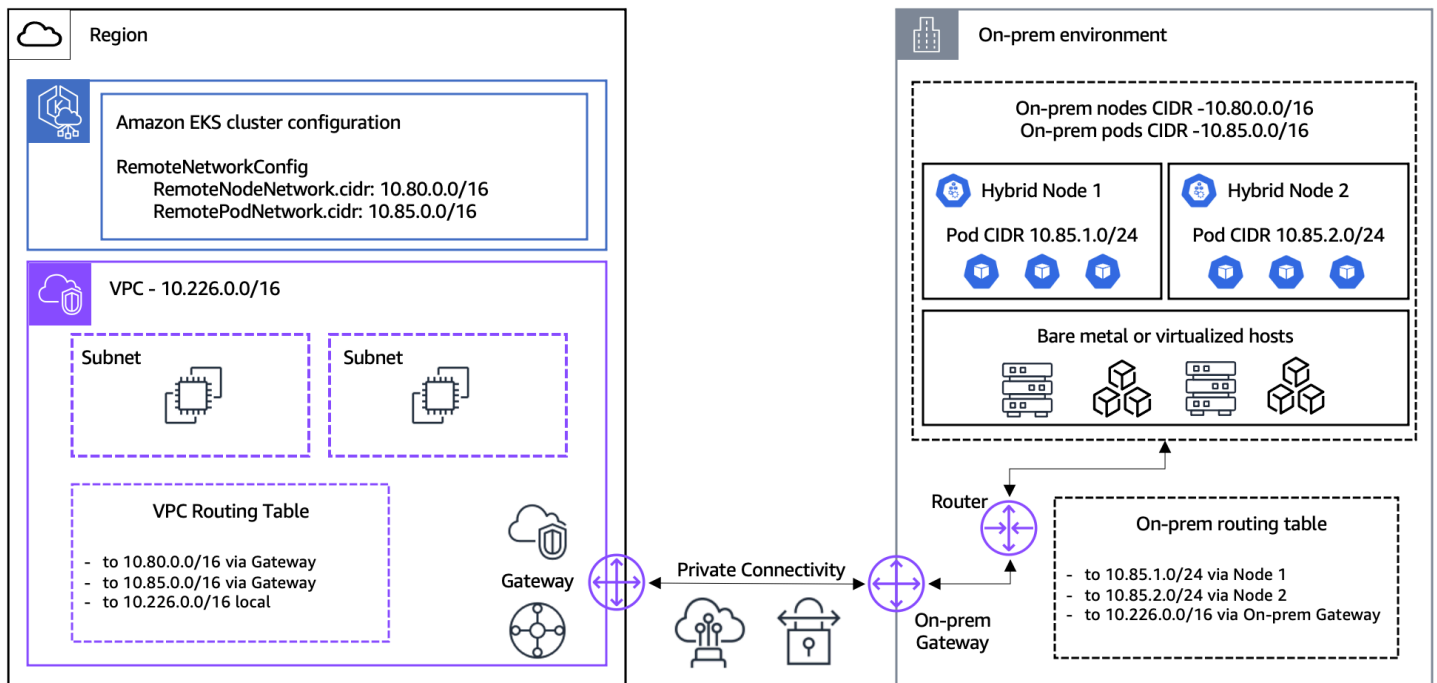
Amazon EKS Hybrid Nodes는 AWS SSM 하이브리드 활성화 또는 AWS IAM Roles Anywhere에서 프로비저닝한 임시 IAM 자격 증명을 사용하여 Amazon EKS 클러스터로 인증합니다. Amazon EKS Hybrid Nodes CLI(nodeadm)와 함께 AWS SSM 하이브리드 활성화 또는 AWS IAM Roles Anywhere

를 사용해야 합니다. 인증 기관(CA)이 있는 기존 퍼블릭 키 인프라(PKI) 및 온프레미스 환경에 대한 인증서가 없는 경우 AWS SSM 하이브리드 활성화를 사용하는 것이 좋습니다. 온프레미스에 기존 PKI 및 인증서가 있는 경우 AWS IAM Roles Anywhere를 사용합니다.

Amazon EC2에서 실행되는 노드의 [the section called “노드 IAM 역할”](#)과 마찬가지로 하이브리드 노드를 Amazon EKS 클러스터에 조인하는 데 필요한 권한이 있는 하이브리드 노드 IAM 역할을 생성합니다. AWS IAM Roles Anywhere를 사용하는 경우 AWS IAM Roles Anywhere가 하이브리드 노드 IAM 역할을 수입하고 하이브리드 노드 IAM 역할을 수입 가능한 역할로 사용하여 AWS IAM Roles Anywhere 프로필을 구성하도록 허용하는 신뢰 정책을 구성합니다. AWS SSM을 사용하는 경우 AWS SSM이 하이브리드 노드 IAM 역할을 수입하고 하이브리드 노드 IAM 역할을 사용하여 하이브리드 활성화를 생성하도록 허용하는 신뢰 정책을 구성합니다. 필요한 권한을 가진 하이브리드 노드 IAM 역할을 생성하는 방법은 [the section called “자격 증명 준비”](#) 섹션을 참조하세요.

하이브리드 노드용 네트워킹 준비

이 주제에서는 Amazon EKS 클러스터를 생성하고 하이브리드 노드를 연결하기 전에 구성해야 하는 네트워킹 설정에 대한 개요를 제공합니다. 이 안내서에서는 [AWS Site-to-Site VPN](#), [AWS Direct Connect](#) 또는 자체 VPN 솔루션을 사용하여 하이브리드 네트워크 연결에 대한 사전 요구 사항을 충족했다고 가정합니다.



온프레미스 네트워킹 구성

최소 네트워크 요구 사항

최적의 경험을 위해 AWS는 AWS 리전의 하이브리드 노드 연결에 최소 100Mbps 및 최대 200ms의 왕복 지연 시간을 제공하는 안정적인 네트워크 연결을 권장합니다. 대역폭 및 지연 시간 요구 사항은 하이브리드 노드 수와 애플리케이션 이미지 크기, 애플리케이션 탄력성, 모니터링 및 로깅 구성, 다른 AWS 서비스에 저장된 데이터에 대한 애플리케이션 종속성과 같은 워크로드 특성에 따라 달라질 수 있습니다.

온프레미스 노드 및 포드 CIDR

하이브리드 노드에 사용할 노드 및 포드 CIDR과 해당 노드에서 실행되는 워크로드를 식별합니다. 노드 CIDR은 온프레미스 네트워크에서 할당되고 포드 CIDR은 CNI에 오버레이 네트워크를 사용하는 경우 컨테이너 네트워크 인터페이스(CNI)에서 할당됩니다. RemoteNodeNetwork 및 RemotePodNetwork 필드를 사용하여 EKS 클러스터를 생성할 때 온프레미스 노드 CIDR과 선택적으로 포드 CIDR을 입력으로 전달합니다.

온프레미스 노드 및 포드 CIDR 블록은 다음 요구 사항을 충족해야 합니다.

1. 다음 IPv4 RFC-1918 범위 10.0.0.0/8, 172.16.0.0/12 또는 192.168.0.0/16 중 하나에 있어야 합니다.
2. 서로, EKS 클러스터의 VPC CIDR 또는 Kubernetes 서비스 IPv4 CIDR과 중첩되지 않습니다.

CNI가 온프레미스 호스트를 떠날 때 포드 트래픽에 대해 네트워크 주소 변환(NAT)을 수행하는 경우, 하이브리드 노드가 워크로드에 대비할 수 있도록 포드 CIDR을 온프레미스 네트워크에 라우팅 가능한 것으로 설정하거나 원격 포드 네트워크로 EKS 클러스터를 구성할 필요가 없습니다. CNI가 온프레미스 호스트를 떠날 때 포드 트래픽에 NAT를 사용하지 않는 경우 온프레미스 네트워크에서 포드 CIDR을 라우팅 가능한 것으로 설정하고 하이브리드 노드가 워크로드에 대비할 수 있도록 원격 포드 네트워크로 EKS 클러스터를 구성해야 합니다.

Border Gateway Protocol(BGP), 정적 경로 또는 기타 사용자 지정 라우팅 솔루션을 포함하여 온프레미스 네트워크에서 포드 CIDR을 라우팅 가능하도록 하는 데 몇 가지 기법을 사용할 수 있습니다. BGP는 사용자 지정 또는 수동 라우팅 구성이 필요한 대체 솔루션보다 더 확장성이 뛰어나고 관리하기 쉽기 때문에 권장되는 솔루션입니다. AWS는 하이브리드 노드 포드 CIDR을 알리는 데 Cilium 및 Calico의 BGP 기능을 지원합니다. 자세한 내용은 [하이브리드 노드용 CNI 구성](#)을 참조하세요.

하이브리드 노드에서 웹후크를 실행하는 경우 포드 CIDR이 온프레미스 네트워크에서 라우팅 가능해야 하고 EKS 컨트롤 플레인도 하이브리드 노드에서 실행되는 웹후크와 직접 통신할 수 있도록 원격 포드 네트워크로 Amazon EKS 클러스터를 구성해야 합니다. 온프레미스 네트워크에서 포드 CIDR을 라우팅 가능하도록 설정할 수 없지만 웹후크를 실행해야 하는 경우 동일한 EKS 클러스터의 클라우드 노드에서 웹후크를 실행하는 것이 좋습니다. 클라우드 노드에서 웹후크를 실행하는 방법에 대한 자세한 내용은 [하이브리드 노드용 웹후크 구성](#)을 참조하세요.

하이브리드 노드 설치 및 업그레이드 중에 필요한 액세스

호스트에 하이브리드 노드 종속성을 설치하는 설치 프로세스 중에 다음 도메인에 액세스할 수 있어야 합니다. 이 프로세스는 운영 체제 이미지를 빌드할 때 한 번 수행하거나 런타임 시 각 호스트에서 수행할 수 있습니다. 여기에는 초기 설치와 하이브리드 노드의 Kubernetes 버전을 업그레이드할 때도 포함됩니다.

구성 요소	URL	프로토콜	Port
EKS 노드 아티팩트 (S3)	https://hybrid-assets.eks.amazonaws.com	HTTPS	443
EKS 서비스 엔드포인트	https://eks.region.amazonaws.com	HTTPS	443
ECR 서비스 엔드포인트	https://api.ecr.region.amazonaws.com	HTTPS	443
EKS ECR 엔드포인트	리전별 엔드포인트는 the section called “Amazon 이미지 레지스트리 보기” 섹션을 참조하세요.	HTTPS	443
SSM 바이너리 엔드포인트 ¹	https://amazon-ssm-region.s3.region.amazonaws.com	HTTPS	443
SSM 서비스 엔드포인트 ¹	https://ssm.region.amazonaws.com	HTTPS	443
IAM Anywhere 바이너리 엔드포인트 ²	https://rolesanywhere.amazonaws.com	HTTPS	443

구성 요소	URL	프로토콜	Port
IAM Anywhere 서비스 엔드포인트 ²	https://rolesanywhere. <i>region</i> .amazonaws.com	HTTPS	443

Note

¹ AWS SSM 엔드포인트에 대한 액세스는 온프레미스 IAM 자격 증명 공급자에 AWS SSM 하이브리드 활성화를 사용하는 경우에만 필요합니다.

² AWS IAM 엔드포인트에 대한 액세스는 온프레미스 IAM 자격 증명 공급자에 AWS IAM Roles Anywhere를 사용하는 경우에만 필요합니다.

지속적인 클러스터 작업에 필요한 액세스

지속적인 클러스터 작업을 위해서는 온프레미스 방화벽에 대한 다음 네트워크 액세스가 필요합니다.

Important

CNI 선택에 따라 CNI 포트에 대한 추가 네트워크 액세스 규칙을 구성해야 합니다. 자세한 내용은 [Cilium documentation](#) 및 [Calico documentation](#)를 참조하세요.

유형	프로토콜	Direction	Port	소스	대상	사용법
HTTPS	TCP	아웃바운드	443	원격 노드 CIDR	EKS 클러스터 IP ¹	kubelet에서 Kubernetes API 서버로
HTTPS	TCP	아웃바운드	443	원격 포트 CIDR	EKS 클러스터 IP ¹	포트에서 Kubernetes API 서버로

유형	프로토콜	Direction	Port	소스	대상	사용법
HTTPS	TCP	아웃바운드	443	원격 노드 CIDR	SSM 서비스 엔드포인트	5분마다 SSM 하 이브리드 활성화 자격 증명 새로 고침 및 SSM 하트 비트
HTTPS	TCP	아웃바운드	443	원격 노드 CIDR	IAM Anywhere 서비스 엔드포인트	IAM Roles Anywhere 자격 증명 새로 고침
HTTPS	TCP	아웃바운드	443	원격 포트 CIDR	STS 리전 엔드포인트	포트에서 STS 엔드포인트로, IRSA에만 필요
HTTPS	TCP	아웃바운드	443	원격 노드 CIDR	Amazon EKS 인증 서비스 엔드포인트	노드에서 Amazon EKS 인증 엔드포인트로, Amazon EKS Pod Identity에 만 필요
HTTPS	TCP	인바운드	10250	EKS 클러스터 IP ¹	원격 노드 CIDR	Kubernetes API 서버에서 kubelet 으로

유형	프로토콜	Direction	Port	소스	대상	사용법
HTTPS	TCP	인바운드	웹훅 포트	EKS 클러스터 IP ¹	원격 포트 CIDR	Kubernetes API 서버에서 웹훅으로
HTTPS	TCP, UDP	인바운드, 아웃바운드	53	원격 포트 CIDR	원격 포트 CIDR	포트와 CoreDNS 간. 클라우드에서 CoreDNS 복제본을 1개 이상 실행하는 경우 CoreDNS가 실행 중인 VPC에 대한 DNS 트래픽을 허용해야 합니다.
사용자 정의	사용자 정의	인바운드, 아웃바운드	앱 포트	원격 포트 CIDR	원격 포트 CIDR	포트 간

 Note

¹ EKS 클러스터의 IP입니다. Amazon EKS 탄력적 네트워크 인터페이스에 대한 다음 섹션을 참조하세요.

Amazon EKS 네트워크 인터페이스

Amazon EKS는 클러스터 생성 중에 전달하는 VPC의 서브넷에 네트워크 인터페이스를 연결하여 EKS 컨트롤 플레인과 VPC 간의 통신을 활성화합니다. Amazon EKS가 생성하는 네트워크 인터페이스

스는 Amazon EC2 콘솔에서 또는 AWS CLI를 사용하여 클러스터를 생성한 후에 찾을 수 있습니다. Kubernetes 버전 업그레이드와 같이 EKS 클러스터에 변경 사항이 적용되면 원래 네트워크 인터페이스가 삭제되고 새 네트워크 인터페이스가 생성됩니다. 클러스터 생성 중에 전달하는 서브넷에 대해 제한된 서브넷 크기를 사용하여 Amazon EKS 네트워크 인터페이스의 IP 범위를 제한할 수 있습니다. 이렇게 하면 알려진 제한된 IP 세트에 대한 인바운드/아웃바운드 연결을 허용하도록 온프레미스 방화벽을 쉽게 구성할 수 있습니다. 네트워크 인터페이스가 생성되는 서브넷을 제어하려면 클러스터를 생성하거나 클러스터를 생성한 후 서브넷을 업데이트할 때 지정하는 서브넷 수를 제한할 수 있습니다.

Amazon EKS에서 프로비저닝한 네트워크 인터페이스에는 Amazon EKS *your-cluster-name* 형식에 대한 설명이 있습니다. Amazon EKS가 프로비저닝하는 네트워크 인터페이스의 IP 주소를 찾는 데 사용할 수 있는 AWS CLI 명령은 아래 예제를 참조하세요. VPC_ID를 클러스터 생성 중에 전달하는 VPC의 ID로 바꿉니다.

```
aws ec2 describe-network-interfaces \
--query 'NetworkInterfaces[?(VpcId == VPC_ID && contains(Description,Amazon EKS))].PrivateIpAddress'
```

AWS VPC 및 서브넷 설정

Amazon EKS에 대한 기존 [VPC 및 서브넷 요구 사항](#)은 하이브리드 노드가 있는 클러스터에 적용됩니다. 또한 VPC CIDR은 온프레미스 노드 및 포드 CIDR과 중첩될 수 없습니다. 온프레미스 노드와 선택적으로 포드 CIDR에 대해 VPC 라우팅 테이블에서 경로를 구성해야 합니다. 이 경로는 일반적으로 가상 프라이빗 게이트웨이(VGW) 또는 전송 게이트웨이(TGW)인 하이브리드 네트워크 연결에 사용 중인 게이트웨이로 트래픽을 라우팅하도록 설정되어야 합니다. TGW 또는 VGW를 사용하여 VPC를 온프레미스 환경에 연결하는 경우 VPC에 대한 TGW 또는 VGW 연결을 생성해야 합니다. VPC는 DNS 호스트 이름과 DNS 확인을 모두 지원해야 합니다.

다음 단계에서는 AWS CLI를 사용합니다. AWS Management Console에서 또는 AWS CloudFormation, AWS CDK, Terraform과 같은 다른 인터페이스를 사용하여 이러한 리소스를 생성할 수도 있습니다.

1단계: VPC 생성

1. 다음 명령을 실행하여 VPC를 생성합니다. VPC_CIDR를 IPv4 RFC-1918(프라이빗) 또는 non-RFC-1918(퍼블릭) CIDR 범위(예: 10.0.0.0/16)로 바꿉니다. 참고: EKS 요구 사항인 DNS 확인은 기본적으로 VPC에 대해 활성화됩니다.

```
aws ec2 create-vpc --cidr-block VPC_CIDR
```

2. VPC의 DNS 호스트 이름을 활성화합니다. 참고로 DNS 확인은 기본적으로 VPC에 대해 활성화됩니다. VPC_ID를 이전 단계에서 생성한 VPC의 ID로 바꿉니다.

```
aws ec2 modify-vpc-attribute --vpc-id VPC_ID --enable-dns-hostnames
```

2단계: 서브넷 생성

서브넷을 2개 이상 생성합니다. Amazon EKS는 클러스터 네트워크 인터페이스에 이러한 서브넷을 사용합니다. 자세한 내용은 [Subnets requirements and considerations](#)을 참조하세요.

1. 다음 명령을 사용하여 AWS 리전의 가용 영역을 찾을 수 있습니다. us-west-2를 해당 리전으로 바꿉니다.

```
aws ec2 describe-availability-zones \
  --query 'AvailabilityZones[?(RegionName == us-west-2)].ZoneName'
```

2. 서브넷을 만듭니다. VPC_ID를 VPC의 ID로 바꿉니다. SUBNET_CIDR을 서브넷의 CIDR 블록(예: 10.0.1.0/24)으로 바꿉니다. AZ를 서브넷이 생성될 가용 영역(예: us-west-2a)으로 바꿉니다. 생성하는 서브넷은 2개 이상의 서로 다른 가용 영역에 있어야 합니다.

```
aws ec2 create-subnet \
  --vpc-id VPC_ID \
  --cidr-block SUBNET_CIDR \
  --availability-zone AZ
```

(선택 사항) 3단계: Amazon VPC Transit Gateway(TGW) 또는 AWS Direct Connect 가상 프라이빗 게이트웨이(VGW)에 VPC 연결

TGW 또는 VGW를 사용하는 경우 VPC를 TGW 또는 VGW에 연결합니다. 자세한 내용은 [Amazon VPC attachments in Amazon VPC Transit Gateways](#) 또는 [AWS Direct Connect virtual private gateway associations](#)를 참조하세요.

전송 게이트웨이

다음 명령을 실행하여 전송 게이트웨이를 연결합니다. VPC_ID를 VPC의 ID로 바꿉니다. SUBNET_ID1 및 SUBNET_ID2를 이전 단계에서 생성한 서브넷의 ID로 바꿉니다. TGW_ID를 TGW의 ID로 바꿉니다.

```
aws ec2 create-transit-gateway-vpc-attachment \
```

```
--vpc-id VPC_ID \
--subnet-ids SUBNET_ID1 SUBNET_ID2 \
--transit-gateway-id TGW_ID
```

가상 프라이빗 게이트웨이

다음 명령을 실행하여 전송 게이트웨이를 연결합니다. VPN_ID를 VGW의 ID로 바꿉니다. VPC_ID를 VPC의 ID로 바꿉니다.

```
aws ec2 attach-vpn-gateway \
  --vpn-gateway-id VPN_ID \
  --vpc-id VPC_ID
```

(선택 사항) 4단계: 라우팅 테이블 생성

VPC의 기본 라우팅 테이블을 수정하거나 사용자 지정 라우팅 테이블을 생성할 수 있습니다. 다음 단계에서는 온프레미스 노드 및 포드 CIDR에 대한 경로가 포함된 사용자 지정 라우팅 테이블을 생성합니다. 자세한 내용은 [Subnet route tables](#)을 참조하세요. VPC_ID를 VPC의 ID로 바꿉니다.

```
aws ec2 create-route-table --vpc-id VPC_ID
```

5단계: 온프레미스 노드 및 포드에 대한 경로 생성

각 온프레미스 원격 노드의 라우팅 테이블에서 경로를 생성합니다. VPC의 기본 라우팅 테이블을 수정하거나 이전 단계에서 생성한 사용자 지정 라우팅 테이블을 사용할 수 있습니다.

아래 예제에서는 온프레미스 노드 및 포드 CIDR에 대한 경로를 생성하는 방법을 보여줍니다. 이 예제에서는 전송 게이트웨이(TGW)를 사용하여 VPC를 온프레미스 환경에 연결합니다. 온프레미스 노드와 포드 CIDR이 여러 개인 경우 각 CIDR에 대해 단계를 반복합니다.

- 인터넷 게이트웨이 또는 가상 프라이빗 게이트웨이(VGW)를 사용하는 경우 --transit-gateway-id를 --gateway-id로 바꿉니다.
- RT_ID를 이전 단계에서 생성한 라우팅 테이블의 ID로 바꿉니다.
- REMOTE_NODE_CIDR을 하이브리드 노드에 사용할 CIDR 범위로 바꿉니다.
- REMOTE_POD_CIDR을 하이브리드 노드에서 실행되는 포드에 사용할 CIDR 범위로 바꿉니다. 포드 CIDR 범위는 온프레미스 오버레이 네트워크를 가장 일반적으로 사용하는 컨테이너 네트워킹 인터페이스(CNI) 구성에 해당합니다. 자세한 내용은 [the section called “CNI 구성”](#) 단원을 참조하십시오.

- TGW_ID를 TGW의 ID로 바꿉니다.

원격 노드 네트워크

```
aws ec2 create-route \
  --route-table-id RT_ID \
  --destination-cidr-block REMOTE_NODE_CIDR \
  --transit-gateway-id TGW_ID
```

원격 포드 네트워크

```
aws ec2 create-route \
  --route-table-id RT_ID \
  --destination-cidr-block REMOTE_POD_CIDR \
  --transit-gateway-id TGW_ID
```

(선택 사항) 6단계: 서브넷을 라우팅 테이블에 연결

이전 단계에서 사용자 지정 라우팅 테이블을 생성한 경우 이전 단계에서 생성한 각 서브넷을 사용자 지정 라우팅 테이블과 연결합니다. VPC 기본 라우팅 테이블을 수정하는 경우 서브넷이 VPC의 기본 라우팅 테이블과 자동으로 연결되며 이 단계를 건너뛸 수 있습니다.

이전 단계에서 생성한 각 서브넷에 대해 다음 명령을 실행합니다. RT_ID를 이전 단계에서 생성한 라우팅 테이블로 바꿉니다. SUBNET_ID를 서브넷의 ID로 바꿉니다.

```
aws ec2 associate-route-table --route-table-id RT_ID --subnet-id SUBNET_ID
```

클러스터 보안 그룹 고려 사항

지속적인 클러스터 작업을 위해서는 EKS 클러스터 보안 그룹에 대한 다음 액세스 권한이 필요합니다.

유형	프로토콜	Direction	Port	소스	대상	사용법
HTTPS	TCP	인바운드	443	원격 노드 CIDR	N/A	Kubelet에서 Kubernetes API 서버로

유형	프로토콜	Direction	Port	소스	대상	사용법
HTTPS	TCP	인바운드	443	원격 포트 CIDR	N/A	CNI가 포트 트래픽에 NAT를 사용하지 않을 때 K8s API 서버에 대한 액세스가 필요한 포트입니다.
HTTPS	TCP	아웃바운드	10250	N/A	원격 노드 CIDR	Kubernetes API 서버에서 Kubelet으로
HTTPS	TCP	아웃바운드	웹훅 포트	N/A	원격 포트 CIDR	Kubernetes API 서버에서 웹훅로(하이브리드 노드에서 웹훅크를 실행하는 경우)

인바운드 액세스 규칙으로 보안 그룹을 생성하려면 다음 명령을 실행합니다. Amazon EKS 클러스터를 생성할 때 이 보안 그룹을 전달해야 합니다. 기본적으로 아래 명령은 모든 아웃바운드 액세스를 허용하는 보안 그룹을 생성합니다. 위의 규칙만 포함하도록 아웃바운드 액세스를 제한할 수 있습니다. 아웃바운드 규칙을 제한하려는 경우 변경된 규칙을 프로덕션 클러스터에 적용하기 전에 모든 애플리케이션 및 포트 연결을 철저히 테스트하는 것이 좋습니다.

- 첫 번째 명령에서 SG_NAME을 보안 그룹의 이름으로 변경
- 첫 번째 명령에서 VPC_ID를 이전 단계에서 생성한 VPC의 ID로 변경
- 두 번째 명령에서 SG_ID를 첫 번째 명령에서 생성한 보안 그룹의 ID로 변경

- 두 번째 명령에서 REMOTE_NODE_CIDR 및 REMOTE_POD_CIDR을 하이브리드 노드 및 온프레미스 네트워크의 값으로 변경

```
aws ec2 create-security-group \
  --group-name SG_NAME \
  --description "security group for hybrid nodes" \
  --vpc-id VPC_ID
```

```
aws ec2 authorize-security-group-ingress \
  --group-id SG_ID \
  --ip-permissions '[{"IpProtocol": "tcp", "FromPort": 443, "ToPort": 443,
  "IpRanges": [{"CidrIp": "REMOTE_NODE_CIDR"}, {"CidrIp": "REMOTE_POD_CIDR"}]]'
```

하이브리드 노드용 운영 체제 준비

Bottlerocket, Ubuntu, Red Hat Enterprise Linux(RHEL) 및 Amazon Linux 2023(AL2023)은 하이브리드 노드의 노드 운영 체제로 사용하기 위해 지속적으로 검증되었습니다. AWS는 이러한 운영 체제와의 하이브리드 노드 통합을 지원하지만 운영 체제 자체는 지원하지 않습니다. AL2023은 Amazon EC2 외부에서 실행되는 경우 AWS 지원 플랜의 적용을 받지 않습니다. AL2023은 온프레미스 가상화 환경에서만 사용할 수 있습니다. 자세한 내용은 [Amazon Linux 2023 사용 설명서](#)를 참조하세요.

운영 체제 프로비저닝 및 관리는 사용자의 책임입니다. 하이브리드 노드를 처음 테스트하는 경우 이미 프로비저닝된 호스트에서 Amazon EKS Hybrid Nodes CLI(nodeadm)를 실행하는 것이 가장 쉽습니다. 프로덕션 배포의 경우 호스트 시작 시 호스트를 Amazon EKS 클러스터에 자동으로 조인하기 위해 systemd 서비스로 실행되도록 구성된 nodeadm을 운영 체제 이미지에 포함하는 것이 좋습니다. Bottlerocket을 vSphere의 노드 운영 체제로 사용하는 경우 Bottlerocket은 하이브리드 노드에 필요한 종속성을 이미 포함하고 호스트 시작 시 구성된 클러스터에 자동으로 연결되므로 nodeadm을 사용할 필요가 없습니다.

버전 호환성

아래 표는 하이브리드 노드의 노드 운영 체제로 사용할 수 있도록 호환되고 검증된 운영 체제 버전을 나타냅니다. 이 표에 포함되지 않은 다른 운영 체제 변형 또는 버전을 사용하는 경우 하이브리드 노드와 운영 체제 변형 또는 버전의 호환성은 AWS 지원에 포함되지 않습니다. 하이브리드 노드는 기본 인프라에 구매받지 않으며 x86 및 ARM 아키텍처를 지원합니다.

운영 체제	버전
Amazon Linux	Amazon Linux 2023(AL2023)
Bottlerocket	Kubernetes v1.28 이상을 실행하는 v1.37.0 이상의 VMware 변형
Ubuntu	Ubuntu 20.04, Ubuntu 22.04, Ubuntu 24.04
Red Hat Enterprise Linux	RHEL 8, RHEL 9

운영 체제 고려 사항

일반

- Amazon EKS Hybrid Nodes CLI(nodeadm)를 사용하여 하이브리드 노드 구성 요소 및 종속성의 설치 및 구성을 간소화할 수 있습니다. 운영 체제 이미지 빌드 파이프라인 중에 또는 각 온프레미스 호스트의 런타임에 `nodeadm install` 프로세스를 실행할 수 있습니다. `nodeadm`이 설치하는 구성 요소에 대한 자세한 내용은 [the section called “하이브리드 노드 nodeadm”](#) 섹션을 참조하세요.
- 온프레미스 환경에서 프록시를 사용하여 인터넷에 연결하는 경우 설치 및 업그레이드 프로세스에서 프록시를 사용하도록 패키지 관리자를 구성하기 위해 추가 운영 체제 구성이 필요합니다. 자세한 내용은 [the section called “프록시 구성”](#) 섹션을 참조하세요.

Bottlerocket

- Bottlerocket 노드를 연결하는 단계 및 도구는 다른 운영 체제의 단계와 다르며 [the section called “하이브리드 노드 연결”](#)의 단계 대신 [the section called “Bottlerocket을 실행하는 하이브리드 노드 연결”](#)에서 별도로 다룹니다.
- Bottlerocket의 단계에서는 하이브리드 노드 CLI 도구인 `nodeadm`을 사용하지 않습니다.
- Bottlerocket 버전 v1.37.0 이상의 VMware 변형만 EKS Hybrid Nodes에서 지원됩니다. Bottlerocket의 VMware 변형은 Kubernetes 버전 v1.28 이상에서 사용할 수 있습니다. [다른 Bottlerocket 변형](#)은 하이브리드 노드 운영 체제로 지원되지 않습니다. 참고: Bottlerocket의 VMware 변형은 x86_64 아키텍처에서만 사용할 수 있습니다.

Containerd

- Containerd는 표준 Kubernetes 컨테이너 런타임이며 하이브리드 노드와 모든 Amazon EKS 노드 컴퓨팅 유형에 대한 종속성입니다. Amazon EKS Hybrid Nodes CLI(nodeadm)는 `nodeadm install` 프로세스 중에 Containerd 설치를 시도합니다. `--containerd-source` 명령줄 옵션을 사용하여 `nodeadm install` 런타임에 Containerd 설치를 구성할 수 있습니다. 유효한 옵션은 `none`, `distro`, `docker`입니다. RHEL을 사용하는 경우 `distro`는 유효한 옵션이 아니며 Docker의 리포지토리에서 Containerd 빌드를 설치하도록 `nodeadm`을 구성하거나 Containerd를 수동으로 설치할 수 있습니다. AL2023 또는 Ubuntu를 사용하는 경우는 `nodeadm`은 기본적으로 운영 체제 배포에서 Containerd를 설치합니다. `nodeadm`이 Containerd를 설치하지 않게 하려면 `--containerd-source none` 옵션을 사용합니다.

Ubuntu

- Ubuntu 20.04를 사용하는 경우 AWS Systems Manager 하이브리드 활성화를 자격 증명 공급자로 사용해야 합니다. AWS IAM Roles Anywhere는 Ubuntu 20.04에서 지원되지 않습니다.
- Ubuntu 24.04를 사용하는 경우 Containerd 버전을 업데이트하거나 포드가 올바르게 종료되게 하는 수정 사항을 채택하도록 AppArmor 구성을 변경해야 할 수 있습니다. [Ubuntu #2065423](#)을 참조하세요. AppArmor 프로필에 변경 사항을 적용하려면 재부팅해야 합니다. 최신 버전의 Ubuntu 24.04에는 패키지 관리자에 수정 사항이 포함되어 업데이트된 Containerd 버전이 있습니다(Containerd 버전 1.7.19 이상).

RHEL

- RHEL 8을 사용하는 경우 AWS Systems Manager 하이브리드 활성화를 자격 증명 공급자로 사용해야 합니다. AWS IAM Roles Anywhere는 RHEL 8에서 지원되지 않습니다.

ARM

- ARM 하드웨어를 사용하는 경우 EKS kube-proxy 추가 기능 버전 1.31 이상을 실행하려면 Cryptography Extension(ARMv8.2+crypto)이 포함된 ARMv8.2 호환 프로세서가 필요합니다. Raspberry Pi 5 이전의 모든 Raspberry Pi 시스템과 Cortex-A72 기반 프로세서는 이 요구 사항을 충족하지 않습니다. 해결 방법으로 2026년 7월에 확장 지원이 종료될 때까지 EKS kube-proxy 추가 기능 1.30 버전을 계속 사용하거나, [the section called “Amazon EKS Kubernetes 릴리스 일정”](#) 섹션을 참조하거나, 업스트림의 사용자 지정 kube-proxy 이미지를 사용할 수 있습니다.
- kube-proxy 로그의 다음 오류 메시지는 이 비호환성을 나타냅니다.

Fatal glibc error: This version of Amazon Linux requires a newer ARM64 processor compliant with at least ARM architecture 8.2-a with Cryptographic extensions. On EC2 this is Graviton 2 or later.

운영 체제 이미지 구축

Amazon EKS가 제공하는 [Packer 템플릿 예제](#)를 사용하여 호스트 시작 시 실행되도록 구성하고 nodeadm을 포함하는 운영 체제 이미지를 생성할 수 있습니다. 이 프로세스는 각 호스트에서 하이브리드 노드 종속성을 개별적으로 가져오지 않도록 하고 하이브리드 노드 부트스트랩 프로세스를 자동화하기 위해 권장됩니다. Packer 템플릿 예제를 Ubuntu 22.04, Ubuntu 24.04, RHEL 8 또는 RHEL 9 ISO 이미지와 함께 사용하고 OVA, Qcow2 또는 원시 형식으로 이미지를 출력할 수 있습니다.

사전 조건

Packer 템플릿 예제를 사용하기 전에 Packer를 실행 중인 시스템에 다음이 설치되어 있어야 합니다.

- Packer 버전 1.11.0 이상. Packer 설치에 대한 지침은 Packer 설명서에서 [Install Packer](#)를 참조하세요.
- OVA를 빌드하는 경우 VMware vSphere 플러그인 1.4.0 이상
- Qcow2를 빌드하거나 원시 이미지인 경우 QEMU 플러그인 버전 1.x

환경 변수 설정

Packer 빌드를 실행하기 전에 Packer를 실행 중인 시스템에서 다음 환경 변수를 설정합니다.

일반

모든 운영 체제 및 출력 형식으로 이미지를 빌드하려면 다음 환경 변수를 설정해야 합니다.

환경 변수	유형	설명
PKR_SSH_PASSWORD	String	Packer는 ssh_username 및 ssh_password 변수를 사용하여 프로비저닝 시 생성된 시스템으로 SSH를 적용합니다. 이는 각 OS의 시작 또는 사용자 데이터 파일 내에서 초기 사용자를 생성하는 데 사용

환경 변수	유형	설명
		되는 암호와 일치해야 합니다. 기본값은 OS에 따라 '빌더' 또는 'ubuntu'로 설정됩니다. 암호를 설정할 때 일치하는 해당 ks.cfg 또는 user-data 파일 내에서 암호를 변경해야 합니다.
ISO_URL	String	사용할 ISO의 URL입니다. 서버에서 다운로드할 웹 링크 또는 로컬 파일의 절대 경로일 수 있습니다.
ISO_CHECKSUM	String	제공된 ISO에 대한 관련 체크섬입니다.
CREDENTIAL_PROVIDER	String	하이브리드 노드의 자격 증명 공급자입니다. 유효한 값은 SSM 하이브리드 활성화의 경우 ssm(기본값), IAM Roles Anywhere의 경우 iam입니다.
K8S_VERSION	String	하이브리드 노드용 Kubernetes 버전(예: 1.31)입니다. 지원되는 Kubernetes 버전은 the section called “Kubernetes 버전” 섹션을 참조하세요.
NODEADM_ARCH	String	nodeadm install용 아키텍처입니다. amd 또는 arm을 선택합니다.

RHEL

RHEL을 사용하는 경우 다음 환경 변수를 설정해야 합니다.

환경 변수	유형	설명
RH_USERNAME	String	RHEL 구독 관리자 사용자 이름
RH_PASSWORD	String	RHEL 구독 관리자 암호
RHEL_VERSION	String	사용 중인 RHEL iso 버전입니다. 유효한 값은 8 또는 9입니다.

Ubuntu

Ubuntu별 환경 변수는 필요하지 않습니다.

vSphere

VMware vSphere OVA를 빌드하는 경우 다음 환경 변수를 설정해야 합니다.

환경 변수	유형	설명
VSPHERE_SERVER	String	vSphere 서버 주소
VSPHERE_USER	String	vSphere 사용자 이름
VSPHERE_PASSWORD	String	vSphere 암호
VSPHERE_DATACENTER	String	vSphere 데이터 센터 이름
VSPHERE_CLUSTER	String	vSphere 클러스터 이름
VSPHERE_DATASTORE	String	vSphere 데이터 저장소 이름
VSPHERE_NETWORK	String	vSphere 네트워크 이름
VSPHERE_OUTPUT_FOLDER	String	템플릿의 vSphere 출력 폴더

QEMU

환경 변수	유형	설명
PACKER_OUTPUT_FORMAT	String	QEMU 빌더의 출력 형식입니다. 유효 값은 qcow2 및 raw입니다.

템플릿 확인

빌드를 실행하기 전에 환경 변수를 설정한 후 다음 명령을 사용하여 템플릿을 확인합니다. 템플릿에 다른 이름을 사용하는 경우 `template.pkr.hcl`을 바꿉니다.

```
packer validate template.pkr.hcl
```

이미지 빌드

다음 명령을 사용하여 이미지를 빌드하고 `-only` 플래그를 사용하여 이미지의 대상 및 운영 체제를 지정합니다. 템플릿에 다른 이름을 사용하는 경우 `template.pkr.hcl`을 바꿉니다.

vSphere OVA

Note

vSphere와 함께 RHEL을 사용하는 경우 시작 파일을 OEMDRV 이미지로 변환하고 부팅할 ISO로 전달해야 합니다. 자세한 내용은 EKS 하이브리드 노드 GitHub 리포지토리의 [Packer Readme](#)를 참조하세요.

Ubuntu 22.04 OVA

```
packer build -only=general-build.vsphere-iso.ubuntu22 template.pkr.hcl
```

Ubuntu 24.04 OVA

```
packer build -only=general-build.vsphere-iso.ubuntu24 template.pkr.hcl
```

RHEL 8 OVA

```
packer build -only=general-build.vsphere-iso.rhel8 template.pkr.hcl
```

RHEL 9 OVA

```
packer build -only=general-build.vsphere-iso.rhel9 template.pkr.hcl
```

QEMU

Note

빌더 호스트와 일치하지 않는 특정 호스트 CPU의 이미지를 빌드하는 경우 호스트 CPU와 일치하는 이름에 대한 [QEMU](#) 설명서를 참조하고 다음 명령을 실행할 때 `-cpu` 플래그를 호스트 CPU 이름과 함께 사용합니다.

Ubuntu 22.04 Qcow2 / 원시

```
packer build -only=general-build.qemu.ubuntu22 template.pkr.hcl
```

Ubuntu 24.04 Qcow2 / 원시

```
packer build -only=general-build.qemu.ubuntu24 template.pkr.hcl
```

RHEL 8 Qcow2 / 원시

```
packer build -only=general-build.qemu.rhel8 template.pkr.hcl
```

RHEL 9 Qcow2 / 원시

```
packer build -only=general-build.qemu.rhel9 template.pkr.hcl
```

사용자 데이터를 통해 nodeadm 구성 전달

Cloud-init을 통해 사용자 데이터에서 nodeadm에 대한 구성을 전달하여 호스트 시작 시 하이브리드 노드를 구성하고 EKS 클러스터에 자동으로 연결할 수 있습니다. 다음은 VMware vSphere를 하이브리드 노드의 인프라로 사용할 때 이를 수행하는 방법의 예입니다.

1. GitHub의 [govc readme](#) 지침에 따라 govc CLI를 설치합니다.

2. 이전 섹션에서 Packer 빌드를 실행하고 템플릿을 프로비저닝한 후 다음을 사용하여 템플릿을 복제하고 여러 노드를 생성할 수 있습니다. 하이브리드 노드에 사용하기 위해 생성하는 새 VM마다 템플릿을 복제해야 합니다. 아래 명령에서 다음 변수를 해당 환경의 값으로 바꿉니다. 아래 명령의 VM_NAME은 metadata.yaml 파일을 통해 VM의 이름을 주입할 때 NODE_NAME으로 사용됩니다.

```
govc vm.clone -vm "/PATH/T0/TEMPLATE" -ds="YOUR_DATASTORE" \
-on=false -template=false -folder=/FOLDER/T0/SAVE/VM "VM_NAME"
```

3. 새로운 각 VM에 대한 템플릿을 복제한 후 VM에 대한 userdata.yaml 및 metadata.yaml을 생성합니다. VM은 동일한 userdata.yaml 및 metadata.yaml을 공유할 수 있으며 아래 단계에서 VM별로 이를 채웁니다. nodeadm 구성은 userdata.yaml의 write_files 섹션에서 생성 및 정의됩니다. 아래 예제에서는 AWS SSM 하이브리드 활성화를 하이브리드 노드의 온프레미스 자격 증명 공급자로 사용합니다. nodeadm 구성에 대한 자세한 내용은 [the section called “하이브리드 노드 nodeadm”](#) 섹션을 참조하세요.

userdata.yaml:

```
#cloud-config
users:
  - name: # username for login. Use 'builder' for RHEL or 'ubuntu' for Ubuntu.
    passwd: # password to login. Default is 'builder' for RHEL.
    groups: [adm, cdrom, dip, plugdev, lxd, sudo]
    lock-passwd: false
    sudo: ALL=(ALL) NOPASSWD:ALL
    shell: /bin/bash

write_files:
  - path: /usr/local/bin/nodeConfig.yaml
    permissions: '0644'
    content: |
      apiVersion: node.eks.aws/v1alpha1
      kind: NodeConfig
      spec:
        cluster:
          name: # Cluster Name
          region: # AWS region
        hybrid:
          ssm:
            activationCode: # Your ssm activation code
            activationId: # Your ssm activation id

runcmd:
```

```
- /usr/local/bin/nodeadm init -c file:///usr/local/bin/nodeConfig.yaml >> /var/log/
nodeadm-init.log 2>&1
```

metadata.yaml:

환경에 대한 metadata.yaml을 생성합니다. "\$NODE_NAME" 변수 형식은 후속 단계의 값으로 채워지므로 파일에 그대로 둡니다.

```
instance-id: "$NODE_NAME"
local-hostname: "$NODE_NAME"
network:
  version: 2
  ethernet:
    nics:
      match:
        name: ens*
      dhcp4: yes
```

4. 다음 명령을 사용하여 userdata.yaml 및 metadata.yaml 파일을 gzip+base64 문자열로 추가합니다. 생성하는 각 VM에 대해 다음 명령을 실행해야 합니다. VM_NAME을 업데이트 중인 VM의 이름으로 바꿉니다.

```
export NODE_NAME="VM_NAME"
export USER_DATA=$(gzip -c9 <userdata.yaml | base64)

govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
  guestinfo.userdata="${USER_DATA}"
govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
  guestinfo.userdata.encoding=gzip+base64

envsubst '$NODE_NAME' < metadata.yaml > metadata.yaml.tmp
export METADATA=$(gzip -c9 <metadata.yaml.tmp | base64)

govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
  guestinfo.metadata="${METADATA}"
govc vm.change -dc="YOUR_DATASTORE" -vm "$NODE_NAME" -e
  guestinfo.metadata.encoding=gzip+base64
```

5. 구성한 EKS 클러스터에 자동으로 연결되어야 하는 새 VM의 전원을 켭니다.

```
govc vm.power -on "${NODE_NAME}"
```

하이브리드 노드에 대한 자격 증명 준비

Amazon EKS Hybrid Nodes는 AWS SSM 하이브리드 활성화 또는 AWS IAM Roles Anywhere에서 프로비저닝한 임시 IAM 자격 증명을 사용하여 Amazon EKS 클러스터로 인증합니다. Amazon EKS Hybrid Nodes CLI(nodeadm)와 함께 AWS SSM 하이브리드 활성화 또는 AWS IAM Roles Anywhere를 사용해야 합니다. AWS SSM 하이브리드 활성화와 AWS IAM Roles Anywhere를 둘 다 사용해서는 안 됩니다. 인증 기관(CA)이 있는 기존 퍼블릭 키 인프라(PKI) 및 온프레미스 환경에 대한 인증서가 없는 경우 AWS SSM 하이브리드 활성화를 사용하는 것이 좋습니다. 온프레미스에 기존 PKI 및 인증서가 있는 경우 AWS IAM Roles Anywhere를 사용합니다.

하이브리드 노드 IAM 역할

하이브리드 노드를 Amazon EKS 클러스터에 연결하려면 먼저 하이브리드 노드 자격 증명에 대해 AWS SSM 하이브리드 활성화 또는 AWS IAM Roles Anywhere와 함께 사용할 IAM 역할을 생성해야 합니다. 클러스터를 생성한 후 이 역할을 Amazon EKS 액세스 항목 또는 aws-auth ConfigMap 항목과 함께 사용하여 IAM 역할을 Kubernetes 역할 기반 액세스 제어(RBAC)에 매핑합니다. 하이브리드 노드 IAM 역할을 Kubernetes RBAC와 연결하는 방법에 대한 자세한 내용은 [the section called “클러스터 액세스 준비”](#) 섹션을 참조하세요.

하이브리드 노드 IAM 역할에는 다음과 같은 권한이 있어야 합니다.

- nodeadm이 eks:DescribeCluster 작업을 사용하여 하이브리드 노드를 클러스터에 연결하는 데 사용되는 클러스터에 대한 정보를 수집할 수 있는 권한입니다. eks:DescribeCluster 작업을 활성화하지 않으면 nodeadm init를 실행할 때 nodeadm에 전달하는 노드 구성에서 Kubernetes API 엔드포인트, 클러스터 CA 번들, 서비스 IPv4 CIDR을 전달해야 합니다.
- [AmazonEC2ContainerRegistryPullOnly](#) 정책의 정의에 따라 kubelet이 Amazon Elastic Container Registry(Amazon ECR)의 컨테이너 이미지를 사용할 수 있는 권한입니다.
- AWS SSM을 사용하는 경우 [aws-managed-policy/latest/reference/AmazonSSMManagedInstanceCore.html](#) 정책에 정의된 대로 nodeadm init가 AWS SSM 하이브리드 활성화를 사용할 수 있는 권한입니다.
- AWS SSM을 사용하는 경우 nodeadm uninstall에서 인스턴스 등록을 취소하기 위해 ssm:DeregisterManagedInstance 작업과 ssm:DescribeInstanceInformation 작업을 사용할 수 있는 권한입니다.
- (선택사항) Amazon EKS Pod Identity 에이전트에서 eks-auth:AssumeRoleForPodIdentity 작업을 사용하여 포드에 대한 보안 인증 정보를 검색할 수 있는 권한.

AWS SSM 하이브리드 활성화 설정

AWS SSM 하이브리드 활성화를 설정하기 전에 하이브리드 노드 IAM 역할을 생성하고 구성해야 합니다. 자세한 내용은 [the section called “하이브리드 노드 IAM 역할 생성”](#) 섹션을 참조하세요. AWS Systems Manager 사용 설명서의 [하이브리드 활성화를 생성하여 Systems Manager에 노드 등록](#) 지침에 따라 하이브리드 노드에 대한 AWS SSM 하이브리드 활성화를 생성합니다. 수신한 활성화 코드와 ID는 호스트를 Amazon EKS 클러스터에 하이브리드 노드로 등록할 때 `nodeadm`과 함께 사용됩니다. 하이브리드 노드용 Amazon EKS 클러스터를 생성하고 준비한 후 나중에 이 단계로 돌아갈 수 있습니다.

Important

활성화를 생성한 방법에 따라, Systems Manager는 활성화 코드 및 ID를 콘솔이나 명령 창에 즉시 반환합니다. 이 정보를 복사하여 안전한 장소에 보관하십시오. 콘솔에서 벗어나거나 명령 창을 닫으면 이 정보가 손실될 수 있습니다. 이 정보가 손실하면 새로운 정품 인증을 생성해야 합니다.

기본적으로 AWS SSM 하이브리드 활성화는 24시간 동안 활성화됩니다. 또는 `2024-08-01T00:00:00`과 같은 타임스탬프 형식으로 하이브리드 활성화를 생성할 때 --
`expiration-date`를 지정할 수 있습니다. AWS SSM을 자격 증명 공급자로 사용하는 경우 하이브리드 노드의 노드 이름은 구성할 수 없으며 AWS SSM에서 자동으로 생성됩니다. AWS Systems Manager 콘솔의 Fleet Manager에서 AWS SSM 관리형 인스턴스를 보고 관리할 수 있습니다. 추가 비용 없이 AWS 리전별로 계정당 최대 1,000개의 표준 [하이브리드 활성화 노드](#)를 등록할 수 있습니다. 그러나 1,000개 이상의 하이브리드 노드를 등록하려면 고급 인스턴스 티어를 활성화해야 합니다. [Amazon EKS Hybrid Nodes 요금](#)에 포함되지 않은 고급 인스턴스 티어를 사용하는 경우 요금이 부과됩니다. 자세한 내용은 [AWS Systems Manager 요금](#)을 참조하세요.

하이브리드 노드 IAM 역할을 사용하여 AWS SSM 하이브리드 활성화를 생성하는 방법은 아래 예제를 참조하세요. 하이브리드 노드 자격 증명에 AWS SSM 하이브리드 활성화를 사용하는 경우 하이브리드 노드의 이름은 `mi-012345678abcdefgh` 형식을 가지며 AWS SSM에서 프로비저닝한 임시 자격 증명은 1시간 동안 유효합니다. AWS SSM을 자격 증명 공급자로 사용하는 경우 노드 이름 또는 자격 증명 기간을 변경할 수 없습니다. 임시 자격 증명은 AWS SSM에 의해 자동으로 교체되며 교체는 노드 또는 애플리케이션의 상태에 영향을 주지 않습니다.

EKS 클러스터당 하나의 AWS SSM 하이브리드 활성화를 사용하여 AWS SSM 하이브리드 활성화와 연결된 인스턴스만 등록 취소할 수 있도록 하이브리드 노드 IAM 역할의 AWS SSM `ssm:DeregisterManagedInstance` 권한 범위를 조정하는 것이 좋습니다. 이 페이지의 예제에서

는 EKS 클러스터 ARN이 있는 태그가 사용되며, 이 태그를 사용하여 AWS SSM 하이브리드 활성화를 EKS 클러스터에 매핑할 수 있습니다. 또는 권한 경계 및 요구 사항에 따라 AWS SSM 권한을 조정하는 기본 태그와 방법을 사용할 수 있습니다. 아래 명령의 REGISTRATION_LIMIT 옵션은 AWS SSM 하이브리드 활성화를 사용할 수 있는 시스템 수를 제한하는 데 사용되는 정수입니다(예: 10).

```
aws ssm create-activation \
  --region AWS_REGION \
  --default-instance-name eks-hybrid-nodes \
  --description "Activation for EKS hybrid nodes" \
  --iam-role AmazonEKSHybridNodesRole \
  --tags Key=EKSclusterARN,Value=arn:aws:eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/CLUSTER_NAME \
  --registration-limit REGISTRATION_LIMIT
```

AWS SSM 하이브리드 활성화에 사용할 수 있는 구성 설정에 대한 자세한 내용은 [하이브리드 활성화를 생성하여 Systems Manager에 노드 등록](#)의 지침을 검토하세요.

AWS IAM Roles Anywhere 설정

IAM Roles Anywhere 사용 설명서의 [IAM Roles Anywhere 시작하기](#)의 지침에 따라 하이브리드 노드 IAM 역할에 대한 임시 IAM 자격 증명에 사용할 트러스트 앵커 및 프로필을 설정합니다. 프로필을 생성할 때 역할을 추가하지 않고도 프로필을 생성할 수 있습니다. 이 프로필을 생성하고, 다음 단계로 돌아가서 하이브리드 노드 IAM 역할을 생성한 다음 생성한 후 프로필에 역할을 추가할 수 있습니다. 또는 이 페이지의 뒷부분에 있는 AWS CloudFormation 단계를 사용하여 하이브리드 노드에 대한 IAM Roles Anywhere 설정을 완료할 수 있습니다.

하이브리드 노드 IAM 역할을 프로필에 추가할 때 AWS IAM Roles Anywhere 콘솔의 프로필 편집 페이지 하단의 사용자 지정 역할 세션 이름 패널에서 사용자 지정 역할 세션 이름 수락을 선택합니다. 이는 CreateProfile API의 [acceptRoleSessionName](#) 필드에 해당합니다. 이렇게 하면 부트스트랩 프로세스 중에 nodeadm으로 전달하는 구성에서 하이브리드 노드의 사용자 지정 노드 이름을 제공할 수 있습니다. nodeadm init 프로세스 중에 사용자 지정 노드 이름을 전달해야 합니다. 프로필을 생성한 후 사용자 지정 역할 세션 이름을 수락하도록 프로필을 업데이트할 수 있습니다.

AWS IAM Roles Anywhere 프로필의 [durationSeconds](#) 필드를 통해 AWS IAM Roles Anywhere로 자격 증명 유효 기간을 구성할 수 있습니다. 기본 지속 시간은 1시간이고 최대 12시간입니다. 하이브리드 노드 IAM 역할의 MaxSessionDuration 설정은 AWS IAM Roles Anywhere 프로필의 durationSeconds 설정보다 커야 합니다. MaxSessionDuration에 대한 자세한 내용은 [UpdateRole API documentation](#)를 참조하세요.

인증 기관(CA)에서 생성하는 시스템별 인증서와 키는 각 하이브리드 노드의 `/etc/iam/pki` 디렉터리에 배치하고 인증서는 `server.pem`, 키는 `server.key`의 파일 이름으로 저장해야 합니다.

하이브리드 노드 IAM 역할 생성

이 섹션의 단계를 실행하려면 AWS 콘솔 또는 AWS CLI를 사용하는 IAM 보안 주체에 다음 권한이 있어야 합니다.

- `iam:CreatePolicy`
- `iam:CreateRole`
- `iam:AttachRolePolicy`
- AWS IAM Roles Anywhere를 사용하는 경우
 - `rolesanywhere:CreateTrustAnchor`
 - `rolesanywhere:CreateProfile`
 - `iam:PassRole`

AWS CloudFormation

아직 하지 않은 경우 AWS CLI를 설치하고 구성합니다. [최신 버전의 AWS CLI 설치 또는 업데이트](#)를 참조하세요.

AWS SSM 하이브리드 활성화 단계

CloudFormation 스택은 위에 설명된 권한을 사용하여 하이브리드 노드 IAM 역할을 생성합니다. CloudFormation 템플릿은 AWS SSM 하이브리드 활성화를 생성하지 않습니다.

1. 하이브리드 노드용 AWS SSM CloudFormation 템플릿을 다운로드합니다.

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/hybrid-ssm-cfn.yaml'
```

2. 다음과 같은 옵션으로 `cfn-ssm-parameters.json`을 생성합니다.

- a. `ROLE_NAME`을 하이브리드 노드 IAM 역할의 이름으로 바꿉니다. 기본적으로 이름을 지정하지 않으면 CloudFormation 템플릿은 `AmazonEKSHybridNodesRole`을 생성하는 역할의 이름으로 사용합니다.
- b. `TAG_KEY`을 `AWSSSM` 하이브리드 활성화를 생성할 때 사용한 AWS SSM 리소스 태그 키로 바꿉니다. 태그 키와 태그 값의 조합은 하이브리드 노드 IAM 역할이 AWS SSM 하이브리드 활성화와 연결된 AWS SSM 관리형 인스턴스의 등록을 취소할 수 있도록 허용하는

ssm:DeregisterManagedInstance의 조건에서 사용됩니다. CloudFormation 템플릿에서 TAG_KEY의 기본값은 EKSClusterARN입니다.

- c. TAG_VALUE를 AWS SSM 하이브리드 활성화를 생성할 때 사용한 AWS SSM 리소스 태그 값으로 바꿉니다. 태그 키와 태그 값의 조합은 하이브리드 노드 IAM 역할이 AWS SSM 하이브리드 활성화와 연결된 AWS SSM 관리형 인스턴스의 등록을 취소할 수 있도록 허용하는 ssm:DeregisterManagedInstance의 조건에서 사용됩니다. TAG_KEY의 기본값인 EKSClusterARN을 사용하는 경우 EKS 클러스터 ARN을 TAG_VALUE로 전달합니다. EKS 클러스터 ARN의 형식은 `arn:aws:eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/CLUSTER_NAME`입니다.

```
{
  "Parameters": {
    "RoleName": "ROLE_NAME",
    "SSMDeregisterConditionTagKey": "TAG_KEY",
    "SSMDeregisterConditionTagValue": "TAG_VALUE"
  }
}
```

3. CloudFormation 스택 배포 STACK_NAME을 CloudFormation 스택의 이름으로 바꿉니다.

```
aws cloudformation deploy \
  --stack-name STACK_NAME \
  --template-file hybrid-ssm-cfn.yaml \
  --parameter-overrides file://cfn-ssm-parameters.json \
  --capabilities CAPABILITY_NAMED_IAM
```

AWS IAM Roles Anywhere의 단계

CloudFormation 스택은 사용자가 구성한 인증 기관(CA)을 사용하여 AWS IAM Roles Anywhere 트러스트 앵커를 생성하고, AWS IAM Roles Anywhere 프로필을 생성하고, 앞서 설명한 권한을 사용하여 하이브리드 노드 IAM 역할을 생성합니다.

1. 인증 기관(CA)을 설정하려면

- a. AWS 프라이빗 CA 리소스를 사용하려면 [AWS 프라이빗 인증 기관 콘솔](#)을 엽니다. [AWS 프라이빗 CA 사용 설명서](#)의 지침을 따릅니다.
- b. 외부 CA를 사용하려면 CA에서 제공하는 지침을 따릅니다. 이후 단계에서 인증서 본문을 제공합니다.
- c. 퍼블릭 CA에서 발급된 인증서는 트러스트 앵커로 사용할 수 없습니다.

2. 하이브리드 노드용 AWS IAM Roles Anywhere CloudFormation 템플릿을 다운로드합니다.

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/hybrid-ira-cfn.yaml'
```

3. 다음과 같은 옵션으로 cfn-iamra-parameters.json을 생성합니다.

- a. `ROLE_NAME`을 하이브리드 노드 IAM 역할의 이름으로 바꿉니다. 기본적으로 이름을 지정하지 않으면 CloudFormation 템플릿은 `AmazonEKSHybridNodesRole`을 생성하는 역할의 이름으로 사용합니다.
- b. `CERT_ATTRIBUTE`를 호스트를 고유하게 식별하는 시스템당 인증서 속성으로 바꿉니다. 사용하는 인증서 속성은 클러스터에 하이브리드 노드를 연결할 때 `nodeadm` 구성에 사용하는 `nodeName`과 일치해야 합니다. 자세한 내용은 [the section called “하이브리드 노드 nodeadm”](#)을 참조하세요. 기본적으로 CloudFormation 템플릿은 `${aws:PrincipalTag/x509Subject/CN}`을 `CERT_ATTRIBUTE`로 사용하며, 이는 시스템별 인증서의 CN 필드에 해당합니다. 또는 `${aws:PrincipalTag/x509SAN/Name/CN}`을 `CERT_ATTRIBUTE`로 전달할 수 있습니다.
- c. `CA_CERT_BODY`를 줄 바꿈 없이 CA의 인증서 본문으로 바꿉니다. `CA_CERT_BODY`는 PEM(Privacy Enhanced Mail) 형식이어야 합니다. PEM 형식의 CA 인증서가 있는 경우 CA 인증서 본문을 `cfn-iamra-parameters.json` 파일에 배치하기 전에 줄 바꿈과 `BEGIN CERTIFICATE` 및 `END CERTIFICATE` 줄을 제거합니다.

```
{
  "Parameters": {
    "RoleName": "ROLE_NAME",
    "CertAttributeTrustPolicy": "CERT_ATTRIBUTE",
    "CABundleCert": "CA_CERT_BODY"
  }
}
```

4. CloudFormation 템플릿을 배포합니다. `STACK_NAME`을 CloudFormation 스택의 이름으로 바꿉니다.

```
aws cloudformation deploy \
  --stack-name STACK_NAME \
  --template-file hybrid-ira-cfn.yaml \
  --parameter-overrides file://cfn-iamra-parameters.json
  --capabilities CAPABILITY_NAMED_IAM
```

AWS CLI

아직 하지 않은 경우 AWS CLI를 설치하고 구성합니다. [최신 버전의 AWS CLI 설치 또는 업데이트](#)를 참조하세요.

EKS 설명 클러스터 정책 생성

1. 다음 콘텐츠를 가진 eks-describe-cluster-policy.json이라는 파일을 생성합니다:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks:DescribeCluster"
      ],
      "Resource": "*"
    }
  ]
}
```

2. 다음 명령을 사용하여 정책을 생성합니다.

```
aws iam create-policy \
  --policy-name EKSDescribeClusterPolicy \
  --policy-document file://eks-describe-cluster-policy.json
```

AWS SSM 하이브리드 활성화 단계

1. 다음 콘텐츠를 가진 eks-hybrid-ssm-policy.json이라는 파일을 생성합니다: 이 정책은 ssm:DescribeInstanceInformation 및 ssm:DeregisterManagedInstance 두 작업에 대한 권한을 부여합니다. 이 정책은 신뢰 정책에 지정한 리소스 태그를 기반으로 AWS SSM 하이브리드 활성화와 연결된 AWS SSM 관리형 인스턴스에 대한 ssm:DeregisterManagedInstance 권한을 제한합니다.

- a. AWS_REGION을 AWS SSM 하이브리드 활성화를 위한 AWS 리전으로 바꿉니다.
- b. AWS_ACCOUNT_ID를 AWS 계정 ID로 바꿉니다.
- c. TAG_KEY을 AWSSSM 하이브리드 활성화를 생성할 때 사용한 AWS SSM 리소스 태그 키로 바꿉니다. 태그 키와 태그 값의 조합은 하이브리드 노드 IAM 역할이 AWS SSM 하이브리드 활성화와 연결된 AWS SSM 관리형 인스턴스의 등록을 취소할 수 있도록 허용하는

ssm:DeregisterManagedInstance의 조건에서 사용됩니다. CloudFormation 템플릿에서 TAG_KEY의 기본값은 EKSClusterARN입니다.

- d. TAG_VALUE를 AWS SSM 하이브리드 활성화를 생성할 때 사용한 AWS SSM 리소스 태그 값으로 바꿉니다. 태그 키와 태그 값의 조합은 하이브리드 노드 IAM 역할이 AWS SSM 하이브리드 활성화와 연결된 AWS SSM 관리형 인스턴스의 등록을 취소할 수 있도록 허용하는 ssm:DeregisterManagedInstance의 조건에서 사용됩니다. TAG_KEY의 기본값인 EKSClusterARN을 사용하는 경우 EKS 클러스터 ARN을 TAG_VALUE로 전달합니다. EKS 클러스터 ARN의 형식은 arn:aws:eks:AWS_REGION:AWS_ACCOUNT_ID:cluster/CLUSTER_NAME입니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ssm:DescribeInstanceInformation",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "ssm:DeregisterManagedInstance",
      "Resource": "arn:aws:ssm:AWS_REGION:AWS_ACCOUNT_ID:managed-instance/*",
      "Condition": {
        "StringEquals": {
          "ssm:resourceTag/TAG_KEY": "TAG_VALUE"
        }
      }
    }
  ]
}
```

2. 다음 명령을 사용하여 정책을 생성합니다.

```
aws iam create-policy \
  --policy-name EKSHybridSSMPolicy \
  --policy-document file://eks-hybrid-ssm-policy.json
```

3. eks-hybrid-ssm-trust.json이라는 이름의 파일을 만듭니다. AWS_REGION을 AWS SSM 하이브리드 활성화의 AWS 리전으로 바꾸고 AWS_ACCOUNT_ID를 AWS 계정 ID로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ssm.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "AWS_ACCOUNT_ID"
        },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:ssm:AWS_REGION:AWS_ACCOUNT_ID:*"
        }
      }
    }
  ]
}
```

4. 다음 명령을 사용하여 역할을 생성합니다.

```
aws iam create-role \
  --role-name AmazonEKSHybridNodesRole \
  --assume-role-policy-document file://eks-hybrid-ssm-trust.json
```

5. 이전 단계에서 생성한 EKSDescribeClusterPolicy와 EKSHybridSSMPolicy를 연결합니다. AWS_ACCOUNT_ID를 AWS 계정 ID로 바꿉니다.

```
aws iam attach-role-policy \
  --role-name AmazonEKSHybridNodesRole \
  --policy-arn arn:aws:iam::AWS_ACCOUNT_ID:policy/EKSDescribeClusterPolicy
```

```
aws iam attach-role-policy \
  --role-name AmazonEKSHybridNodesRole \
  --policy-arn arn:aws:iam::AWS_ACCOUNT_ID:policy/EKSHybridSSMPolicy
```

6. AmazonEC2ContainerRegistryPullOnly 및 AmazonSSMManagedInstanceCore AWS 관리형 정책을 연결합니다.

```
aws iam attach-role-policy \
  --role-name AmazonEKSHybridNodesRole \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

```
aws iam attach-role-policy \
  --role-name AmazonEKSHybridNodesRole \
  --policy-arn arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
```

AWS IAM Roles Anywhere의 단계

AWS IAM Roles Anywhere를 사용하려면 하이브리드 노드 IAM 역할을 생성하기 전에 AWS IAM Roles Anywhere 트러스트 앵커를 설정해야 합니다. 자세한 내용은 [the section called “AWS IAM Roles Anywhere 설정”](#) 섹션을 참조하세요.

1. eks-hybrid-iamra-trust.json이라는 이름의 파일을 만듭니다. TRUST_ANCHOR ARN을 [the section called “AWS IAM Roles Anywhere 설정”](#) 단계에서 생성한 트러스트 앵커의 ARN으로 바꿉니다. 이 신뢰 정책의 조건은 역할 세션 이름이 하이브리드 노드에 설치된 x509 인증서의 CN과 일치하는 경우에만 AWS IAM Roles Anywhere가 하이브리드 노드 IAM 역할을 수임하여 임시 IAM 자격 증명을 교환하는 기능을 제한합니다. 또는 다른 인증서 속성을 사용하여 노드를 고유하게 식별할 수 있습니다. 신뢰 정책에서 사용하는 인증서 속성은 nodeadm 구성에서 설정한 nodeName과 일치해야 합니다. 자세한 내용은 [the section called “하이브리드 노드 nodeadm”](#)을 참조하세요.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "rolesanywhere.amazonaws.com"
      },
      "Action": [
        "sts:TagSession",
        "sts:SetSourceIdentity"
      ],
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "TRUST_ANCHOR_ARN"
        }
      }
    }
  ]
}
```

```

    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "rolesanywhere.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}"
        },
        "ArnEquals": {
          "aws:SourceArn": "TRUST_ANCHOR_ARN"
        }
      }
    }
  ]
}

```

2. 다음 명령을 사용하여 역할을 생성합니다.

```

aws iam create-role \
  --role-name AmazonEKSHybridNodesRole \
  --assume-role-policy-document file:///eks-hybrid-iamra-trust.json

```

3. 이전 단계에서 생성한 EKSDescribeClusterPolicy를 연결합니다. AWS_ACCOUNT_ID를 AWS 계정 ID로 바꿉니다.

```

aws iam attach-role-policy \
  --role-name AmazonEKSHybridNodesRole \
  --policy-arn arn:aws:iam::AWS_ACCOUNT_ID:policy/EKSDescribeClusterPolicy

```

4. AmazonEC2ContainerRegistryPullOnly AWS 관리형 정책을 연결합니다.

```

aws iam attach-role-policy \
  --role-name AmazonEKSHybridNodesRole \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly

```

AWS Management Console

EKS 설명 클러스터 정책 생성

1. [Amazon IAM 콘솔](#)을 엽니다.
2. 왼쪽 탐색 창에서 정책을 선택합니다.
3. 정책 페이지에서 정책 생성을 선택합니다.
4. 권한 지정 페이지의 서비스 선택 패널에서 EKS를 선택합니다.
 - a. DescribeCluster에 대한 작업을 필터링하고 DescribeCluster 읽기 작업을 선택합니다.
 - b. 다음을 선택합니다.
5. 검토 및 생성 페이지에서
 - a. 정책에 정책 이름(예: EKSDescribeClusterPolicy)을 입력합니다.
 - b. 정책 생성을 선택합니다.

AWS SSM 하이브리드 활성화 단계

1. [Amazon IAM 콘솔](#)을 엽니다.
2. 왼쪽 탐색 창에서 정책을 선택합니다.
3. 정책 페이지에서 정책 생성을 선택합니다.
4. 권한 지정 페이지의 정책 편집기 오른쪽 상단에서 JSON을 선택합니다. 다음 조각을 붙여 넣습니다. AWS_REGION을 AWS SSM 하이브리드 활성화의 AWS 리전으로 바꾸고 AWS_ACCOUNT_ID를 AWS 계정 ID로 바꿉니다. TAG_KEY 및 TAG_VALUE를 AWS SSM 하이브리드 활성화를 생성할 때 사용한 AWS SSM 리소스 태그 키로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "ssm:DescribeInstanceInformation",
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": "ssm:DeregisterManagedInstance",
      "Resource": "arn:aws:ssm:AWS_REGION:AWS_ACCOUNT_ID:managed-instance/*",
      "Condition": {
        "StringEquals": {
          "ssm:resourceTag/TAG_KEY": "TAG_VALUE"
        }
      }
    }
  ]
}
```



```

    }
  ]
}
```

a. 다음을 선택합니다.

5. 검토 및 생성 페이지에서

- a. 정책에 정책 이름(예: EKSHybridSSMPolicy)을 입력합니다.
- b. 정책 생성을 선택하세요.

6. 왼쪽 탐색 창에서 역할을 선택합니다.

7. 역할 페이지에서 역할 생성을 선택합니다.

8. 신뢰할 수 있는 엔터티 선택 페이지에서 다음을 수행합니다.

- a. 신뢰할 수 있는 엔터티 유형 섹션에서 사용자 지정 신뢰 정책을 선택합니다. 사용자 지정 신뢰 정책 편집기에 다음을 붙여 넣습니다. AWS_REGION을 AWS SSM 하이브리드 활성화의 AWS 리전으로 바꾸고 AWS_ACCOUNT_ID를 AWS 계정 ID로 바꿉니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ssm.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "AWS_ACCOUNT_ID"
        },
        "ArnEquals": {
          "aws:SourceArn": "arn:aws:ssm:AWS_REGION:AWS_ACCOUNT_ID:*"
        }
      }
    }
  ]
}
```

b. [Next]를 선택합니다.

9. 권한 추가 페이지에서 사용자 지정 정책을 연결하거나 다음과 같이 합니다.

- a. 정책 필터링 상자에 EKSDescribeClusterPolicy 또는 위에서 생성한 정책의 이름을 입력합니다. 검색 결과의 정책 이름 왼쪽에 있는 확인란을 선택합니다.
- b. 정책 필터링 상자에 EKSHybridSSMPolicy 또는 위에서 생성한 정책의 이름을 입력합니다. 검색 결과의 정책 이름 왼쪽에 있는 확인란을 선택합니다.
- c. 필터 정책 상자에 AmazonEC2ContainerRegistryPullOnly를 입력합니다. 검색 결과의 AmazonEC2ContainerRegistryPullOnly 왼쪽에 있는 확인란을 선택합니다.
- d. 필터 정책 상자에 AmazonSSMManagedInstanceCore를 입력합니다. 검색 결과의 AmazonSSMManagedInstanceCore 왼쪽에 있는 확인란을 선택합니다.
- e. 다음을 선택합니다.

10. 이름, 검토 및 생성 페이지에서 다음을 수행합니다.

- a. 역할 이름에 역할의 고유한 이름(예: AmazonEKSHybridNodesRole)을 입력합니다.
- b. 설명(Description)에서 현재 텍스트를 설명이 포함된 텍스트(예: Amazon EKS - Hybrid Nodes role)로 바꿉니다.
- c. 역할 생성을 선택합니다.

AWS IAM Roles Anywhere의 단계

AWS IAM Roles Anywhere를 사용하려면 하이브리드 노드 IAM 역할을 생성하기 전에 AWS IAM Roles Anywhere 트러스트 앵커를 설정해야 합니다. 자세한 내용은 [the section called “AWS IAM Roles Anywhere 설정”](#) 섹션을 참조하세요.

1. [Amazon IAM 콘솔](#)을 엽니다.
2. 왼쪽 탐색 창에서 역할을 선택합니다.
3. 역할 페이지에서 역할 생성을 선택합니다.
4. 신뢰할 수 있는 엔터티 선택 페이지에서 다음을 수행합니다.
 - a. 신뢰할 수 있는 엔터티 유형 섹션에서 사용자 지정 신뢰 정책을 선택합니다. 사용자 지정 신뢰 정책 편집기에 다음을 붙여 넣습니다. TRUST_ANCHOR ARN을 [the section called “AWS IAM Roles Anywhere 설정”](#) 단계에서 생성한 트러스트 앵커의 ARN으로 바꿉니다. 이 신뢰 정책의 조건은 역할 세션 이름이 하이브리드 노드에 설치된 x509 인증서의 CN과 일치하는 경우에만 AWS IAM Roles Anywhere가 하이브리드 노드 IAM 역할을 수임하여 임시 IAM 자격 증명을 교환하는 기능을 제한합니다. 또는 다른 인증서 속성을 사용하여 노드를 고유하게 식별할 수 있습니다. 신뢰 정책에서 사용하는 인증서 속성은 nodeadm 구성에서 설정한 nodeName과 일치해야 합니다. 자세한 내용은 [the section called “하이브리드 노드 nodeadm”](#)을 참조하세요.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "rolesanywhere.amazonaws.com"
      },
      "Action": [
        "sts:TagSession",
        "sts:SetSourceIdentity"
      ],
      "Condition": {
        "ArnEquals": {
          "aws:SourceArn": "TRUST_ANCHOR_ARN"
        }
      }
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "rolesanywhere.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}"
        },
        "ArnEquals": {
          "aws:SourceArn": "TRUST_ANCHOR_ARN"
        }
      }
    }
  ]
}
```

b. [Next]를 선택합니다.

5. 권한 추가 페이지에서 사용자 지정 정책을 연결하거나 다음과 같이 합니다.

a. 정책 필터링 상자에 EKSDescribeClusterPolicy 또는 위에서 생성한 정책의 이름을 입력합니다. 검색 결과의 정책 이름 왼쪽에 있는 확인란을 선택합니다.

- b. 필터 정책 상자에 AmazonEC2ContainerRegistryPullOnly를 입력합니다. 검색 결과의 AmazonEC2ContainerRegistryPullOnly 왼쪽에 있는 확인란을 선택합니다.
 - c. 다음을 선택합니다.
6. 이름, 검토 및 생성 페이지에서 다음을 수행합니다.
- a. 역할 이름에 역할의 고유한 이름(예: AmazonEKSHybridNodesRole)을 입력합니다.
 - b. 설명(Description)에서 현재 텍스트를 설명이 포함된 텍스트(예: Amazon EKS - Hybrid Nodes role)로 바꿉니다.
 - c. 역할 생성을 선택합니다.

하이브리드 노드를 사용하여 Amazon EKS 클러스터 생성

이 주제에서는 사용 가능한 옵션에 대한 개요와 하이브리드 노드 지원 Amazon EKS 클러스터를 생성할 때 고려해야 할 사항을 설명합니다. EKS Hybrid Nodes는 표준 및 확장 지원을 포함하여 클라우드 노드가 있는 Amazon EKS 클러스터와 동일한 [Kubernetes 버전 지원](#)을 제공합니다.

EKS Hybrid Nodes를 사용할 계획이 없는 경우 [the section called “클러스터 생성”](#)의 기본 Amazon EKS 클러스터 생성 설명서를 참조하세요.

사전 조건

- [the section called “사전 조건”](#)가 완료되었습니다. 하이브리드 노드 지원 클러스터를 생성하기 전에 온프레미스 노드와 선택적으로 포드 CIDR을 식별하고, EKS 요구 사항 및 하이브리드 노드 요구 사항에 따라 VPC 및 서브넷을 생성하고, 온프레미스와 선택적으로 포드 CIDR에 대한 인바운드 규칙이 있는 보안 그룹을 지정해야 합니다. 이러한 사전 조건에 대한 자세한 내용은 [the section called “네트워킹 준비”](#) 섹션을 참조하세요.
- 장치에 최신 버전의 AWS Command Line Interface(AWS CLI)가 설치 및 구성되어 있습니다. 현재 버전을 확인하려면 `aws --version`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서의 [최신 버전의 AWS CLI 설치 또는 업데이트 및 AWS CLI 설정 구성](#)을 참조하세요.
- IAM 역할을 생성하고 정책을 연결하며 EKS 클러스터를 생성하고 설명할 권한이 있는 [IAM 보안 주제](#)

고려 사항

- 클러스터는 클러스터 인증 모드에 API 또는 API_AND_CONFIG_MAP을 사용해야 합니다.

- 클러스터는 IPv4 주소 패밀리를 사용해야 합니다.
- 클러스터는 퍼블릭 또는 프라이빗 클러스터 엔드포인트 연결을 사용해야 합니다. Amazon EKS Kubernetes API 서버 엔드포인트는 VPC 외부에서 실행되는 하이브리드 노드의 퍼블릭 IP로 확인되므로 클러스터는 '퍼블릭 및 프라이빗' 클러스터 엔드포인트 연결을 사용할 수 없습니다.
- OIDC 인증은 하이브리드 노드가 있는 EKS 클러스터에서 지원됩니다.
- 기존 클러스터의 하이브리드 노드 구성을 추가, 변경 또는 제거할 수 있습니다. 자세한 내용은 [the section called “기존 클러스터”](#) 섹션을 참조하세요.

1단계: 클러스터 IAM 역할 만들기

이미 클러스터 IAM 역할이 있거나 eksctl 또는 AWS CloudFormation을 사용하여 클러스터를 생성하려는 경우 이 단계를 건너뛸 수 있습니다. 기본적으로 eksctl 및 AWS CloudFormation 템플릿은 클러스터 IAM 역할을 생성합니다.

1. 다음 명령을 실행하여 IAM 신뢰 정책 JSON 파일을 생성합니다.

```
cat >eks-cluster-role-trust-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

2. Amazon EKS 클러스터 IAM 역할을 생성합니다. 필요한 경우 eks-cluster-role-trust-policy.json 앞에 이전 단계에서 파일을 작성한 컴퓨터의 경로를 붙입니다. 명령은 사용자가 이전 단계에서 생성한 신뢰 정책을 역할에 연결합니다. [IAM 역할](#)을 생성하려면 역할을 생성하는 IAM 보안 주체에 iam:CreateRole 작업(권한)을 할당해야 합니다.

```
aws iam create-role \
  --role-name myAmazonEKSClusterRole \
  --assume-role-policy-document file://eks-cluster-role-trust-policy.json
```

3. Amazon EKS 관리형 정책을 할당하거나 사용자 지정 정책을 생성할 수 있습니다. 사용자 지정 정책에서 사용해야 하는 최소 권한은 [the section called “노드 IAM 역할”](#)을(를) 참조하십시오. Amazon EKS 관리형 AmazonEKSClusterPolicy를 역할에 연결합니다. IAM 정책을 [IAM 보안 주체](#)에 연결하려면 정책을 연결하는 보안 주체에 iam:AttachUserPolicy 또는 iam:AttachRolePolicy의 IAM 작업(권한) 중 하나를 할당해야 합니다.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy \
  --role-name myAmazonEKSClusterRole
```

2단계: 하이브리드 노드 지원 클러스터 생성

다음을 사용하여 클러스터를 만들 수 있습니다.

- [eksctl](#)
- [AWS CloudFormation](#)
- [AWS CLI](#)
- [AWS Management Console](#)

하이브리드 노드 지원 클러스터 생성- eksctl

eksctl 명령줄 도구의 최신 버전을 설치해야 합니다. eksctl을 설치 또는 업그레이드하려면 eksctl 설명서에서 [설치](#)를 참조하세요.

1. cluster-config.yaml을 생성하여 하이브리드 노드 지원 Amazon EKS IPv4 클러스터를 정의합니다. cluster-config.yaml에서 다음과 같은 교체를 수행합니다. 전체 설정 목록은 [eksctl 설명서](#)를 참조하세요.
 - a. CLUSTER_NAME를 클러스터 이름으로 바꿉니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다.
 - b. AWS_REGION을 클러스터를 생성할 AWS 리전으로 바꿉니다.
 - c. K8S_VERSION를 모든 [Amazon EKS 지원 버전](#)으로 변경합니다.
 - d. [the section called “자격 증명 준비”](#)의 단계에서 구성한 자격 증명 공급자를 기반으로 CREDS_PROVIDER를 ssm 또는 ira로 바꿉니다.

- e. AWS IAM Roles Anywhere를 자격 증명 공급자로 사용하는 ira로 자격 증명 공급자가 설정된 경우 CA_BUNDLE_CERT를 바꿉니다. CA_BUNDLE_CERT는 CA(인증 기관) 인증서 본문이며 CA 선택에 따라 달라집니다. 인증서는 PEM(Privacy Enhanced Mail) 형식이어야 합니다.
- f. GATEWAY_ID를 VPC에 연결할 가상 프라이빗 게이트웨이 또는 전송 게이트웨이의 ID로 바꿉니다.
- g. REMOTE_NODE_CIDRS를 하이브리드 노드의 온프레미스 노드 CIDR로 바꿉니다.
- h. REMOTE_POD_CIDRS를 하이브리드 노드에서 실행되는 워크로드의 온프레미스 포트 CIDR로 바꾸거나 하이브리드 노드에서 웹후크를 실행하지 않는 경우 구성에서 라인을 제거합니다. 포트 트래픽이 온프레미스 호스트에서 나갈 때 CN이 포트 IP 주소에 네트워크 주소 변환(NAT) 또는 매스커레이딩을 사용하지 않는 경우 REMOTE_POD_CIDRS를 구성해야 합니다. 하이브리드 노드에서 웹후크를 실행하는 경우 REMOTE_POD_CIDRS를 구성해야 합니다. [the section called “웹후크 구성”](#)에서 자세한 내용을 참조하세요.
- i. 온프레미스 노드 및 포트 CIDR 블록은 다음 요구 사항을 충족해야 합니다.
 - i. IPv4 RFC-1918 범위 10.0.0.0/8, 172.16.0.0/12 또는 192.168.0.0/16 중 하나에 있어야 합니다.
 - ii. 서로, 클러스터의 VPC CIDR 또는 Kubernetes 서비스 IPv4 CIDR과 중첩되지 않습니다.

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: CLUSTER_NAME
  region: AWS_REGION
  version: "K8S_VERSION"

remoteNetworkConfig:
  iam:
    provider: CREDS_PROVIDER # default SSM, can also be set to IRA
    # caBundleCert: CA_BUNDLE_CERT
  vpcGatewayID: GATEWAY_ID
  remoteNodeNetworks:
    - cidrs: ["REMOTE_NODE_CIDRS"]
  remotePodNetworks:
    - cidrs: ["REMOTE_POD_CIDRS"]
```

2. 다음 명령 실행:

```
eksctl create cluster -f cluster-config.yaml
```

클러스터 프로비저닝에는 몇 분 정도 걸립니다. 클러스터가 생성되는 동안 여러 줄의 출력이 나타납니다. 출력의 마지막 줄은 다음 예제와 유사합니다.

```
[#] EKS cluster "CLUSTER_NAME" in "REGION" region is ready
```

3. [the section called “3단계: kubeconfig 업데이트”](#) 항목을 계속 진행합니다.

하이브리드 노드 지원 클러스터 생성-AWS CloudFormation

CloudFormation 스택은 사용자가 지정하는 RemoteNodeNetwork 및 RemotePodNetwork를 사용하여 EKS 클러스터 IAM 역할과 EKS 클러스터를 생성합니다. CloudFormation 템플릿에 노출되지 않은 EKS 클러스터의 설정을 사용자 지정해야 하는 경우 CloudFormation 템플릿을 수정합니다.

1. CloudFormation 템플릿을 다운로드하십시오.

```
curl -OL 'https://raw.githubusercontent.com/aws/eks-hybrid/refs/heads/main/example/hybrid-eks-cfn.yaml'
```

2. cfn-eks-parameters.json을 생성하고 각 값에 대한 구성을 지정합니다.

- a. CLUSTER_NAME: 생성할 EKS 클러스터의 이름입니다.
- b. CLUSTER_ROLE_NAME: 생성할 EKS 클러스터 IAM 역할의 이름입니다. 템플릿의 기본값은 'EKSClusterRole'입니다.
- c. SUBNET1_ID: 사전 조건 단계에서 생성한 첫 번째 서브넷의 ID입니다.
- d. SUBNET2_ID: 사전 조건 단계에서 생성한 두 번째 서브넷의 ID입니다.
- e. SG_ID: 이전 단계에서 만든 보안 그룹 ID입니다.
- f. REMOTE_NODE_CIDRS: 하이브리드 노드에 대한 온프레미스 노드 CIDR입니다.
- g. REMOTE_POD_CIDRS: 하이브리드 노드에서 실행되는 워크로드에 대한 온프레미스 포트 CIDR입니다. 포트 트래픽이 온프레미스 호스트에서 나갈 때 CNI가 포트 IP 주소에 네트워크 주소 변환 (NAT) 또는 매스커레이딩을 사용하지 않는 경우 REMOTE_POD_CIDRS를 구성해야 합니다. 하이브리드 노드에서 웹후크를 실행하는 경우 REMOTE_POD_CIDRS를 구성해야 합니다. [the section called “웹후크 구성”](#)에서 자세한 내용을 참조하세요.
- h. 온프레미스 노드 및 포트 CIDR 블록은 다음 요구 사항을 충족해야 합니다.
 - i. IPv4 RFC-1918 범위 10.0.0.0/8, 172.16.0.0/12 또는 192.168.0.0/16 중 하나에 있어야 합니다.
 - ii. 서로, 클러스터의 VPC CIDR 또는 Kubernetes 서비스 IPv4 CIDR과 중첩되지 않습니다.

- i. CLUSTER_AUTH: 클러스터의 클러스터 인증 모드입니다. 유효 값은 API 및 API_AND_CONFIG_MAP입니다. 템플릿의 기본값은 API_AND_CONFIG_MAP입니다.
- j. CLUSTER_ENDPOINT: 클러스터의 클러스터 엔드포인트 연결입니다. 유효한 값은 '퍼블릭' 또는 '프라이빗'입니다. 템플릿의 기본값은 프라이빗입니다. 즉, VPC 내에서만 Kubernetes API 엔드포인트에 연결할 수 있습니다.
- k. K8S_VERSION: 클러스터에 사용할 Kubernetes 버전입니다. [the section called “Kubernetes 버전”](#)을(를) 참조하세요.

```
{
  "Parameters": {
    "ClusterName": "CLUSTER_NAME",
    "ClusterRoleName": "CLUSTER_ROLE_NAME",
    "SubnetId1": "SUBNET1_ID",
    "SubnetId2": "SUBNET2_ID",
    "SecurityGroupId": "SG_ID",
    "RemoteNodeCIDR": "REMOTE_NODE_CIDRS",
    "RemotePodCIDR": "REMOTE_POD_CIDRS",
    "ClusterAuthMode": "CLUSTER_AUTH",
    "ClusterEndpointConnectivity": "CLUSTER_ENDPOINT",
    "K8sVersion": "K8S_VERSION"
  }
}
```

3. CloudFormation 스택 배포 STACK_NAME을 CloudFormation 스택의 이름으로 바꾸고 AWS_REGION을 클러스터를 생성할 AWS 리전으로 바꿉니다.

```
aws cloudformation deploy \
  --stack-name STACK_NAME \
  --region AWS_REGION \
  --template-file hybrid-eks-cfn.yaml \
  --parameter-overrides file://cfn-eks-parameters.json \
  --capabilities CAPABILITY_NAMED_IAM
```

클러스터 프로비저닝에는 몇 분 정도 걸립니다. 다음 명령을 사용하여 스택의 상태를 확인할 수 있습니다. STACK_NAME을 CloudFormation 스택의 이름으로 바꾸고 AWS_REGION을 클러스터를 생성할 AWS 리전으로 바꿉니다.

```
aws cloudformation describe-stacks \
  --stack-name STACK_NAME \
  --region AWS_REGION \
```

```
--query 'Stacks[].StackStatus'
```

4. [the section called “3단계: kubeconfig 업데이트”](#) 항목을 계속 진행합니다.

하이브리드 노드 지원 클러스터 생성-AWS CLI

1. 다음 명령을 실행하여 하이브리드 노드 지원 EKS 클러스터를 생성합니다. 명령을 실행하기 전에 다음을 실제 설정으로 바꿉니다. 전체 설정 목록은 [the section called “클러스터 생성”](#) 설명서를 참조하세요.
 - a. CLUSTER_NAME: 생성할 EKS 클러스터의 이름입니다.
 - b. AWS_REGION: 클러스터가 생성될 AWS 리전입니다.
 - c. K8S_VERSION: 클러스터에 사용할 Kubernetes 버전입니다. 지원되는 Amazon EKS 버전을 참조하세요.
 - d. ROLE_ARN: 클러스터에 대해 구성한 Amazon EKS 클러스터 역할입니다. 자세한 내용은 Amazon EKS 클러스터 IAM 역할을 참조하세요.
 - e. SUBNET1_ID: 사전 조건 단계에서 생성한 첫 번째 서브넷의 ID입니다.
 - f. SUBNET2_ID: 사전 조건 단계에서 생성한 두 번째 서브넷의 ID입니다.
 - g. SG_ID: 이전 단계에서 만든 보안 그룹 ID입니다.
 - h. 클러스터 액세스 인증 모드에서 API 및 API_AND_CONFIG_MAP을 사용할 수 있습니다. 아래 명령에서 클러스터 액세스 인증 모드는 API_AND_CONFIG_MAP으로 설정됩니다.
 - i. endpointPublicAccess 및 endpointPrivateAccess 파라미터를 사용하여 클러스터의 Kubernetes API 서버 엔드포인트에 대한 퍼블릭 및 프라이빗 액세스를 활성화하거나 비활성화할 수 있습니다. 아래 명령에서 endpointPublicAccess는 false로 설정되고 endpointPrivateAccess는 true로 설정됩니다.
 - j. REMOTE_NODE_CIDRS: 하이브리드 노드에 대한 온프레미스 노드 CIDR입니다.
 - k. REMOTE_POD_CIDRS (선택 사항): 하이브리드 노드에서 실행되는 워크로드에 대한 온프레미스 포트 CIDR입니다.
 - l. 온프레미스 노드 및 포트 CIDR 블록은 다음 요구 사항을 충족해야 합니다.
 - i. IPv4 RFC-1918 범위 10.0.0.0/8, 172.16.0.0/12 또는 192.168.0.0/16 중 하나에 있어야 합니다.
 - ii. 서로, Amazon EKS 클러스터의 VPC CIDR 또는 Kubernetes 서비스 IPv4 CIDR과 중첩되지 않습니다.

```
aws eks create-cluster \
```

```
--name CLUSTER_NAME \
--region AWS_REGION \
--kubernetes-version K8S_VERSION \
--role-arn ROLE_ARN \
--resources-vpc-config
subnetIds=SUBNET1_ID,SUBNET2_ID,securityGroupIds=SG_ID,endpointPrivateAccess=true,endpointPublicAccess=true \
--access-config authenticationMode=API_AND_CONFIG_MAP \
--remote-network-config '{"remoteNodeNetworks":[{"cidrs":["REMOTE_NODE_CIDRS"]}],"remotePodNetworks":[{"cidrs":["REMOTE_POD_CIDRS"]}]}'
```

2. 클러스터를 프로비저닝하는 데 몇 분 정도 걸립니다. 다음 명령을 사용하여 클러스터의 상태를 쿼리할 수 있습니다. CLUSTER_NAME을 생성 중인 클러스터의 이름으로 바꾸고 AWS_REGION을 클러스터가 생성 중인 AWS 리전으로 바꿉니다. 반환되는 출력이 ACTIVE가 되면 다음 단계로 진행하세요.

```
aws eks describe-cluster \
--name CLUSTER_NAME \
--region AWS_REGION \
--query "cluster.status"
```

3. [the section called “3단계: kubeconfig 업데이트”](#) 항목을 계속 진행합니다.

하이브리드 노드 지원 클러스터 생성 - AWS Management Console

1. [Amazon EKS 콘솔](#)에서 Amazon EKS 콘솔을 엽니다.
2. 클러스터 추가를 선택하고 생성을 선택합니다.
3. 클러스터 구성 페이지에서 다음 필드를 입력합니다.
 - a. 이름 - 클러스터의 이름 이름에는 영숫자(대소문자 구분)와 하이픈, 밑줄만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다.
 - b. 클러스터 IAM 역할-생성한 Amazon EKS 클러스터 IAM 역할을 선택하여 Kubernetes 컨트롤 플레인이 사용자 대신 AWS 리소스를 관리하게 할 수 있습니다.
 - c. Kubernetes 버전(Kubernetes version) - 클러스터에 대해 사용할 Kubernetes 버전 이전 버전이 필요한 경우가 아니면 최신 버전을 선택하는 것이 좋습니다.
 - d. 업그레이드 정책-확장 또는 표준을 선택합니다.
 - i. 확장: 이 옵션은 릴리스 날짜 이후 26개월 동안 Kubernetes 버전을 지원합니다. 연장 지원 기간에는 표준 지원 기간이 종료된 후 시작되는 시간당 추가 비용이 발생합니다. 확장 지원이 종료되면 클러스터가 다음 버전으로 자동 업그레이드됩니다.

- ii. 표준: 이 옵션은 릴리스 날짜 이후 14개월 동안 Kubernetes 버전을 지원합니다. 추가 비용은 없습니다. 표준 지원이 종료되면 클러스터가 다음 버전으로 자동 업그레이드됩니다.
 - e. 클러스터 액세스-클러스터 관리자 액세스를 허용하거나 허용하지 않도록 선택하고 인증 모드를 선택합니다. 하이브리드 노드 지원 클러스터에는 다음 인증 모드가 지원됩니다.
 - i. EKS API: 클러스터는 EKS 액세스 항목 API에서만 인증된 IAM 보안 주체를 가져옵니다.
 - ii. EKS API 및 ConfigMap: 클러스터는 EKS 액세스 항목 API와 aws-auth ConfigMap 모두에서 인증된 IAM 보안 주체를 가져옵니다.
 - f. 비밀 암호화(Secrets encryption) - (선택 사항) KMS 키를 사용하여 Kubernetes 비밀의 비밀 암호화를 사용 설정하도록 선택합니다. 클러스터를 생성한 후에도 이 기능을 사용 설정할 수 있습니다. 이 기능을 사용 설정하기 전에 [the section called “비밀 암호화 활성화”](#)의 정보를 숙지해야 합니다.
 - g. ARC 영역 전환-활성화하면 EKS는 클러스터를 ARC 영역 전환에 등록하고 영역 전환을 사용하여 애플리케이션 트래픽을 AZ에서 멀리 이동할 수 있습니다.
 - h. 태그-(선택사항) 클러스터에 태그를 추가합니다. 자세한 내용은 [the section called “리소스에 태깅”](#) 섹션을 참조하세요.
 - i. 이 페이지를 모두 완료하면 다음을 선택합니다.
4. 네트워킹 지정 페이지에서 다음 필드의 값을 선택합니다.
- a. VPC-[the section called “VPC 및 서브넷 요구 사항”](#) 및 [Amazon EKS Hybrid Nodes 요구 사항](#)을 충족하는 기존 VPC를 선택합니다. VPC를 선택하기 전에 VPC, 서브넷, 하이브리드 노드에 대한 Amazon EKS 네트워킹 요구 사항 보기의 모든 요구 사항과 고려 사항을 숙지하는 것이 좋습니다. 클러스터 생성 후에는 사용하려는 VPC를 변경할 수 없습니다. 목록에 기존 VPC가 없는 경우 먼저 VPC를 생성해야 합니다. 자세한 내용은 [the section called “VPC 생성”](#) 및 [Amazon EKS Hybrid Nodes 네트워킹 요구 사항](#)을 참조하세요.
 - b. 서브넷-기본적으로 이전 필드에서 지정한 VPC에서 사용 가능한 모든 서브넷이 사전 선택됩니다. 두 개 이상을 선택해야 합니다.
 - c. 보안 그룹 - (선택사항) 생성하는 네트워크 인터페이스에 Amazon EKS가 연결하려는 보안 그룹을 하나 이상 지정합니다. 지정하는 보안 그룹 중 하나 이상에 온프레미스 노드와 선택적으로 포트 CIDR에 대한 인바운드 규칙이 있어야 합니다. 자세한 내용은 [Amazon EKS Hybrid Nodes networking requirements](#)을 참조하세요. 보안 그룹 선택 여부에 관계없이 Amazon EKS는 클러스터와 VPC 간에 통신을 사용 설정할 수 있는 보안 그룹을 생성합니다. Amazon EKS는 이 보안 그룹과 사용자가 선택한 보안 그룹을 생성하는 네트워크 인터페이스에 연결합니다. Amazon EKS가 생성하는 클러스터 보안 그룹에 대한 자세한 내용은 [the section called “보안 그룹 요구 사항”](#) 섹션을 참조하세요. Amazon EKS가 생성하는 클러스터 보안 그룹의 규칙을 수정할 수 있습니다.
 - d. 클러스터 IP 주소 패밀리 선택-하이브리드 노드 지원 클러스터의 경우 IPv4를 선택해야 합니다.

- e. (선택 사항) Kubernetes 서비스 IP 주소 범위 구성을 선택하고 서비스 IPv4 범위를 지정합니다.
 - f. 하이브리드 노드를 활성화하도록 원격 네트워크 구성을 선택하고 하이브리드 노드에 대한 온프레미스 노드와 포트 CIDR을 지정합니다.
 - g. 포트 트래픽이 온프레미스 호스트에서 나갈 때 CNI가 포트 IP 주소에 네트워크 주소 변환(NAT) 또는 매스커레이딩을 사용하지 않는 경우 원격 포트 CIDR을 구성해야 합니다. 하이브리드 노드에서 웹후크를 실행하는 경우 원격 포트 CIDR을 구성해야 합니다.
 - h. 온프레미스 노드 및 포트 CIDR 블록은 다음 요구 사항을 충족해야 합니다.
 - i. IPv4 RFC-1918 범위 10.0.0.0/8, 172.16.0.0/12 또는 192.168.0.0/16 중 하나에 있어야 합니다.
 - ii. 서로, 클러스터의 VPC CIDR 또는 Kubernetes 서비스 IPv4 CIDR과 중첩되지 않습니다.
 - i. 클러스터 엔드포인트 액세스에서 옵션을 선택합니다. 클러스터를 생성한 후에는 이 옵션을 변경할 수 있습니다. 하이브리드 노드 지원 클러스터의 경우 퍼블릭 또는 프라이빗을 선택해야 합니다. 기본값이 아닌 옵션을 선택하기 전에 옵션과 그 의미를 숙지해야 합니다. 자세한 내용은 [the section called “클러스터 엔드포인트 액세스”](#) 섹션을 참조하세요.
 - j. 이 페이지를 모두 완료하면 다음을 선택합니다.
5. (선택사항) 관찰성 구성 페이지에서 설정할 지표 및 컨트롤 플레인 로깅 옵션을 선택합니다. 기본적으로 각 로그 유형은 해제되어 있습니다.
- a. Prometheus 지표 옵션에 자세한 내용은 [the section called “Prometheus 지표”](#) 섹션을 참조하세요.
 - b. EKS 컨트롤 로깅 옵션에 대한 자세한 내용은 [the section called “컨트롤 플레인 로그”](#) 섹션을 참조하세요.
 - c. 이 페이지를 모두 완료하면 다음을 선택합니다.
6. 추가 기능 선택 페이지에서 클러스터에 추가할 추가 기능을 선택합니다.
- a. 필요한 만큼 Amazon EKS 애드온 및 AWS 마켓플레이스 애드온을 선택할 수 있습니다. 하이브리드 노드와 호환되지 않는 Amazon EKS 추가 기능은 '하이브리드 노드와 호환되지 않음'으로 표시되며 추가 기능에는 하이브리드 노드에서 실행되지 않게 하는 반선택도 규칙이 있습니다. 자세한 내용은 하이브리드 노드용 추가 기능 구성을 참조하세요. 설치하려는 AWS 마켓플레이스 애드온이 목록에 없는 경우 검색창에 텍스트를 입력하여 사용 가능한 AWS 마켓플레이스 애드온을 검색할 수 있습니다. 카테고리, 공급업체 또는 요금 모델로 검색한 다음에 검색 결과 중에서 추가 기능을 선택할 수도 있습니다.
 - b. CoreDNS, kube-proxy와 같은 일부 추가 기능은 기본적으로 설치됩니다. 기본 추가 기능을 비활성화하면 Kubernetes 애플리케이션 실행 기능에 영향을 미칠 수 있습니다.
 - c. 이 페이지를 모두 완료하면 Next을 선택합니다.

7. 선택한 추가 기능 설정 구성 페이지에서 설치할 버전을 선택합니다.
 - a. 클러스터 생성 후 언제든지 최신 버전으로 업데이트할 수 있습니다. 클러스터 생성 후 각 추가 기능의 구성을 업데이트할 수 있습니다. 추가 기능 구성에 대한 자세한 내용은 [the section called “추가 기능 업데이트”](#) 섹션을 참조하세요. 하이브리드 노드와 호환되는 추가 기능 버전은 [the section called “추가 기능 구성”](#) 섹션을 참조하세요.
 - b. 이 페이지를 모두 완료하면 다음을 선택합니다.
8. 검토 및 생성 페이지에서 이전 페이지에서 입력하거나 선택한 정보를 검토합니다. 변경해야 하는 경우 편집을 선택합니다. 만족하는 경우 생성을 선택합니다. 클러스터가 프로비저닝되는 동안 상태 필드에는 생성 중이 표시됩니다. 클러스터 프로비저닝에는 몇 분 정도 걸립니다.
9. [the section called “3단계: kubeconfig 업데이트”](#) 항목을 계속 진행합니다.

3단계: kubeconfig 업데이트

eksctl을 사용하여 클러스터를 생성한 경우 이 단계를 건너뛸 수 있습니다. 이는 eksctl이 사용자 대신 이 단계를 이미 완료했기 때문입니다. kubectl 구성 파일에 새 컨텍스트를 추가하여 이 클러스터와 통신하도록 kubectl을 활성화합니다. 파일 생성 및 업데이트 방법에 대한 자세한 내용을 알아보려면 [the section called “kubectl을 사용하여 클러스터에 액세스”](#) 부분을 참조하세요.

```
aws eks update-kubeconfig --name CLUSTER_NAME --region AWS_REGION
```

예제 출력은 다음과 같습니다.

```
Added new context arn:aws:eks:AWS_REGION:111122223333:cluster/CLUSTER_NAME to /home/username/.kube/config
```

다음 명령을 실행하여 클러스터와의 통신을 확인합니다.

```
kubectl get svc
```

예제 출력은 다음과 같습니다.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.100.0.1	<none>	443/TCP	28h

4단계: 클러스터 설정

다음 단계로 하이브리드 노드가 클러스터에 조인할 수 있도록 액세스를 활성화하려면 [the section called “클러스터 액세스 준비”](#) 섹션을 참조하세요.

기존 Amazon EKS 클러스터에서 하이브리드 노드 활성화 또는 구성 수정

이 주제에서는 사용 가능한 옵션에 대한 개요를 제공하고 Amazon EKS 클러스터의 하이브리드 노드 구성을 추가, 변경 또는 제거할 때 고려해야 할 사항을 설명합니다.

Amazon EKS 클러스터에서 하이브리드 노드를 사용할 수 있게 하려면 RemoteNetworkConfig 구성에서 온프레미스 노드와 포드 네트워크(선택 사항)의 IP 주소 CIDR 범위를 추가합니다. EKS는 이 CIDR 목록을 사용하여 클러스터와 온프레미스 네트워크 간의 연결을 활성화합니다. 클러스터 구성을 업데이트할 때의 전체 옵션 목록은 Amazon EKS API 참조의 [UpdateClusterConfig](#)를 참조하세요.

클러스터의 EKS Hybrid Nodes 네트워킹 구성에 대해 다음 작업 중 하나를 수행할 수 있습니다.

- [원격 네트워크 구성을 추가하여 기존 클러스터에서 EKS Hybrid Nodes를 활성화합니다.](#)
- [기존 클러스터에서 원격 노드 네트워크나 원격 포드 네트워크를 추가, 변경 또는 제거합니다.](#)
- [모든 원격 노드 네트워크 CIDR 범위를 제거하여 기존 클러스터에서 EKS Hybrid Nodes를 비활성화합니다.](#)

사전 조건

- 하이브리드 노드에 대해 Amazon EKS 클러스터를 활성화하기 전에 환경이 [the section called “사전 조건”](#)에 요약되어 있고 [the section called “네트워킹 준비”](#), [the section called “운영 체제 준비”](#) 및 [the section called “자격 증명 준비”](#)에 자세히 설명된 요구 사항을 충족하는지 확인합니다.
- 클러스터는 IPv4 주소 패밀리를 사용해야 합니다.
- 클러스터는 클러스터 인증 모드에 API 또는 API_AND_CONFIG_MAP을 사용해야 합니다. 클러스터 인증 모드를 수정하는 프로세스는 [the section called “인증 모드”](#)에 설명되어 있습니다.
- Amazon EKS Kubernetes API 서버 엔드포인트에 퍼블릭 또는 프라이빗 엔드포인트 액세스 중 하나만 사용하는 것이 좋습니다. '퍼블릭 및 프라이빗'을 선택하면 Amazon EKS Kubernetes API 서버 엔드포인트가 항상 VPC 외부에서 실행되는 하이브리드 노드의 퍼블릭 IP로 확인되므로 하이브리드 노드가 클러스터에 조인하지 못할 수 있습니다. 클러스터에 대한 네트워크 액세스를 수정하는 프로세스는 [the section called “클러스터 엔드포인트 액세스”](#)에 설명되어 있습니다.
- 장치에 최신 버전의 AWS Command Line Interface(AWS CLI)가 설치 및 구성되어 있습니다. 현재 버전을 확인하려면 `aws --version`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서의 [최신 버전의 AWS CLI 설치 또는 업데이트 및 AWS CLI 설정 구성](#)을 참조하세요.
- Amazon EKS 클러스터에서 [UpdateClusterConfig](#)를 직접적으로 호출할 수 있는 권한이 있는 [IAM 위탁자](#)입니다.

- 하이브리드 노드와 호환되는 버전으로 추가 기능을 업데이트합니다. 하이브리드 노드와 호환되는 추가 기능 버전은 [the section called “추가 기능 구성”](#) 섹션을 참조하세요.
- 하이브리드 노드와 호환되지 않는 추가 기능을 실행하는 경우 하이브리드 노드에 배포를 방지하는 다음 선호도 규칙이 추가 기능 [DaemonSet](#) 또는 [배포](#)에 있는지 확인합니다. 아직 없으면 다음 선호도 규칙을 추가합니다.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
            - key: eks.amazonaws.com/compute-type
              operator: NotIn
              values:
                - hybrid
```

고려 사항

remoteNetworkConfig JSON 객체는 업데이트 중 다음과 같은 동작을 합니다.

- 지정하지 않은 구성의 기존 부분은 변경되지 않습니다. remoteNodeNetworks 또는 remotePodNetworks 중 하나를 지정하지 않는 경우 해당 부분은 동일하게 유지됩니다.
- CIDR의 remoteNodeNetworks 또는 remotePodNetworks 목록 중 하나를 수정하는 경우 최종 구성에서 원하는 CIDR의 전체 목록을 지정해야 합니다. remoteNodeNetworks 또는 remotePodNetworks CIDR 목록 중 하나에 대한 변경 사항을 지정하는 경우 업데이트 중 EKS가 원래 목록을 대체합니다.
- 온프레미스 노드 및 포드 CIDR 블록은 다음 요구 사항을 충족해야 합니다.
 1. IPv4 RFC-1918 범위인 10.0.0.0/8, 172.16.0.0/12 또는 192.168.0.0/16 중 하나에 속해야 합니다.
 2. 서로, Amazon EKS 클러스터의 모든 VPC CIDR 또는 Kubernetes 서비스 IPv4 CIDR과 중첩되지 않아야 합니다.

기존 클러스터에서 하이브리드 노드 활성화

다음을 사용하여 기존 클러스터에서 EKS Hybrid Nodes를 활성화할 수 있습니다.

- [AWS CloudFormation](#)
- [AWS CLI](#)

- [AWS Management Console](#)

기존 클러스터에서 EKS Hybrid Nodes 활성화 - AWS CloudFormation

1. 클러스터에서 EKS Hybrid Nodes를 활성화하려면 CloudFormation 템플릿에 RemoteNodeNetwork와 RemotePodNetwork(선택 사항)를 추가하고 스택을 업데이트합니다. RemoteNodeNetwork는 최대 하나의 Cidrs 항목이 포함된 목록이고, Cidrs는 여러 IP CIDR 범위의 목록입니다.

```
RemoteNetworkConfig:
  RemoteNodeNetworks:
    - Cidrs: [RemoteNodeCIDR]
  RemotePodNetworks:
    - Cidrs: [RemotePodCIDR]
```

2. 계속해서 [the section called “클러스터 액세스 준비”](#)로 이동하세요.

기존 클러스터에서 EKS Hybrid Nodes 활성화 - AWS CLI

1. 다음 명령을 실행하여 EKS 클러스터의 EKS Hybrid Nodes에 대해 RemoteNetworkConfig를 활성화합니다. 명령을 실행하기 전에 다음을 실제 설정으로 바꿉니다. 전체 설정 목록은 Amazon EKS API 참조의 [UpdateClusterConfig](#)를 참조하세요.
 - a. CLUSTER_NAME: 업데이트할 EKS 클러스터의 이름입니다.
 - b. AWS_REGION: EKS 클러스터가 실행 중인 AWS 리전입니다.
 - c. REMOTE_NODE_CIDRS: 하이브리드 노드에 대한 온프레미스 노드 CIDR입니다.
 - d. REMOTE_POD_CIDRS (선택 사항): 하이브리드 노드에서 실행되는 워크로드에 대한 온프레미스 포트 CIDR입니다.

```
aws eks update-cluster-config \
  --name CLUSTER_NAME \
  --region AWS_REGION \
  --remote-network-config '{"remoteNodeNetworks":[{"cidrs":
["REMOTE_NODE_CIDRS"]}], "remotePodNetworks":[{"cidrs":["REMOTE_POD_CIDRS"]}]}'
```

2. 클러스터를 업데이트하는 데 몇 분 정도 걸립니다. 다음 명령을 사용하여 클러스터의 상태를 쿼리할 수 있습니다. CLUSTER_NAME을 수정 중인 클러스터의 이름으로 바꾸고 AWS_REGION을 클러스터가 실행되고 있는 AWS 리전으로 바꿉니다. 반환되는 출력이 ACTIVE가 되면 다음 단계로 진행하세요.

```
aws eks describe-cluster \
  --name CLUSTER_NAME \
  --region AWS_REGION \
  --query "cluster.status"
```

3. 계속해서 [the section called “클러스터 액세스 준비”](#)로 이동하세요.

기존 클러스터에서 EKS Hybrid Nodes 활성화 - AWS Management Console

1. [Amazon EKS 콘솔](#)에서 Amazon EKS 콘솔을 엽니다.
2. 클러스터 이름을 선택하여 클러스터 정보를 표시합니다.
3. 네트워킹 탭을 선택하고 관리를 선택합니다.
4. 드롭다운에서 원격 네트워크를 선택합니다.
5. 하이브리드 노드를 활성화하도록 원격 네트워크 구성을 선택하고 하이브리드 노드에 대한 온프레미스 노드와 포드 CIDR을 지정합니다.
6. 변경 사항 저장을 선택하여 종료합니다. 클러스터가 활성 상태로 돌아갈 때까지 기다립니다.
7. 계속해서 [the section called “클러스터 액세스 준비”](#)로 이동하세요.

기존 클러스터에서 하이브리드 노드 구성 업데이트

다음 중 하나를 사용하여 기존 하이브리드 클러스터에서 `remoteNetworkConfig`를 수정할 수 있습니다.

- [AWS CloudFormation](#)
- [AWS CLI](#)
- [AWS Management Console](#)

기존 클러스터에서 하이브리드 구성 업데이트 - AWS CloudFormation

1. 새 네트워크 CIDR 값으로 CloudFormation 템플릿을 업데이트합니다.

```
RemoteNetworkConfig:
  RemoteNodeNetworks:
    - Cidrs: [NEW_REMOTE_NODE_CIDRS]
  RemotePodNetworks:
    - Cidrs: [NEW_REMOTE_POD_CIDRS]
```

Note

RemoteNodeNetworks 또는 RemotePodNetworks CIDR 목록을 업데이트할 때 모든 CIDR(신규 및 기존)을 포함합니다. EKS는 업데이트 중 전체 목록을 대체합니다. 업데이트 요청에서 이러한 필드를 생략하면 기존 구성이 유지됩니다.

- 수정된 템플릿으로 CloudFormation 스택을 업데이트하고 스택 업데이트가 완료될 때까지 기다립니다.

기존 클러스터에서 하이브리드 구성 업데이트 - AWS CLI

- 원격 네트워크 CIDR을 수정하려면 다음 명령을 실행합니다. 값을 실제 설정으로 바꿉니다.

```
aws eks update-cluster-config
--name CLUSTER_NAME
--region AWS_REGION
--remote-network-config '{"remoteNodeNetworks":[{"cidrs":
["NEW_REMOTE_NODE_CIDRS"]}], "remotePodNetworks":[{"cidrs":
["NEW_REMOTE_POD_CIDRS"]}]}'
```

Note

remoteNodeNetworks 또는 remotePodNetworks CIDR 목록을 업데이트할 때 모든 CIDR(신규 및 기존)을 포함합니다. EKS는 업데이트 중 전체 목록을 대체합니다. 업데이트 요청에서 이러한 필드를 생략하면 기존 구성이 유지됩니다.

- 계속 진행하기 전에 클러스터가 활성화된 상태로 돌아갈 때까지 기다립니다.

기존 클러스터에서 하이브리드 구성 업데이트 - AWS Management Console

- [Amazon EKS 콘솔](#)에서 Amazon EKS 콘솔을 엽니다.
- 클러스터 이름을 선택하여 클러스터 정보를 표시합니다.
- 네트워킹 탭을 선택하고 관리를 선택합니다.
- 드롭다운에서 원격 네트워크를 선택합니다.
- 필요에 따라 Remote pod networks - Optional 및 Remote node networks에서 CIDR을 업데이트합니다.

6. 변경 사항 저장을 선택하고 클러스터 상태가 활성으로 돌아갈 때까지 기다립니다.

기존 클러스터에서 하이브리드 노드 비활성화

다음을 사용하여 기존 클러스터에서 EKS Hybrid Nodes를 비활성화할 수 있습니다.

- [AWS CloudFormation](#)
- [AWS CLI](#)
- [AWS Management Console](#)

기존 클러스터에서 EKS Hybrid Nodes 비활성화 - AWS CloudFormation

1. 클러스터에서 EKS Hybrid Nodes를 비활성화하려면 CloudFormation 템플릿에서 RemoteNodeNetworks와 RemotePodNetworks를 빈 어레이로 설정하고 스택을 업데이트합니다.

```
RemoteNetworkConfig:
  RemoteNodeNetworks: []
  RemotePodNetworks: []
```

기존 클러스터에서 EKS Hybrid Nodes 비활성화 - AWS CLI

1. 다음 명령을 실행하여 EKS 클러스터에서 RemoteNetworkConfig를 제거합니다. 명령을 실행하기 전에 다음을 실제 설정으로 바꿉니다. 전체 설정 목록은 Amazon EKS API 참조의 [UpdateClusterConfig](#)를 참조하세요.
 - a. CLUSTER_NAME: 업데이트할 EKS 클러스터의 이름입니다.
 - b. AWS_REGION: EKS 클러스터가 실행 중인 AWS 리전입니다.

```
aws eks update-cluster-config \
  --name CLUSTER_NAME \
  --region AWS_REGION \
  --remote-network-config '{"remoteNodeNetworks":[],"remotePodNetworks":[]}'
```

2. 클러스터를 업데이트하는 데 몇 분 정도 걸립니다. 다음 명령을 사용하여 클러스터의 상태를 쿼리할 수 있습니다. CLUSTER_NAME을 수정 중인 클러스터의 이름으로 바꾸고 AWS_REGION을 클러스터가 실행되고 있는 AWS 리전으로 바꿉니다. 반환되는 출력이 ACTIVE가 되면 다음 단계로 진행하세요.

```
aws eks describe-cluster \
  --name CLUSTER_NAME \
  --region AWS_REGION \
  --query "cluster.status"
```

기존 클러스터에서 EKS Hybrid Nodes 비활성화 - AWS Management Console

1. [Amazon EKS 콘솔](#)에서 Amazon EKS 콘솔을 엽니다.
2. 클러스터 이름을 선택하여 클러스터 정보를 표시합니다.
3. 네트워킹 탭을 선택하고 관리를 선택합니다.
4. 드롭다운에서 원격 네트워크를 선택합니다.
5. Remote node networks 및 Remote pod networks - Optional에서 하이브리드 노드를 사용하도록 원격 네트워크 구성을 선택하고 모든 CIDR을 제거합니다.
6. 변경 사항 저장을 선택하여 종료합니다. 클러스터가 활성 상태로 돌아갈 때까지 기다립니다.

하이브리드 노드에 대한 클러스터 액세스 준비

하이브리드 노드를 Amazon EKS 클러스터에 연결하기 전에 Kubernetes 권한이 있는 하이브리드 노드 IAM 역할을 활성화하여 클러스터에 조인해야 합니다. 하이브리드 노드 IAM 역할 생성 방법에 대한 자세한 내용은 [the section called “자격 증명 준비”](#) 섹션을 참조하세요. Amazon EKS는 IAM 보안 주체를 Kubernetes 역할 기반 액세스 제어(RBAC), Amazon EKS 액세스 항목, aws-auth ConfigMap과 연결하는 두 가지 방법을 지원합니다. Amazon EKS 액세스 관리에 대한 자세한 내용은 [the section called “Kubernetes API 액세스”](#) 섹션을 참조하세요.

아래 절차에 따라 하이브리드 노드 IAM 역할을 Kubernetes 권한과 연결합니다. Amazon EKS 액세스 항목을 사용하려면 클러스터가 API 또는 API_AND_CONFIG_MAP 인증 모드로 생성되어야 합니다. aws-auth ConfigMap을 사용하려면 클러스터가 API_AND_CONFIG_MAP 인증 모드로 생성되어야 합니다. 하이브리드 노드 지원 Amazon EKS 클러스터에는 CONFIG_MAP 전용 인증 모드가 지원되지 않습니다.

하이브리드 노드 IAM 역할에 Amazon EKS 액세스 항목 사용

IAM 역할과 함께 사용할 수 있는 하이브리드 노드의 Amazon EKS 액세스 항목 유형인 HYBRID_LINUX가 있습니다. 이 액세스 항목 유형에서는 사용자 이름이 system:node:{{SessionName}}으로 자동 설정됩니다. 액세스 항목 생성에 대한 자세한 내용은 [the section called “액세스 항목 생성”](#) 섹션을 참조하세요.

AWS CLI

1. 장치에 최신 버전의 AWS CLI가 설치 및 구성되어 있어야 합니다. 현재 버전을 확인하려면 `aws --version`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 설치 및 `aws config`를 사용하여 빠른 구성을 참조하세요.
2. 다음 명령을 사용하여 액세스 항목을 생성합니다. CLUSTER_NAME을 클러스터 이름으로 바꾸고 HYBRID_NODES_ROLE_ARN을 [the section called “자격 증명 준비”](#)의 단계에서 생성한 역할의 ARN으로 바꿉니다.

```
aws eks create-access-entry --cluster-name CLUSTER_NAME \  
  --principal-arn HYBRID_NODES_ROLE_ARN \  
  --type HYBRID_LINUX
```

AWS Management Console

1. [Amazon EKS 콘솔](#)에서 Amazon EKS 콘솔을 엽니다.
2. 하이브리드 노드 지원 클러스터의 이름을 선택합니다.
3. 액세스 탭을 선택합니다.
4. 액세스 항목 생성을 선택합니다.
5. IAM 보안 주체의 경우 [the section called “자격 증명 준비”](#)의 단계에서 생성한 하이브리드 노드 IAM 역할을 선택합니다.
6. 유형에서 하이브리드 Linux를 선택합니다.
7. (선택사항) 태그에서 액세스 항목에 레이블을 할당합니다. 예를 들어, 태그가 동일한 모든 리소스를 더 쉽게 찾을 수 있습니다.
8. 검토 및 생성 건너뛰기를 선택합니다. 하이브리드 Linux 액세스 항목에 정책을 추가하거나 액세스 범위를 변경할 수 없습니다.
9. 액세스 항목의 구성을 검토합니다. 문제가 있는 것 같으면 이전을 선택하여 이전 단계로 돌아가서 오류를 수정합니다. 구성이 올바르면 생성을 선택합니다.

하이브리드 노드 IAM 역할에 aws-auth ConfigMap 사용

다음 단계에서는 [the section called “자격 증명 준비”](#)의 단계에서 생성한 하이브리드 노드 IAM 역할의 ARN으로 `aws-auth ConfigMap`을 생성하거나 업데이트합니다.

1. 클러스터에 대한 기존 `aws-auth ConfigMap`이 있는지 확인합니다. 특정 `kubeconfig` 파일을 사용하는 경우 `--kubeconfig` 플래그를 사용합니다.

```
kubectl describe configmap -n kube-system aws-auth
```

2. `aws-auth ConfigMap`이 표시되면 필요에 따라 이를 업데이트합니다.

- a. 편집을 위해 `ConfigMap`을 엽니다.

```
kubectl edit -n kube-system configmap/aws-auth
```

- b. 필요에 따라 새 `mapRoles` 항목을 추가합니다. `HYBRID_NODES_ROLE_ARN`을 하이브리드 노드 IAM 역할의 ARN으로 바꿉니다. 참고로 `{{SessionName}}`은 `ConfigMap`에 저장할 올바른 템플릿 형식입니다. 다른 값으로 바꾸지 마세요.

```
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: HYBRID_NODES_ROLE_ARN
      username: system:node:{{SessionName}}
```

- c. 파일을 저장하고 텍스트 편집기를 종료합니다.

3. 클러스터에 대한 기존 `aws-auth ConfigMap`이 없는 경우 다음 명령을 사용하여 생성합니다. `HYBRID_NODES_ROLE_ARN`을 하이브리드 노드 IAM 역할의 ARN으로 바꿉니다. `{{SessionName}}`은 `ConfigMap`에 저장할 올바른 템플릿 형식입니다. 다른 값으로 바꾸지 마세요.

```
kubectl apply -f=/dev/stdin <<-EOF
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: HYBRID_NODES_ROLE_ARN
      username: system:node:{{SessionName}}
```

하이브리드 노드에서 온프레미스 워크로드 실행

하이브리드 노드가 활성화된 EKS 클러스터에서는 AWS 클라우드에서 사용하는 것과 동일한 Amazon EKS 클러스터, 기능, 도구를 사용하여 자체 인프라에서 온프레미스 및 엣지 애플리케이션을 실행할 수 있습니다.

다음 섹션에는 하이브리드 노드 사용을 위한 단계별 지침이 포함되어 있습니다.

주제

- [하이브리드 노드 연결](#)
- [Bottlerocket을 실행하는 하이브리드 노드 연결](#)
- [클러스터의 하이브리드 노드 업그레이드](#)
- [하이브리드 노드의 패치 보안 업데이트](#)
- [하이브리드 노드 제거](#)

하이브리드 노드 연결

Note

다음 단계는 Bottlerocket을 제외한 호환 운영 체제를 실행하는 하이브리드 노드에 적용됩니다. Bottlerocket을 실행하는 하이브리드 노드를 연결하는 단계는 [the section called “Bottlerocket을 실행하는 하이브리드 노드 연결”](#) 섹션을 참조하세요.

이 주제에서는 하이브리드 노드를 Amazon EKS 클러스터에 연결하는 방법을 설명합니다. 하이브리드 노드가 클러스터에 조인되면 Amazon EKS 콘솔이나 kubectl과 같은 Kubernetes 호환 도구에 준비되지 않은 상태로 표시됩니다. 이 페이지의 단계를 완료한 후 [the section called “CNI 구성”](#)로 이동하여 하이브리드 노드가 애플리케이션을 실행할 준비가 되게 합니다.

사전 조건

하이브리드 노드를 Amazon EKS 클러스터에 연결하기 전에 사전 조건 단계를 완료했는지 확인합니다.

- 온프레미스 환경에서 Amazon EKS 클러스터를 호스팅하는 AWS 리전으로 네트워크 연결이 가능합니다. 자세한 정보는 [the section called “네트워킹 준비”](#)를 참조하세요.

- 온프레미스 호스트에 하이브리드 노드용 호환 운영 체제가 설치되어 있습니다. 자세한 정보는 [the section called “운영 체제 준비”](#)을 참조하세요.
- 하이브리드 노드 IAM 역할을 생성하고 온프레미스 자격 증명 공급자(AWS Systems Manager 하이브리드 활성화 또는 AWS IAM Roles Anywhere)를 설정했습니다. 자세한 정보는 [the section called “자격 증명 준비”](#)을 참조하세요.
- 하이브리드 노드가 활성화된 Amazon EKS 클러스터를 생성했습니다. 자세한 정보는 [the section called “클러스터 생성”](#)을 참조하세요.
- 하이브리드 노드 IAM 역할을 Kubernetes 역할 기반 액세스 제어(RBAC) 권한과 연결했습니다. 자세한 정보는 [the section called “클러스터 액세스 준비”](#)을 참조하세요.

1단계: 각 온프레미스 호스트에 하이브리드 노드 CLI(**nodeadm**) 설치

사전 구축된 운영 체제 이미지에 Amazon EKS Hybrid Nodes CLI(**nodeadm**)를 포함하는 경우 이 단계를 건너뛸 수 있습니다. 하이브리드 노드 버전 **nodeadm**에 대한 자세한 내용은 [the section called “하이브리드 노드 nodeadm”](#) 섹션을 참조하세요.

하이브리드 노드 버전 **nodeadm**은 Amazon CloudFront가 제공하는 Amazon S3에서 호스팅됩니다. 각 온프레미스 호스트에 **nodeadm**을 설치하려면 온프레미스 호스트에서 다음 명령을 실행할 수 있습니다.

x86_64 호스트의 경우:

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/amd64/nodeadm'
```

ARM 호스트의 경우

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/arm64/nodeadm'
```

각 호스트에서 다운로드한 바이너리에 실행 파일 권한을 추가합니다.

```
chmod +x nodeadm
```

2단계: **nodeadm**을 사용하여 하이브리드 노드 종속성 설치

사전 구축된 운영 체제 이미지에 하이브리드 노드 종속성을 설치하는 경우 이 단계를 건너뛸 수 있습니다. **nodeadm install** 명령을 사용하여 하이브리드 노드에 필요한 모든 종속성을 설치할 수

있습니다. 하이브리드 노드 종속성에는 Containerd, kubelet, kubectI 및 AWS SSM 또는 AWS IAM Roles Anywhere 구성 요소가 포함됩니다. `nodeadm install`에서 설치한 구성 요소 및 파일 위치에 대한 자세한 내용은 [the section called “하이브리드 노드 nodeadm”](#) 섹션을 참조하세요. `nodeadm install` 프로세스를 위해 온프레미스 방화벽에서 허용해야 하는 도메인에 대한 자세한 내용은 하이브리드 노드용 [the section called “네트워킹 준비”](#)를 참조하세요.

아래 명령을 실행하여 온프레미스 호스트에 하이브리드 노드 종속성을 설치합니다. 아래 명령은 호스트에서 `sudo/root` 액세스 권한이 있는 사용자와 함께 실행해야 합니다.

Important

하이브리드 노드 CLI(`nodeadm`)는 호스트에서 `sudo/root` 액세스 권한이 있는 사용자와 함께 실행해야 합니다.

- 예를 들어 `K8S_VERSION`을 Amazon EKS 클러스터의 Kubernetes 마이너 버전(예: 1.31)으로 바꿉니다. 지원되는 Kubernetes 버전의 전체 목록은 [the section called “Kubernetes 버전”](#) 섹션을 참조하세요.
- `CREDS_PROVIDER`를 사용 중인 온프레미스 자격 증명 공급자로 바꿉니다. 유효한 값은 `ssm`(AWS SSM) 및 `iam-ra`(AWS IAM Roles Anywhere)입니다.

```
nodeadm install K8S_VERSION --credential-provider CREDS_PROVIDER
```

3단계: 클러스터에 하이브리드 노드 연결

하이브리드 노드를 클러스터에 연결하기 전에 Amazon EKS 컨트롤 플레인과 하이브리드 노드 간 통신을 주고받기 위해 클러스터의 온프레미스 방화벽 및 보안 그룹에서 필요한 액세스를 허용했는지 확인합니다. 이 단계에서 발생하는 대부분의 문제는 방화벽 구성, 보안 그룹 구성 또는 하이브리드 노드 IAM 역할 구성과 관련이 있습니다.

Important

하이브리드 노드 CLI(`nodeadm`)는 호스트에서 `sudo/root` 액세스 권한이 있는 사용자와 함께 실행해야 합니다.

1. 각 호스트에 배포 값을 포함하는 `nodeConfig.yaml` 파일을 생성합니다. 사용 가능한 구성 설정에 대한 전체 설명은 [the section called “하이브리드 노드 nodeadm”](#) 섹션을 참조하세요. 하이브리드 노드 IAM 역할에 `eks:DescribeCluster` 작업에 대한 권한이 없는 경우 `nodeConfig.yaml`의 클러스터 섹션에서 Kubernetes API 엔드포인트, 클러스터 CA 번들, Kubernetes 서비스 IPv4 CIDR을 전달해야 합니다.
 - a. 온프레미스 자격 증명 공급자에 대해 AWS SSM 하이브리드 활성화를 사용하는 경우 아래 `nodeConfig.yaml` 예제를 사용합니다.
 - i. `CLUSTER_NAME`를 클러스터 이름으로 바꿉니다.
 - ii. `AWS_REGION`을 클러스터를 호스팅하는 AWS 리전으로 바꿉니다. 예를 들어 `us-west-2`입니다.
 - iii. `ACTIVATION_CODE`를 AWS SSM 하이브리드 활성화를 생성할 때 받은 활성화 코드로 바꿉니다. 자세한 정보는 [the section called “자격 증명 준비”](#)을 참조하세요.
 - iv. `ACTIVATION_ID`를 AWS SSM 하이브리드 활성화를 생성할 때 받은 활성화 ID로 바꿉니다. AWS Systems Manager 콘솔 또는 AWS CLI `aws ssm describe-activations` 명령에서 이 정보를 검색할 수 있습니다.

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: CLUSTER_NAME
    region: AWS_REGION
  hybrid:
    ssm:
      activationCode: ACTIVATION_CODE
      activationId: ACTIVATION_ID
```

- b. 온프레미스 자격 증명 공급자에 AWS IAM Roles Anywhere를 사용하는 경우 아래 `nodeConfig.yaml` 예제를 사용합니다.
 - i. `CLUSTER_NAME`를 클러스터 이름으로 바꿉니다.
 - ii. `AWS_REGION`을 클러스터를 호스팅하는 AWS 리전으로 바꿉니다. 예를 들어 `us-west-2`입니다.
 - iii. `NODE_NAME`을 노드의 이름으로 바꿉니다. 하이브리드 노드 IAM 역할의 신뢰 정책을 `"sts:RoleSessionName": "${aws:PrincipalTag/x509Subject/CN}"` 리소스 조건으로 구성한 경우 노드 이름은 호스트에서 인증서의 CN과 일치해야 합니다. 사용하는 `nodeName`은 64자를 초과할 수 없습니다.

- iv. 하이브리드 노드에 대한 자격 증명 준비 단계에서 구성한 트러스트 앵커의 ARN으로 TRUST_ANCHOR_ARN을 바꿉니다.
- v. [the section called “자격 증명 준비”](#) 단계에서 구성한 트러스트 앵커의 ARN으로 PROFILE_ARN을 바꿉니다.
- vi. ROLE_ARN을 하이브리드 노드 IAM 역할의 ARN으로 바꿉니다.
- vii. CERTIFICATE_PATH를 노드 인증서의 디스크 경로로 바꿉니다. 지정하지 않으면 기본값은 /etc/iam/pki/server.pem입니다.
- viii. KEY_PATH를 인증서 프라이빗 키의 디스크 경로로 바꿉니다. 지정하지 않으면 기본값은 /etc/iam/pki/server.key입니다.

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name: CLUSTER_NAME
    region: AWS_REGION
  hybrid:
    iamRolesAnywhere:
      nodeName: NODE_NAME
      trustAnchorArn: TRUST_ANCHOR_ARN
      profileArn: PROFILE_ARN
      roleArn: ROLE_ARN
      certificatePath: CERTIFICATE_PATH
      privateKeyPath: KEY_PATH
```

2. nodeConfig.yaml로 nodeadm init 명령을 실행하여 하이브리드 노드를 Amazon EKS 클러스터에 연결합니다.

```
nodeadm init -c file:///nodeConfig.yaml
```

위 명령이 성공적으로 완료되면 하이브리드 노드가 Amazon EKS 클러스터에 조인된 것입니다. 클러스터의 컴퓨팅 탭으로 이동하거나([IAM 보안 주체가 볼 수 있는 권한이 있는지 확인](#)) kubectl get nodes를 사용하여 Amazon EKS 콘솔에서 이를 확인할 수 있습니다.

⚠ Important

노드의 상태는 Not Ready이며, 이는 하이브리드 노드에서 실행되는 CNI가 없기 때문에 이러한 상태가 예상됩니다. 노드가 클러스터에 조인하지 않은 경우 [the section called “문제 해결”](#) 섹션을 참조하세요.

4단계: 하이브리드 노드에 대한 CNI 구성

하이브리드 노드가 애플리케이션을 실행할 수 있게 하려면 [the section called “CNI 구성”](#)의 단계를 계속합니다.

Bottlerocket을 실행하는 하이브리드 노드 연결

이 주제에서는 Bottlerocket을 실행하는 하이브리드 노드를 Amazon EKS 클러스터에 연결하는 방법을 설명합니다. [Bottlerocket](#)은 AWS에서 후원하고 지원하는 오픈 소스 Linux 배포판입니다. Bottlerocket은 컨테이너 워크로드 호스팅용으로 특별히 설계되었습니다. Bottlerocket을 사용하면 컨테이너 인프라 업데이트를 자동화하여 컨테이너화된 배포의 가용성을 개선하고 운영 비용을 절감할 수 있습니다. Bottlerocket에는 컨테이너를 실행하는 데 필수적인 소프트웨어만 포함되어 있어 리소스 사용량이 개선되고 보안 위협과 관리 오버헤드가 감소합니다.

Bottlerocket 버전 v1.37.0 이상의 VMware 변형만 EKS Hybrid Nodes에서 지원됩니다. Bottlerocket의 VMware 변형은 Kubernetes 버전 v1.28 이상에서 사용할 수 있습니다. 이러한 변형의 OS 이미지에는 kubelet, containerd, aws-iam-authenticator 및 기타 EKS Hybrid Nodes용 소프트웨어 사전 조건이 포함됩니다. Bottlerocket 부트스트랩 및 관리자 컨테이너에 대한 base64 인코딩된 사용자 데이터가 포함된 Bottlerocket [설정](#) 파일을 사용하여 이러한 구성 요소를 구성할 수 있습니다. 이러한 설정을 구성하면 Bottlerocket이 하이브리드 노드 자격 증명 공급자를 사용하여 하이브리드 노드를 클러스터에 인증할 수 있습니다. 하이브리드 노드가 클러스터에 조인되면 Amazon EKS 콘솔이나 kubectl과 같은 Kubernetes 호환 도구에 Not Ready 상태로 표시됩니다. 이 페이지의 단계를 완료한 후 [the section called “CNI 구성”](#)로 이동하여 하이브리드 노드가 애플리케이션을 실행할 준비가 되게 합니다.

사전 조건

하이브리드 노드를 Amazon EKS 클러스터에 연결하기 전에 사전 조건 단계를 완료했는지 확인합니다.

- 온프레미스 환경에서 Amazon EKS 클러스터를 호스팅하는 AWS 리전으로 네트워크 연결이 가능합니다. 자세한 정보는 [the section called “네트워킹 준비”](#)을 참조하세요.

- 하이브리드 노드 IAM 역할을 생성하고 온프레미스 자격 증명 공급자(AWS Systems Manager 하이브리드 활성화 또는 AWS IAM Roles Anywhere)를 설정했습니다. 자세한 정보는 [the section called “자격 증명 준비”](#)을 참조하세요.
- 하이브리드 노드가 활성화된 Amazon EKS 클러스터를 생성했습니다. 자세한 정보는 [the section called “클러스터 생성”](#)을 참조하세요.
- 하이브리드 노드 IAM 역할을 Kubernetes 역할 기반 액세스 제어(RBAC) 권한과 연결했습니다. 자세한 정보는 [the section called “클러스터 액세스 준비”](#)을 참조하세요.

1단계: Bottlerocket 설정 TOML 파일 생성

하이브리드 노드에 대해 Bottlerocket을 구성하려면 필요한 구성으로 `settings.toml` 파일을 생성해야 합니다. TOML 파일의 내용은 사용 중인 자격 증명 공급자(SSM 또는 IAM Roles Anywhere)에 따라 달라집니다. 이 파일은 Bottlerocket 인스턴스를 프로비저닝할 때 사용자 데이터로 전달됩니다.

SSM

AWS Systems Manager를 자격 증명 공급자로 사용하는 경우 다음 내용이 포함된 `settings.toml` 파일을 생성합니다.

```
[settings.kubernetes]
cluster-name = "<cluster-name>"
api-server = "<api-server-endpoint>"
cluster-certificate = "<cluster-certificate-authority>"
hostname-override = "<hostname>"
provider-id = "eks-hybrid:///<region>/<cluster-name>/<hostname>"
authentication-mode = "aws"

[settings.network]
hostname = "<hostname>"

[settings.aws]
region = "<region>"

[settings.kubernetes.node-labels]
"eks.amazonaws.com/compute-type" = "hybrid"
"eks.amazonaws.com/hybrid-credential-provider" = "ssm"

[settings.host-containers.admin]
enabled = true
user-data = "<base64-encoded-admin-container-userdata>"
```

```
[settings.bootstrap-containers.eks-hybrid-setup]
mode = "always"
user-data = "<base64-encoded-bootstrap-container-userdata>"

[settings.host-containers.control]
enabled = true
```

자리 표시자를 다음 값으로 바꿉니다.

- **<cluster-name>**: Amazon ECS 클러스터의 이름입니다.
- **<api-server-endpoint>**: 클러스터의 API 서버 엔드포인트입니다.
- **<cluster-certificate-authority>**: 클러스터의 base64로 인코딩된 CA 번들입니다.
- **<region>**: 클러스터를 호스팅하는 AWS 리전입니다(예: "us-east-1").
- **<hostname>**: Bottlerocket 인스턴스의 호스트 이름으로, 노드 이름으로도 구성됩니다. 이 이름은 임의의 고유한 값일 수 있지만 [Kubernetes 객체 이름 지정 규칙](#)을 따라야 합니다. 또한 사용하는 호스트 이름은 64자를 초과할 수 없습니다. 참고: SSM 공급자를 사용하는 경우 인스턴스가 SSM에 등록된 후 이 호스트 이름 및 노드 이름이 관리형 인스턴스 ID(예: mi-* ID)로 바뀝니다.
- **<base64-encoded-admin-container-userdata>**: Bottlerocket 관리자 컨테이너 구성의 base64로 인코딩된 내용입니다. 관리자 컨테이너를 활성화하면 시스템 탐색 및 디버깅을 위해 SSH를 사용하여 Bottlerocket 인스턴스에 연결할 수 있습니다. 필수 설정은 아니지만 용이한 문제 해결을 위해 활성화하는 것이 좋습니다. 관리자 컨테이너를 사용한 인증에 대한 자세한 내용은 [Bottlerocket 관리자 컨테이너 설명서](#)를 참조하세요. 관리자 컨테이너는 예를 들어 SSH 사용자 및 키 입력을 JSON 형식으로 사용합니다.

```
{
  "user": "<ssh-user>",
  "ssh": {
    "authorized-keys": [
      "<ssh-authorized-key>"
    ]
  }
}
```

- **<base64-encoded-bootstrap-container-userdata>**: Bottlerocket 부트스트랩 컨테이너 구성의 base64로 인코딩된 내용입니다. 구성에 대한 자세한 내용은 [Bottlerocket 부트스트랩 컨테이너 설명서](#)를 참조하세요. 부트스트랩 컨테이너는 인스턴스를 AWS SSM 관리형 인스턴스로 등록하고 Amazon EKS 클러스터에서 Kubernetes 노드로 조인하는 역할을 합니다. 부트스트랩 컨테이너로 전

달되는 사용자 데이터는 이전에 생성한 SSM 하이브리드 활성화 코드 및 ID를 입력으로 수락하는 명령 간접 호출의 형태를 취합니다.

```
eks-hybrid-ssm-setup --activation-id=<activation-id> --activation-code=<activation-code> --region=<region>
```

IAM Roles Anywhere

AWS IAM Roles Anywhere를 자격 증명 공급자로 사용하는 경우 다음 내용이 포함된 `settings.toml` 파일을 생성합니다.

```
[settings.kubernetes]
cluster-name = "<cluster-name>"
api-server = "<api-server-endpoint>"
cluster-certificate = "<cluster-certificate-authority>"
hostname-override = "<hostname>"
provider-id = "eks-hybrid:///<region>/<cluster-name>/<hostname>"
authentication-mode = "aws"

[settings.network]
hostname = "<hostname>"

[settings.aws]
region = "<region>"
config = "<base64-encoded-aws-config-file>"

[settings.kubernetes.node-labels]
"eks.amazonaws.com/compute-type" = "hybrid"
"eks.amazonaws.com/hybrid-credential-provider" = "iam-ra"

[settings.host-containers.admin]
enabled = true
user-data = "<base64-encoded-admin-container-userdata>"

[settings.bootstrap-containers.eks-hybrid-setup]
mode = "always"
user-data = "<base64-encoded-bootstrap-container-userdata>"
```

자리 표시자를 다음 값으로 바꿉니다.

- `<cluster-name>`: Amazon ECS 클러스터의 이름입니다.

- **<api-server-endpoint>**: 클러스터의 API 서버 엔드포인트입니다.
- **<cluster-certificate-authority>**: 클러스터의 base64로 인코딩된 CA 번들입니다.
- **<region>**: 클러스터를 호스팅하는 AWS 리전입니다(예: "us-east-1")
- **<hostname>**: Bottlerocket 인스턴스의 호스트 이름으로, 노드 이름으로도 구성됩니다. 이 이름은 임의의 고유한 값일 수 있지만 [Kubernetes 객체 이름 지정 규칙](#)을 따라야 합니다. 또한 사용하는 호스트 이름은 64자를 초과할 수 없습니다. 참고: IAM-RA 공급자를 사용하는 경우 하이브리드 노드 IAM 역할의 신뢰 정책을 "sts:RoleSessionName": "\${aws:PrincipalTag/x509Subject/CN}" 리소스 조건으로 구성했으면 노드 이름은 호스트에서 인증서의 CN과 일치해야 합니다.
- **<base64-encoded-aws-config-file>**: AWS config 파일의 base64로 인코딩된 내용입니다. 파일의 내용은 다음과 같습니다.

```
[default]
credential_process = aws_signing_helper credential-process --certificate /root/.aws/node.crt --private-key /root/.aws/node.key --profile-arn <profile-arn> --role-arn <role-arn> --trust-anchor-arn <trust-anchor-arn> --role-session-name <role-session-name>
```

- **<base64-encoded-admin-container-userdata>**: Bottlerocket 관리자 컨테이너 구성의 base64로 인코딩된 내용입니다. 관리자 컨테이너를 활성화하면 시스템 탐색 및 디버깅을 위해 SSH를 사용하여 Bottlerocket 인스턴스에 연결할 수 있습니다. 필수 설정은 아니지만 용이한 문제 해결을 위해 활성화하는 것이 좋습니다. 관리자 컨테이너를 사용한 인증에 대한 자세한 내용은 [Bottlerocket 관리자 컨테이너 설명서](#)를 참조하세요. 관리자 컨테이너는 예를 들어 SSH 사용자 및 키 입력을 JSON 형식으로 사용합니다.

```
{
  "user": "<ssh-user>",
  "ssh": {
    "authorized-keys": [
      "<ssh-authorized-key>"
    ]
  }
}
```

- **<base64-encoded-bootstrap-container-userdata>**: Bottlerocket 부트스트랩 컨테이너 구성의 base64로 인코딩된 내용입니다. 구성에 대한 자세한 내용은 [Bottlerocket 부트스트랩 컨테이너 설명서](#)를 참조하세요. 부트스트랩 컨테이너는 인스턴스에서 IAM Roles Anywhere 호스트 인증서 및

인증서 프라이빗 키 파일을 생성하는 역할을 합니다. 그런 다음 `aws_signing_helper`가 Amazon EKS 클러스터로 인증하기 위한 임시 자격 증명을 얻기 위해 사용합니다. 부트스트랩 컨테이너로 전달되는 사용자 데이터는 이전에 생성한 인증서 및 프라이빗 키의 내용을 입력으로 수락하는 명령 간접 호출의 형태를 취합니다.

```
eks-hybrid-iam-ra-setup --certificate=<certificate> --key=<private-key>
```

2단계: 사용자 데이터로 Bottlerocket vSphere VM 프로비저닝

TOML 파일을 구성했으면 vSphere VM 생성 중에 이 파일을 사용자 데이터로 전달합니다. VM이 처음 켜지기 전에 사용자 데이터가 구성되어 있어야 합니다. 따라서 인스턴스를 생성할 때 사용자 데이터를 제공해야 합니다. VM을 미리 생성하려는 경우에는 사용자 데이터를 구성할 때까지 VM이 `poweredOff` 상태여야 합니다. 예를 들어 `govc` CLI를 사용하는 경우:

처음으로 VM 생성

```
govc vm.create \
  -on=true \
  -c=2 \
  -m=4096 \
  -net.adapter=<network-adapter> \
  -net=<network-name> \
  -e guestinfo.userdata.encoding="base64" \
  -e guestinfo.userdata="$(base64 -w0 settings.toml)" \
  -template=<template-name> \
  <vm-name>
```

기존 VM의 사용자 데이터 업데이트

```
govc vm.create \
  -on=false \
  -c=2 \
  -m=4096 \
  -net.adapter=<network-adapter> \
  -net=<network-name> \
  -template=<template-name> \
  <vm-name>

govc vm.change
  -vm <vm-name> \
  -e guestinfo.userdata="$(base64 -w0 settings.toml)" \
```

```
-e guestinfo.userdata.encoding="base64"
```

```
govc vm.power -on <vm-name>
```

위 섹션에서 `-e guestinfo.userdata.encoding="base64"` 옵션은 사용자 데이터가 base64로 인코딩되도록 지정합니다. `-e guestinfo.userdata` 옵션은 `settings.toml` 파일의 base64로 인코딩된 내용을 사용자 데이터로 Bottlerocket 인스턴스에 전달합니다. 자리 표시자를 Bottlerocket OVA 템플릿 및 네트워킹 세부 정보와 같은 특정 값으로 바꿉니다.

3단계: 하이브리드 노드 연결 확인

Bottlerocket 인스턴스가 시작되면 Amazon EKS 클러스터에 조인을 시도합니다. 클러스터의 컴퓨팅 탭으로 이동하거나 다음 명령을 실행하여 Amazon EKS 콘솔에서 연결을 확인할 수 있습니다.

```
kubectl get nodes
```

Important

노드의 상태는 `Not Ready`이며, 이는 하이브리드 노드에서 실행되는 CNI가 없기 때문에 이러한 상태가 예상됩니다. 노드가 클러스터에 조인하지 않은 경우 [the section called “문제 해결”](#) 섹션을 참조하세요.

4단계: 하이브리드 노드에 대한 CNI 구성

하이브리드 노드가 애플리케이션을 실행할 수 있게 하려면 [the section called “CNI 구성”](#)의 단계를 계속합니다.

클러스터의 하이브리드 노드 업그레이드

하이브리드 노드 업그레이드 지침은 Amazon EC2에서 실행되는 자체 관리형 Amazon EKS 노드와 유사합니다. 대상 Kubernetes 버전에서 새 하이브리드 노드를 생성하고, 기존 애플리케이션을 새 Kubernetes 버전의 하이브리드 노드로 정상적으로 마이그레이션하고, 클러스터에서 이전 Kubernetes 버전의 하이브리드 노드를 제거할 수 있습니다. 업그레이드를 시작하기 전에 업그레이드에 대한 [Amazon EKS 모범 사례](#)를 검토해야 합니다. Amazon EKS Hybrid Nodes는 표준 및 확장 지원을 포함하여 클라우드 노드가 있는 Amazon EKS 클러스터에 대해 동일한 [Kubernetes 버전 지원](#)을 제공합니다.

Amazon EKS Hybrid Nodes는 노드에 대해 업스트림 Kubernetes와 동일한 [버전 스큐 정책](#)을 따릅니다. Amazon EKS Hybrid Nodes는 Amazon EKS 컨트롤 플레인보다 최신 버전일 수 없으며, 하이브리드

드 노드는 Amazon EKS 컨트롤 플레인 마이너 버전보다 최대 3개의 Kubernetes 마이너 버전일 수 있습니다.

전환 마이그레이션 업그레이드 전략을 위해 대상 Kubernetes 버전에 새 하이브리드 노드를 생성할 예비 용량이 없는 경우 Amazon EKS Hybrid Nodes CLI(nodeadm)를 사용하여 하이브리드 노드의 Kubernetes 버전을 업그레이드할 수 있습니다.

Important

nodeadm을 사용하여 하이브리드 노드를 현재 상태로 업그레이드하는 경우 이전 버전의 Kubernetes 구성 요소가 종료되고 새 Kubernetes 버전 구성 요소가 설치 및 시작되는 프로세스 중에 노드에 대한 가동 중지가 발생합니다.

사전 조건

업그레이드하기 전에 다음 사전 조건을 완료했는지 확인합니다.

- 하이브리드 노드 업그레이드의 대상 Kubernetes 버전은 Amazon EKS 컨트롤 플레인 버전과 같거나 낮아야 합니다.
- 전환 마이그레이션 업그레이드 전략을 따르는 경우 대상 Kubernetes 버전에 설치 중인 새 하이브리드 노드가 [the section called “사전 조건”](#) 요구 사항을 충족해야 합니다. 여기에는 Amazon EKS 클러스터 생성 중에 전달한 원격 노드 네트워크 CIDR 내에 IP 주소가 있는 것이 포함됩니다.
- 전환 마이그레이션과 인플레이스 업그레이드의 경우 하이브리드 노드는 하이브리드 노드 종속성의 새 버전을 가져오는 데 [필요한 도메인](#)에 액세스할 수 있어야 합니다.
- Amazon EKS Kubernetes API 엔드포인트와 상호 작용하는 데 사용 중인 로컬 시스템 또는 인스턴스에 kubectl이 설치되어 있어야 합니다.
- CNI 버전은 업그레이드하려는 Kubernetes 버전을 지원해야 합니다. 그렇지 않은 경우 하이브리드 노드를 업그레이드하기 전에 CNI 버전을 업그레이드합니다. 자세한 정보는 [the section called “CNI 구성”](#)을 참조하세요.

전환 마이그레이션(블루-그린) 업그레이드

전환 마이그레이션 업그레이드는 대상 Kubernetes 버전을 사용하여 새 호스트에서 새 하이브리드 노드를 생성하고, 기존 애플리케이션을 대상 Kubernetes 버전의 새 하이브리드 노드로 정상적으로 마이그레이션하고, 클러스터에서 이전 Kubernetes 버전의 하이브리드 노드를 제거하는 프로세스를 나타냅니다. 이 전략을 블루-그린 마이그레이션이라고도 합니다.

1. [the section called “하이브리드 노드 연결”](#) 단계에 따라 새 호스트를 하이브리드 노드로 연결합니다. `nodeadm install` 명령을 실행할 때 대상 Kubernetes 버전을 사용합니다.
2. 대상 Kubernetes 버전의 새 하이브리드 노드와 이전 Kubernetes 버전의 하이브리드 노드 간에 통신을 활성화합니다. 이 구성을 사용하면 워크로드를 대상 Kubernetes 버전의 하이브리드 노드로 마이그레이션하는 동안 포드가 서로 통신할 수 있습니다.
3. 대상 Kubernetes 버전의 하이브리드 노드가 클러스터에 성공적으로 조인되었고 준비됨 상태인지 확인합니다.
4. 다음 명령을 사용하여 제거하려는 각 노드를 예약 불가로 표시합니다. 이는 바꾸고 있는 노드에서 새 포드가 예약되거나 다시 예약되지 않도록 하기 위한 것입니다. 자세한 내용은 Kubernetes 문서의 [kubectl cordon](#)을 참조하세요. `NODE_NAME`을 이전 Kubernetes 버전의 하이브리드 노드 이름으로 바꿉니다.

```
kubectl cordon NODE_NAME
```

다음 코드 조각을 사용하여 특정 Kubernetes 버전(이 경우 1.28)의 모든 노드를 식별하고 연결할 수 있습니다.

```
K8S_VERSION=1.28
for node in $(kubectl get nodes -o json | jq --arg K8S_VERSION
"$K8S_VERSION" -r '.items[] | select(.status.nodeInfo.kubeletVersion |
match("\($K8S_VERSION)")).metadata.name')
do
    echo "Cordoning $node"
    kubectl cordon $node
done
```

5. 현재 배포가 하이브리드 노드에서 2개 미만의 CoreDNS 복제본을 실행하고 있는 경우, 배포를 복제본 2개 이상으로 확장합니다. 정상 작업 중에 복원력을 위해 하이브리드 노드에서 최소 2개의 CoreDNS 복제본을 실행하는 것이 좋습니다.

```
kubectl scale deployments/coredns --replicas=2 -n kube-system
```

6. 다음 명령을 사용하여 클러스터에서 제거할 기존 Kubernetes 버전의 각 하이브리드 노드를 드레인합니다. 노드 드레이닝에 대한 자세한 내용은 Kubernetes 설명서의 [Safely Drain a Node](#)을 참조하세요. `NODE_NAME`을 이전 Kubernetes 버전의 하이브리드 노드 이름으로 바꿉니다.

```
kubectl drain NODE_NAME --ignore-daemonsets --delete-emptydir-data
```

다음 코드 조각을 사용하여 특정 Kubernetes 버전(이 경우 1.28)의 모든 노드를 식별하고 드레이닝할 수 있습니다.

```
K8S_VERSION=1.28
for node in $(kubectl get nodes -o json | jq --arg K8S_VERSION
"$K8S_VERSION" -r '.items[] | select(.status.nodeInfo.kubeletVersion |
match("\($K8S_VERSION)")).metadata.name')
do
    echo "Draining $node"
    kubectl drain $node --ignore-daemonsets --delete-emptydir-data
done
```

7. nodeadm을 사용하여 호스트에서 하이브리드 노드 아티팩트를 중지하고 제거할 수 있습니다. root/sudo 권한이 있는 사용자와 함께 nodeadm을 실행해야 합니다. 노드에 포드가 남아 있는 경우 기본적으로 nodeadm uninstall이 진행되지 않습니다. 자세한 정보는 [the section called “하이브리드 노드 nodeadm”](#) 섹션을 참조하세요.

```
nodeadm uninstall
```

8. 하이브리드 노드 아티팩트가 중지 및 제거된 상태에서, 클러스터에서 노드 리소스를 제거합니다.

```
kubectl delete node node-name
```

다음 코드 조각을 사용하여 특정 Kubernetes 버전(이 경우 1.28)의 모든 노드를 식별하고 삭제할 수 있습니다.

```
K8S_VERSION=1.28
for node in $(kubectl get nodes -o json | jq --arg K8S_VERSION
"$K8S_VERSION" -r '.items[] | select(.status.nodeInfo.kubeletVersion |
match("\($K8S_VERSION)")).metadata.name')
do
    echo "Deleting $node"
    kubectl delete node $node
done
```

9. CNI 선택에 따라 위의 단계를 실행한 후 하이브리드 노드에 아티팩트가 남아 있을 수 있습니다. 자세한 정보는 [the section called “CNI 구성”](#)을 참조하세요.

인플레이스 업그레이드

인플레이스 업그레이드 프로세스는 `nodeadm upgrade`를 사용하여 새로운 물리적 또는 가상 호스트 및 전환 마이그레이션 전략을 사용하지 않고 하이브리드 노드용 Kubernetes 버전을 업그레이드하는 것을 말합니다. 이 `nodeadm upgrade` 프로세스는 하이브리드 노드에서 실행되는 기존 구형 Kubernetes 구성 요소를 종료하고, 기존 구형 Kubernetes 구성 요소를 제거하고, 새 대상 Kubernetes 구성 요소를 설치하고, 새 대상 Kubernetes 구성 요소를 시작합니다. 하이브리드 노드에서 실행되는 애플리케이션에 미치는 영향을 최소화하기 위해 한 번에 하나의 노드를 업그레이드하는 것이 좋습니다. 이 프로세스의 지속 시간은 네트워크 대역폭 및 지연 시간에 따라 달라집니다.

1. 다음 명령을 사용하여 업그레이드하는 노드를 예약 불가로 표시합니다. 이는 업그레이드하는 노드에서 새 포드가 예약되거나 다시 예약되지 않도록 하기 위한 것입니다. 자세한 내용은 Kubernetes 문서의 [kubectl cordon](#)을 참조하세요. `NODE_NAME`을 업그레이드하려는 하이브리드 노드의 이름으로 바꿉니다.

```
kubectl cordon NODE_NAME
```

2. 다음 명령을 사용하여 업그레이드하려는 노드를 드레이닝합니다. 노드 드레이닝에 대한 자세한 내용은 Kubernetes 설명서의 [Safely Drain a Node](#)을 참조하세요. `NODE_NAME`을 업그레이드하려는 하이브리드 노드의 이름으로 바꿉니다.

```
kubectl drain NODE_NAME --ignore-daemonsets --delete-emptydir-data
```

3. 업그레이드하려는 하이브리드 노드에서 `nodeadm upgrade`를 실행합니다. `root/sudo` 권한이 있는 사용자와 함께 `nodeadm`을 실행해야 합니다. 노드 이름은 AWS SSM 및 AWS IAM Roles Anywhere 자격 증명 공급자 모두에 대한 업그레이드를 통해 보존됩니다. 업그레이드 프로세스 중에는 자격 증명 공급자를 변경할 수 없습니다. [the section called “하이브리드 노드 nodeadm”](#)의 구성 값은 `nodeConfig.yaml` 섹션을 참조하세요. `K8S_VERSION`을 업그레이드할 대상 Kubernetes 버전으로 바꿉니다.

```
nodeadm upgrade K8S_VERSION -c file://nodeConfig.yaml
```

4. 업그레이드한 후 노드에서 포드를 예약하도록 허용하려면 다음을 입력합니다. `NODE_NAME`을 노드의 이름으로 바꿉니다.

```
kubectl uncordon NODE_NAME
```

5. 하이브리드 노드의 상태를 확인하고 노드가 종료될 때까지 기다렸다가 준비 상태로 새 Kubernetes 버전에서 다시 시작합니다.

```
kubectl get nodes -o wide -w
```

하이브리드 노드의 패치 보안 업데이트

이 주제에서는 하이브리드 노드에서 실행되는 특정 패키지 및 종속성에 대한 보안 업데이트의 실행 중 패치 적용을 수행하는 절차를 설명합니다. 가장 좋은 방법은 하이브리드 노드를 정기적으로 업데이트 하여 CVE 및 보안 패치를 받는 것입니다.

Kubernetes 버전을 업그레이드하는 단계는 [the section called “하이브리드 노드 업그레이드”](#) 섹션을 참조하세요.

보안 패치가 필요할 수 있는 소프트웨어의 예로는 containerd가 있습니다.

Containerd

containerd는 EKS Hybrid Nodes의 표준 Kubernetes 컨테이너 런타임 및 핵심 종속성으로, 이미지 가져오기 및 컨테이너 실행 관리를 포함하여 컨테이너 수명 주기를 관리하는 데 사용됩니다. 하이브리드 노드에서는 [nodeadm CLI](#)를 통해 또는 수동으로 containerd를 설치할 수 있습니다. 노드의 운영 체제에 따라 nodeadm은 OS 배포 패키지 또는 Docker 패키지에서 containerd를 설치합니다.

containerd의 CVE가 게시되면 하이브리드 노드에서 다음 옵션으로 패치된 버전의 containerd로 업그레이드합니다.

1단계: 패치가 패키지 관리자에 게시되었는지 확인

해당 보안 게시판을 참조하여 containerd CVE 패치가 각 OS 패키지 관리자에 게시되었는지 확인할 수 있습니다.

- [Amazon Linux 2023](#)
- [RHEL](#)
- [Ubuntu 22.04](#)
- [Ubuntu 22.04](#)
- [Ubuntu 24.04](#)

Docker 리포지토리를 containerd의 소스로 사용하는 경우 [Docker 보안 공지](#)를 확인하여 Docker 리포지토리에서 패치된 버전의 가용성을 식별할 수 있습니다.

2단계: 패치를 설치할 방법 선택

노드에 보안 업그레이드를 패치하고 설치하는 세 가지 방법이 있습니다. 사용할 수 있는 방법은 패키지 관리자의 운영 체제에서 패치가 제공되는지 여부에 따라 다릅니다.

1. 패키지 관리자에 게시되는 `nodeadm upgrade`를 사용하여 패치를 설치합니다. [2단계 a](#)를 참조하세요.
2. 패키지 관리자를 사용하여 패치를 직접 설치합니다. [2단계 b](#)를 참조하세요.
3. 패키지 관리자에 게시되지 않은 사용자 지정 패치를 설치합니다. `containerd`의 사용자 지정 패치에는 특별한 고려 사항이 있습니다. [2단계 c](#)를 참조하세요.

2단계 a: **nodeadm upgrade**를 사용하여 패치 적용

`containerd` CVE 패치가 OS 또는 Docker 리포지토리(Apt 또는 RPM)에 게시되었는지 확인한 후 `nodeadm upgrade` 명령을 사용하여 `containerd`의 최신 버전으로 업그레이드할 수 있습니다. Kubernetes 버전 업그레이드가 아니므로 현재 Kubernetes 버전을 `nodeadm` 업그레이드 명령에 전달해야 합니다.

```
nodeadm upgrade K8S_VERSION --config-source file:///root/nodeConfig.yaml
```

2단계 b: 운영 체제 패키지 관리자로 패치 적용

아니면 해당 패키지 관리자를 통해 업데이트하고 이를 사용하여 다음과 같이 `containerd` 패키지를 업그레이드할 수도 있습니다.

Amazon Linux 2023

```
sudo yum update -y
sudo yum install -y containerd
```

RHEL

```
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo
sudo yum update -y
sudo yum install -y containerd
```

Ubuntu

```

sudo mkdir -p /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/
docker.asc
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update -y
sudo apt install -y --only-upgrade containerd.io

```

2단계 c: 패키지 관리자에 **Containerd** CVE 패치가 게시되지 않음

GitHub 릴리스와 같이 패키지 관리자가 아닌 다른 방법으로만 패치 적용된 containerd 버전이 제공된 경우 공식 GitHub 사이트에서 containerd를 설치할 수 있습니다.

1. 머신이 이미 클러스터에 하이브리드 노드로 조인한 경우 `nodeadm uninstall` 명령을 실행해야 합니다.
2. 공식 containerd 바이너리를 설치합니다. GitHub에서 단계 [공식 설치 단계](#)를 사용할 수 있습니다.
3. `--containerd-source` 인수가 `none`로 설정된 상태에서 `nodeadm install` 명령을 실행하면 nodeadm를 통해 containerd 설치를 건너뛵니다. 노드가 실행 중인 모든 운영 체제에 대해 containerd 소스의 `none` 값을 사용할 수 있습니다.

```

nodeadm install K8S_VERSION --credential-provider CREDS_PROVIDER --containerd-source
none

```

하이브리드 노드 제거

이 주제에서는 Amazon EKS 클러스터에서 하이브리드 노드를 삭제하는 방법을 설명합니다. [kubecti](#)과 같은 Kubernetes 호환 도구를 선택하여 하이브리드 노드를 삭제해야 합니다. 노드 객체가 Amazon EKS 클러스터에서 제거되면 하이브리드 노드에 대한 요금 부과가 중지됩니다. 하이브리드 노드 요금에 대한 자세한 내용은 [Amazon EKS Pricing](#)을 참조하세요.

Important

노드를 제거하면 노드에서 실행되는 워크로드가 중단됩니다. 하이브리드 노드를 삭제하기 전에 먼저 노드를 드레인하여 포드를 다른 활성 노드로 이동하는 것이 좋습니다. 노드 드레인에 대한 자세한 내용은 Kubernetes 설명서의 [Safely Drain a Node](#)을 참조하세요.

Amazon EKS 클러스터의 Kubernetes API 엔드포인트와 상호 작용하는 데 사용하는 로컬 시스템 또는 인스턴스에서 아래 `kubectl` 단계를 실행합니다. 특정 `kubeconfig` 파일을 사용하는 경우 `--kubeconfig` 플래그를 사용합니다.

1단계: 노드 나열

```
kubectl get nodes
```

2단계: 노드 드레인

`kubectl drain` 명령에 대한 자세한 내용은 Kubernetes 설명서의 [kubectl drain](#)을 참조하세요.

```
kubectl drain --ignore-daemonsets <node-name>
```

3단계: 하이브리드 노드 아티팩트 중지 및 제거

Amazon EKS Hybrid Nodes CLI(`nodeadm`)를 사용하여 호스트에서 하이브리드 노드 아티팩트를 중지하고 제거할 수 있습니다. `root/sudo` 권한이 있는 사용자와 함께 `nodeadm`를 실행해야 합니다. 노드에 포드가 남아 있는 경우 기본적으로 `nodeadm uninstall`이 진행되지 않습니다. AWS Systems Manager(SSM)를 자격 증명 공급자로 사용하는 경우 `nodeadm uninstall` 명령은 호스트를 AWS SSM 관리형 인스턴스로 등록 취소합니다. 자세한 내용은 [the section called “하이브리드 노드 nodeadm”](#) 단원을 참조하십시오.

```
nodeadm uninstall
```

4단계: 클러스터에서 노드 삭제

하이브리드 노드 아티팩트가 중지 및 제거된 상태에서, 클러스터에서 노드 리소스를 제거합니다.

```
kubectl delete node <node-name>
```

5단계: 남은 아티팩트 확인

CNI 선택에 따라 위의 단계를 실행한 후 하이브리드 노드에 아티팩트가 남아 있을 수 있습니다. 자세한 내용은 [the section called “CNI 구성”](#)를 참조하십시오.

하이브리드 노드에 대한 CNI, 추가 기능 및 웹훅 구성

하이브리드 노드용 클러스터를 구성한 후 하이브리드 노드에는 컨테이너 네트워크 인터페이스(CNI), 추가 기능, 웹훅 및 가능한 프록시 설정에 대한 구성이 필요합니다. 하이브리드 노드와 호환되는 EKS 및 커뮤니티 추가 기능의 전체 목록은 [the section called “추가 기능 구성”](#) 섹션을 참조하세요.

EKS 클러스터 인사이트 EKS에는 클러스터 또는 워크로드의 기능을 손상시킬 수 있는 EKS 하이브리드 노드 설정의 잘못된 구성에 대한 인사이트 검사가 포함됩니다. 클러스터 인사이트에 대한 자세한 내용은 [the section called “클러스터 인사이트”](#) 섹션을 참조하세요.

다음은 하이브리드 노드에 대해 구성이 다른 추가 기능 및 기타 구성 요소를 나열합니다.

- 컨테이너 네트워크 인터페이스(CNI): [Cilium](#) 및 [Calico](#)의 핵심 기능은 하이브리드 노드와 함께 사용할 수 있습니다. 원하는 도구(예: Helm)를 사용하여 하이브리드 노드에서 CNI를 관리할 수 있습니다. AWS VPC CNI는 하이브리드 노드와 함께 사용할 수 없습니다. 자세한 내용은 [the section called “CNI 구성”](#) 섹션을 참조하세요.
- CoreDNS 및 **kube-proxy**: CoreDNS 및 kube-proxy는 하이브리드 노드가 EKS 클러스터에 조인할 때 자동으로 설치됩니다. 이러한 추가 기능은 클러스터 생성 후 EKS 추가 기능으로 관리할 수 있습니다.
- 수신 및 로드 밸런싱: AWS Direct Connect 또는 AWS Site-to-Site VPN과 연결된 하이브리드 노드의 워크로드에 대상 유형 ip와 함께 AWS 로드 밸런서 컨트롤러 및 Application Load Balancer(ALB) 또는 Network Load Balancer(NLB)를 사용할 수 있습니다. 또는 온프레미스 환경에 로컬로 유지되는 애플리케이션 트래픽에 수신 컨트롤러 또는 로드 밸런서를 사용할 수 있습니다.
- 지표: 하이브리드 노드에서 Amazon Managed Service for Prometheus(AMP) 에이전트 없는 스크레이퍼, AWS Distro for Open Telemetry(ADOT), Amazon CloudWatch 관찰성 에이전트를 사용할 수 있습니다. 하이브리드 노드의 포드 지표에 AMP 에이전트 없는 스크레이퍼를 사용하려면 EKS 클러스터에 사용하는 VPC에서 포드에 액세스할 수 있어야 합니다.
- 로그: 하이브리드 노드 지원 클러스터에 대해 EKS 컨트롤 플레인 로깅을 활성화할 수 있습니다. 하이브리드 노드 및 포드 로깅에 ADOT EKS 추가 기능과 Amazon CloudWatch 관찰성 에이전트 EKS 추가 기능을 사용할 수 있습니다.
- Pod Identity 및 IRSA: 하이브리드 노드에서 실행되는 애플리케이션과 함께 EKS Pod Identity와 서비스 계정에 대한 IAM 역할(IRSA)을 사용하여 다른 AWS 서비스의 하이브리드 노드에서 실행되는 포드에 대한 세분화된 액세스를 활성화할 수 있습니다.
- 웹훅: 웹훅을 실행하는 경우 온프레미스 포드 네트워크를 라우팅할 수 없을 때 선택적으로 클라우드 노드에서 웹훅을 실행하기 위한 고려 사항 및 단계는 [the section called “웹훅 구성”](#) 섹션을 참조하세요.

- 프록시: 데이터 센터 또는 엣지 환경에서 나가는 트래픽에 대해 온프레미스 환경에서 프록시 서버를 사용하는 경우 프록시 서버를 사용하도록 하이브리드 노드 및 클러스터를 구성할 수 있습니다. 자세한 내용은 [the section called “프록시 구성”](#) 섹션을 참조하세요.

주제

- [하이브리드 노드에 대한 CNI 구성](#)
- [하이브리드 노드에 대한 추가 기능 구성](#)
- [하이브리드 노드용 웹훅 구성](#)
- [하이브리드 노드용 프록시 구성](#)

하이브리드 노드에 대한 CNI 구성

Cilium 및 Calico는 Amazon EKS Hybrid Nodes용 컨테이너 네트워킹 인터페이스(CNI)로 지원됩니다. 워크로드에 대비하려면 하이브리드 노드용 CNI를 설치해야 합니다. 하이브리드 노드는 CNI가 실행될 때까지 Not Ready 상태로 표시됩니다. Helm과 같은 도구를 선택하여 이러한 CNI를 관리할 수 있습니다. Amazon VPC CNI는 하이브리드 노드와 호환되지 않으며 VPC CNI는 `eks.amazonaws.com/compute-type: hybrid` 레이블에 대한 반선택으로 구성됩니다.

CNI 버전 호환성

Amazon EKS에서 지원되는 모든 Kubernetes 버전에 대해 EKS 하이브리드 노드에 대해 Cilium 버전 `v1.16.x`가 지원되고 권장됩니다.

Calico 버전 `v3.29.x`는 Amazon EKS에서 지원되는 모든 Kubernetes 버전에 대해 EKS 하이브리드 노드에 지원되고 권장됩니다.

Amazon EKS에서 지원되는 Kubernetes 버전 목록은 [Kubernetes 버전 지원](#)을 참조하세요. EKS Hybrid Nodes는 클라우드 노드가 있는 Amazon EKS 클러스터와 동일한 Kubernetes 버전 지원을 제공합니다.

지원되는 기능

AWS는 하이브리드 노드에서 사용할 수 있는 Cilium 및 Calico의 다음 기능을 지원합니다. AWS 지원 범위 외에 기능을 사용하려는 경우 해당 플러그인에 대한 상용 지원을 받거나 CNI 플러그인 프로젝트의 문제를 해결하고 프로젝트 수정에 기여할 수 있는 전문 지식을 내부적으로 보유하는 것이 좋습니다.

Feature	Cilium	Calico
Kubernetes 네트워크 적합성	예	예
컨트롤 플레인과 노드 연결	예	예
컨트롤 플레인과 포드 연결	예	예
수명 주기 관리	설치, 업그레이드, 삭제	설치, 업그레이드, 삭제
네트워킹 모드	VXLAN	VXLAN
IP 주소 관리(IPAM)	클러스터 범위(Cilium IPAM)	Calico IPAM
IP 패밀리	IPv4	IPv4
BGP	예(Cilium 컨트롤 플레인)	예

Cilium 고려 사항

- 기본적으로 Cilium은 [캡슐화 방법](#)으로 VXLAN을 사용하여 오버레이/터널 모드에서 실행되도록 구성됩니다. 이 모드는 기본 물리적 네트워크에 대한 요구 사항이 가장 적습니다.
- 기본적으로 Cilium은 클러스터를 떠나는 모든 포드 트래픽의 소스 IP 주소를 노드의 IP 주소로 [마스커레이딩](#)합니다. 이를 통해 클러스터에 원격 포드 네트워크가 구성되어 있는지 여부와 무관하게 하이브리드 노드에서 Cilium을 실행할 수 있습니다. 마스커레이딩을 비활성화한 경우 온프레미스 네트워크에서 포드 CIDR이 라우팅 가능해야 하고 원격 포드 네트워크로 Amazon EKS 클러스터를 구성해야 합니다.
- 하이브리드 노드에서 웹후크를 실행하는 경우 포드 CIDR이 온프레미스 네트워크에서 라우팅 가능해야 하고 원격 포드 네트워크로 Amazon EKS 클러스터를 구성해야 합니다. 온프레미스 네트워크에서 포드 CIDR이 라우팅 가능하지 않은 경우 동일한 클러스터의 클라우드 노드에서 웹후크를 실행하는 것이 좋습니다. 자세한 내용은 [하이브리드 노드용 웹후크 구성](#)을 참조하세요.
- 포드 CIDR이 온프레미스 네트워크에서 라우팅 가능하도록 설정하는 일반적인 방법은 BGP로 포드 주소를 알리는 것입니다. Cilium에서 BGP를 사용하려면 Helm 구성에서 `bgpControlPlane.enabled: true`를 설정해야 합니다. Cilium의 BGP 지원에 대한 자세한 내용은 Cilium 설명서의 [Cilium BGP Control Plane](#)을 참조하세요.
- Cilium의 기본 IP Address Management(IPAM)를 [Cluster Scope](#)라고 하며, 여기서 Cilium 연산자는 사용자가 구성한 포드 CIDR을 기반으로 각 노드에 IP 주소를 할당합니다. 포드 CIDR은

`clusterPoolIPv4PodCIDRList` Helm 값으로 구성되며, 이는 Amazon EKS 클러스터에 대해 구성된 원격 포드 네트워크 CIDR과 일치해야 합니다. Cilium은 `clusterPoolIPv4PodCIDRList`에서 세그먼트를 각 노드에 할당합니다. 노드 세그먼트당 크기는 `clusterPoolIPv4MaskSize` Helm 값으로 구성됩니다. `clusterPoolIPv4PodCIDRList` 및 `clusterPoolIPv4MaskSize`에 대한 자세한 내용은 Cilium 설명서의 [클러스터 풀 확장](#)을 참조하세요.

하이브리드 노드에 Cilium 설치

1. 명령줄 환경에 Helm CLI를 설치했는지 확인합니다. 설치 지침은 [헬름 설정](#)을 참조하세요.
2. Cilium Helm 리포지토리를 설치합니다.

```
helm repo add cilium https://helm.cilium.io/
```

3. `cilium-values.yaml`이라는 YAML 파일을 생성합니다. 다음 예제에서는 `eks.amazonaws.com/compute-type: hybrid` 레이블에 대한 선호도를 설정하여 하이브리드 노드에서만 실행되도록 Cilium을 구성합니다.
 - 원격 포드 네트워크로 Amazon EKS 클러스터를 구성한 경우 `clusterPoolIPv4PodCIDRList`에 대해 동일한 포드 CIDR을 구성합니다. 예를 들어 `10.100.0.0/24`입니다. 오버레이/터널 모드에서 CNI를 실행할 때 온프레미스 포드 CIDR이 온프레미스 노드 CIDR과 중첩되지 않아야 합니다.
 - 노드당 필요한 포드를 기반으로 `clusterPoolIPv4MaskSize`를 구성합니다. 예를 들어, 25로 구성하면 노드당 128개 포드의 세그먼트 크기는 `/25`가 됩니다.
 - 클러스터에 Cilium을 배포한 후에는 `clusterPoolIPv4PodCIDRList` 또는 `clusterPoolIPv4MaskSize`를 변경해서는 안 됩니다. Cilium 설명서의 [클러스터 풀 확장](#)을 참조하세요.
 - Cilium의 Helm 값 전체 목록은 Cilium 설명서의 [Helm reference](#)를 참조하세요.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
            - key: eks.amazonaws.com/compute-type
              operator: In
              values:
                - hybrid
  ipam:
    mode: cluster-pool
```

```

operator:
  clusterPoolIPv4MaskSize: 25
  clusterPoolIPv4PodCIDRList:
    - POD_CIDR
loadBalancer:
  serviceTopology: true
operator:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: eks.amazonaws.com/compute-type
                operator: In
              values:
                - hybrid
  unmanagedPodWatcher:
    restart: false
envoy:
  enabled: false

```

4. 클러스터에 Cilium을 설치합니다.

- CILIUM_VERSION을 Cilium 버전으로 바꿉니다(예: v1.17.0). Cilium 마이너 버전에 대한 최신 패치 버전을 실행하는 것이 좋습니다. 지정된 마이너 Cilium 릴리스에 대한 최신 패치 릴리스는 Cilium 설명서의 [안정적인 릴리스 섹션](#)에서 확인할 수 있습니다.
- 배포에 대해 BGP를 활성화하는 경우 아래 명령에 --set bgpControlPlane.enabled=true 플래그를 추가합니다.
- 특정 kubeconfig 파일을 사용하는 경우 Helm 설치 명령과 함께 --kubeconfig 플래그를 사용합니다.

```

helm install cilium cilium/cilium \
  --version CILIUM_VERSION \
  --namespace kube-system \
  --values cilium-values.yaml

```

5. 다음 명령을 사용하여 Cilium 설치가 성공했는지 확인할 수 있습니다. cilium-operator 배포와 각 하이브리드 노드에서 실행 중인 cilium-agent가 표시됩니다. 또한 하이브리드 노드는 이제 Ready 상태여야 합니다. Cilium용 BGP를 구성하는 방법에 대한 자세한 내용은 다음 단계로 진행하세요.


```
kubectl get pods -n kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
cilium-jjjn8	1/1	Running	0	11m
cilium-operator-d4f4d7fcb-sc5xn	1/1	Running	0	11m

```
kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
mi-04a2cf999b7112233	Ready	<none>	19m	v1.31.0-eks-a737599

6. BGP를 Cilium과 함께 사용하여 온프레미스 네트워크에 포트 주소를 알리려

면bgpControlPlane.enabled: true로 Cilium을 설치해야 합니다. 기존 Cilium 배포에 대해 BGP를 활성화하는 경우 BGP가 이전에 활성화되지 않은 경우 BGP 구성을 적용하려면 Cilium 연산자를 다시 시작해야 합니다. Helm 값에서 operator.rollOutPods를 true로 설정하여 Helm 설치/업그레이드 프로세스의 일부로 Cilium 연산자를 다시 시작할 수 있습니다.

Cilium에서 BGP를 구성하려면 먼저 CiliumBGPClusterConfig 정의가 포함된 cilium-bgp-cluster.yaml이라는 파일을 생성합니다. 네트워크 관리자로부터 다음 정보를 얻어야 할 수 있습니다.

- Cilium을 실행하는 노드의 ASN으로 localASN을 구성합니다.
- 온프레미스 라우터의 ASN으로 peerASN을 구성합니다.
- Cilium을 실행하는 각 노드가 피어링할 온프레미스 라우터 IP로 peerAddress를 구성합니다.

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPClusterConfig
metadata:
  name: cilium-bgp
spec:
  nodeSelector:
    matchExpressions:
      - key: eks.amazonaws.com/compute-type
        operator: In
        values:
          - hybrid
  bgpInstances:
    - name: "rack0"
      localASN: NODES_ASN
```

```
peers:
- name: "onprem-router"
  peerASN: ONPREM_ROUTER_ASN
  peerAddress: ONPREM_ROUTER_IP
  peerConfigRef:
    name: "cilium-peer"
```

7. 클러스터에 Cilium BGP 클러스터 구성을 적용합니다.

```
kubectl apply -f cilium-bgp-cluster.yaml
```

8. CiliumBGPPeerConfig 리소스는 BGP 피어 구성을 정의합니다. 여러 피어가 동일한 구성을 공유하고 공통 CiliumBGPPeerConfig 리소스에 대한 참조를 제공할 수 있습니다. cilium-bgp-peer.yaml이라는 파일을 생성하여 온프레미스 네트워크에 대한 피어 구성을 진행합니다. 구성 옵션의 전체 목록은 Cilium 설명서의 [BGP 피어 구성](#)을 참조하세요.

```
apiVersion: cilium.io/v2alpha1
kind: CiliumBGPPeerConfig
metadata:
  name: cilium-peer
spec:
  timers:
    holdTimeSeconds: 30
    keepAliveTimeSeconds: 10
  gracefulRestart:
    enabled: true
    restartTimeSeconds: 120
  families:
    - afi: ipv4
      safi: unicast
      advertisements:
        matchLabels:
          advertise: "bgp"
```

9. 클러스터에 Cilium BGP 피어 구성을 적용합니다.

```
kubectl apply -f cilium-bgp-peer.yaml
```

10. CiliumBGPAdvertisement 리소스는 다양한 광고 유형 및 이와 관련된 속성을 정의하는 데 사용됩니다. cilium-bgp-advertisement.yaml이라는 파일을 생성하고 실제 설정으로 CiliumBGPAdvertisement 리소스를 구성합니다.

```

apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAvertisement
metadata:
  name: bgp-advertisements
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "PodCIDR"
    - advertisementType: "Service"
    service:
      addresses:
        - ClusterIP
        - ExternalIP
        - LoadBalancerIP

```

11. 클러스터에 Cilium BGP 광고 구성을 적용합니다.

```
kubectl apply -f cilium-bgp-advertisement.yaml
```

`cilium bgp peers` 명령을 사용하여 [Cilium CLI](#)에서 작동하는 BGP 피어링을 확인할 수 있습니다. 환경에 대한 출력에 올바른 값이 표시되고 세션 상태가 `established`로 설정되어야 합니다. 문제 해결에 대한 자세한 내용은 Cilium 설명서의 [Troubleshooting and Operations Guide](#)를 참조하세요.

하이브리드 노드에서 Cilium 업그레이드

Cilium 배포를 업그레이드하기 전에 [Cilium 업그레이드 설명서](#)와 업그레이드 정보를 주의 깊게 검토하여 대상 Cilium 버전의 변경 사항을 이해합니다.

1. 명령줄 환경에 `helm` CLI를 설치했는지 확인합니다. 설치 지침은 [Helm 설명서](#)를 참조하세요.
2. Cilium Helm 리포지토리를 설치합니다.

```
helm repo add cilium https://helm.cilium.io/
```

3. Cilium 업그레이드 사전 검사를 실행합니다. `CILIUM_VERSION`을 대상 Cilium 버전으로 바꿉니다. Cilium 마이너 버전에 대한 최신 패치 버전을 실행하는 것이 좋습니다. 지정된 마이너 Cilium 릴리스에 대한 최신 패치 릴리스는 Cilium 설명서의 [안정적인 릴리스 섹션](#)에서 확인할 수 있습니다.

```
helm install cilium-preflight cilium/cilium --version CILIUM_VERSION \
  --namespace=kube-system \
  --set preflight.enabled=true \
  --set agent=false \
  --set operator.enabled=false
```

4. cilium-preflight.yaml을 적용한 후 READY 포트 수가 실행 중인 Cilium 포트 수와 동일한지 확인합니다.

```
kubectl get ds -n kube-system | sed -n '1p;/cilium/p'
```

NAME	DESIRED	CURRENT	READY	UP-TO-DATE	AVAILABLE	NODE
SELECTOR						
AGE						
cilium	2	2	2	2	2	<none>
1h20m						
cilium-pre-flight-check	2	2	2	2	2	<none>
7m15s						

5. READY 포트 수가 같으면 Cilium 사전 배포도 READY 1/1로 표시되어야 합니다. READY 0/1이 표시되면 업그레이드를 계속하기 전에 [CNP 검증](#) 섹션을 참조하고 배포 관련 문제를 해결합니다.

```
kubectl get deployment -n kube-system cilium-pre-flight-check -w
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
cilium-pre-flight-check	1/1	1	0	12s

6. 사전 배포 삭제

```
helm uninstall cilium-preflight --namespace kube-system
```

7. 정상적인 클러스터 작업 중에는 모든 Cilium 구성 요소가 동일한 버전을 실행해야 합니다. 다음 단계에서는 모든 구성 요소를 하나의 안정적인 릴리스에서 이후 안정적인 릴리스로 업그레이드하는 방법을 설명합니다. 한 마이너 릴리스에서 다른 마이너 릴리스로 업그레이드할 때는 먼저 기존 Cilium 마이너 버전이 최신 패치 릴리스로 업그레이드하는 것이 좋습니다. 종단을 최소화하려면 upgradeCompatibility 옵션을 이 클러스터에 설치된 초기 Cilium 버전으로 설정해야 합니다.

helm 업그레이드 명령을 실행하기 전에 cilium-values.yaml에서 배포 값을 보존하거나 설정에 --set 명령줄 옵션을 사용합니다. 업그레이드 작업은 Cilium ConfigMap을 덮어쓰므로 업그레이드

할 때 구성 값을 전달하는 것이 중요합니다. BGP를 사용하는 경우 값 파일에 이 정보를 제공하는 대신 `--set bgpControlPlane=true` 명령줄 옵션을 사용하는 것이 좋습니다.

```
helm upgrade cilium cilium/cilium --version CILIUM_VERSION \
  --namespace kube-system \
  --set upgradeCompatibility=1.X \
  -f cilium-values.yaml
```

8. (선택 사항) 문제로 인해 업그레이드를 롤백해야 하는 경우 다음 명령을 실행합니다.

```
helm history cilium --namespace kube-system
helm rollback cilium [REVISION] --namespace kube-system
```

하이브리드 노드에서 Cilium 삭제

1. 다음 명령을 실행하여 클러스터에서 모든 Cilium 구성 요소를 제거합니다. 참고로 CNI를 제거하면 노드 및 포드의 상태에 영향을 미칠 수 있으므로 프로덕션 클러스터에서 수행해서는 안 됩니다.

```
helm uninstall cilium --namespace kube-system
```

CNI가 클러스터에서 제거되면 Cilium에서 구성한 인터페이스 및 경로가 기본적으로 제거되지 않습니다. 자세한 내용은 [GitHub issue](#)를 참조하세요.

2. 온디스크 구성 파일 및 리소스를 정리하려면 표준 구성 디렉토리를 사용하는 경우 GitHub의 Cilium 리포지토리에서 [cni-uninstall.sh 스크립트](#)에 표시된 대로 파일을 제거할 수 있습니다.
3. 클러스터에서 Cilium 사용자 지정 리소스 정의(CRD)를 제거하려면 다음 명령을 실행할 수 있습니다.

```
kubectl get crds -oname | grep "cilium" | xargs kubectl delete
```

Calico 고려 사항

- VXLAN을 [캡슐화 방법](#)으로 사용하여 오버레이/터널 모드에서 Calico를 실행하는 것이 좋습니다. 이 모드는 기본 물리적 네트워크에 대한 요구 사항이 가장 적습니다. 다양한 Calico 네트워킹 모드에 대한 자세한 내용은 Calico 설명서의 [Determining the best networking option](#)을 참조하세요.
- natOutgoing을 true로 설정하여 Calico를 실행하는 것이 좋습니다. natOutgoing이 true로 설정되면 클러스터에서 나가는 모든 포드 트래픽의 소스 IP 주소가 노드의 IP 주소로 변환됩니다. 이를

통해 클러스터에 원격 포트 네트워크가 구성되어 있는지 여부와 무관하게 Amazon EKS 클러스터에서 Calico를 실행할 수 있습니다. natOutgoing을 비활성화한 경우 온프레미스 네트워크에서 포트 CIDR이 라우팅 가능해야 하고 원격 포트 네트워크로 Amazon EKS 클러스터를 구성해야 합니다.

- 하이브리드 노드에서 웹후크를 실행하는 경우 포트 CIDR이 온프레미스 네트워크에서 라우팅 가능해야 하고 원격 포트 네트워크로 Amazon EKS 클러스터를 구성해야 합니다. 온프레미스 네트워크에서 포트 CIDR이 라우팅 가능하지 않은 경우 동일한 클러스터의 클라우드 노드에서 웹후크를 실행하는 것이 좋습니다. 자세한 내용은 [하이브리드 노드용 웹후크 구성](#)을 참조하세요.
- 포트 CIDR이 온프레미스 네트워크에서 라우팅 가능하도록 설정하는 일반적인 방법은 BGP로 포트 주소를 알리는 것입니다. Calico에서 BGP를 사용하려면 Helm 구성에서 `installation.calicoNetwork.bgp: Enabled`를 설정해야 합니다. Calico의 BGP 지원에 대한 자세한 내용은 Calico 설명서의 [BGP 피어링 구성](#)을 참조하세요.
- Cilium의 기본 IP Address Management(IPAM)를 [Calico IPAM](#)이라고 하며, 여기서 `calico-ipam` 플러그인은 사용자가 구성한 포트 CIDR을 기반으로 각 노드에 IP 주소를 할당합니다. 포트 CIDR은 `installation.calicoNetwork.ipPools.cidr` Helm 값으로 구성되며, 이는 Amazon EKS 클러스터에 대해 구성한 원격 포트 네트워크 CIDR과 일치해야 합니다. Calico는 `ipPools.cidr`에서 세그먼트를 각 노드에 할당합니다. 노드 세그먼트당 크기는 `ipPools.blockSize` Helm 값으로 구성됩니다. Calico를 사용하는 IPAM에 대한 자세한 내용은 Calico 설명서의 [IP 주소 관리 시작하기](#)를 참조하세요.

하이브리드 노드에 Calico 설치

1. 명령줄 환경에 helm CLI를 설치했는지 확인합니다. 설치 지침은 [Helm 설명서](#)를 참조하세요.
2. Cilium Helm 리포지토리를 설치합니다.

```
helm repo add projectcalico https://docs.tigera.io/calico/charts
```

3. `calico-values.yaml`이라는 YAML 파일을 생성합니다. 다음 예제에서는 `eks.amazonaws.com/compute-type: hybrid` 레이블에 대한 선호도를 설정하여 하이브리드 노드에서만 실행되도록 모든 Calico 구성 요소를 구성합니다.
 - `POD_CIDR`을 포트의 CIDR 범위로 바꿉니다. Amazon EKS 클러스터를 원격 포트 네트워크로 구성한 경우 Calico에 지정하는 `POD_CIDR`은 원격 포트 네트워크와 동일해야 합니다. 예를 들어 `10.100.0.0/24`입니다. 오버레이/터널 모드에서 CNI를 실행할 때 온프레미스 포트 CIDR이 온프레미스 노드 CIDR과 중첩되지 않아야 합니다.
 - `CIDR_SIZE`를 각 노드에 할당하려는 CIDR 세그먼트의 크기로 바꿉니다. 예를 들어, 25로 구성하면 노드당 128개 포트 주소의 세그먼트 크기는 `/25`가 됩니다. CIDR `blockSize` 및 `blockSize` 변경에 대한 자세한 내용은 Calico 설명서의 [Change IP pool block size](#)을 참조하세요.

- 아래 예제에서는 natOutgoing이 활성화되고 bgp가 비활성화됩니다. 대상 구성을 기반으로 이러한 값을 수정합니다.

```
installation:
  enabled: true
  cni:
    type: Calico
    ipam:
      type: Calico
  calicoNetwork:
    bgp: Disabled
    ipPools:
      - cidr: POD_CIDR
        blockSize: CIDR_SIZE
        encapsulation: VXLAN
        natOutgoing: Enabled
        nodeSelector: eks.amazonaws.com/compute-type == "hybrid"
  controlPlaneReplicas: 1
  controlPlaneNodeSelector:
    eks.amazonaws.com/compute-type: hybrid
  calicoNodeDaemonSet:
    spec:
      template:
        spec:
          nodeSelector:
            eks.amazonaws.com/compute-type: hybrid
  csiNodeDriverDaemonSet:
    spec:
      template:
        spec:
          nodeSelector:
            eks.amazonaws.com/compute-type: hybrid
  calicoKubeControllersDeployment:
    spec:
      template:
        spec:
          nodeSelector:
            eks.amazonaws.com/compute-type: hybrid
  typhaDeployment:
    spec:
      template:
        spec:
          nodeSelector:
```

```
eks.amazonaws.com/compute-type: hybrid
```

4. 클러스터에 Calico를 설치합니다.

- CALICO_VERSION을 Calico 버전(예: v3.29.0)으로 바꿉니다. [Calico releases](#)를 참조하여 Calico 마이너 버전에 대한 최신 패치 릴리스를 찾습니다. Calico 마이너 버전에 대한 최신 패치 버전을 실행하는 것이 좋습니다.
- 특정 kubeconfig 파일을 사용하는 경우 --kubeconfig 플래그를 사용합니다.

```
helm install calico projectcalico/tigera-operator \
  --version CALICO_VERSION \
  --namespace kube-system \
  -f calico-values.yaml
```

5. 다음 명령을 사용하여 Calico 설치가 성공했는지 확인할 수 있습니다. tigera-operator 배포, 각 하이브리드 노드에서 실행되는 calico-node 에이전트, 배포된 calico-apiserver, csi-node-driver, calico-kube-controllers가 표시됩니다. 또한 하이브리드 노드는 이제 Ready 상태여야 합니다. natOutgoing: Disabled를 사용하는 경우 온프레미스 네트워크에 포트 주소를 알리기 전까지는 모든 Calico 구성 요소가 성공적으로 시작되지 않습니다. Calico용 BGP를 구성하는 방법에 대한 자세한 내용은 다음 단계로 진행하세요.

```
kubectl get pods -A
```

NAMESPACE	NAME	READY	STATUS	
RESTARTS	AGE			
calico-apiserver	calico-apiserver-6c77bb6d46-2n8mq	1/1	Running	0
69s				
calico-system	calico-kube-controllers-7c5f8556b5-7h267	1/1	Running	0
68s				
calico-system	calico-node-s5nnk	1/1	Running	0
68s				
calico-system	calico-typha-6487cc9d8c-wc5jm	1/1	Running	0
69s				
calico-system	csi-node-driver-cv42d	2/2	Running	0
68s				
kube-system	coredns-7bb495d866-2lc9v	1/1	Running	0
6m27s				
kube-system	coredns-7bb495d866-2t8ln	1/1	Running	0
157m				
kube-system	kube-proxy-lxzxh	1/1	Running	0
18m				

kube-system 90s	tigera-operator-f8bc97d4c-28b4d	1/1	Running	0

```
kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
mi-0c6ec2f6f79176565	Ready	<none>	5h13m	v1.31.0-eks-a737599

6. BGP 없이 Calico를 설치한 경우 이 단계를 건너뛰십시오. BGP를 구성하려면 BGPPeer 구성 및 BGPConfiguration을 사용하여 calico-bgp.yaml이라는 파일을 생성합니다. BGPPeer와 BGPConfiguration을 구분하는 것이 중요합니다. BGPPeer는 Calico 클러스터의 노드가 피어링할 BGP 지원 라우터 또는 원격 리소스입니다. BGPPeer 구성의 asNumber는 Cilium 설정 peerASN과 유사합니다. BGPConfiguration은 각 Calico 노드에 적용되며 BGPConfiguration의 asNumber는 Cilium 설정 localASN과 동일합니다. 아래 예제의 ONPREM_ROUTER_IP, ONPREM_ROUTER_ASN, LOCAL_ASN을 네트워크 관리자로부터 얻어야 할 수 있는 온프레미스 환경의 값으로 바꿉니다. keepOriginalNextHop: true 설정은 각 노드가 소유한 포트 네트워크 CIDR만 광고하도록 하는 데 사용됩니다.

```
apiVersion: projectcalico.org/v3
kind: BGPPeer
metadata:
  name: calico-hybrid-nodes
spec:
  peerIP: ONPREM_ROUTER_IP
  asNumber: ONPREM_ROUTER_ASN
  keepOriginalNextHop: true
---
apiVersion: projectcalico.org/v3
kind: BGPConfiguration
metadata:
  name: default
spec:
  nodeToNodeMeshEnabled: false
  asNumber: LOCAL_ASN
```

7. 파일을 클러스터에 적용합니다.

```
kubectl apply -f calico-bgp.yaml
```

8. 다음 명령을 사용하여 Calico 포드가 실행 중인지 확인합니다.

```
kubectl get pods -n calico-system -w
```

NAMESPACE	NAME	READY	STATUS	
RESTARTS	AGE			
calico-apiserver	calico-apiserver-598bf99b6c-2vltk	1/1	Running	0
	3h24m			
calico-system	calico-kube-controllers-75f84bbfd6-zwmnx	1/1	Running	31
(59m ago)	3h20m			
calico-system	calico-node-9b2pg	1/1	Running	0
	5h17m			
calico-system	calico-typha-7d55c76584-kxtnq	1/1	Running	0
	5h18m			
calico-system	csi-node-driver-dmnmm	2/2	Running	0
	5h18m			
kube-system	coredns-7bb495d866-dtn4z	1/1	Running	0
	6h23m			
kube-system	coredns-7bb495d866-mk7j4	1/1	Running	0
	6h19m			
kube-system	kube-proxy-vms28	1/1	Running	0
	6h12m			
kube-system	tigera-operator-55f9d9d565-jj9bg	1/1	Running	0
	73m			

이 단계에서 문제가 발생하면 Calico 설명서의 [troubleshooting guidance](#)를 참조하세요.

하이브리드 노드에서 Calico 업그레이드

Calico 배포를 업그레이드하기 전에 [Calico 업그레이드 설명서](#)와 [릴리스 정보](#)를 주의 깊게 검토하여 대상 Calico 버전의 변경 사항을 이해합니다. 업그레이드 단계는 Helm 사용 여부, Calico 연산자, 데이터 저장소 유형에 따라 달라집니다. 아래 단계에서는 Helm 사용을 가정합니다.

1. 업그레이드하려는 Calico 버전의 연산자 매니페스트를 다운로드합니다. 예를 들어 CALICO_VERSION을 업그레이드하려는 버전으로 바꿉니다(예: v3.29.0). major.minor.patch 앞에 v를 추가해야 합니다.

```
kubectl apply --server-side --force-conflicts \
  -f https://raw.githubusercontent.com/projectcalico/calico/CALICO_VERSION/
  manifests/operator-crd.yaml
```

2. `helm upgrade`를 실행하여 Calico 배포를 업그레이드합니다. 예를 들어 `CALICO_VERSION`을 업그레이드하려는 버전으로 바꿉니다(예: `v3.29.0`). Calico를 설치하는 데 사용한 구성 값에서 `calico-values.yaml` 파일을 생성합니다.

```
helm upgrade calico projectcalico/tigera-operator \
  --version CALICO_VERSION \
  --namespace kube-system \
  -f calico-values.yaml
```

하이브리드 노드에서 Calico 삭제

1. 다음 명령을 실행하여 클러스터에서 Calico 구성 요소를 제거합니다. CNI를 제거하면 노드 및 포드의 상태에 영향을 미칠 수 있으므로 프로덕션 클러스터에서 수행해서는 안 됩니다. Calico를 `kube-system`과 다른 네임스페이스에 설치한 경우 아래 명령의 네임스페이스를 변경합니다.

```
helm uninstall calico --namespace kube-system
```

클러스터에서 CNI를 제거하면 Calico에서 구성한 인터페이스 및 경로가 기본적으로 제거되지 않습니다.

2. 온디스크 구성 파일 및 리소스를 정리하려면 `/opt/cni` 및 `/etc/cni` 디렉터리에서 Calico 파일을 제거합니다.
3. 클러스터에서 Calico CRD를 제거하려면 다음 명령을 실행합니다.

```
kubectl get crds -oname | grep "calico" | xargs kubectl delete
```

```
kubectl get crds -oname | grep "tigera" | xargs kubectl delete
```

하이브리드 노드에 대한 추가 기능 구성

이 페이지에서는 Amazon EKS Hybrid Nodes에서 AWS 추가 기능 및 커뮤니티 추가 기능을 실행하기 위한 고려 사항을 설명합니다. Amazon EKS 추가 기능과 클러스터에서 추가 기능을 생성, 업그레이드, 제거하는 프로세스에 대한 자세한 내용은 [the section called “Amazon EKS 추가 기능”](#) 섹션을 참조하세요. 이 페이지에서 별도로 명시되지 않는 한 Amazon EKS 추가 기능을 생성, 업그레이드, 제거하는 프로세스는 하이브리드 노드가 있는 Amazon EKS 클러스터의 경우 AWS 클라우드에서 실행 중인 노드가 있는 Amazon EKS 클러스터의 경우와 동일합니다. 이 페이지에 포함된 추가 기능만 Amazon EKS Hybrid Nodes와의 호환성이 검증되었습니다.

다음 AWS 추가 기능은 Amazon EKS Hybrid Nodes와 호환됩니다.

AWS 추가 기능	호환되는 추가 기능 버전
kube-proxy	v1.25.14-eksbuild.2 이상
CoreDNS	v1.9.3-eksbuild.7 이상
AWS Distro for OpenTelemetry(ADOT)	v0.102.1-eksbuild.2 이상
CloudWatch Observability Agent	v2.2.1-eksbuild.1 이상
EKS Pod Identity 에이전트	- v1.3.3-eksbuild.1 이상(Bottlerocket 제외) - v1.3.7-eksbuild.2 이상(Bottlerocket용)
노드 모니터링 에이전트	v1.2.0-eksbuild.1 이상
CSI 스냅샷 컨트롤러	v8.1.0-eksbuild.1 이상
AWS Private CA Connector for Kubernetes	v1.6.0-eksbuild.1 이상

다음 커뮤니티 추가 기능은 Amazon EKS Hybrid Nodes와 호환됩니다. 커뮤니티 추가 기능에 대한 자세한 내용은 [the section called “커뮤니티 추가 기능”](#) 섹션을 참조하세요.

커뮤니티 추가 기능	호환되는 추가 기능 버전
Kubernetes 지표 서버	v0.7.2-eksbuild.1 이상
cert-manager	v1.17.2-eksbuild.1 이상
Prometheus 익스포터	v1.9.1-eksbuild.2 이상
kube-state-metrics	v2.15.0-eksbuild.4 이상

위의 표에 있는 Amazon EKS 추가 기능 외에도 [Amazon Managed Service for Prometheus Collector](#)와 [애플리케이션 수신](#)(HTTP) 및 [로드 밸런싱](#)(TCP/UDP)용 [AWS 로드 밸런서 컨트롤러](#)는 하이브리드 노드와 호환됩니다.

Amazon EKS Hybrid Nodes와 호환되지 않는 AWS 추가 기능 및 커뮤니티 추가 기능이 있습니다. 이러한 추가 기능의 최신 버전에는 하이브리드 노드에 적용되는 기본 `eks.amazonaws.com/compute-type: hybrid` 레이블에 대한 반선택도 규칙이 있습니다. 이렇게 하면 클러스터에 배포할 때 하이브리드 노드에서 실행되지 않습니다. 클러스터에 하이브리드 노드와 AWS 클라우드에서 실행되는 노드가 모두 있는 경우 AWS 클라우드에서 실행되는 노드에 클러스터의 추가 기능을 배포할 수 있습니다. Amazon VPC CNI는 하이브리드 노드와 호환되지 않으며, Cilium 및 Calico는 Amazon EKS Hybrid Nodes용 컨테이너 네트워킹 인터페이스(CNI)로 지원됩니다. 자세한 정보는 [the section called “CNI 구성”](#)을 참조하세요.

AWS 추가 기능

다음 섹션에서는 다른 Amazon EKS 컴퓨팅 유형과 비교하여 하이브리드 노드에서 호환되는 AWS 추가 기능을 실행하는 것의 차이점을 설명합니다.

kube-proxy 및 CoreDNS

EKS는 AWS CLI를 포함하여 AWS API 및 AWS SDK를 사용하여 EKS 클러스터를 생성할 때 기본적으로 kube-proxy 및 CoreDNS를 자체 관리형 추가 기능으로 설치합니다. 클러스터 생성 후 Amazon EKS 추가 기능으로 이러한 추가 기능을 덮어쓸 수 있습니다. [the section called “kube-proxy”](#) 및 [the section called “CoreDNS”](#)에 대한 자세한 내용은 EKS 설명서를 참조하세요. 하이브리드 노드와 AWS 클라우드의 노드가 있는 혼합 모드 클러스터를 실행하는 경우 하이브리드 노드에는 하나 이상의 CoreDNS 복제본을, AWS 클라우드의 노드에는 하나 이상의 CoreDNS 복제본을 사용하는 것이 좋습니다. 구성 단계는 [the section called “CoreDNS 복제본 구성”](#) 섹션을 참조하세요.

CloudWatch Observability Agent

CloudWatch Observability Agent 연산자는 [웹후크](#)를 사용합니다. 하이브리드 노드에서 연산자를 실행하는 경우 온프레미스 포트 CIDR은 온프레미스 네트워크에서 라우팅 가능해야 하며 원격 포트 네트워크로 EKS 클러스터를 구성해야 합니다. 자세한 내용은 [하이브리드 노드용 웹후크 구성](#)을 참조하세요.

[CloudWatch Container Insights](#)가 노드 수준 지표에 대한 [인스턴스 메타데이터 서비스\(IMDS\)](#)의 가용성에 좌우되기 때문에 노드 수준 지표는 하이브리드 노드에 사용할 수 없습니다. 클러스터, 워크로드, 포트, 컨테이너 수준 지표는 하이브리드 노드에 사용할 수 있습니다.

[Amazon CloudWatch 관찰성을 사용하여 CloudWatch 에이전트 설치](#)에 설명된 단계에 따라 추가 기능을 설치한 후에 추가 기능 매니페스트를 업데이트해야 에이전트가 하이브리드 노드에서 성공적으로 실행될 수 있습니다. 클러스터의 `amazoncloudwatchagents` 리소스를 편집하여 아래와 같이 `RUN_WITH_IRSA` 환경 변수를 추가합니다.

```
kubectl edit amazoncloudwatchagents -n amazon-cloudwatch cloudwatch-agent
```

```

apiVersion: v1
items:
- apiVersion: cloudwatch.aws.amazon.com/v1alpha1
  kind: AmazonCloudWatchAgent
  metadata:
    ...
    name: cloudwatch-agent
    namespace: amazon-cloudwatch
    ...
  spec:
    ...
    env:
    - name: RUN_WITH_IRSA # <-- Add this
      value: "True" # <-- Add this
    - name: K8S_NODE_NAME
      valueFrom:
        fieldRef:
          fieldPath: spec.nodeName
    ...

```

하이브리드 노드용 Amazon Managed Prometheus 관리형 수집기

Amazon Managed Service for Prometheus(AMP) 관리형 수집기는 Amazon EKS 클러스터의 리소스에서 지표를 검색하고 수집하는 스크레이퍼로 구성됩니다. AMP가 스크레이퍼를 대신 관리하므로 인스턴스, 에이전트 또는 스크레이퍼를 직접 관리할 필요가 없습니다.

하이브리드 노드에 특정한 추가 구성 없이 AMP 관리형 수집기를 사용할 수 있습니다. 그러나 하이브리드 노드의 애플리케이션에 대한 지표 엔드포인트는 VPC에서 원격 포드 네트워크 CIDR로의 경로와 온프레미스 방화벽에서 열린 포트를 포함하여 VPC에서 연결할 수 있어야 합니다. 또한 클러스터에는 [프라이빗 클러스터 엔드포인트 액세스](#) 권한이 있어야 합니다.

Amazon Managed Service for Prometheus 사용 설명서의 [AWS 관리형 수집기 사용](#)의 단계를 따릅니다.

AWS Distro for OpenTelemetry(ADOT)

AWS Distro for OpenTelemetry(ADOT) 추가 기능을 사용하여 하이브리드 노드에서 실행되는 애플리케이션의 지표, 로그 및 추적 데이터를 수집할 수 있습니다. ADOT 연산자는 승인 [웹훅](#)을 사용하여 수집기 사용자 지정 리소스 요청을 변경 및 검증합니다. 하이브리드 노드에서 ADOT 연산자를 실행하는 경우 온프레미스 포드 CIDR은 온프레미스 네트워크에서 라우팅 가능해야 하며 원격 포드 네트워크로 EKS 클러스터를 구성해야 합니다. 자세한 내용은 [하이브리드 노드용 웹훅 구성](#)을 참조하세요.

AWS Distro for OpenTelemetry 설명서의 [EKS 추가 기능을 사용하여 AWS Distro for OpenTelemetry 시작하기](#)의 단계를 따릅니다.

AWS 로드 밸런서 컨트롤러

AWS Direct Connect 또는 AWS Site-to-Site VPN과 연결된 하이브리드 노드의 워크로드에 대상 유형 ip와 함께 [AWS 로드 밸런서 컨트롤러](#) 및 Application Load Balancer(ALB) 또는 Network Load Balancer(NLB)를 사용할 수 있습니다. ALB 또는 NLB에서 사용하는 IP 대상은 AWS에서 라우팅 가능해야 합니다. AWS Load Balancer Controller 역시 [웹훅](#)을 사용합니다. 하이브리드 노드에서 AWS Load Balancer Controller 연산자를 실행하는 경우 온프레미스 포드 CIDR은 온프레미스 네트워크에서 라우팅 가능해야 하며 원격 포드 네트워크로 EKS 클러스터를 구성해야 합니다. 자세한 내용은 [하이브리드 노드용 웹훅 구성](#)을 참조하세요.

AWS 로드 밸런서 컨트롤러를 설치하려면 [the section called “Helm을 사용하여 설치”](#) 또는 [the section called “매니페스트를 사용하여 설치”](#)의 단계를 따릅니다.

ALB로 수신하려면 아래 주석을 지정해야 합니다. 자세한 내용은 [the section called “애플리케이션 로드 밸런싱”](#) 섹션을 참조하세요.

```
alb.ingress.kubernetes.io/target-type: ip
```

NLB로 로드 밸런싱하려면 아래 주석을 지정해야 합니다. 자세한 내용은 [the section called “네트워크 로드 밸런싱”](#) 섹션을 참조하세요.

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip"
```

EKS Pod Identity 에이전트

Note

Bottlerocket을 실행하는 하이브리드 노드에 EKS Pod Identity Agent 추가 기능을 성공적으로 배포하려면 Bottlerocket 버전이 v1.39.0 이상이어야 합니다. 하이브리드 노드 환경의 이전 Bottlerocket 버전에서는 Pod Identity Agent가 지원되지 않습니다.

원래 Amazon EKS Pod Identity 에이전트 DaemonSet는 노드에서 EC2 IMDS를 사용하여 필요한 AWS ID를 얻습니다. 버전 1.3.3-eksbuild.1부터는 하이브리드 노드에서 IMDS를 사용할 수 없으므로 Pod Identity Agent 추가 기능은 필요한 자격 증명을 탑재하는 DaemonSet를 선택적으로 배포합니다. Bottlerocket을 실행하는 하이브리드 노드는 자격 증명을 탑재하려면 다른 방법이 필요하며, 버전

1.3.7-eksbuild.2부터 Pod Identity Agent 추가 기능은 선택적으로 Bottlerocket 하이브리드 노드를 특별히 대상으로 하는 DaemonSet를 배포합니다. 다음 섹션에서는 선택적 DaemonSets를 활성화하는 프로세스를 설명합니다.

Ubuntu/RHEL/AL2023

1. Ubuntu/RHEL/AL2023 하이브리드 노드에서 Pod Identity 에이전트를 사용하려면 아래와 같이 nodeadm 구성의 하이브리드 섹션에서 enableCredentialsFile: true를 설정합니다.

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  hybrid:
    enableCredentialsFile: true # <-- Add this
```

이렇게 하면 /eks-hybrid/.aws/credentials에서 노드에 구성할 자격 증명 파일을 생성하도록 nodeadm이 구성되며, 이 파일은 eks-pod-identity-agent 포드에서 사용됩니다. 이 자격 증명 파일에는 주기적으로 새로 고침되는 임시 AWS 자격 증명이 포함됩니다.

2. 각 노드에서 nodeadm 구성을 업데이트한 후 nodeConfig.yaml을 사용하여 다음 nodeadm init 명령을 실행하고 하이브리드 노드를 Amazon EKS 클러스터에 조인합니다. 노드가 이전에 클러스터에 조인한 경우 init 명령을 다시 실행합니다.

```
nodeadm init -c file:///nodeConfig.yaml
```

3. AWS CLI 또는 AWS Management Console을 사용하여 하이브리드 노드에 대한 지원이 활성화된 상태로 eks-pod-identity-agent를 설치합니다.
 - a. AWS CLI: 클러스터 관리에 사용하는 시스템에서 다음 명령을 실행하여 하이브리드 노드에 대한 지원이 활성화된 상태로 eks-pod-identity-agent를 설치합니다. my-cluster를 클러스터 이름으로 바꿉니다.

```
aws eks create-addon \
  --cluster-name my-cluster \
  --addon-name eks-pod-identity-agent \
  --configuration-values '{"daemonsets":{"hybrid":{"create": true}}}'
```

- b. AWS Management Console: AWS 콘솔을 통해 Pod Identity 에이전트 추가 기능을 설치하는 경우 선택적 구성에 다음을 추가하여 하이브리드 노드를 대상으로 하는 DaemonSet를 배포합니다.

```
{"daemonsets":{"hybrid":{"create": true}}}
```


Bottlerocket

1. Bottlerocket 하이브리드 노드에서 Pod Identity 에이전트를 사용하려면 [the section called “Bottlerocket을 실행하는 하이브리드 노드 연결”](#)에 설명된 대로 Bottlerocket 부트스트랩 컨테이너 사용자 데이터에 사용되는 명령에 `--enable-credentials-file=true` 플래그를 추가합니다.

- a. SSM 자격 증명 공급자를 사용하는 경우 명령은 다음과 같아야 합니다.

```
eks-hybrid-ssm-setup --activation-id=<activation-id> --activation-
code=<activation-code> --region=<region> --enable-credentials-file=true
```

- b. IAM Roles Anywhere 자격 증명 공급자를 사용하는 경우 명령은 다음과 같아야 합니다.

```
eks-hybrid-iam-ra-setup --certificate=<certificate> --key=<private-key> --enable-
credentials-file=true
```

이렇게 하면 `/var/eks-hybrid/.aws/credentials`에서 노드에 구성할 자격 증명 파일을 생성하도록 부트스트랩 스크립트가 구성되며, 이 파일은 `eks-pod-identity-agent` 포드에서 사용됩니다. 이 자격 증명 파일에는 주기적으로 새로 고침되는 임시 AWS 자격 증명이 포함됩니다.

2. AWS CLI 또는 AWS Management Console을 사용하여 Bottlerocket 하이브리드 노드에 대한 지원이 활성화된 상태로 `eks-pod-identity-agent`를 설치합니다.

- a. AWS CLI: 클러스터 관리에 사용하는 시스템에서 다음 명령을 실행하여 Bottlerocket 하이브리드 노드에 대한 지원이 활성화된 상태로 `eks-pod-identity-agent`를 설치합니다. `my-cluster`를 클러스터 이름으로 바꿉니다.

```
aws eks create-addon \
  --cluster-name my-cluster \
  --addon-name eks-pod-identity-agent \
  --configuration-values '{"daemonsets":{"hybrid-bottlerocket":{"create":
true}}}'
```

- b. AWS Management Console: AWS 콘솔을 통해 Pod Identity 에이전트 추가 기능을 설치하는 경우 선택적 구성에 다음을 추가하여 Bottlerocket 하이브리드 노드를 대상으로 하는 DaemonSet을 배포합니다.

```
{"daemonsets":{"hybrid-bottlerocket":{"create": true}}}
```

CSI 스냅샷 컨트롤러

v8.1.0-eksbuild.2 버전 부터 [CSI 스냅샷 컨트롤러 추가 기능](#)은 하이브리드 노드에 대해 소프트웨어 반선호도 규칙을 적용하며, 컨트롤러 deployment가 Amazon EKS 컨트롤 플레인과 동일한 AWS 리전의 EC2에서 실행되는 것을 선호합니다. Amazon EKS 컨트롤 플레인과 동일한 AWS 리전에서 deployment를 공동 배치하면 지연 시간이 개선됩니다.

커뮤니티 추가 기능

다음 섹션에서는 다른 Amazon EKS 컴퓨팅 유형과 비교하여 하이브리드 노드에서 호환되는 커뮤니티 추가 기능을 실행하는 것의 차이점을 설명합니다.

Kubernetes 지표 서버

컨트롤 플레인 Metrics Server의 포트 IP(또는 hostNetwork가 활성화된 경우 노드 IP)에 도달해야 합니다. 따라서 hostNetwork 모드에서 Metrics Server를 실행하지 않는 한 Amazon EKS 클러스터를 생성할 때 원격 포트 네트워크를 구성해야 하고, 포트 IP 주소를 라우팅 가능하도록 설정해야 합니다. CNI를 사용하여 Border Gateway Protocol(BGP)을 구현하는 것은 포트 IP 주소를 라우팅 가능하도록 설정하는 일반적인 방법 중 하나입니다.

cert-manager

cert-manager는 [웹훅](#)을 사용합니다. 하이브리드 노드에서 cert-manager를 실행하는 경우 온프레미스 포트 CIDR은 온프레미스 네트워크에서 라우팅 가능해야 하며 원격 포트 네트워크로 EKS 클러스터를 구성해야 합니다. 자세한 내용은 [하이브리드 노드용 웹훅 구성](#)을 참조하세요.

하이브리드 노드용 웹훅 구성

이 페이지에서는 하이브리드 노드에서 웹훅을 실행하는 것과 관련된 고려 사항을 자세히 설명합니다. 웹훅은 Kubernetes 애플리케이션과 AWS 로드 밸런서 컨트롤러 및 CloudWatch 관찰성 에이전트와 같은 오픈 소스 프로젝트에서 런타임 시 변경 및 검증 기능을 수행하는 데 사용됩니다.

하이브리드 노드에서 웹훅을 실행하는 경우 온프레미스 포트 CIDR이 온프레미스 네트워크에서 라우팅 가능해야 합니다. 또한 EKS 컨트롤 플레인이 하이브리드 노드에서 실행되는 웹훅과 통신할 수 있도록 원격 포트 네트워크로 EKS 클러스터를 구성해야 합니다.

Border Gateway Protocol(BGP), 정적 경로 또는 기타 사용자 지정 라우팅 솔루션을 포함하여 온프레미스 네트워크에서 온프레미스 포트 CIDR을 라우팅 가능하도록 하는 데 몇 가지 기법을 사용할 수 있습니다. BGP는 사용자 지정 또는 수동 라우팅 구성이 필요한 대체 솔루션보다 더 확장성이 뛰어나고 관

리하기 쉽기 때문에 권장되는 솔루션입니다. AWS는 하이브리드 노드 포드 CIDR을 알리는 데 Cilium 및 Calico의 BGP 기능을 지원합니다. 자세한 내용은 [하이브리드 노드용 CNI 구성](#)을 참조하세요.

온프레미스 네트워크에서 온프레미스 포드 CIDR을 라우팅 가능하도록 할 수 없고 웹후크를 실행해야 하는 경우 AWS 클라우드에서 모든 웹후크를 실행하는 것이 좋습니다. 웹후크가 작동하려면 하이브리드 노드와 동일한 EKS 클러스터에서 실행되어야 합니다.

혼합 모드 클러스터 고려 사항

혼합 모드 클러스터는 하이브리드 노드와 노드가 모두 AWS 클라우드에서 실행되는 EKS 클러스터로 정의됩니다. 혼합 모드 클러스터를 실행할 때는 다음 권장 사항을 고려하세요.

- AWS 클라우드의 노드에서 VPC CNI를 실행하고 하이브리드 노드에서 Cilium 또는 Calico를 실행합니다. Cilium 및 Calico는 AWS 클라우드의 노드에서 실행할 때 AWS에서 지원되지 않습니다.
- 애플리케이션이 하이브리드 노드에서 실행되는 포드와 직접 통신("동서 통신")하기 위해 AWS 클라우드의 노드에서 실행되는 포드가 필요하고, 하이브리드 노드에서 오버레이/터널 모드로 AWS 클라우드 및 Cilium 또는 Calico의 노드에서 VPC CNI를 사용하는 경우 온프레미스 포드 CIDR은 온프레미스 네트워크에서 라우팅 가능해야 합니다.
- AWS 클라우드의 노드에서 하나 이상의 CoreDNS 복제본을 실행하고 하이브리드 노드에서 하나 이상의 CoreDNS 복제본을 실행합니다. 구성 단계는 [혼합 모드 클러스터에 대한 추가 기능 및 웹후크 구성](#)을 참조하세요.
- AWS 클라우드의 노드에서 실행되도록 웹후크를 구성합니다. 혼합 모드 클러스터를 실행할 때에서 사용하는 AWS에서 사용하는 웹후크와 커뮤니티 추가 기능을 구성하는 방법은 [추가 기능을 위해 웹후크 구성](#)을 참조하세요.
- 하이브리드 노드에서 실행되는 워크로드 트래픽에 Application Load Balancer(ALB) 또는 Network Load Balancer(NLB)를 사용하는 경우 ALB 또는 NLB와 함께 사용되는 IP 대상이 AWS에서 라우팅 가능해야 합니다.
- Metrics Server 추가 기능을 사용하려면 EKS 컨트롤 플레인에서 Metrics Server 포드 IP 주소로의 연결이 필요합니다. 하이브리드 노드에서 Metrics Server 추가 기능을 실행하는 경우 온프레미스 포드 CIDR이 온프레미스 네트워크에서 라우팅 가능해야 합니다.
- Amazon Managed Service for Prometheus(AMP) 관리형 수집기를 사용하여 하이브리드 노드에 대한 지표를 수집하려면 온프레미스 포드 CIDR이 온프레미스 네트워크에서 라우팅 가능해야 합니다. 또는 AWS 클라우드에서 실행되는 EKS 컨트롤 플레인 지표 및 노드에 AMP 관리형 수집기를 사용하고, AWS Distro for OpenTelemetry(ADOT) 추가 기능을 사용하여 하이브리드 노드에 대한 지표를 수집할 수 있습니다.

혼합 모드 클러스터에 대한 추가 기능 및 웹훅 구성

클러스터에서 실행되는 변형 및 검증 웹훅을 보려면 클러스터의 EKS 콘솔의 리소스 패널에서 확장 리소스 유형을 보거나 다음 명령을 사용할 수 있습니다. 또한 EKS는 클러스터 관찰성 대시보드에 웹훅 지표를 보고합니다. 자세한 내용은 [the section called “관찰성 대시보드”](#) 섹션을 참조하세요.

```
kubectl get mutatingwebhookconfigurations
```

```
kubectl get validatingwebhookconfigurations
```

CoreDNS 복제본 구성

하이브리드 노드와 AWS 클라우드의 노드가 있는 혼합 모드 클러스터를 실행하는 경우 하이브리드 노드에는 하나 이상의 CoreDNS 복제본을, AWS 클라우드의 노드에는 하나 이상의 CoreDNS 복제본을 사용하는 것이 좋습니다. 혼합 모드 클러스터 설정에서 지연 및 네트워크 문제를 방지하려면 [서비스 트래픽 분산](#)을 사용하여 가장 가까운 CoreDNS 복제본을 선호하도록 CoreDNS 서비스를 구성할 수 있습니다.

서비스 트래픽 분산(EKS의 Kubernetes 버전 1.31 이상에서 사용 가능)은 예측 가능성이 더 높기 때문에 [토폴로지 인식 라우팅](#)보다 권장되는 솔루션입니다. 서비스 트래픽 분산에서는 영역 내의 정상 엔드포인트는 해당 영역의 모든 트래픽을 수신합니다. 토폴로지 인식 라우팅에서는 사용자 지정 라우팅을 적용하려면 각 서비스가 해당 영역의 여러 조건을 충족해야 합니다. 그렇지 않으면 트래픽을 모든 엔드포인트로 균등하게 라우팅합니다. 다음 단계에서는 서비스 트래픽 분산을 구성합니다.

Cilium을 CNI로 사용하는 경우 enable-service-topology가 true로 설정된 CNI를 실행하여 서비스 트래픽 분산을 활성화해야 합니다. Helm 설치 플래그 `--set loadBalancer.serviceTopology=true`를 사용하여 이 구성을 전달하거나 Cilium CLI 명령 `cilium config set enable-service-topology true`를 사용하여 기존 설치를 업데이트할 수 있습니다. 기존 설치의 구성을 업데이트한 후 각 노드에서 실행되는 Cilium 에이전트를 다시 시작해야 합니다.

1. `topology.kubernetes.io/zone`: onprem과 같이 각 하이브리드 노드에 대한 토폴로지 영역 레이블을 추가합니다. 또는 nodeadm 구성에서 레이블을 지정하여 nodeadm init 단계에서 레이블을 설정할 수 있습니다. [the section called “kubelet 사용자 지정을 위한 노드 구성\(선택 사항\)”](#) 섹션을 참조하세요. AWS 클라우드에서 실행되는 노드는 노드의 가용 영역(AZ)에 해당하는 토폴로지 영역 레이블이 자동 적용됩니다.

```
kubectl label node hybrid-node-name topology.kubernetes.io/zone=zone
```

2. 토폴로지 영역 키가 있는 CoreDNS 배포에 podAntiAffinity를 추가합니다. 또는 설치 중에 EKS 추가 기능을 사용하여 CoreDNS 배포를 구성할 수 있습니다.

```
kubectl edit deployment coredns -n kube-system
```

```
spec:
  template:
    spec:
      affinity:
        ...
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution:
            - podAffinityTerm:
                labelSelector:
                  matchExpressions:
                    - key: k8s-app
                      operator: In
                      values:
                        - kube-dns
                topologyKey: kubernetes.io/hostname
                weight: 100
            - podAffinityTerm:
                labelSelector:
                  matchExpressions:
                    - key: k8s-app
                      operator: In
                      values:
                        - kube-dns
                topologyKey: topology.kubernetes.io/zone
                weight: 50
        ...
```

3. kube-dns 서비스 구성에 trafficDistribution: PreferClose 설정을 추가하여 토폴로지 인식 라우팅을 활성화합니다.

```
kubectl patch svc kube-dns -n kube-system --type=merge -p '{
  "spec": {
    "trafficDistribution": "PreferClose"
  }
}'
```

4. kube-dns 서비스의 엔드포인트 조각을 확인하여 서비스 트래픽 분산이 활성화되었는지 확인할 수 있습니다. 엔드포인트 조각에 서비스 트래픽 분산이 활성화되었음을 확인하는 토폴로지 영역 레이블의 hints가 표시되어야 합니다. 각 엔드포인트 주소에 대해 hints가 표시되지 않으면 서비스 트래픽 분산이 활성화되지 않은 것입니다.

```
kubectl get endpointslice -A | grep "kube-dns"
```

```
kubectl get endpointslice [.replaceable]`kube-dns-<id>` -n kube-system -o yaml
```

```
addressType: IPv4
apiVersion: discovery.k8s.io/v1
endpoints:
- addresses:
  - <your-hybrid-node-pod-ip>
  hints:
    forZones:
      - name: onprem
    nodeName: <your-hybrid-node-name>
    zone: onprem
- addresses:
  - <your-cloud-node-pod-ip>
  hints:
    forZones:
      - name: us-west-2a
    nodeName: <your-cloud-node-name>
    zone: us-west-2a
```

추가 기능을 위한 웹훅 구성

다음 추가 기능은 웹훅을 사용하고 하이브리드 노드에서 사용할 수 있습니다.

- AWS 로드 밸런서 컨트롤러
- CloudWatch 관찰성 에이전트
- AWS Distro for OpenTelemetry(ADOT)
- cert-manager

이러한 추가 기능이 AWS 클라우드의 노드에서 실행하도록 사용하는 웹훅을 구성하려면 다음 섹션을 참조하세요.

AWS 로드 밸런서 컨트롤러

혼합 모드 클러스터 설정에서 AWS 로드 밸런서 컨트롤러를 사용하려면 AWS 클라우드의 노드에서 컨트롤러를 실행해야 합니다. 이렇게 하려면 Helm 값 구성에 다음을 추가하거나 EKS 추가 기능 구성을 사용하여 값을 지정합니다.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
      - matchExpressions:
        - key: eks.amazonaws.com/compute-type
          operator: NotIn
          values:
            - hybrid
```

CloudWatch 관찰성 에이전트

CloudWatch 관찰성 에이전트 추가 기능에는 웹후크를 사용하는 Kubernetes 연산자가 있습니다. 혼합 모드 클러스터 설정의 AWS 클라우드에 있는 노드에서 연산자를 실행하려면 CloudWatch Observability Agent 연산자 구성을 편집합니다. Helm 및 EKS 추가 기능을 사용하여 설치하는 동안에는 연산자 선호도를 구성할 수 없습니다([컨테이너-로드맵 문제 #2431](#) 참조).

```
kubectl edit -n amazon-cloudwatch deployment amazon-cloudwatch-observability-
controller-manager
```

```
spec:
  ...
  template:
    ...
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: eks.amazonaws.com/compute-type
                operator: NotIn
                values:
                  - hybrid
```

AWS Distro for OpenTelemetry(ADOT)

AWS Distro for OpenTelemetry(ADOT) 추가 기능에는 웹후크를 사용하는 Kubernetes 연산자가 있습니다. 혼합 모드 클러스터 설정의 AWS 클라우드에 있는 노드에서 연산자를 실행하려면 Helm 값 구성에 다음을 추가하거나 EKS 추가 기능 구성을 사용하여 값을 지정합니다.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
            - key: eks.amazonaws.com/compute-type
              operator: NotIn
              values:
                - hybrid
```

포드 CIDR이 온프레미스 네트워크에서 라우팅 가능하지 않은 경우 하이브리드 노드 및 해당 노드에서 실행 중인 워크로드에서 지표를 스크레이프할 수 있도록 ADOT Collector가 하이브리드 노드에서 실행되어야 합니다. 이렇게 하려면 사용자 지정 리소스 정의(CRD)를 편집합니다.

```
kubectl -n opentelemetry-operator-system edit opentelemetrycollectors.opentelemetry.io
adot-col-prom-metrics
```

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: eks.amazonaws.com/compute-type
                operator: In
                values:
                  - hybrid
```

ADOT Collector CRD 구성의 각 `scrape_configs`에 다음 `relabel_configs`를 추가하여 하이브리드 노드 및 하이브리드 노드에서 실행되는 리소스의 지표만 스크레이프하도록 ADOT Collector를 구성할 수 있습니다.

```
relabel_configs:
  - action: keep
    regex: hybrid
```



```
source_labels:
- __meta_kubernetes_node_label_eks_amazonaws_com_compute_type
```

ADOT 추가 기능에는 ADOT 연산자 웹후크에서 사용하는 TLS 인증서에 대해 cert-manager를 설치하기 위한 사전 요구 사항이 있습니다. 또한 cert-manager는 웹후크를 실행하고 다음 Helm 값 구성을 사용하여 AWS 클라우드의 노드에서 실행되도록 구성할 수 있습니다.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
            - key: eks.amazonaws.com/compute-type
              operator: NotIn
              values:
                - hybrid
webhook:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: eks.amazonaws.com/compute-type
                operator: NotIn
                values:
                  - hybrid
cainjector:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: eks.amazonaws.com/compute-type
                operator: NotIn
                values:
                  - hybrid
startupapicheck:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: eks.amazonaws.com/compute-type
```

```
operator: NotIn
values:
- hybrid
```

cert-manager

또한 cert-manager 추가 기능은 웹훅을 실행하며, 다음 Helm 값 구성을 사용하여 AWS 클라우드의 노드에서 실행되도록 구성할 수 있습니다.

```
affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
            - key: eks.amazonaws.com/compute-type
              operator: NotIn
              values:
                - hybrid
webhook:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: eks.amazonaws.com/compute-type
                operator: NotIn
                values:
                  - hybrid
cainjector:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
          - matchExpressions:
              - key: eks.amazonaws.com/compute-type
                operator: NotIn
                values:
                  - hybrid
startupapicheck:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
```

```
- matchExpressions:
  - key: eks.amazonaws.com/compute-type
    operator: NotIn
    values:
      - hybrid
```

하이브리드 노드용 프록시 구성

데이터 센터 또는 엣지 환경에서 나가는 트래픽에 대해 온프레미스 환경에서 프록시 서버를 사용하는 경우 프록시 서버를 사용하려면 노드와 클러스터를 개별적으로 구성해야 합니다.

클러스터

클러스터에서 프록시 서버를 사용하도록 kube-proxy를 구성해야 합니다. Amazon EKS 클러스터를 생성한 후 kube-proxy를 구성해야 합니다.

노드

노드에서 프록시 서버를 사용하도록 운영 체제, containerd, kubelet 및 Amazon SSM 에이전트를 구성해야 합니다. 운영 체제 이미지의 빌드 프로세스 중에 또는 각 하이브리드 노드에서 nodeadm init을 실행하기 전에 변경할 수 있습니다.

노드 수준 구성

운영 체제 이미지에 또는 각 하이브리드 노드에서 nodeadm init를 실행하기 전에 다음 구성을 적용해야 합니다.

containerd 프록시 구성

containerd는 Kubernetes의 기본 컨테이너 관리 런타임입니다. 인터넷 액세스에 프록시를 사용하는 경우 Kubernetes 및 Amazon EKS에 필요한 컨테이너 이미지를 가져올 수 있도록 containerd를 구성해야 합니다.

/etc/systemd/system/containerd.service.d 디렉터리의 http-proxy.conf라는 각 하이브리드 노드에 다음 콘텐츠로 파일을 생성합니다. proxy-domain과 port를 해당 환경의 값으로 바꿉니다.

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

사용자 데이터의 **containerd** 구성

이 파일에 대해 `containerd.service.d` 디렉터리를 생성해야 합니다. 재부팅하지 않고 구성 파일을 픽업하려면 `systemd`를 다시 로드해야 합니다. AL2023에서는 스크립트가 실행될 때 서비스가 이미 실행 중일 수 있으므로 서비스를 다시 시작해야 합니다.

```
mkdir -p /etc/systemd/system/containerd.service.d
echo '[Service]' > /etc/systemd/system/containerd.service.d/http-proxy.conf
echo 'Environment="HTTP_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
containerd.service.d/http-proxy.conf
echo 'Environment="HTTPS_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
containerd.service.d/http-proxy.conf
echo 'Environment="NO_PROXY=localhost"' >> /etc/systemd/system/containerd.service.d/
http-proxy.conf
systemctl daemon-reload
systemctl restart containerd
```

kubelet 프록시 구성

`kubelet`는 각 Kubernetes 노드에서 실행되는 Kubernetes 노드 에이전트이며 해당 노드에서 실행되는 노드 및 포드를 관리할 책임이 있습니다. 온프레미스 환경에서 프록시를 사용하는 경우 Amazon EKS 클러스터의 퍼블릭 또는 프라이빗 엔드포인트와 통신할 수 있도록 `kubelet`를 구성해야 합니다.

`/etc/systemd/system/kubelet.service.d/` 디렉터리의 `http-proxy.conf`라는 각 하이브리드 노드에 다음 콘텐츠로 파일을 생성합니다. `proxy-domain`과 `port`를 해당 환경의 값으로 바꿉니다.

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

kubelet 사용자 데이터의 구성

이 파일에 대한 `kubelet.service.d` 디렉터리를 생성해야 합니다. 재부팅하지 않고 구성 파일을 픽업하려면 `systemd`를 다시 로드해야 합니다. AL2023에서는 스크립트가 실행될 때 서비스가 이미 실행 중일 수 있으므로 서비스를 다시 시작해야 합니다.

```
mkdir -p /etc/systemd/system/kubelet.service.d
echo '[Service]' > /etc/systemd/system/kubelet.service.d/http-proxy.conf
echo 'Environment="HTTP_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
kubelet.service.d/http-proxy.conf
```

```
echo 'Environment="HTTPS_PROXY=http://proxy-domain:port"' >> /etc/systemd/system/
kubelet.service.d/http-proxy.conf
echo 'Environment="NO_PROXY=localhost"' >> /etc/systemd/system/kubelet.service.d/http-
proxy.conf
systemctl daemon-reload
systemctl restart kubelet
```

ssm 프록시 구성

ssm은 하이브리드 노드를 초기화하는 데 사용할 수 있는 자격 증명 공급자 중 하나입니다. ssm은 AWS를 사용하여 인증하고 kubelet에서 사용하는 임시 자격 증명을 생성할 책임이 있습니다. 온프레미스 환경에서 프록시를 사용하고 노드에서 자격 증명 공급자로 ssm을 사용하는 경우 Amazon SSM 서비스 엔드포인트와 통신할 수 있도록 ssm을 구성해야 합니다.

운영 체제에 따라 아래 경로에서 http-proxy.conf라는 파일을 각 하이브리드 노드에 생성합니다.

- Ubuntu - /etc/systemd/system/snap.amazon-ssm-agent.amazon-ssm-agent.service.d/http-proxy.conf
- Amazon Linux 2023 및 Red Hat Enterprise Linux - /etc/systemd/system/amazon-ssm-agent.service.d/http-proxy.conf

다음 콘텐츠로 파일을 채웁니다. proxy-domain과 port를 해당 환경의 값으로 바꿉니다.

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

ssm 사용자 데이터의 구성

이 파일에 대한 ssm systemd 서비스 파일 디렉토리를 생성해야 합니다. 디렉터리 경로는 노드에서 사용되는 운영 체제에 따라 다릅니다.

- Ubuntu - /etc/systemd/system/snap.amazon-ssm-agent.amazon-ssm-agent.service.d
- Amazon Linux 2023 및 Red Hat Enterprise Linux - /etc/systemd/system/amazon-ssm-agent.service.d

노드에서 사용되는 운영 체제에 따라 아래 재시작 명령의 시스템 서비스 이름을 바꿉니다.

- Ubuntu - `snap.amazon-ssm-agent.amazon-ssm-agent`
- Amazon Linux 2023 및 Red Hat Enterprise Linux - `amazon-ssm-agent`

```
mkdir -p systemd-service-file-directory
echo '[Service]' > [.replaceable]#systemd-service-file-directory/http-proxy.conf
echo 'Environment="HTTP_PROXY=http://[.replaceable]#proxy-domain:port"' >> systemd-
service-file-directory/http-proxy.conf
echo 'Environment="HTTPS_PROXY=http://[.replaceable]#proxy-domain:port"' >>
[.replaceable]#systemd-service-file-directory/http-proxy.conf
echo 'Environment="NO_PROXY=localhost"' >> [.replaceable]#systemd-service-file-
directory/http-proxy.conf
systemctl daemon-reload
systemctl restart [.replaceable]#systemd-service-name
```

운영 체제 프록시 구성

인터넷 액세스에 프록시를 사용하는 경우 운영 체제의 패키지 관리자에서 하이브리드 노드 종속성을 가져올 수 있도록 운영 체제를 구성해야 합니다.

Ubuntu

1. 다음 명령으로 프록시를 사용하도록 snap을 구성합니다.

```
sudo snap set system proxy.https=http://proxy-domain:port
sudo snap set system proxy.http=http://proxy-domain:port
```

2. apt에 대한 프록시를 활성화하려면 `/etc/apt/` 디렉터리에서 `apt.conf`라는 파일을 생성합니다. 프록시 도메인과 포트를 해당 환경의 값으로 바꿉니다.

```
Acquire::http::Proxy "http://proxy-domain:port";
Acquire::https::Proxy "http://proxy-domain:port";
```

Amazon Linux 2023

1. 프록시를 사용하도록 dnf을 구성합니다. 환경의 프록시 도메인 및 포트 값을 사용하여 `/etc/dnf/` `dnf.conf` 파일을 생성합니다.

```
proxy=http://proxy-domain:port
```

Red Hat Enterprise Linux

1. 프록시를 사용하도록 yum을 구성합니다. 환경의 프록시 도메인 및 포트 값을 사용하여 `/etc/yum.conf` 파일을 생성합니다.

```
proxy=http://proxy-domain:port
```

IAM Roles Anywhere 프록시 구성

IAM Roles Anywhere 자격 증명 공급자 서비스는 `enableCredentialsFile` 플래그와 함께 IAM Roles Anywhere를 사용할 때 자격 증명을 새로 고치는 역할을 합니다([the section called “EKS Pod Identity 에이전트”](#) 참조). 온프레미스 환경에서 프록시를 사용하는 경우 IAM Roles Anywhere 엔드포인트와 통신할 수 있도록 서비스를 구성해야 합니다.

각 하이브리드 노드의 `/etc/systemd/system/aws_signing_helper_update.service.d/` 디렉터리에 다음 콘텐츠로 `http-proxy.conf`라는 파일을 생성합니다. `proxy-domain`과 `port`를 해당 환경의 값으로 바꿉니다.

```
[Service]
Environment="HTTP_PROXY=http://proxy-domain:port"
Environment="HTTPS_PROXY=http://proxy-domain:port"
Environment="NO_PROXY=localhost"
```

클러스터 전체 구성

이 섹션의 구성은 Amazon EKS 클러스터를 생성한 후 각 하이브리드 노드에서 `nodeadm init`를 실행하기 전에 적용해야 합니다.

kube-proxy 프록시 구성

Amazon EKS는 하이브리드 노드가 클러스터에 조인할 때 각 하이브리드 노드에 DaemonSet로 `kube-proxy`를 자동으로 설치합니다. `kube-proxy`를 사용하면 Amazon EKS 클러스터의 포드가 지원하는 서비스 간에 라우팅할 수 있습니다. 각 호스트를 구성하려면 `kube-proxy`에서 Amazon EKS 클러스터 엔드포인트에 대한 DNS 확인이 필요합니다.

1. 다음 명령을 사용하여 `kube-proxy` DaemonSet를 편집합니다.

```
kubectl -n kube-system edit ds kube-proxy
```

그러면 구성된 편집기에서 kube-proxy DaemonSet 정의가 열립니다.

2. HTTP_PROXY 및 HTTPS_PROXY에 대한 환경 변수를 추가합니다. NODE_NAME 환경 변수가 구성에 이미 있어야 합니다. proxy-domain과 port를 해당 환경의 값으로 바꿉니다.

```
containers:
  - command:
    - kube-proxy
    - --v=2
    - --config=/var/lib/kube-proxy-config/config - --hostname-override=$(NODE_NAME)
    env:
    - name: HTTP_PROXY
      value: http://proxy-domain:port
    - name: HTTPS_PROXY
      value: http://proxy-domain:port
    - name: NODE_NAME
      valueFrom:
        fieldRef:
          apiVersion: v1
          fieldPath: spec.nodeName
```

하이브리드 노드의 개념

Amazon EKS Hybrid Nodes를 사용하면 온프레미스 또는 엣지 환경에서 실행되는 물리적 또는 가상 머신을 AWS 클라우드에서 실행되는 Amazon EKS 클러스터에 조인할 수 있습니다. 이 접근 방식은 많은 이점을 제공하지만 단일 네트워크 환경에서 Kubernetes 클러스터를 실행하는 데 익숙한 사용자를 위한 새로운 네트워킹 개념과 아키텍처도 도입합니다.

다음 섹션에서는 EKS Hybrid Nodes의 Kubernetes 및 네트워킹 개념을 자세히 살펴보고 하이브리드 아키텍처를 통해 트래픽이 흐르는 방식을 자세히 설명합니다. 이 섹션에서는 포드, 노드, 서비스, Kubernetes 컨트롤 플레인, kubelet 및 kube-proxy의 개념과 같은 기본 Kubernetes 네트워킹 지식을 숙지해야 합니다.

[the section called “네트워킹 개념”](#)부터 시작하여, [the section called “Kubernetes 개념”](#), 마지막으로 [the section called “트래픽 흐름”](#) 순으로 이 페이지를 읽는 것이 좋습니다.

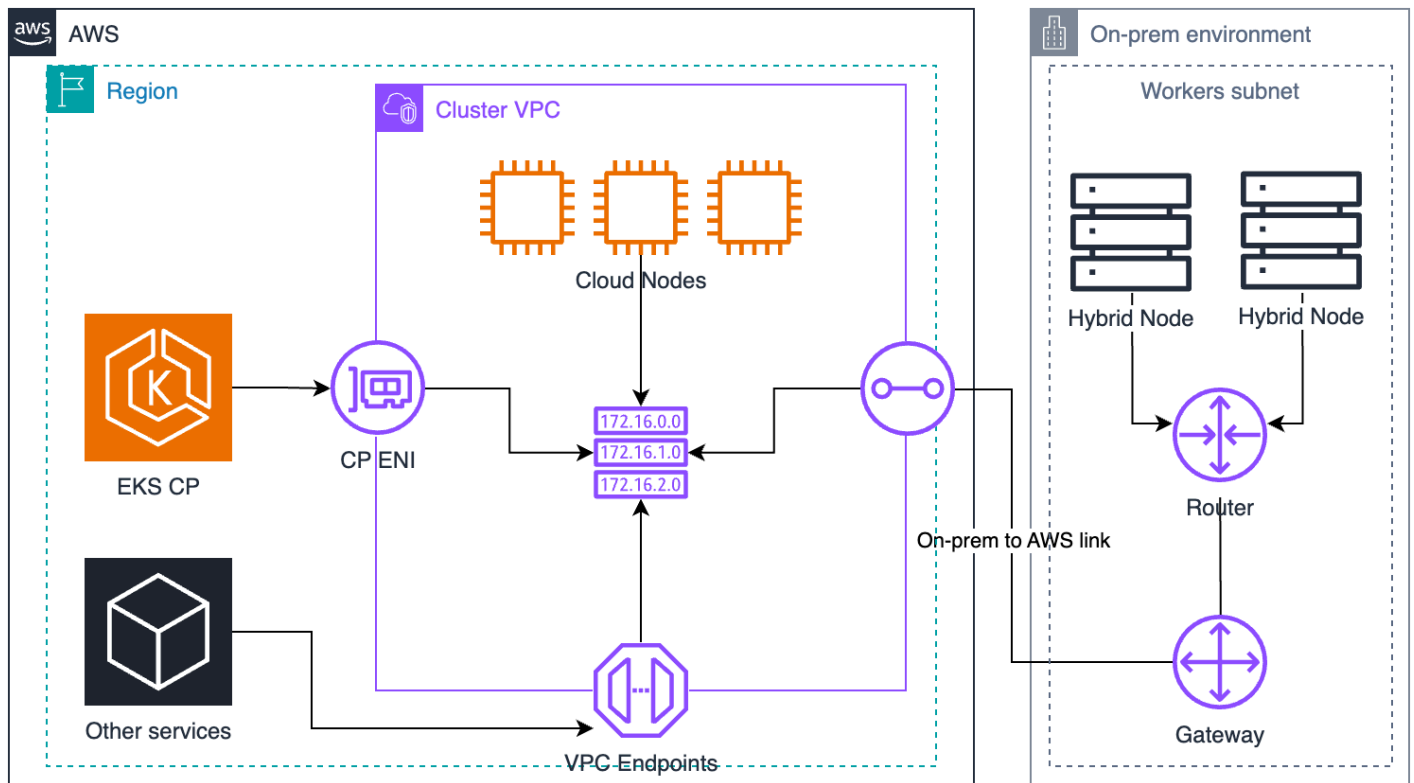
주제

- [하이브리드 노드의 네트워킹 개념](#)
- [하이브리드 노드에 대한 Kubernetes 개념](#)
- [하이브리드 노드의 네트워크 트래픽 흐름](#)

하이브리드 노드의 네트워킹 개념

이 섹션에서는 EKS Hybrid Nodes용 네트워크 토폴로지를 설계할 때 고려해야 하는 핵심 네트워킹 개념과 제약 조건을 자세히 설명합니다.

EKS Hybrid Nodes의 네트워킹 개념



네트워크 허브로서의 VPC

클라우드 경계를 통과하는 모든 트래픽은 VPC를 통해 라우팅됩니다. 여기에는 AWS에서 실행되는 EKS 컨트롤 플레인 또는 포드와 해당 포드에서 실행되는 하이브리드 노드 또는 포드 간의 트래픽이 포함됩니다. 클러스터의 VPC는 하이브리드 노드와 나머지 클러스터 사이의 네트워크 허브라고 생각할 수 있습니다. 이 아키텍처를 사용하면 트래픽과 라우팅을 완벽하게 제어할 수 있지만 VPC에 대한 경로, 보안 그룹 및 방화벽을 올바르게 구성하는 것도 사용자의 책임입니다.

VPC에 대한 EKS 컨트롤 플레인

EKS 컨트롤 플레인(VPC)에 탄력적 네트워크 인터페이스(ENI)를 연결합니다. 이러한 ENI는 EKS API 서버와의 트래픽을 처리합니다. EKS는 클러스터 생성 시 전달한 서브넷에 ENI를 연결하므로 클러스터를 구성할 때 EKS 컨트롤 플레인 ENI의 배치를 제어할 수 있습니다.

EKS는 서브넷에 연결하는 ENI에 보안 그룹을 연결합니다. 이러한 보안 그룹은 ENI를 통해 EKS 컨트롤 플레인을 오가는 트래픽을 허용합니다. 이는 EKS Hybrid Nodes에 중요합니다. 하이브리드 노드와 해당 노드에서 실행되는 포드에서 EKS 컨트롤 플레인 ENI로의 트래픽을 허용해야 하기 때문입니다.

원격 노드 네트워크

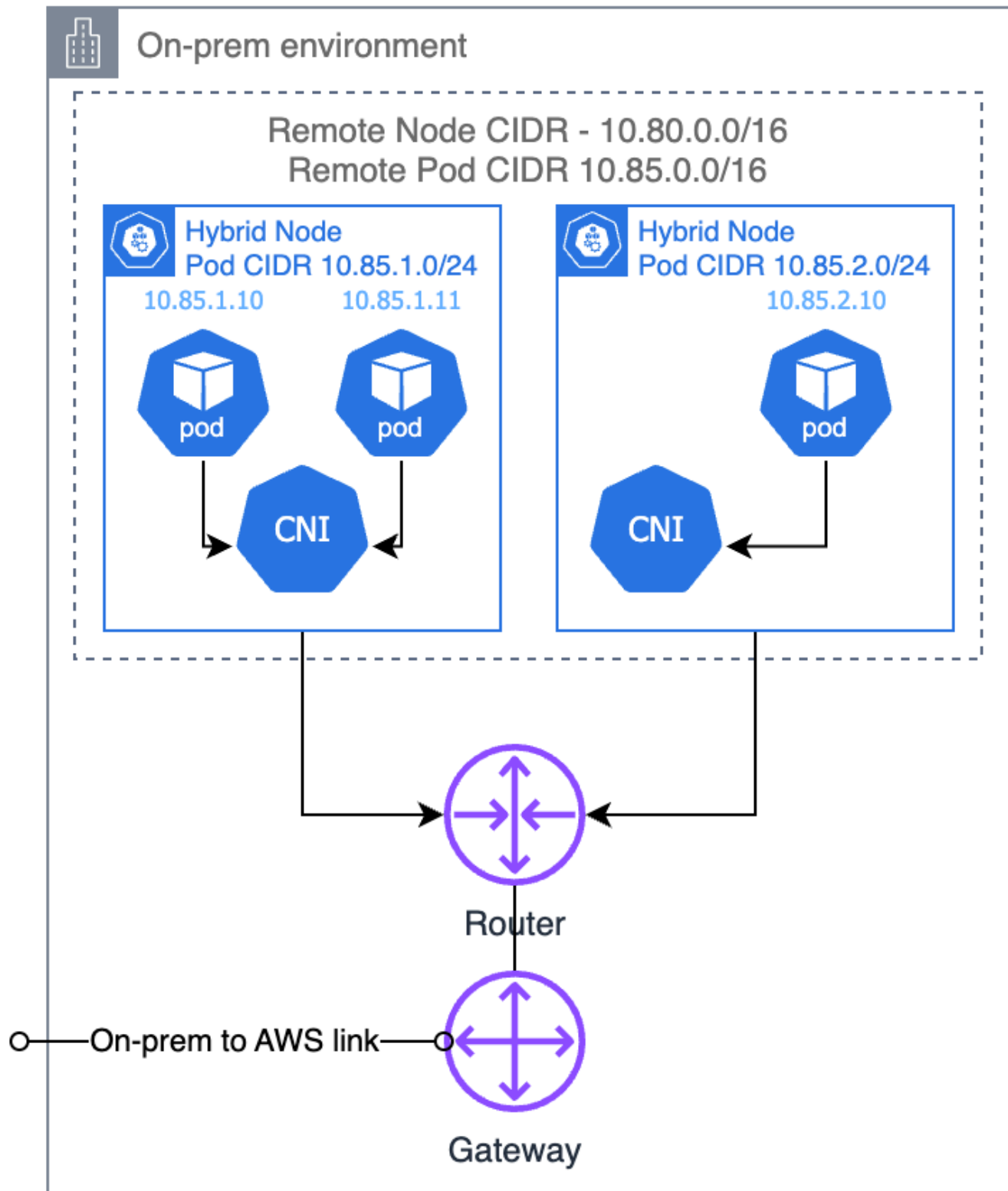
원격 노드 네트워크, 특히 원격 노드 CIDR은 하이브리드 노드로 사용하는 머신에 할당된 IP의 범위입니다. 하이브리드 노드를 프로비저닝하면 해당 노드는 EKS 컨트롤 플레인 및 VPC와 다른 네트워크 도메인인 온프레미스 데이터 센터 또는 엣지 로케이션에 상주합니다. 각 하이브리드 노드에는 VPC의 서브넷과 구별되는 원격 노드 CIDR의 IP 주소 또는 주소가 있습니다.

EKS가 kubelet API에 대한 요청과 같이 클러스터 VPC를 통해 하이브리드 노드 IP로 향하는 모든 트래픽을 라우팅하는 것을 알도록 이러한 원격 노드 CIDR로 EKS 클러스터를 구성합니다.

원격 포드 네트워크

원격 포드 네트워크는 하이브리드 노드에서 실행되는 포드에 할당된 IP의 범위입니다. 일반적으로 이러한 범위로 CNI를 구성하면 CNI의 IP Address Manager(IPAM) 기능이 이러한 범위의 조각을 각 하이브리드 노드에 할당합니다. 포드를 생성하면 CNI는 포드가 예약된 노드에 할당된 조각에서 포드에 IP를 할당합니다.

이러한 원격 포드 CIDR로 EKS 클러스터를 구성하면 EKS 컨트롤 플레인이 웹훅과의 통신과 같이 클러스터의 VPC를 통해 하이브리드 노드에서 실행되는 포드로 향하는 모든 트래픽을 라우팅하는 것을 압니다.



온프레미스에서 VPC로

하이브리드 노드에 사용하는 온프레미스 네트워크는 EKS 클러스터에 사용하는 VPC로 라우팅해야 합니다. 온프레미스 네트워크를 VPC에 연결하는 데 사용할 수 있는 여러 가지 [네트워크-Amazon VPC 연결 옵션](#)이 있습니다. 자체 VPN 솔루션을 사용할 수도 있습니다.

VPC와 온프레미스 네트워크의 AWS 클라우드 측에서 라우팅을 올바르게 구성하는 것이 중요합니다. 이렇게 하면 두 네트워크 모두 두 네트워크의 연결을 통해 올바른 트래픽을 라우팅할 수 있습니다.

VPC에서 원격 노드 및 원격 포드 네트워크로 이동하는 모든 트래픽은 연결을 통해 온프레미스 네트워크('게이트웨이'라고 함)로 라우팅되어야 합니다. 일부 서브넷의 라우팅 테이블이 다른 경우 하이브리드 노드의 경로와 해당 노드에서 실행되는 포드로 각 라우팅 테이블을 구성해야 합니다. 이는 EKS 컨트롤 플레인 ENI가 연결된 서브넷과 하이브리드 노드와 통신해야 하는 EC2 노드 또는 포드가 포함된 서브넷에 해당합니다.

온프레미스 네트워크에서 EKS 클러스터의 VPC 및 하이브리드 노드에 필요한 기타 AWS 서비스와의 트래픽을 허용하도록 네트워크를 구성해야 합니다. EKS 클러스터의 트래픽은 게이트웨이를 양방향으로 통과합니다.

네트워킹 제약 조건

완전히 라우팅된 네트워크

주요 제약 조건은 EKS 컨트롤 플레인과 모든 노드, 클라우드 또는 하이브리드 노드가 완전히 라우팅된 네트워크를 구성해야 한다는 것입니다. 즉, 모든 노드가 IP 주소를 기준으로 계층 3에서 서로 연결할 수 있어야 합니다.

EKS 컨트롤 플레인과 클라우드 노드는 일반 네트워크(VPC)에 있기 때문에 이미 서로 연결할 수 있습니다. 그러나 하이브리드 노드는 다른 네트워크 도메인에 있습니다. 따라서 하이브리드 노드와 클러스터의 나머지 부분 간에 트래픽을 라우팅하려면 VPC 및 온프레미스 네트워크에서 추가 라우팅을 구성해야 합니다. 하이브리드 노드가 서로 및 VPC에서 연결 가능한 경우 하이브리드 노드는 단일 일반 네트워크 또는 여러 세그먼트 네트워크에 있을 수 있습니다.

라우팅 가능한 원격 포드 CIDR

EKS 컨트롤 플레인이 하이브리드 노드(예: 웹후크 또는 지표 서버)에서 실행되는 포드와 통신하거나 클라우드 노드에서 실행되는 포드가 하이브리드 노드에서 실행되는 포드와 통신하려면(워크로드 동시 통신) 원격 포드 CIDR이 VPC에서 라우팅 가능해야 합니다. 즉, VPC는 게이트웨이를 통해 온프레미스 네트워크로 포드 CIDR로 트래픽을 라우팅할 수 있어야 하며 온프레미스 네트워크는 포드의 트래픽을 올바른 노드로 라우팅할 수 있어야 합니다.

VPC의 포트 라우팅 요구 사항과 온프레미스의 포트 라우팅 요구 사항을 구분하는 것이 중요합니다. VPC는 원격 포드로 이동하는 모든 트래픽이 게이트웨이를 통과해야 한다는 것을 알기만 하면 됩니다. 원격 포트 CIDR이 하나만 있는 경우 경로가 하나만 필요합니다.

이 요구 사항은 온프레미스 네트워크의 모든 홉에서 하이브리드 노드와 동일한 서브넷의 로컬 라우터까지 적용됩니다. 이는 각 노드에 할당된 포트 CIDR 조각을 인식해야 하는 유일한 라우터로, 특정 포트에 대한 트래픽이 포트가 예약된 노드로 전달되도록 합니다.

로컬 온프레미스 라우터에서 VPC 라우팅 테이블로 온프레미스 포트 CIDR에 대한 이러한 경로를 전파하도록 선택할 수 있지만 반드시 그럴 필요는 없습니다. 온프레미스 포트 CIDR이 자주 변경되고 VPC 라우팅 테이블을 변경된 포트 CIDR을 반영하도록 업데이트해야 하는 경우 온프레미스 포트 CIDR을 VPC 라우팅 테이블로 전파하는 것이 좋지만 이는 혼하지 않습니다.

온프레미스 포트 CIDR을 라우팅 가능하게 만드는 제약 조건은 선택 사항입니다. 하이브리드 노드에서 웹훅을 실행할 필요가 없거나 클라우드 노드의 포트가 하이브리드 노드의 포트와 통신하는 경우 온프레미스 네트워크에서 포트 CIDR에 대한 라우팅을 구성할 필요가 없습니다.

온프레미스 포트 CIDR을 하이브리드 노드로 라우팅해야 하는 이유는 무엇인가요?

클라우드 노드에 VPC CNI와 함께 EKS를 사용하는 경우 VPC CNI는 VPC에서 포트에 IP를 직접 할당합니다. 즉, 클라우드 포트와 EKS 컨트롤 플레인 모두 포트 IP에 직접 도달할 수 있으므로 특별한 라우팅이 필요하지 않습니다.

온프레미스 및 클라우드의 다른 CNI에서 실행하는 경우 포트는 일반적으로 격리된 오버레이 네트워크에서 실행되고 CNI는 포트 간 트래픽 전달을 처리합니다. 이는 일반적으로 캡슐화를 통해 수행됩니다. CNI는 포트 간 트래픽을 노드 간 트래픽으로 변환하여 양쪽 끝에서 캡슐화 및 캡슐화 해제를 처리합니다. 이렇게 하면 노드 및 라우터에서 추가 구성이 필요하지 않습니다.

하이브리드 노드를 사용한 네트워킹은 두 가지 토폴로지의 조합을 제공한다는 점에서 독특합니다. EKS 컨트롤 플레인과 클라우드 노드(VPC CNI 사용)는 노드와 포드를 포함한 플랫폼 네트워크를 기대하는 반면, 하이브리드 노드에서 실행되는 포트는 캡슐화를 위해 VXLAN을 사용하여 오버레이 네트워크에 있습니다(기본적으로 Cilium에서). 온프레미스 네트워크가 VPC로 라우팅될 수 있다고 가정하면 하이브리드 노드에서 실행되는 포트는 EKS 컨트롤 플레인과 클라우드 노드에서 실행되는 포트에 도달할 수 있습니다. 그러나 온프레미스 네트워크의 포트 CIDR에 대한 라우팅이 없으면 네트워크가 오버레이 네트워크에 도달하고 올바른 노드로 라우팅하는 방법을 모르는 경우 온프레미스 포트 IP로 다시 들어오는 모든 트래픽이 결국 삭제됩니다.

하이브리드 노드에 대한 Kubernetes 개념

이 페이지에서는 EKS Hybrid Nodes 시스템 아키텍처의 기반이 되는 주요 Kubernetes 개념을 자세히 설명합니다.

VPC의 EKS 컨트롤 플레인

EKS 컨트롤 플레인 ENI의 IP는 default 네임스페이스의 kubernetes Endpoints 객체에 저장됩니다. EKS는 새 ENI를 생성하거나 이전 ENI를 제거할 때 IP 목록이 항상 최신 상태로 유지되도록 이 객체를 업데이트합니다.

이러한 엔드포인트는 default 네임스페이스에서도 kubernetes 서비스를 통해 사용할 수 있습니다. ClusterIP 유형의 이 서비스는 항상 클러스터 서비스 CIDR의 첫 번째 IP 주소를 할당받습니다. 예를 들어 서비스 CIDR 172.16.0.0/16의 경우 서비스 IP는 172.16.0.1이 됩니다.

일반적으로 이 방식으로 포드(클라우드 또는 하이브리드 노드에서 실행되는지 여부에 관계 없음)가 EKS Kubernetes API 서버에 액세스합니다. 포드는 서비스 IP를 대상 IP로 사용하며, 이는 EKS 컨트롤 플레인 ENI 중 하나의 실제 IP로 변환됩니다. 프라이머리 예외는 kube-proxy인데, 이는 변환을 설정하기 때문입니다.

EKS API 서버 엔드포인트

kubernetes 서비스 IP가 EKS API 서버에 액세스하는 유일한 방법은 아닙니다. 또한 EKS는 클러스터를 생성할 때 Route53 DNS 이름을 생성합니다. 이는 EKS DescribeCluster API 작업을 직접적으로 호출할 때 EKS 클러스터의 endpoint 필드입니다.

```
{
  "cluster": {
    "endpoint": "https://xxxxxxxxxxxxxxxxxxxxxxxxxxxxx.gr7.us-west-2.eks.amazonaws.com",
    "name": "my-cluster",
    "status": "ACTIVE"
  }
}
```

퍼블릭 엔드포인트 액세스 또는 퍼블릭 및 프라이빗 엔드포인트 액세스 클러스터에서 하이브리드 노드는 기본적으로 이 DNS 이름을 인터넷을 통해 라우팅 가능한 퍼블릭 IP로 확인합니다. 프라이빗 엔드포인트 액세스 클러스터에서 DNS 이름은 EKS 컨트롤 플레인 ENI의 프라이빗 IP로 확인됩니다.

이것이 kubelet과 kube-proxy가 Kubernetes API 서버에 액세스하는 방식입니다. 모든 Kubernetes 클러스터 트래픽이 VPC를 통해 흐르도록 하려면 클러스터를 프라이빗 액세스 모드로 구성하거나 온 프레미스 DNS 서버를 수정하여 EKS 클러스터 엔드포인트를 EKS 컨트롤 플레인 ENI의 프라이빗 IP로 확인해야 합니다.

kubelet 엔드포인트

kubelet은 여러 REST 엔드포인트를 노출하여 시스템의 다른 부분이 각 노드와 상호 작용하고 정보를 수집할 수 있도록 합니다. 대부분의 클러스터에서 kubelet 서버로 전송되는 트래픽의 대부분은 컨트롤 플레인에서 발생하지만, 특정 모니터링 에이전트도 kubelet 서버와 상호 작용할 수 있습니다.

이 인터페이스를 통해 kubelet은 로그 가져오기(`kubectl logs`), 컨테이너 내부에서 명령 실행(`kubectl exec`), 포트 포워딩 트래픽(`kubectl port-forward`) 등 다양한 요청을 처리합니다. 이러한 각 요청은 kubelet을 통해 기본 컨테이너 런타임과 상호 작용하여 클러스터 관리자와 개발자에게 원활하게 나타납니다.

이 API의 가장 일반적인 소비자는 Kubernetes API 서버입니다. 앞서 언급한 `kubectl` 명령 중 하나를 사용할 때 `kubectl`은 API 서버에 API 요청을 보낸 다음 포드가 실행되고 있는 노드의 kubelet API를 직접적으로 호출합니다. 이것이 EKS 컨트롤 플레인에서 노드 IP에 도달할 수 있어야 하는 주된 이유이며, 노드 경로가 잘못 구성되면 포드가 실행 중이더라도 로그나 `exec`에 접근할 수 없는 이유이기도 합니다.

노드 IP

EKS 컨트롤 플레인이 노드와 통신할 때 Node 객체 상태(`status.addresses`)에 보고된 주소 중 하나를 사용합니다.

EKS 클라우드 노드를 사용하면 노드 등록 중 kubelet이 EC2 인스턴스의 프라이빗 IP를 `InternalIP`로 보고하는 것이 일반적입니다. 이후 Cloud Controller Manager(CCM)가 이 IP를 검증하여 해당 IP가 EC2 인스턴스에 속하는지 확인합니다. 또한 CCM은 일반적으로 인스턴스의 퍼블릭 IP(`ExternalIP`)와 DNS 이름(`InternalDNS` 및 `ExternalDNS`)을 노드 상태에 추가합니다.

그러나 하이브리드 노드용 CCM은 없습니다. EKS Hybrid Nodes CLI(`nodeadm`)를 사용하여 하이브리드 노드를 등록하면 CCM 없이 노드 상태에서 머신의 IP를 직접 보고하도록 kubelet이 구성됩니다.

```
apiVersion: v1
kind: Node
metadata:
  name: my-node-1
spec:
  providerID: eks-hybrid:///us-west-2/my-cluster/my-node-1
status:
  addresses:
    - address: 10.1.1.236
      type: InternalIP
```

```
- address: my-node-1
  type: Hostname
```

머신에 여러 IP가 있는 경우 kubelet은 자체 로직에 따라 IP 중 하나를 선택합니다.

spec.kubelet.flags의 nodeadm 구성에서 전달할 수 있는 `--node-ip` 플래그를 사용하여 선택한 IP를 제어할 수 있습니다. Node 객체에 보고된 IP에만 VPC의 경로가 필요합니다. 머신에는 클라우드에서 연결할 수 없는 다른 IP가 있을 수 있습니다.

kube-proxy

kube-proxy는 각 노드의 네트워킹 계층에서 서비스 추상화 구현을 담당합니다. 또한 Kubernetes Services로 향하는 트래픽에 대한 네트워크 프록시 및 로드 밸런서 역할을 합니다. kube-proxy는 Kubernetes API 서버에서 서비스 및 엔드포인트와 관련된 변경 사항을 지속적으로 감시해서 기본 호스트의 네트워킹 규칙을 동적으로 업데이트하여 트래픽이 제대로 전달되도록 합니다.

iptables 모드에서 kube-proxy는 여러 개의 netfilter 체인을 프로그래밍하여 서비스 트래픽을 처리합니다. 규칙은 다음과 같은 계층 구조를 형성합니다.

1. KUBE-SERVICES 체인: 모든 서비스 트래픽의 진입점입니다. 각 서비스의 ClusterIP 및 포트와 일치하는 규칙이 있습니다.
2. KUBE-SVC-XXX 체인: 서비스별 체인에는 각 서비스에 대한 로드 밸런싱 규칙이 있습니다.
3. KUBE-SEP-XXX 체인: 엔드포인트별 체인에는 실제 DNAT 규칙이 있습니다.

default 네임스페이스의 서비스 test-server에서 어떤 일이 발생하는지 살펴보겠습니다. * 서비스 클러스터 IP: 172.16.31.14 * 서비스 포트: 80 * 백업 포트: 10.2.0.110, 10.2.1.39 및 10.2.2.254

iptables 규칙을 검사할 때(iptables-save | grep -A10 KUBE-SERVICES 사용)

1. KUBE-SERVICES 체인에서 서비스와 일치하는 규칙을 찾습니다.

```
-A KUBE-SERVICES -d 172.16.31.14/32 -p tcp -m comment --comment "default/test-server cluster IP" -m tcp --dport 80 -j KUBE-SVC-XYZABC123456
```

- 이 규칙은 172.16.31.14:80을 대상으로 하는 패킷과 일치합니다.
 - 주석은 이 규칙의 용도를 나타냅니다(default/test-server cluster IP).
 - 일치하는 패킷은 KUBE-SVC-XYZABC123456 체인으로 이동합니다.
2. KUBE-SVC-XYZABC123456 체인에는 확률 기반 로드 밸런싱 규칙이 있습니다.


```
-A KUBE-SVC-XYZABC123456 -m statistic --mode random --probability 0.33333333349 -j
KUBE-SEP-POD1XYZABC
-A KUBE-SVC-XYZABC123456 -m statistic --mode random --probability 0.50000000000 -j
KUBE-SEP-POD2XYZABC
-A KUBE-SVC-XYZABC123456 -j KUBE-SEP-POD3XYZABC
```

- 첫 번째 규칙: 33.3%의 확률로 KUBE-SEP-POD1XYZABC로 이동합니다.
- 두 번째 규칙: 50%의 확률로 나머지 트래픽(전체의 33.3%)이 KUBE-SEP-POD2XYZABC로 이동합니다.
- 마지막 규칙: 나머지 모든 트래픽(전체의 33.3%)이 KUBE-SEP-POD3XYZABC로 이동합니다.

3. 개별 KUBE-SEP-XXX 체인이 DNAT(Destination NAT)를 수행합니다.

```
-A KUBE-SEP-POD1XYZABC -p tcp -m tcp -j DNAT --to-destination 10.2.0.110:80
-A KUBE-SEP-POD2XYZABC -p tcp -m tcp -j DNAT --to-destination 10.2.1.39:80
-A KUBE-SEP-POD3XYZABC -p tcp -m tcp -j DNAT --to-destination 10.2.2.254:80
```

- 이러한 DNAT 규칙은 대상 IP와 포트를 다시 작성하여 트래픽을 특정 포드로 전달합니다.
- 각 규칙은 트래픽의 약 33.3%를 처리하여 10.2.0.110, 10.2.1.39 및 10.2.2.254 간에 균일한 로드 밸런싱을 제공합니다.

이러한 멀티 레벨 체인 구조를 통해 kube-proxy는 데이터 경로에 프록시 프로세스가 필요 없이 커널 레벨 패킷 조작을 통해 서비스 로드 밸런싱 및 리디렉션을 효율적으로 구현할 수 있습니다.

Kubernetes 작업에 미치는 영향

노드에서 kube-proxy가 손상되면 해당 노드가 서비스 트래픽을 제대로 라우팅할 수 없어 클러스터 서비스에 의존하는 포드에 제한 시간 초과나 연결 실패가 발생합니다. 이는 노드가 처음 등록될 때 특히 방해가 될 수 있습니다. CNI는 포드 네트워킹을 구성하기 전에 노드의 포드 CIDR과 같은 정보를 얻기 위해 Kubernetes API 서버와 통신해야 합니다. 이를 위해 CNI는 kubernetes 서비스 IP를 사용합니다. 그러나 kube-proxy가 시작되지 못하거나 올바른 iptables 규칙을 설정하지 못한 경우 kubernetes 서비스 IP로 전송되는 요청은 EKS 컨트롤 플레인 ENI의 실제 IP로 변환되지 않습니다. 따라서 CNI는 충돌 루프에 들어가고 포드 중 어느 것도 제대로 실행되지 않습니다.

포드가 kubernetes 서비스 IP를 사용하여 Kubernetes API 서버와 통신한다는 것을 알고 있지만, 이를 작동시키려면 먼저 kube-proxy가 iptables 규칙을 설정해야 합니다.

kube-proxy는 API 서버와 어떻게 통신하나요?

kube-proxy는 Kubernetes API 서버의 실제 IP 또는 해당 IP로 확인되는 DNS 이름을 사용하도록 구성해야 합니다. EKS의 경우 EKS는 클러스터 생성 시 EKS가 생성하는 Route53 DNS 이름을 가리키도록 기본 kube-proxy를 구성합니다. 이 값은 kube-system 네임스페이스의 kube-proxy ConfigMap에서 확인할 수 있습니다. 이 ConfigMap의 내용은 kube-proxy 포트에 삽입되는 kubeconfig이므로 clusters0.cluster.server 필드를 확인하세요. 이 값은 EKS DescribeCluster API 직접 호출 시 EKS 클러스터의 endpoint 필드와 일치합니다.

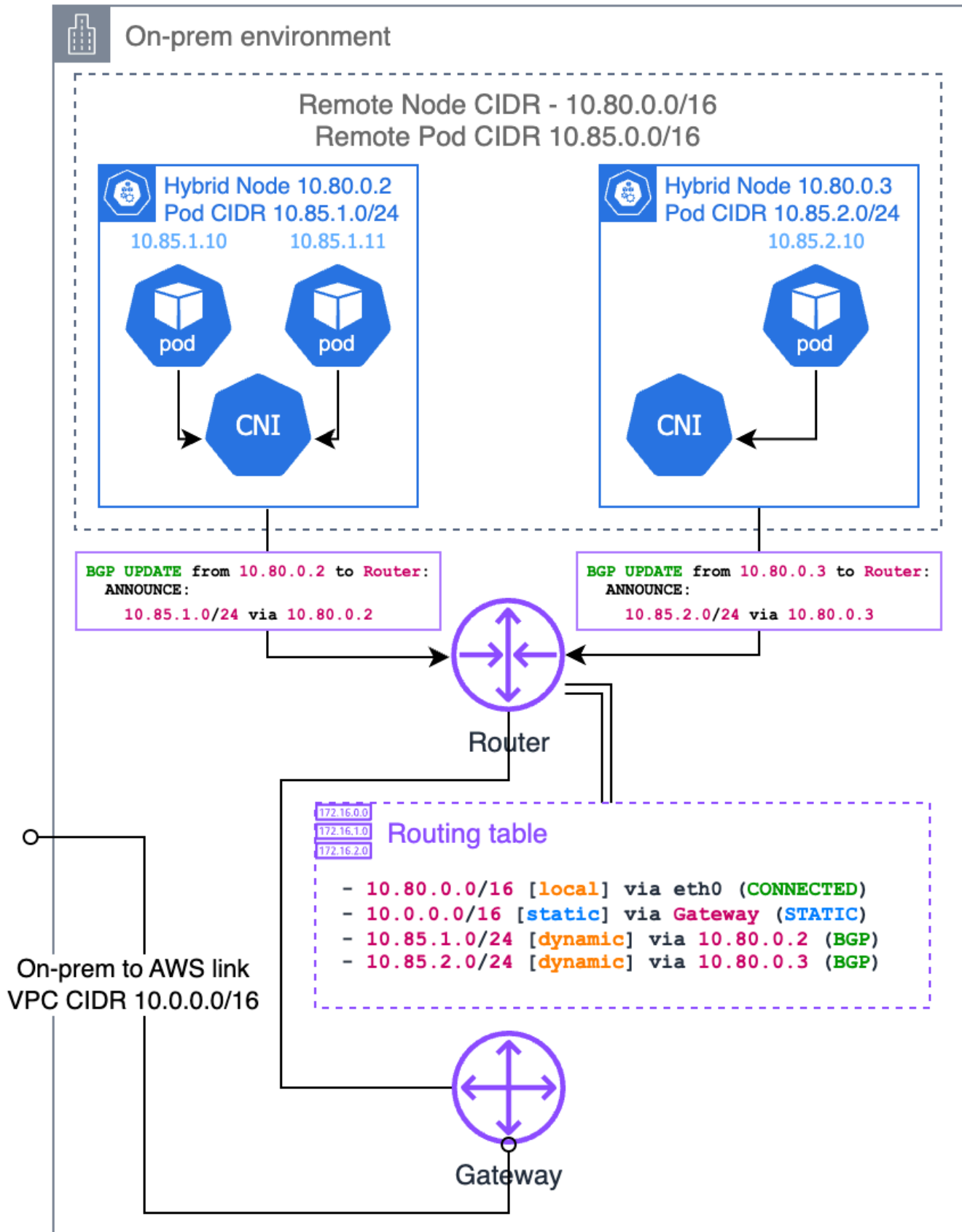
```
apiVersion: v1
data:
  kubeconfig: |-
    kind: Config
    apiVersion: v1
    clusters:
    - cluster:
        certificate-authority: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        server: https://xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.gr7.us-
west-2.eks.amazonaws.com
        name: default
    contexts:
    - context:
        cluster: default
        namespace: default
        user: default
        name: default
    current-context: default
    users:
    - name: default
      user:
        tokenFile: /var/run/secrets/kubernetes.io/serviceaccount/token
kind: ConfigMap
metadata:
  name: kube-proxy
  namespace: kube-system
```

라우팅 가능한 원격 포트 CIDR

이 [the section called “네트워킹 개념”](#) 페이지에서는 하이브리드 노드에서 웹후크를 실행하거나 클라우드 노드에서 실행되는 포트가 하이브리드 노드에서 실행되는 포트와 통신하도록 하는 요구 사항을 자세히 설명합니다. 핵심 요구 사항은 온프레미스 라우터가 특정 포트 IP를 담당하는 노드를 알아야 한다는 것입니다. Border Gateway Protocol(BGP), 정적 경로, 주소 확인 프로토콜(ARP) 프록시 등 이를 달성하는 방법에는 여러 가지가 있습니다. 이들은 다음 섹션에서 다룹니다.

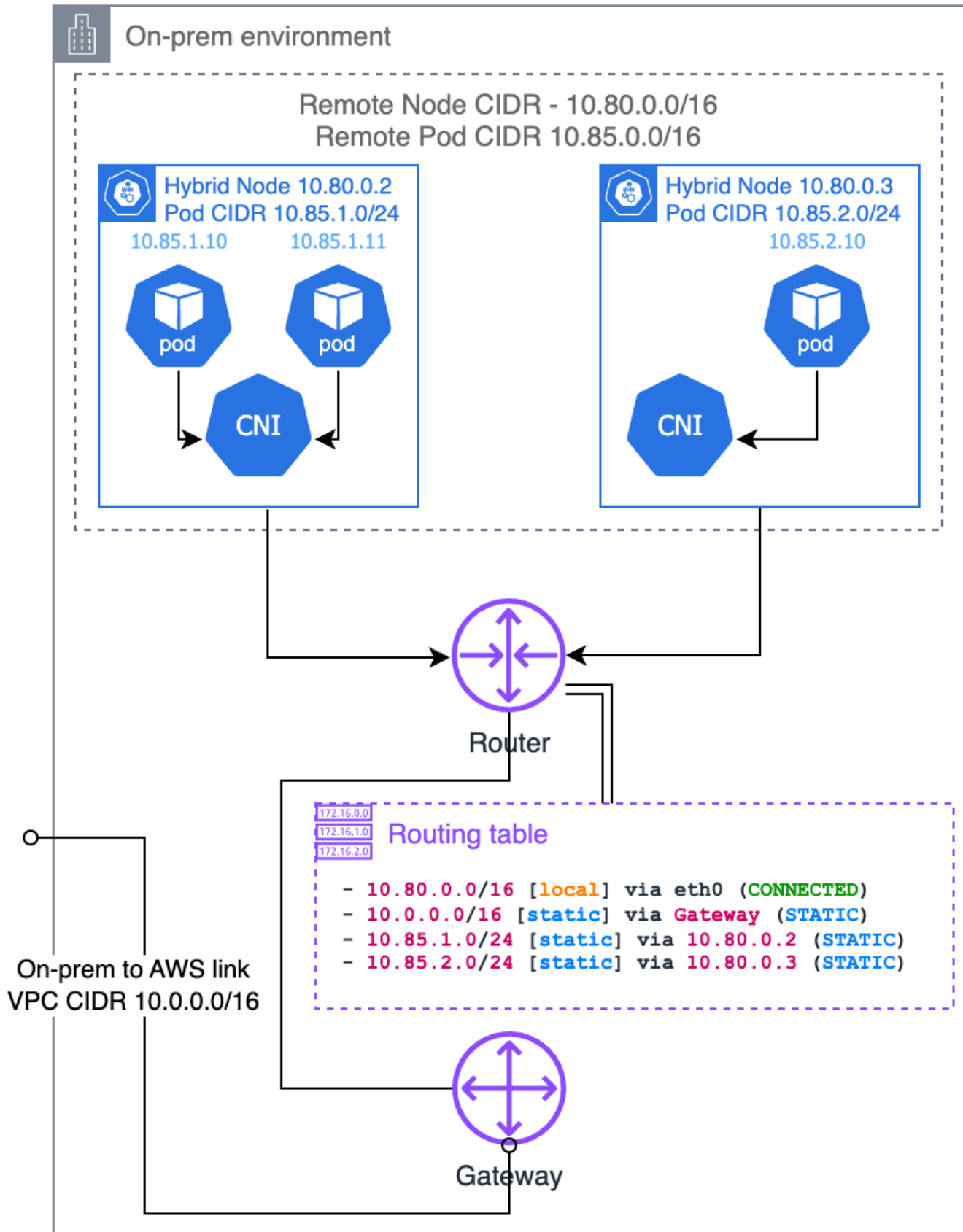
Border Gateway Protocol(BGP)

CNI가 이를 지원하는 경우(예: Cilium 및 Calico), CNI의 BGP 모드를 사용하여 노드에서 로컬 라우터로 노드별 포트 CIDR에 대한 경로를 전파할 수 있습니다. CNI의 BGP 모드를 사용할 때 CNI는 가상 라우터 역할을 하므로 로컬 라우터는 포트 CIDR이 다른 서브넷에 속하고 노드가 해당 서브넷의 게이트웨이라고 간주합니다.



정적 경로

또는 로컬 라우터에서 정적 경로를 구성할 수 있습니다. 이는 온프레미스 포트 CIDR을 VPC로 라우팅하는 가장 간단한 방법이지만 가장 오류가 발생하기 쉽고 유지 관리하기 어렵습니다. 기존 노드와 할당된 포트 CIDR을 사용하여 경로가 항상 최신 상태인지 확인해야 합니다. 노드 수가 적고 인프라가 정적이면 실행 가능한 옵션이며 라우터에서 BGP를 지원할 필요가 없습니다. 이 옵션을 선택하는 경우 IPAM이 결정하도록 하는 대신 각 노드에 할당하려는 포트 CIDR 조각으로 CNI를 구성하는 것이 좋습니다.



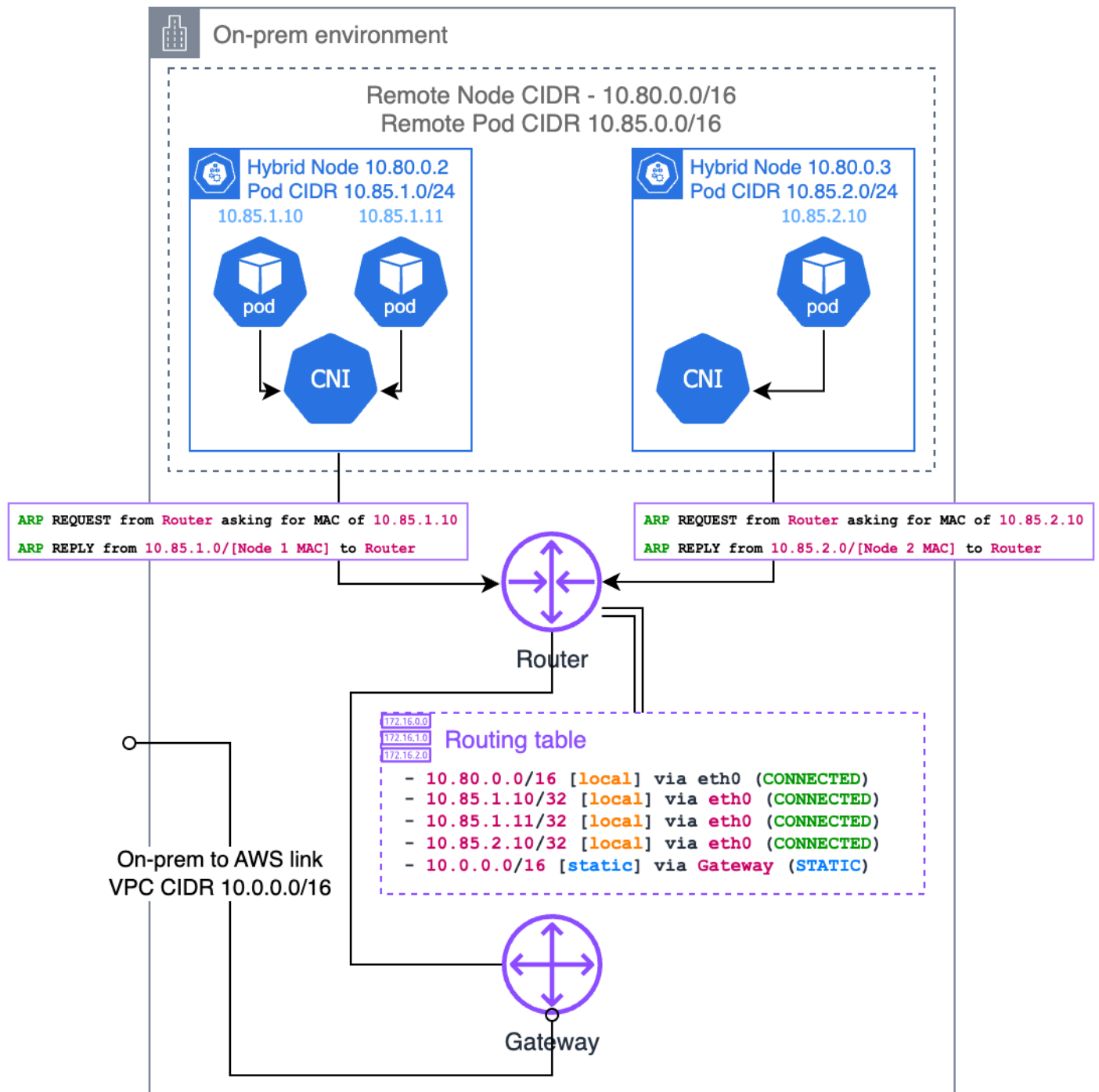
주소 확인 프로토콜(ARP) 프록시

ARP 프록시는 온프레미스 포트 IP를 라우팅 가능하게 만드는 또 다른 접근 방식이며, 하이브리드 노드가 로컬 라우터와 동일한 계층 2 네트워크에 있을 때 특히 유용합니다. ARP 프록싱이 활성화되면 노드는 자신이 호스팅하는 포트 IP에 대한 ARP 요청에 응답하는데, 해당 IP가 다른 서브넷에 속하더라도 마찬가지입니다.

로컬 네트워크의 디바이스가 포트 IP에 도달하려고 할 때 먼저 '누가 이 IP를 가지고 있나요?'라는 ARP 요청을 전송합니다. 포트가 자체 MAC 주소로 응답하는 하이브리드 노드 호스팅으로, '해당 IP에 대한 트래픽을 처리할 수 있습니다.'라는 메시지가 표시됩니다. 이를 통해 라우터 구성 없이 로컬 네트워크의 디바이스와 포트 간에 직접 경로가 생성됩니다.

이렇게 하려면 CNI가 프록시 ARP 기능을 지원해야 합니다. Cilium에는 구성을 통해 활성화할 수 있는 프록시 ARP에 대한 지원이 내장되어 있습니다. 가장 중요한 고려 사항은 포트 CIDR이 환경 내의 다른 네트워크와 겹치지 않아야 한다는 것입니다. 겹치면 라우팅 충돌이 발생할 수 있습니다.

이 접근 방식에는 다음과 같은 몇 가지 이점이 있습니다. * BGP로 라우터를 구성하거나 정적 경로를 유지할 필요가 없습니다. * 라우터 구성을 제어할 수 없는 환경에서 잘 작동합니다.



포드 간 캡슐화

온프레미스 환경에서 CNI는 일반적으로 캡슐화 프로토콜을 사용하여 물리적 네트워크상에서 작동할 수 있는 오버레이 네트워크를 생성하므로 네트워크를 재구성할 필요가 없습니다. 이 섹션에서는 이러한 캡슐화가 어떻게 작동하는지 설명합니다. 일부 세부 정보는 사용하는 CNI에 따라 다를 수 있습니다.

캡슐화는 원래의 포트 네트워크 패킷을 기본 물리적 네트워크를 통해 라우팅될 수 있는 다른 네트워크 패킷으로 묶습니다. 이를 통해 포트는 물리적 네트워크가 해당 포트 CIDR을 라우팅하는 방법을 알 필요 없이 동일한 CNI를 실행하는 노드 간에 통신할 수 있습니다.

Kubernetes에서 가장 일반적으로 사용되는 캡슐화 프로토콜은 VXLAN(Virtual Extensible LAN)이지만 CNI에 따라 Geneve 등의 다른 프로토콜도 사용할 수 있습니다.

VXLAN 캡슐화

VXLAN은 UDP 패킷 내에서 계층 2 이더넷 프레임을 캡슐화합니다. 포트가 다른 노드의 다른 포드로 트래픽을 전송하면 CNI는 다음을 수행합니다.

1. CNI가 포트 A에서 패킷을 가로챍니다.
2. CNI가 원래 패킷을 VXLAN 헤더로 래핑합니다.
3. 그러면 이 래핑된 패킷이 노드의 일반 네트워킹 스택을 통해 대상 노드로 전송됩니다.
4. 대상 노드의 CNI가 패킷을 언래핑하여 포트 B로 전송합니다.

VXLAN 캡슐화 중 패킷 구조에는 다음과 같은 일이 일어납니다.

원래 포트 간 패킷:

```
+-----+-----+-----+-----+
| Ethernet Header | IP Header   | TCP/UDP   | Payload       |
| Src: Pod A MAC  | Src: Pod A IP | Src Port  |                |
| Dst: Pod B MAC  | Dst: Pod B IP | Dst Port  |                |
+-----+-----+-----+-----+
```

VXLAN 캡슐화 후

```
+-----+-----+-----+-----+
+-----+
| Outer Ethernet  | Outer IP    | Outer UDP  | VXLAN         | Original Pod-to-Pod
|                |              |             |               |
| Src: Node A MAC | Src: Node A | Src: Random | VNI: xx      | Packet (unchanged
|                |              |             |               |
| Dst: Node B MAC | Dst: Node B | Dst: 4789   |               | from above)
|                |              |             |               |
+-----+-----+-----+-----+
+-----+
```

VXLAN 네트워크 식별자(VNI)는 서로 다른 오버레이 네트워크를 구분합니다.

포드 통신 시나리오

동일한 하이브리드 노드의 포드

동일한 하이브리드 노드의 포드가 통신하는 경우 일반적으로 캡슐화가 필요하지 않습니다. CNI는 노드의 내부 가상 인터페이스를 통해 포드 간 트래픽을 보내는 로컬 경로를 설정합니다.

```
Pod A -> veth0 -> node's bridge/routing table -> veth1 -> Pod B
```

패킷은 절대 노드를 벗어나지 않으며 캡슐화가 불필요합니다.

다양한 하이브리드 노드의 포드

서로 다른 하이브리드 노드의 포드 간 통신에는 캡슐화가 필요합니다.

```
Pod A -> CNI -> [VXLAN encapsulation] -> Node A network -> router or gateway -> Node B network -> [VXLAN decapsulation] -> CNI -> Pod B
```

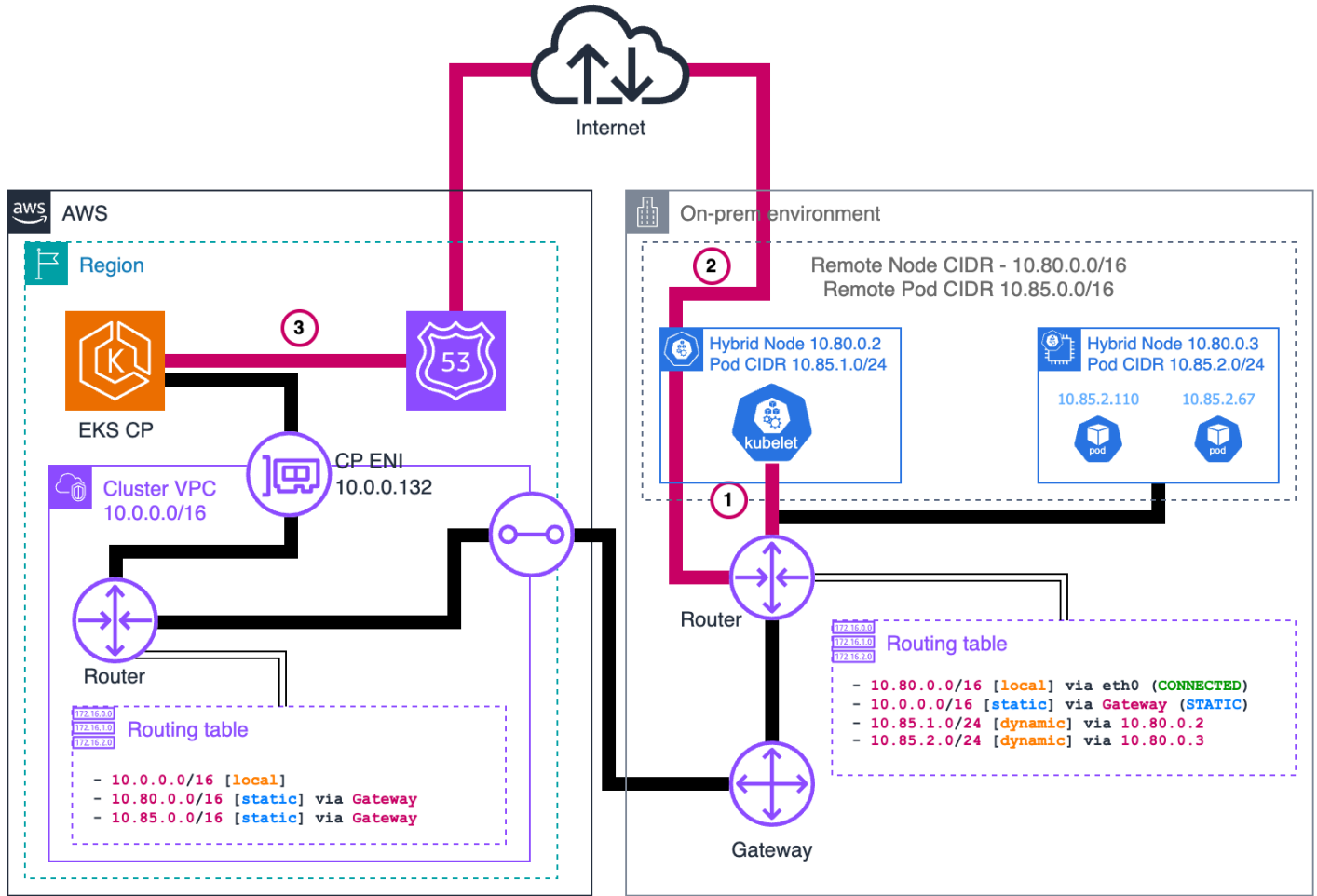
이렇게 하면 물리적 네트워크가 포드 IP 라우팅을 이해할 필요 없이 포드 트래픽이 물리적 네트워크 인프라를 통과할 수 있습니다.

하이브리드 노드의 네트워크 트래픽 흐름

이 페이지에서는 다양한 트래픽 유형에 대한 엔드 투 엔드 네트워크 경로를 보여주는 다이어그램과 함께 EKS Hybrid Nodes의 네트워크 트래픽 흐름을 자세히 설명합니다.

다음 트래픽 흐름이 포함됩니다.

- [the section called “하이브리드 노드 kubelet에서 EKS 컨트롤 플레인으로”](#)
- [the section called “EKS 컨트롤 플레인에서 하이브리드 노드로\(kubelet 서버\)”](#)
- [the section called “하이브리드 노드에서 실행되는 포드에서 EKS 컨트롤 플레인으로”](#)
- [the section called “하이브리드 노드에서 실행되는 포드에 대한 EKS 컨트롤 플레인\(웹후크\)”](#)
- [the section called “하이브리드 노드에서 실행되는 포드 간”](#)
- [the section called “클라우드 노드의 포드에서 하이브리드 노드의 포드로\(동서 트래픽\)”](#)

하이브리드 노드 **kubelet**에서 EKS 컨트롤 플레인으로

요청

1. **kubelet** 요청 시작

하이브리드 노드의 **kubelet**이 EKS 컨트롤 플레인과 통신해야 하는 경우(예: 노드 상태 보고 또는 포트 사양 가져오기), 노드 등록 중 제공된 kubeconfig 파일을 사용합니다. 이 kubeconfig에는 직접 IP 주소가 아닌 API 서버 엔드포인트 URL(Route53 DNS 이름)이 있습니다.

kubelet은 엔드포인트(예: <https://xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx.gr7.us-west-2.eks.amazonaws.com>)에 대한 DNS 조회를 수행합니다. 퍼블릭 액세스 클러스터에서는 AWS에서 실행되는 EKS 서비스에 속하는 퍼블릭 IP 주소(예: 54.239.118.52)로 확인됩니다. 그런 다음 **kubelet**은 이 엔드포인트에 대한 보안 HTTPS 요청을 생성합니다. 초기 패킷은 다음과 같습니다.

```
+-----+-----+-----+-----+
```

IP Header	TCP Header	Payload
Src: 10.80.0.2	Src: 52390 (random)	
Dst: 54.239.118.52	Dst: 443	

2. 로컬 라우터 라우팅

대상 IP가 로컬 네트워크의 일부가 아닌 퍼블릭 IP 주소이므로 kubelet은 이 패킷을 기본 게이트웨이(로컬 온프레미스 라우터)로 전송합니다. 라우터는 대상 IP를 검사하고 퍼블릭 IP 주소인지 확인합니다.

퍼블릭 트래픽의 경우 라우터는 일반적으로 인터넷으로의 아웃바운드 트래픽을 처리하는 인터넷 게이트웨이 또는 경계 라우터로 패킷을 전달합니다. 이는 다이어그램에서 생략되며 온프레미스 네트워크 설정 방식에 따라 달라집니다. 패킷은 온프레미스 네트워크 인프라를 통과하여 결국 인터넷 서비스 제공업체의 네트워크에 도달합니다.

3. EKS 컨트롤 플레인으로 전송

패킷은 AWS의 네트워크에 도달할 때까지 퍼블릭 인터넷과 전송 네트워크를 통해 이동합니다. AWS의 네트워크는 패킷을 적절한 리전의 EKS 서비스 엔드포인트로 라우팅합니다. 패킷이 EKS 서비스에 도달하면 클러스터의 실제 EKS 컨트롤 플레인으로 전달됩니다.

퍼블릭 인터넷을 통한 이 라우팅은 다른 트래픽 흐름에서 볼 수 있는 프라이빗 VPC 라우팅 경로와 다릅니다. 주요 차이점은 퍼블릭 액세스 모드를 사용할 때 온프레미스 kubelet(포드가 아님)에서 EKS 컨트롤 플레인으로의 트래픽이 VPC를 통과하지 않고 대신 글로벌 인터넷 인프라를 사용한다는 것입니다.

응답

EKS 컨트롤 플레인이 kubelet 요청을 처리한 후 응답을 다시 전송합니다.

3. EKS 컨트롤 플레인에서 응답 전송

EKS 컨트롤 플레인은 응답 패킷을 생성합니다. 이 패킷에는 소스로 퍼블릭 IP, 대상으로 하이브리드 노드의 IP가 있습니다.

IP Header	TCP Header	Payload
Src: 54.239.118.52	Src: 443	
Dst: 10.80.0.2	Dst: 52390	

2. 인터넷 라우팅

응답 패킷은 인터넷 서비스 제공업체가 결정한 라우팅 경로에 따라 온프레미스 네트워크 엣지 라우터에 도달할 때까지 인터넷을 통해 다시 이동합니다.

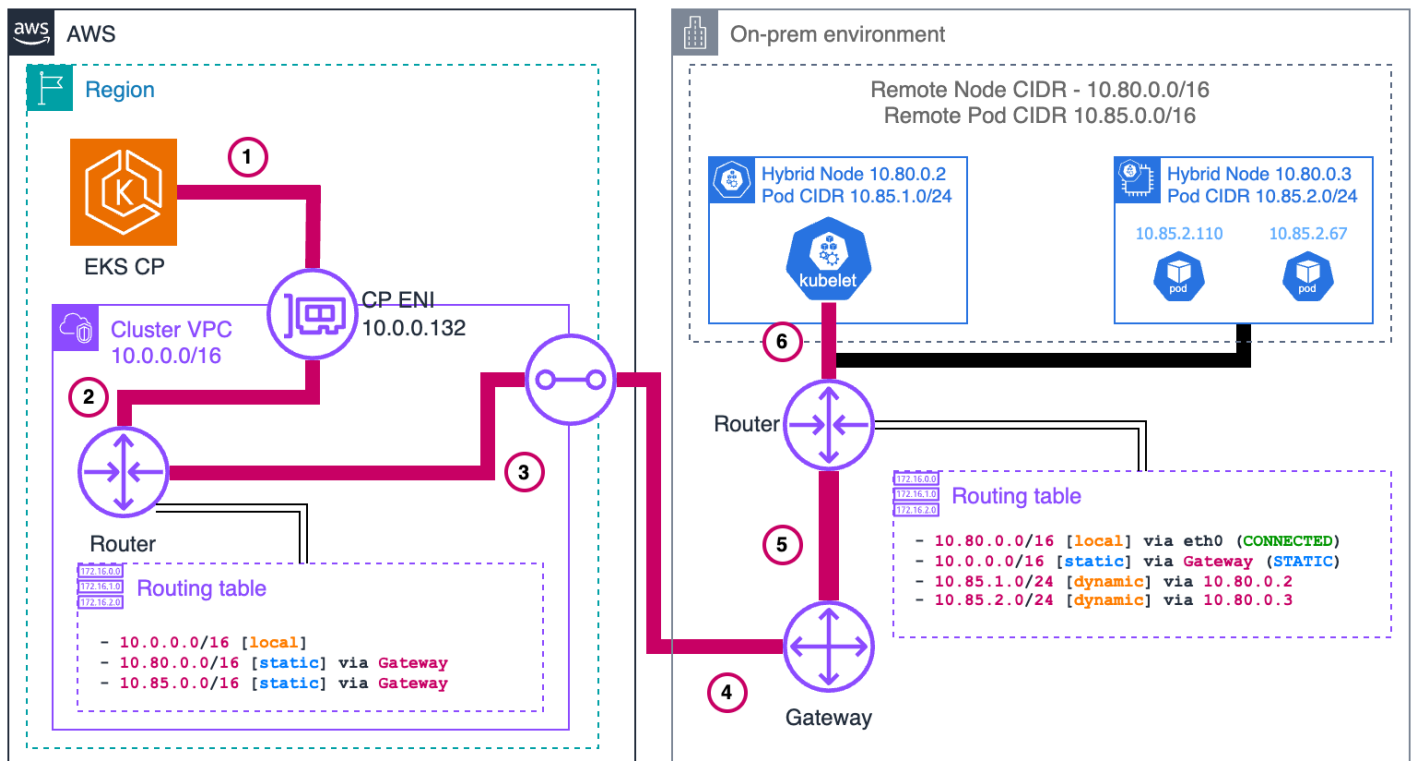
1: 로컬 전송

온프레미스 라우터는 패킷을 수신하고 대상 IP(10.80.0.2)를 로컬 네트워크에 속하는 것으로 인식합니다. 패킷이 대상 하이브리드 노드에 도달할 때까지 로컬 네트워크 인프라를 통해 패킷을 전달하고, 여기서 kubelet이 응답을 수신하고 처리합니다.

하이브리드 노드 **kube-proxy**에서 EKS 컨트롤 플레인으로

클러스터에 대해 퍼블릭 엔드포인트 액세스를 활성화하면 반환 트래픽은 퍼블릭 인터넷을 사용합니다. 이 트래픽은 하이브리드 노드의 kube-proxy에서 시작되어 EKS 컨트롤 플레인으로 이동하며 kubelet에서 EKS 컨트롤 플레인으로 이동하는 트래픽과 동일한 경로를 따릅니다.

EKS 컨트롤 플레인에서 하이브리드 노드로(**kubelet** 서버)



요청

1: EKS Kubernetes API 서버가 요청 시작

EKS Kubernetes API 서버는 노드 객체의 상태에서 노드의 IP 주소(10.80.0.2)를 검색합니다. 그런 다음 대상 IP가 구성된 원격 노드 CIDR(10.80.0.0/16)에 속하므로 VPC의 ENI를 통해 이 요청을 라우팅합니다. 초기 패킷은 다음과 같습니다.

```
+-----+-----+-----+
| IP Header | TCP Header | Payload |
| Src: 10.0.0.132 | Src: 67493 (random) | |
| Dst: 10.80.0.2 | Dst: 10250 | |
+-----+-----+-----+
```

2. VPC 네트워크 처리

패킷은 ENI를 떠나 VPC 네트워킹 계층으로 들어가고, 이후 라우팅을 위해 서브넷의 게이트웨이로 전달됩니다.

3. VPC 라우팅 테이블 조회

EKS 컨트롤 플레인 ENI가 포함된 서브넷의 VPC 라우팅 테이블에는 원격 노드 CIDR에 대한 특정 경로(다이어그램의 두 번째 경로)가 있습니다. 이 라우팅 규칙에 따라 패킷은 VPC-온프레미스 게이트웨이로 전달됩니다.

4. 교차 경계 전송

게이트웨이는 설정된 연결(예: Direct Connect 또는 VPN)을 통해 클라우드 경계를 넘어 온프레미스 네트워크로 패킷을 전송합니다.

5. 온프레미스 네트워크 수신

패킷은 하이브리드 노드가 위치한 서브넷의 트래픽을 처리하는 로컬 온프레미스 라우터에 도착합니다.

6. 최종 전송

로컬 라우터는 대상 IP(10.80.0.2) 주소가 직접 연결된 네트워크에 속하는 것을 식별하고 패킷을 대상 하이브리드 노드로 직접 전달하며, 여기서 kubelet이 요청을 수신하고 처리합니다.

응답

하이브리드 노드의 kubelet이 요청을 처리한 후, 동일한 경로를 역으로 응답을 다시 전송합니다.

```
+-----+-----+-----+
```

IP Header	TCP Header	Payload	
Src: 10.80.0.2	Src: 10250		
Dst: 10.0.0.132	Dst: 67493		
+-----+	+-----+	+-----+	+-----+

6. kubelet 응답 전송

하이브리드 노드(10.80.0.2)의 kubelet은 원본 소스 IP를 대상으로 하는 응답 패킷을 생성합니다. 대상은 로컬 네트워크에 속하지 않으므로 호스트의 기본 게이트웨이인 로컬 라우터로 전송됩니다.

5. 로컬 라우터 라우팅

라우터는 대상 IP(10.0.0.132)가 AWS로 연결되는 게이트웨이를 가리키는 경로를 가진 10.0.0.0/16에 속한다고 판단합니다.

4. 교차 경계 반환

패킷은 동일한 온프레미스를 통해 VPC 연결(예: Direct Connect 또는 VPN)로 다시 이동하며 반대 방향으로 클라우드 경계를 통과합니다.

3. VPC 라우팅

패킷이 VPC에 도착하면 라우팅 테이블은 대상 IP가 VPC CIDR에 속함을 식별합니다. 패킷은 VPC 내에서 라우팅됩니다.

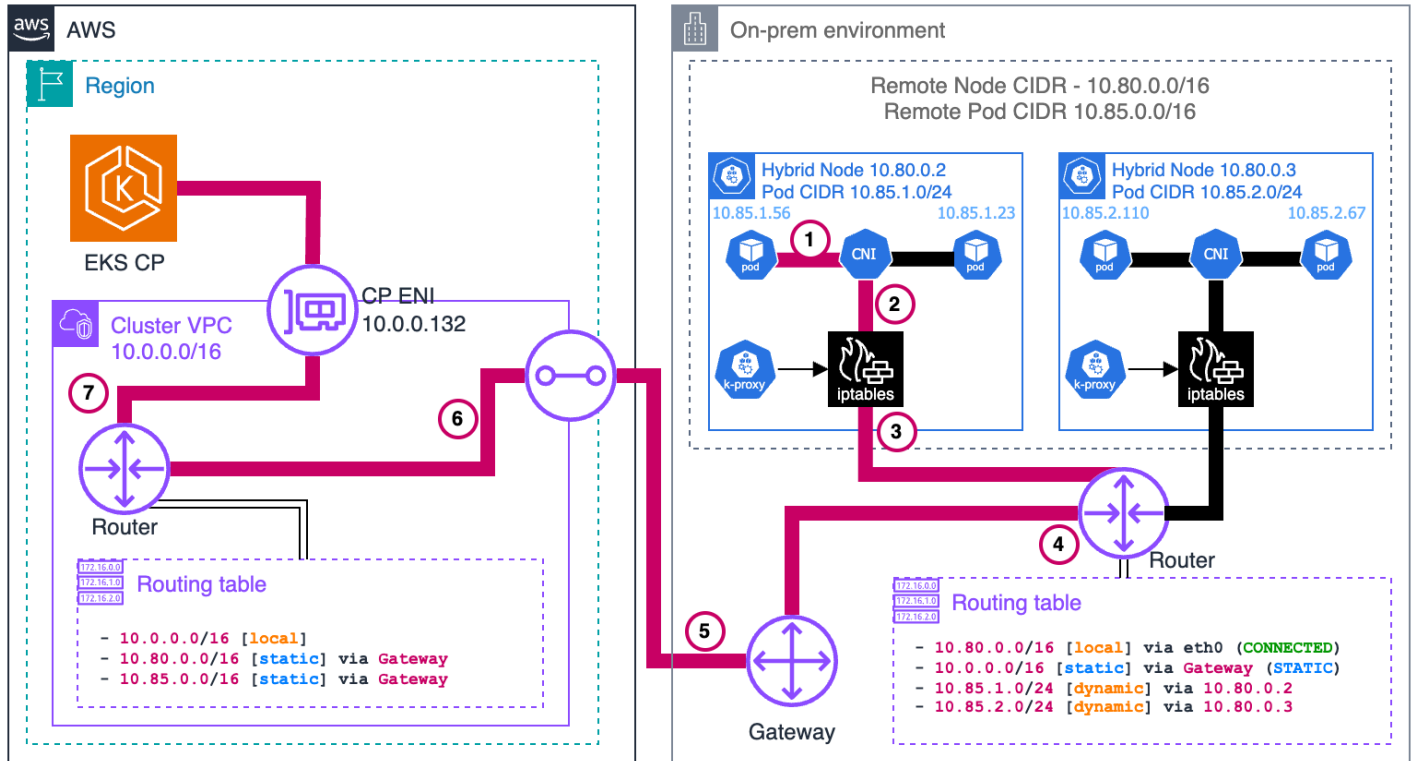
2. VPC 네트워크 전송

VPC 네트워킹 계층은 EKS 컨트롤 플레인 ENI(10.0.0.132)를 사용하여 패킷을 서브넷으로 전달합니다.

1: ENI 수신

패킷은 Kubernetes API 서버에 연결된 EKS 컨트롤 플레인 ENI에 도달하여 왕복을 완료합니다.

하이브리드 노드에서 실행되는 포드에서 EKS 컨트롤 플레인으로



CNI NAT 사용 안 함

요청

포드는 일반적으로 kubernetes 서비스를 통해 Kubernetes API 서버와 통신합니다. 서비스 IP는 클러스터 서비스 CIDR의 첫 번째 IP입니다. 이 규칙을 사용하면 CoreDNS를 사용할 수 있기 전에 실행해야 하는 포드(예: CNI)가 API 서버에 도달할 수 있습니다. 요청은 서비스 IP를 대상으로 하여 포드를 떠납니다. 예를 들어 서비스 CIDR이 172.16.0.0/16인 경우 서비스 IP는 172.16.0.1이 됩니다.

1: 포드에서 요청 시작

포드는 임의의 소스 포트에서 API 서버 포트(443)의 kubernetes 서비스 IP(172.16.0.1)로 요청을 전송합니다. 패킷은 다음과 같습니다.

```
+-----+
| IP Header | TCP Header | Payload |
| Src: 10.85.1.56 | Src: 67493 (random) |
| Dst: 172.16.0.1 | Dst: 443 |
+-----+
```


2. CNI 처리

CNI는 대상 IP가 관리하는 어떤 포드 CIDR에도 속하지 않음을 감지합니다. 발신 NAT가 비활성화되어 있으므로 CNI는 패킷을 수정하지 않고 호스트 네트워크 스택으로 전달합니다.

3. 노드 네트워크 처리

패킷은 netfilter 후크가 kube-proxy에서 설정한 iptables 규칙을 트리거하는 노드의 네트워크 스택으로 들어갑니다. 다음과 같은 순서로 여러 규칙이 적용됩니다.

1. 패킷은 먼저 KUBE-SERVICES 체인에 도달합니다. 여기에는 각 서비스의 ClusterIP 및 포트와 일치하는 규칙이 포함됩니다.
2. 일치하는 규칙은 kubernetes 서비스(172.16.0.1:443으로 전송되는 패킷)에 대한 KUBE-SVC-XXX 체인으로 이동합니다.
3. 로드 밸런싱 규칙은 컨트롤 플레인 ENI IP(10.0.0.132 또는 10.0.1.23)에 대해 KUBE-SEP-XXX 체인 중 하나를 무작위로 선택합니다.
4. 선택한 KUBE-SEP-XXX 체인에는 대상 IP를 서비스 IP에서 선택한 IP로 변경하는 실제 규칙이 있습니다. 이를 DNAT(Destination Network Address Translation)이라고 합니다.

이러한 규칙이 적용된 후 선택한 EKS 컨트롤 플레인 ENI의 IP가 10.0.0.132라고 가정하면 패킷은 다음과 같습니다.

```
+-----+-----+-----+
| IP Header | TCP Header | Payload |
| Src: 10.85.1.56 | Src: 67493 (random) | |
| Dst: 10.0.0.132 | Dst: 443 | |
+-----+-----+-----+
```

대상 IP가 로컬 네트워크에 없기 때문에 노드는 패킷을 기본 게이트웨이로 전달합니다.

4. 로컬 라우터 라우팅

로컬 라우터는 대상 IP(10.0.0.132)가 VPC CIDR(10.0.0.0/16)에 속한다고 판단하고 이를 AWS에 연결하는 게이트웨이로 전달합니다.

5. 교차 경계 전송

패킷은 설정된 연결(예: Direct Connect 또는 VPN)을 통해 클라우드 경계를 넘어 VPC로 이동합니다.

6. VPC 네트워크 전송

VPC 네트워킹 계층은 EKS 컨트롤 플레인 ENI(10.0.0.132)가 위치한 올바른 서브넷으로 패킷을 라우팅합니다.

7. ENI 수신

패킷은 Kubernetes API 서버에 연결된 EKS 컨트롤 플레인 ENI에 도달합니다.

응답

EKS 컨트롤 플레인이 요청을 처리한 후 포드에 응답을 다시 전송합니다.

7. API 서버 전송 응답

EKS Kubernetes API 서버는 원본 소스 IP를 대상으로 하는 응답 패킷을 생성합니다. 패킷은 다음과 같습니다.

```
+-----+-----+-----+
| IP Header | TCP Header | Payload |
| Src: 10.0.0.132 | Src: 443 | |
| Dst: 10.85.1.56 | Dst: 67493 | |
+-----+-----+-----+
```

대상 IP는 구성된 원격 포드 CIDR(10.85.0.0/16)에 속하므로 서브넷의 라우터를 다음 홉으로 사용하여 VPC의 ENI를 통해 전송합니다.

6. VPC 라우팅

VPC 라우팅 테이블에는 원격 포드 CIDR(10.85.0.0/16)에 대한 항목이 포함되어 이 트래픽을 VPC-온프레미스 게이트웨이로 전달합니다.

5. 교차 경계 전송

게이트웨이는 설정된 연결(예: Direct Connect 또는 VPN)을 통해 클라우드 경계를 넘어 온프레미스 네트워크로 패킷을 전송합니다.

4. 온프레미스 네트워크 수신

패킷이 로컬 온프레미스 라우터에 도착합니다.

3. 노드로 전송

라우터 테이블에는 10.85.1.0/24에 대한 항목이 있고, 다음 흐름은 10.80.0.2로 패킷을 노드에 전달합니다.

2. 노드 네트워크 처리

패킷이 노드의 네트워크 스택에서 처리되면 conntrack(netfilter의 일부)은 패킷을 포드가 처음 설정한 연결과 일치시킵니다. 원래 DNAT가 적용되었으므로 conntrack은 EKS 컨트롤 플레인 ENI의 IP에서 kubernetes 서비스 IP로 소스 IP를 다시 작성하여 DNAT를 되돌립니다.

```
+-----+-----+-----+
| IP Header | TCP Header | Payload |
| Src: 172.16.0.1 | Src: 443 | |
| Dst: 10.85.1.56 | Dst: 67493 | |
+-----+-----+-----+
```

1: CNI 처리

CNI는 대상 IP가 네트워크의 포드에 속함을 식별하고 패킷을 올바른 포드 네트워크 네임스페이스로 전송합니다.

이 흐름은 원격 포드 CIDR이 VPC에서 각 포드를 호스팅하는 특정 노드까지 제대로 라우팅되어야 하는 이유를 보여줍니다. 전체 반환 경로는 클라우드 및 온프레미스 네트워크 모두에서 포드 IP의 적절한 라우팅에 따라 달라집니다.

CNI NAT 사용

이 흐름은 CNI NAT가 없는 흐름과 매우 유사하지만 한 가지 주요 차이점이 있습니다. CNI는 패킷을 노드의 네트워크 스택으로 전송하기 전에 패킷에 소스 NAT(SNAT)를 적용한다는 점입니다. 이렇게 하면 패킷의 소스 IP가 노드의 IP로 변경되므로 추가 라우팅 구성 없이 패킷을 노드로 다시 라우팅할 수 있습니다.

요청

1: 포드에서 요청 시작

포드는 임의의 소스 포트에서 EKS Kubernetes API 서버 포트(443)의 kubernetes 서비스 IP(172.16.0.1)로 요청을 전송합니다. 패킷은 다음과 같습니다.

```
+-----+-----+-----+
| IP Header | TCP Header | Payload |
+-----+-----+-----+
```

```

| Src: 10.85.1.56 | Src: 67493 (random) |
| Dst: 172.16.0.1 | Dst: 443          |
+-----+-----+-----+

```

2. CNI 처리

CNI는 대상 IP가 관리하는 어떤 포드 CIDR에도 속하지 않음을 감지합니다. 발신 NAT가 활성화되었으므로 CNI는 패킷에 SNAT를 적용하여 소스 IP를 노드의 네트워크 스택에 전달하기 전에 노드의 IP로 변경합니다.

```

+-----+-----+-----+
| IP Header | TCP Header | Payload |
| Src: 10.80.0.2 | Src: 67493 (random) |
| Dst: 172.16.0.1 | Dst: 443   |
+-----+-----+-----+

```

참고: 예에서는 명확성을 위해 CNI와 iptables를 별도의 블록으로 표시했지만 실제로는 일부 CNI가 iptables를 사용하여 NAT를 적용할 수 있습니다.

3. 노드 네트워크 처리

여기서 kube-proxy 설정한 iptables 규칙은 이전 예제에서와 동일하게 동작하여 패킷을 EKS 컨트롤 플레인 ENI 중 하나로 로드 밸런싱합니다. 패킷은 이제 다음과 같습니다.

```

+-----+-----+-----+
| IP Header | TCP Header | Payload |
| Src: 10.80.0.2 | Src: 67493 (random) |
| Dst: 10.0.0.132 | Dst: 443   |
+-----+-----+-----+

```

대상 IP가 로컬 네트워크에 없기 때문에 노드는 패킷을 기본 게이트웨이로 전달합니다.

4. 로컬 라우터 라우팅

로컬 라우터는 대상 IP(10.0.0.132)가 VPC CIDR(10.0.0.0/16)에 속한다고 판단하고 이를 AWS에 연결하는 게이트웨이로 전달합니다.

5. 교차 경계 전송

패킷은 설정된 연결(예: Direct Connect 또는 VPN)을 통해 클라우드 경계를 넘어 VPC로 이동합니다.

6. VPC 네트워크 전송

VPC 네트워킹 계층은 EKS 컨트롤 플레인 ENI(10.0.0.132)가 위치한 올바른 서브넷으로 패킷을 라우팅합니다.

7. ENI 수신

패킷은 Kubernetes API 서버에 연결된 EKS 컨트롤 플레인 ENI에 도달합니다.

응답

EKS 컨트롤 플레인이 요청을 처리한 후 포트에 응답을 다시 전송합니다.

7. API 서버 전송 응답

EKS Kubernetes API 서버는 원본 소스 IP를 대상으로 하는 응답 패킷을 생성합니다. 패킷은 다음과 같습니다.

IP Header	TCP Header	Payload
Src: 10.0.0.132	Src: 443	
Dst: 10.80.0.2	Dst: 67493	

대상 IP는 구성된 원격 노드 CIDR(10.80.0.0/16)에 속하므로 서브넷의 라우터를 다음 홉으로 사용하여 VPC의 ENI를 통해 전송합니다.

6. VPC 라우팅

VPC 라우팅 테이블에는 원격 노드 CIDR(10.80.0.0/16)에 대한 항목이 포함되어 이 트래픽을 VPC-온프레미스 게이트웨이로 전달합니다.

5. 교차 경계 전송

게이트웨이는 설정된 연결(예: Direct Connect 또는 VPN)을 통해 클라우드 경계를 넘어 온프레미스 네트워크로 패킷을 전송합니다.

4. 온프레미스 네트워크 수신

패킷이 로컬 온프레미스 라우터에 도착합니다.

3. 노드로 전송

로컬 라우터는 대상 IP(10.80.0.2) 주소가 직접 연결된 네트워크에 속하는 것을 식별하고 패킷을 대상 하이브리드 노드로 직접 전달합니다.

2. 노드 네트워크 처리

패킷이 노드의 네트워크 스택에서 처리될 때 conntrack(netfilter의 일부)은 패킷을 포드가 처음에 설정한 연결과 일치시키고 DNAT가 원래 적용되었기 때문에 EKS 컨트롤 플레인 ENI의 IP에서 kubernetes 서비스 IP로 소스 IP를 다시 작성하여 이를 반대로 수행합니다.

```
+-----+-----+-----+
| IP Header | TCP Header | Payload |
| Src: 172.16.0.1 | Src: 443 | |
| Dst: 10.80.0.2 | Dst: 67493 | |
+-----+-----+-----+
```

1: CNI 처리

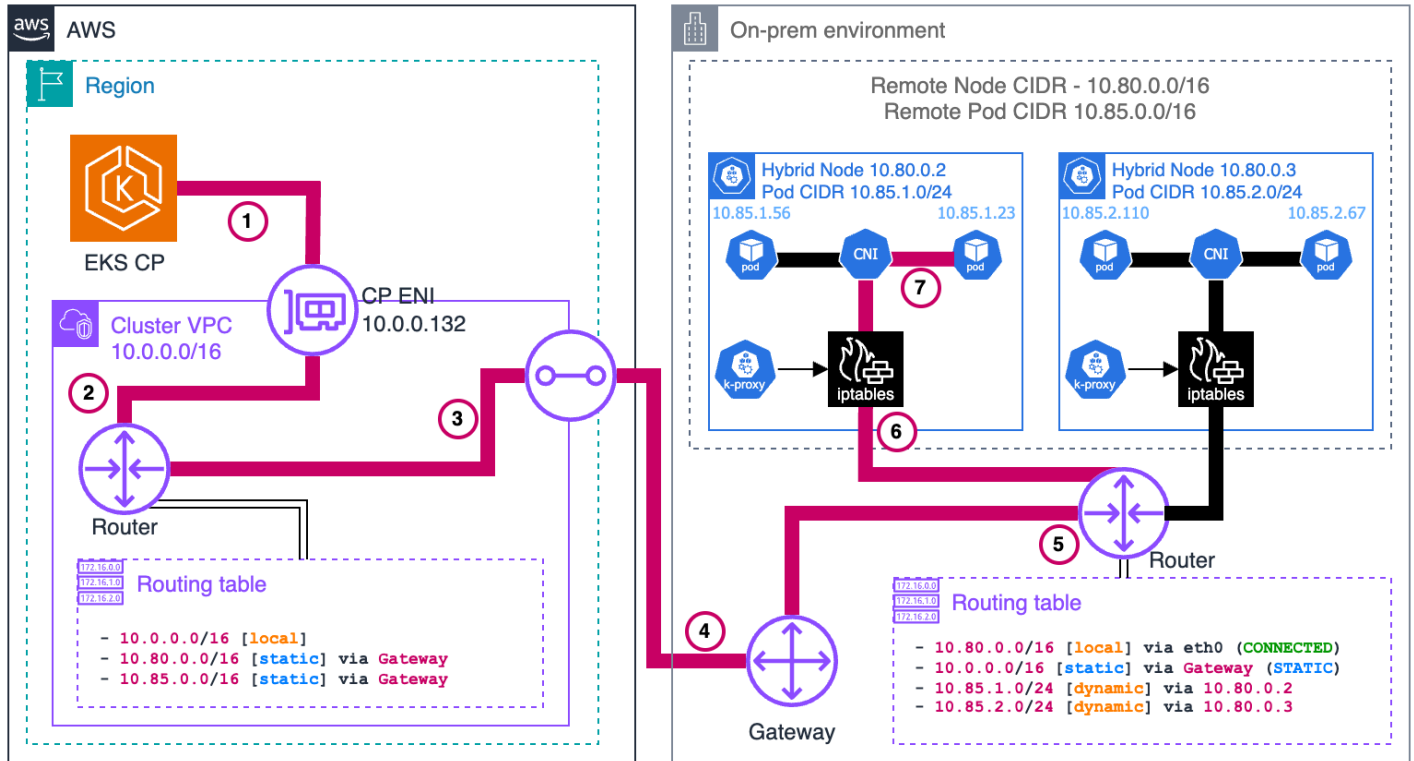
CNI는 이 패킷이 이전에 SNAT를 적용한 연결에 속함을 식별합니다. SNAT를 반대로 실행하여 대상 IP를 포드의 IP로 다시 변경합니다.

```
+-----+-----+-----+
| IP Header | TCP Header | Payload |
| Src: 172.16.0.1 | Src: 443 | |
| Dst: 10.85.1.56 | Dst: 67493 | |
+-----+-----+-----+
```

CNI는 대상 IP가 네트워크의 포드에 속함을 감지하고 패킷을 올바른 포드 네트워크 네임스페이스로 전송합니다.

이 흐름은 CNI NAT-ing이 Pod CIDR에 대한 추가 라우팅을 요구하지 않고도 패킷을 노드로 다시 라우팅할 수 있도록 하여 구성을 간소화하는 방법을 보여줍니다.

하이브리드 노드에서 실행되는 포드에 대한 EKS 컨트롤 플레인(웹훅)



이 트래픽 패턴은 EKS 컨트롤 플레인이 하이브리드 노드의 포드에서 실행되는 웹훅 서버에 대한 연결을 직접 시작해야 하는 웹훅에서 가장 일반적으로 볼 수 있습니다. 리소스 검증 또는 변형 프로세스 중 API 서버에서 직접적으로 호출하는 승인 웹훅의 검증 및 변형을 예로 들 수 있습니다.

요청

1: EKS Kubernetes API 서버가 요청 시작

클러스터에 웹훅이 구성되고 관련 API 작업이 이를 트리거하면 EKS Kubernetes API 서버는 웹훅 서버 포드에 직접 연결해야 합니다. API 서버는 먼저 웹훅과 연결된 서비스 또는 엔드포인트 리소스에서 포드의 IP 주소를 조회합니다.

IP가 10.85.1.23인 하이브리드 노드에서 웹훅 포드가 실행 중이라는 가정하에 EKS Kubernetes API 서버는 웹훅 엔드포인트에 HTTPS 요청을 생성합니다. 대상 IP 10.85.1.23이 구성된 원격 포드 CIDR(10.85.0.0/16)에 속하기 때문에 초기 패킷은 VPC의 EKS 컨트롤 플레인 ENI를 통해 전송됩니다. 패킷은 다음과 같습니다.

```

+-----+-----+-----+
| IP Header | TCP Header | Payload |
| Src: 10.0.0.132 | Src: 41892 (random) | |

```

```
| Dst: 10.85.1.23 | Dst: 8443 |
+-----+-----+-----+
```

2. VPC 네트워크 처리

패킷은 EKS 컨트롤 플레인 ENI를 떠나 서브넷의 라우터를 다음 홉으로 하여 VPC 네트워킹 계층으로 들어갑니다.

3. VPC 라우팅 테이블 조회

EKS 컨트롤 플레인 ENI가 포함된 서브넷의 VPC 라우팅 테이블에는 원격 포드 CIDR에 대한 특정 경로(10.85.0.0/16)가 포함되어 있습니다. 이 라우팅 규칙은 패킷을 VPC-온프레미스 게이트웨이(예: Direct Connect 또는 VPN 연결을 위한 가상 프라이빗 게이트웨이)로 전달합니다.

Destination	Target
10.0.0.0/16	local
10.85.0.0/16	vgw-id (VPC-to-onprem gateway)

4. 교차 경계 전송

게이트웨이는 설정된 연결(예: Direct Connect 또는 VPN)을 통해 클라우드 경계를 넘어 온프레미스 네트워크로 패킷을 전송합니다. 패킷은 이 연결을 통과할 때 원본 소스 및 대상 IP 주소를 유지합니다.

5. 온프레미스 네트워크 수신

패킷이 로컬 온프레미스 라우터에 도착합니다. 라우터는 라우팅 테이블을 참조하여 10.85.1.23 주소에 도달하는 방법을 결정합니다. 이를 위해서는 온프레미스 네트워크에 트래픽을 적절한 하이브리드 노드로 전달하는 포드 CIDR에 대한 경로가 있어야 합니다.

이 경우 라우터의 라우팅 테이블에는 IP가 10.80.0.2인 하이브리드 노드를 통해 10.85.1.0/24 서브넷에 도달할 수 있음을 나타내는 항목이 포함되어 있습니다.

Destination	Next Hop
10.85.1.0/24	10.80.0.2

6. 노드로 전송

라우팅 테이블 항목을 기반으로 라우터는 패킷을 하이브리드 노드(10.80.0.2)로 전달합니다. 패킷이 노드에 도착하면 대상 IP가 여전히 포드의 IP인 것을 제외하면 EKS Kubernetes API 서버가 패킷을 전송했을 때와 동일하게 보입니다.

7. CNI 처리

노드의 네트워크 스택은 패킷을 수신하고 대상 IP가 노드의 자체 IP가 아님을 확인한 후 처리를 위해 CNI에 전달합니다. CNI는 대상 IP가 이 노드에서 로컬로 실행되는 포드에 속함을 식별하고 적절한 가상 인터페이스를 통해 패킷을 올바른 포드로 전달합니다.

```
Original packet -> node routing -> CNI -> Pod's network namespace
```

포드의 웹훅 서버는 요청을 수신하고 처리합니다.

응답

웹훅 포드가 이 요청을 처리한 후, 동일한 경로를 역으로 응답을 다시 전송합니다.

7. 포드에서 응답 전송

웹훅 포드는 자체 IP를 소스로, 원래 요청자(EKS 컨트롤 플레인 ENI)를 대상으로 응답 패킷을 생성합니다.

```
+-----+-----+-----+
| IP Header | TCP Header | Payload |
| Src: 10.85.1.23 | Src: 8443 |      |
| Dst: 10.0.0.132 | Dst: 41892 |      |
+-----+-----+-----+
```

CNI는 이 패킷이 로컬 포드가 아닌 외부 네트워크로 이동함을 식별하고 원본 소스 IP가 보존된 노드의 네트워크 스택으로 패킷을 전달합니다.

6. 노드 네트워크 처리

노드는 대상 IP(10.0.0.132)가 로컬 네트워크에 없다고 판단하고 패킷을 기본 게이트웨이(로컬 라우터)로 전달합니다.

5. 로컬 라우터 라우팅

로컬 라우터는 라우팅 테이블을 참조하여 대상 IP(10.0.0.132)가 VPC CIDR(10.0.0.0/16)에 속한다고 판단하고 패킷을 AWS에 연결하는 게이트웨이로 전달합니다.

4. 교차 경계 전송

패킷은 동일한 온프레미스를 통해 VPC 연결로 다시 이동하며 반대 방향으로 클라우드 경계를 통과합니다.

3. VPC 라우팅

패킷이 VPC에 도착하면 라우팅 테이블은 대상 IP가 VPC 내의 서브넷에 속함을 식별합니다. 패킷은 VPC 내에서 그에 따라 라우팅됩니다.

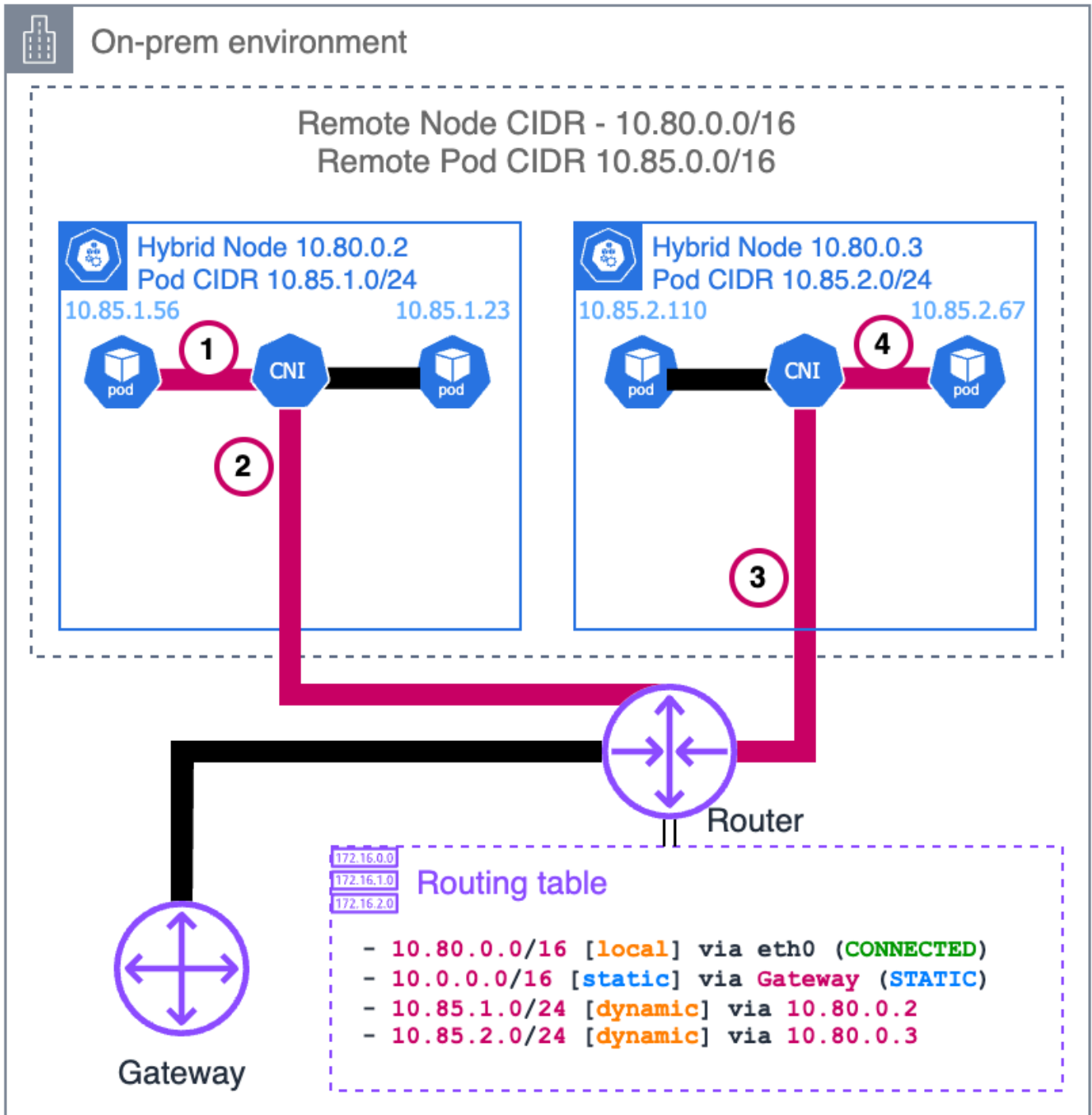
2. 및 1. EKS 컨트롤 플레인 ENI 수신

패킷이 EKS Kubernetes API 서버에 연결된 ENI에 도달하여 왕복을 완료합니다. API 서버는 웹훅 응답을 수신하고 이 응답을 기반으로 원래 API 요청을 계속 처리합니다.

이 트래픽 흐름은 원격 포드 CIDR을 올바르게 구성하고 라우팅해야 하는 이유를 보여줍니다.

- VPC에는 온프레미스 게이트웨이를 가리키는 원격 포드 CIDR에 대한 경로가 있어야 합니다.
- 온프레미스 네트워크에는 해당 포드를 호스팅하는 특정 노드로 트래픽을 전달하는 포드 CIDR에 대한 경로가 있어야 합니다.
- 이 라우팅 구성이 없으면 EKS 컨트롤 플레인이 하이브리드 노드의 포드에서 실행되는 웹훅과 기타 유사 서비스에 연결할 수 없습니다.

하이브리드 노드에서 실행되는 포드 간



이 섹션에서는 서로 다른 하이브리드 노드에서 실행되는 포드가 서로 통신하는 방법을 설명합니다. 이 예에서는 CNI가 캡슐화를 위해 VXLAN을 사용한다고 가정하며, 이는 Cilium 또는 Calico와 같은 CNI에 일반적입니다. 전체 프로세스는 Geneve 또는 IP-in-IP 등의 다른 캡슐화 프로토콜과 유사합니다.

요청

1: 포트 A에서 통신 시작

노드 1의 포트 A(10.85.1.56)가 노드 2의 포트 B(10.85.2.67)로 트래픽을 전송하려고 합니다. 초기 패킷은 다음과 같습니다.

```
+-----+-----+-----+-----+
| Ethernet Header | IP Header      | TCP/UDP      | Payload      |
| Src: Pod A MAC  | Src: 10.85.1.56 | Src: 43721   |              |
| Dst: Gateway MAC | Dst: 10.85.2.67 | Dst: 8080    |              |
+-----+-----+-----+-----+
```

2. CNI에서 패킷을 가로채고 처리

포트 A의 패킷이 네트워크 네임스페이스를 벗어나면 CNI가 이를 가로칩니다. CNI는 라우팅 테이블을 참조하여 다음을 확인합니다. - 대상 IP(10.85.2.67)는 포트 CIDR에 속합니다. - 이 IP는 로컬 노드에 없지만 노드 2(10.80.0.3)에 속합니다. - 패킷은 VXLAN으로 캡슐화되어야 합니다.

캡슐화 결정은 기본 물리적 네트워크가 포트 CIDR을 직접 라우팅하는 방법을 모르고 노드 IP 간의 트래픽을 라우팅하는 방법만 알기 때문에 중요합니다.

CNI는 VXLAN 프레임 내에서 전체 원본 패킷을 캡슐화합니다. 이렇게 하면 새로운 헤더가 포함된 '패킷 내의 패킷'이 효과적으로 생성됩니다.

```
+-----+-----+-----+-----+
+-----+
| Outer Ethernet | Outer IP      | Outer UDP    | VXLAN        | Original Pod-to-Pod
|               |               |               |              |
| Src: Node1 MAC | Src: 10.80.0.2 | Src: Random  | VNI: 42      | Packet (unchanged
|               | Dst: 10.80.0.3 | Dst: 8472    |              | from above)
|               |
+-----+-----+-----+-----+
+-----+
```

이 캡슐화의 핵심 사항: - 외부 패킷은 노드 1(10.80.0.2)에서 노드 2(10.80.0.3)로 주소가 지정됩니다. - UDP 포트 8472는 Cilium이 기본적으로 사용하는 VXLAN 포트입니다. - VXLAN 네트워크 식별자(VNI)는 이 패킷이 속하는 오버레이 네트워크를 식별합니다. - 전체 원본 패킷(포트 A의 IP를 소스로, 포트 B의 IP를 대상으로 함)은 내부에 그대로 보존됩니다.

캡슐화된 패킷은 이제 노드 1의 일반 네트워킹 스택에 들어가서 다른 패킷과 동일한 방식으로 처리됩니다.

1. 노드 네트워크 처리: 노드 1의 네트워크 스택은 대상(10.80.0.3)을 기반으로 패킷을 라우팅합니다.
2. 로컬 네트워크 전송:
 - 두 노드가 동일한 계층 2 네트워크에 있는 경우 패킷이 노드 2로 직접 전송됩니다.
 - 서로 다른 서브넷에 있는 경우 패킷이 먼저 로컬 라우터로 전달됩니다.
3. 라우터 처리: 라우터가 라우팅 테이블을 기반으로 패킷을 전달하여 노드 2로 전송합니다.

3. 수신 노드 처리

캡슐화된 패킷이 노드 2(10.80.0.3)에 도착하면

1. 노드의 네트워크 스택이 이를 수신하고 VXLAN 패킷(UDP 포트 4789)으로 식별합니다.
2. 패킷이 처리를 위해 CNI의 VXLAN 인터페이스로 전달됩니다.

4. VXLAN 캡슐화 해제

노드 2의 CNI가 VXLAN 패킷을 처리합니다.

1. 외부 헤더(이더넷, IP, UDP 및 VXLAN)를 제거합니다.
2. 원래 내부 패킷을 추출합니다.
3. 이제 패킷이 원래 형식으로 돌아갑니다.

```
+-----+-----+-----+-----+
| Ethernet Header | IP Header   | TCP/UDP    | Payload    |
| Src: Pod A MAC  | Src: 10.85.1.56 | Src: 43721  |            |
| Dst: Gateway MAC | Dst: 10.85.2.67 | Dst: 8080   |            |
+-----+-----+-----+-----+
```

노드 2의 CNI는 대상 IP(10.85.2.67)를 검사하고 다음을 수행합니다.

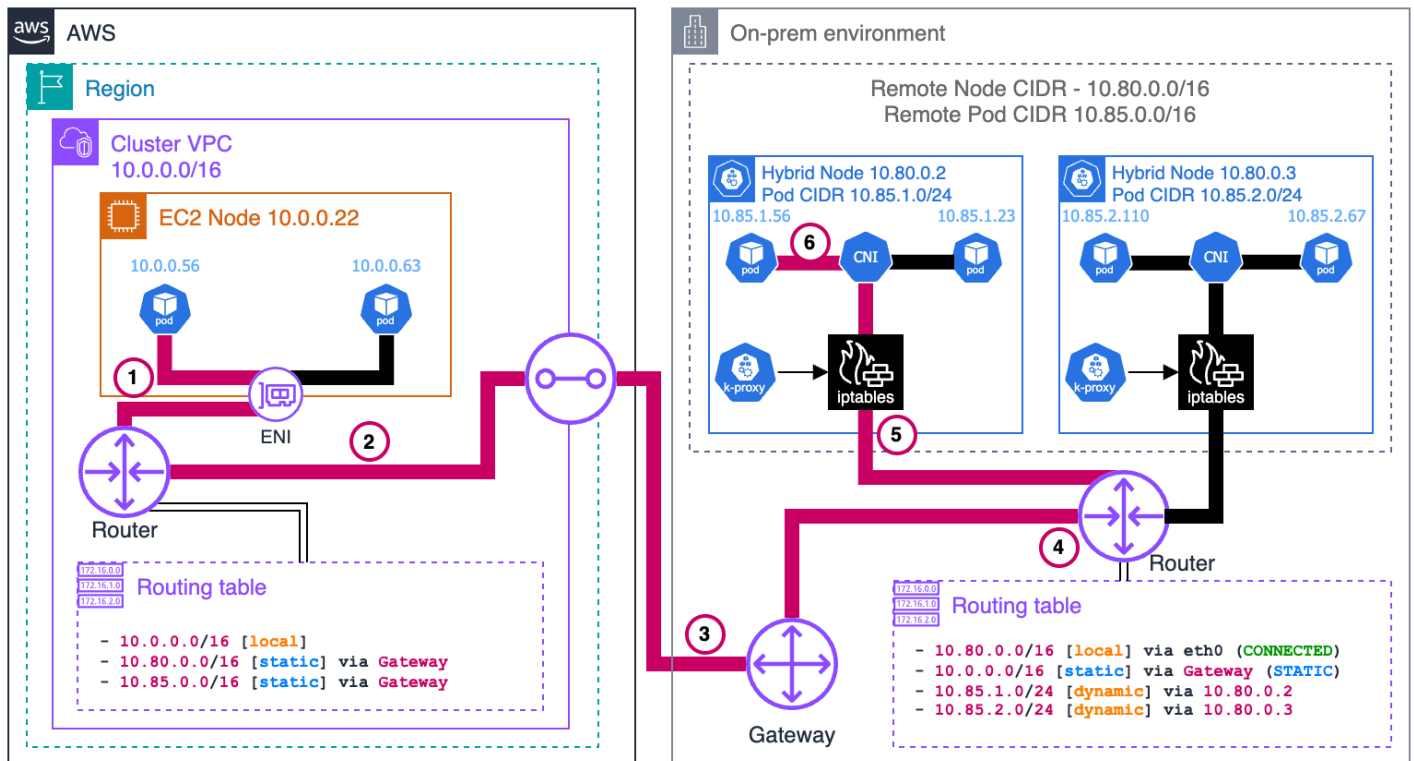
1. 이 IP가 로컬 포드에 속함을 식별합니다.
2. 적절한 가상 인터페이스를 통해 패킷을 라우팅합니다.
3. 포트 B의 네트워크 네임스페이스로 패킷을 전송합니다.

응답

포트 B가 포트 A에 응답하면 전체 프로세스가 반대로 수행됩니다.

4. 포트 B가 포트 A(10.85.1.56)로 패킷을 전송합니다.
5. 노드 2의 CNI는 노드 1(10.80.0.2)로 대상을 설정하여 VXLAN으로 캡슐화합니다.
6. 캡슐화된 패킷이 노드 1로 전달됩니다.
7. 노드 1의 CNI는 이를 캡슐화 해제하고 포트 A에 원래 응답을 전달합니다.

클라우드 노드의 포트에서 하이브리드 노드의 포트(동서 트래픽)



요청

1: 포트 A에서 통신 시작

EC2 노드의 포트 A(10.0.0.56)가 하이브리드 노드의 포트 B(10.85.1.56)로 트래픽을 전송하려고 합니다. 초기 패킷은 다음과 같습니다.

```
+-----+-----+-----+
| IP Header | TCP Header | Payload |
| Src: 10.0.0.56 | Src: 52390 (random) | |
```

```
| Dst: 10.85.1.56 | Dst: 8080 |
+-----+-----+-----+
```

VPC CNI를 사용하면 포트 A는 VPC CIDR의 IP를 가지며 EC2 인스턴스의 ENI에 직접 연결됩니다. 포트의 네트워크 네임스페이스는 VPC 네트워크에 연결되므로 패킷은 VPC 라우팅 인프라에 직접 들어갑니다.

2. VPC 라우팅

VPC 라우팅 테이블에는 원격 포트 CIDR(10.85.0.0/16)에 대한 특정 경로가 포함되어 있으며, 이 트래픽을 VPC-온프레미스 게이트웨이로 전달합니다.

Destination	Target
10.0.0.0/16	local
10.85.0.0/16	vgw-id (VPC-to-onprem gateway)

이 라우팅 규칙에 따라 패킷은 온프레미스 네트워크에 연결하는 게이트웨이로 전달됩니다.

3. 교차 경계 전송

게이트웨이는 설정된 연결(예: Direct Connect 또는 VPN)을 통해 클라우드 경계를 넘어 온프레미스 네트워크로 패킷을 전송합니다. 패킷은 이 전송 과정에서 원본 소스 및 대상 IP 주소를 유지합니다.

4. 온프레미스 네트워크 수신

패킷이 로컬 온프레미스 라우터에 도착합니다. 라우터는 라우팅 테이블을 참조하여 10.85.1.56 주소에 도달하기 위한 다음 홉을 결정합니다. 온프레미스 라우터에 트래픽을 적절한 하이브리드 노드로 전달하는 포트 CIDR에 대한 경로가 있어야 합니다.

라우터의 테이블에는 IP가 10.80.0.2인 하이브리드 노드를 통해 10.85.1.0/24 서브넷에 연결할 수 있음을 나타내는 항목이 있습니다.

Destination	Next Hop
10.85.1.0/24	10.80.0.2

5. 노드 네트워크 처리

라우터는 패킷을 하이브리드 노드(10.80.0.2)로 전달합니다. 패킷이 노드에 도착해도 여전히 포트 A의 IP가 소스이고 포트 B의 IP가 대상입니다.

6. CNI 처리

노드의 네트워크 스택은 패킷을 수신하고 대상 IP가 자체가 아님을 확인한 후 처리를 위해 CNI에 전달합니다. CNI는 대상 IP가 이 노드에서 로컬로 실행되는 포드에 속함을 식별하고 적절한 가상 인터페이스를 통해 패킷을 올바른 포드로 전달합니다.

```
Original packet -> node routing -> CNI -> Pod B's network namespace
```

포드 B는 패킷을 수신하고 필요에 따라 처리합니다.

응답

6. 포드 B에서 응답 전송

포드 B는 자체 IP를 소스로, 포드 A의 IP를 대상으로 하는 응답 패킷을 생성합니다.

```
+-----+-----+-----+
| IP Header | TCP Header | Payload |
| Src: 10.85.1.56 | Src: 8080 | |
| Dst: 10.0.0.56 | Dst: 52390 | |
+-----+-----+-----+
```

CNI는 이 패킷이 외부 네트워크로 향하는 것을 식별하여 노드의 네트워크 스택으로 전달합니다.

5. 노드 네트워크 처리

노드는 대상 IP(10.0.0.56)가 로컬 네트워크에 속하지 않는다고 판단하고 패킷을 기본 게이트웨이(로컬 라우터)로 전달합니다.

4. 로컬 라우터 라우팅

로컬 라우터는 라우팅 테이블을 참조하여 대상 IP(10.0.0.56)가 VPC CIDR(10.0.0.0/16)에 속한다고 판단하고 패킷을 AWS에 연결하는 게이트웨이로 전달합니다.

3. 교차 경계 전송

패킷은 동일한 온프레미스를 통해 VPC 연결로 다시 이동하며 반대 방향으로 클라우드 경계를 통과합니다.

2. VPC 라우팅

패킷이 VPC에 도착하면 라우팅 시스템은 대상 IP가 VPC 내의 서브넷에 속함을 식별합니다. 패킷은 VPC 네트워크를 통해 포드 A를 호스팅하는 EC2 인스턴스로 라우팅됩니다.

1: 포드 A에서 응답 수신

패킷은 EC2 인스턴스에 도착하고 연결된 ENI를 통해 포트 A로 직접 전달됩니다. VPC CNI는 VPC의 포트에 오버레이 네트워킹을 사용하지 않으므로 추가 캡슐화 해제가 필요하지 않습니다. 패킷은 원래 헤더가 그대로 유지된 상태로 도착합니다.

이 동서 트래픽 흐름은 원격 포트 CIDR을 올바르게 구성하고 양방향으로 라우팅해야 하는 이유를 보여줍니다.

- VPC에는 온프레미스 게이트웨이를 가리키는 원격 포트 CIDR에 대한 경로가 있어야 합니다.
- 온프레미스 네트워크에는 해당 포트를 호스팅하는 특정 노드로 트래픽을 전달하는 포트 CIDR에 대한 경로가 있어야 합니다.

하이브리드 노드 **nodeadm** 참조

Amazon EKS Hybrid Nodes CLI(nodeadm)는 하이브리드 노드 구성 요소의 설치, 구성, 등록 및 제거를 간소화합니다. 운영 체제 이미지에 nodeadm을 포함하여 하이브리드 노드 부트스트랩을 자동화할 수 있습니다. 자세한 내용은 [the section called “운영 체제 준비”](#) 섹션을 참조하세요.

하이브리드 노드용 nodeadm 버전은 Amazon EC2 인스턴스를 Amazon EKS 클러스터의 노드로 부트스트랩하는 데 사용되는 nodeadm 버전과 다릅니다. 적절한 nodeadm 버전의 설명서 및 참조를 따릅니다. 이 설명서 페이지는 하이브리드 노드 nodeadm 버전용입니다.

하이브리드 노드 nodeadm의 소스 코드는 <https://github.com/aws/eks-hybrid> GitHub 리포지토리에 게시됩니다.

Important

root/sudo 권한이 있는 사용자와 함께 nodeadm을 실행해야 합니다.

nodeadm 다운로드

하이브리드 노드 버전 nodeadm은 Amazon CloudFront가 제공하는 Amazon S3에서 호스팅됩니다. 각 온프레미스 호스트에 nodeadm을 설치하려면 온프레미스 호스트에서 다음 명령을 실행할 수 있습니다.

x86_64 호스트의 경우

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/amd64/nodeadm'
```

ARM 호스트의 경우

```
curl -OL 'https://hybrid-assets.eks.amazonaws.com/releases/latest/bin/linux/arm64/nodeadm'
```

각 호스트에서 다운로드한 바이너리에 실행 파일 권한을 추가합니다.

```
chmod +x nodeadm
```

nodeadm install

`nodeadm install` 명령은 하이브리드 노드를 실행하고 Amazon EKS 클러스터에 조인하는 데 필요한 아티팩트 및 종속성을 설치하는 데 사용됩니다. `nodeadm install` 명령을 각 하이브리드 노드에서 개별적으로 실행하거나 이미지 빌드 파이프라인 중에 실행하여 운영 체제 이미지에 하이브리드 노드 종속성을 사전 설치할 수 있습니다.

사용량

```
nodeadm install [KUBERNETES_VERSION] [flags]
```

위치 인수

(필수) KUBERNETES_VERSION 설치할 EKS Kubernetes의 major.minor 버전(예: 1.32)

Flags

명칭	필수	설명
-p, --credential-provider	TRUE	설치할 자격 증명 공급자입니다. 지원되는 값은 iam-ra 및 ssm입니다. 자세한 정보는 the section called “자격 증명 준비” 을 참조하세요.
-s, --containerd-source	FALSE	containerd 소스입니다. nodeadm은 OS distro, Docker 패키지에서 containerd 설치 또는 containerd 설치 건너뛰기를 지원합니다.

명칭	필수	설명
		값 distro-기본값입니다. nodeadm은 노드 OS에서 배포한 containerd 패키지를 설치합니다. distro는 Red Hat Enterprise Linux(RHEL) 운영 체제에서 지원되지 않는 값입니다. docker-nodeadm은 Docker에서 빌드 및 배포한 containerd 패키지를 설치합니다. docker는 Amazon Linux 2023에서 지원되지 않는 값입니다. none-nodeadm은 containerd 패키지를 설치하지 않습니다. nodeadm init를 실행하기 전에 containerd를 수동으로 설치해야 합니다.
-r, --region	FALSE	SSM 에이전트와 같은 아티팩트를 다운로드할 AWS 리전을 지정합니다. 기본값은 us-west-2 입니다.
-t, --timeout	FALSE	최대 설치 명령 기간입니다. 입력은 기간 형식을 따릅니다. 예: 1h23m. 설치 명령의 기본 다운로드 제한 시간은 20분으로 설정됩니다.

명칭	필수	설명
-h, --help	FALSE	사용 가능한 플래그, 하위 명령, 위치 값 파라미터가 포함된 도움말 메시지가 표시됩니다.

예시

AWS Systems Manager(SSM)를 자격 증명 공급자로 사용하여 Kubernetes 버전 1.32 설치

```
nodeadm install 1.32 --credential-provider ssm
```

AWS Systems Manager(SSM)를 자격 증명 공급자로, Docker를 Containerd 소스로 사용하여 Kubernetes 버전 1.32을 설치하고 다운로드 제한 시간은 20분입니다.

```
nodeadm install 1.32 --credential-provider ssm --containerd-source docker --timeout 20m
```

AWS IAM Roles Anywhere를 자격 증명 공급자로 사용하여 Kubernetes 버전 1.32 설치

```
nodeadm install 1.32 --credential-provider iam-ra
```

nodeadm config check

nodeadm config check 명령은 제공된 노드 구성에 오류가 있는지 확인합니다. 이 명령을 사용하여 하이브리드 노드 구성 파일의 정확성을 확인하고 검증할 수 있습니다.

사용량

```
nodeadm config check [flags]
```

Flags

명칭	필수	설명
-c, --config-source	TRUE	nodeadm 구성 소스입니다. 하이브리드 노드의 경우 입력은

명칭	필수	설명
		파일 체계의 URI를 따라야 합니다.
-h, --help	FALSE	사용 가능한 플래그, 하위 명령, 위치 값 파라미터가 포함된 도움말 메시지가 표시됩니다.

예시

```
nodeadm config check -c file:///nodeConfig.yaml
```

nodeadm init

nodeadm init 명령은 하이브리드 노드를 시작하고 구성된 Amazon EKS 클러스터와 연결합니다. nodeConfig.yaml 파일을 구성하는 방법에 대한 자세한 내용은 [the section called “SSM 하이브리드 활성화를 위한 노드 구성”](#) 또는 [the section called “IAM Roles Anywhere를 위한 노드 구성”](#) 섹션을 참조하세요.

사용량

```
nodeadm init [flags]
```

Flags

명칭	필수	설명
-c, --config-source	TRUE	nodeadm 구성 소스입니다. 하이브리드 노드의 경우 입력은 파일 체계의 URI를 따라야 합니다.
-s, --skip	FALSE	건너뛴 init의 단계입니다. 문제 해결에 도움이 되지 않는 한 단계를 건너뛰는 것은 권장되지 않습니다.

명칭	필수	설명
		값 <code>install-validation</code> 은 진행 중인 설치 명령이 성공적으로 실행되었는지 확인하는 단계를 건너뛵니다. <code>cni-validation</code> 은 노드에서 방화벽이 활성화된 경우 Cilium 또는 Calico CNI의 VXLAN 포트가 열려 있는지 확인하는 단계를 건너뛵니다. <code>node-ip-validation</code> 은 노드 IP가 원격 노드 네트워크의 CIDR에 속하는지 확인하는 단계를 건너뛵니다.
<code>-h, --help</code>	FALSE	사용 가능한 플래그, 하위 명령, 위치 값 파라미터가 포함된 도움말 메시지가 표시됩니다.

예시

```
nodeadm init -c file://nodeConfig.yaml
```

nodeadm upgrade

`nodeadm upgrade` 명령은 설치된 모든 아티팩트를 최신 버전으로 업그레이드하고, 노드를 부트스트랩하여 업그레이드된 아티팩트를 구성하고 AWS에서 EKS 클러스터에 조인합니다. 업그레이드는 노드에서 실행되는 워크로드에 대한 중단 명령입니다. 업그레이드를 실행하기 전에 워크로드를 다른 노드로 이동하세요.

사용량

```
nodeadm upgrade [KUBERNETES_VERSION] [flags]
```

위치 인수

(필수) KUBERNETES_VERSION 설치할 EKS Kubernetes의 major.minor 버전(예: 1.32)

Flags

명칭	필수	설명
-c, --config-source	TRUE	nodeadm 구성 소스입니다. 하이브리드 노드의 경우 입력은 파일 체계의 URI를 따라야 합니다.
-t, --timeout	FALSE	아티팩트 다운로드 제한 시간입니다. 입력은 기간 형식을 따릅니다. 예를 들면 1h23m입니다. 업그레이드 명령의 기본 다운로드 제한 시간은 10분으로 설정됩니다.
-s, --skip	FALSE	건너뛸 업그레이드 단계입니다. 문제 해결에 도움이 되지 않는 한 단계를 건너뛰는 것은 권장되지 않습니다. 값 pod-validation 은 대몬 세트 및 정적 포드를 제외하고 노드에서 모든 포드가 실행 중인 지 확인하는 단계를 건너뛵니다. node-validation 은 노드가 차단되었는지 확인하는 단계를 건너뛵니다. init-validation 은 업그레이드를 실행하기 전에 노드

명칭	필수	설명
		가 성공적으로 초기화되었는지 확인하는 단계를 건너뛸니다.
-h, --help	FALSE	사용 가능한 플래그, 하위 명령, 위치 값 파라미터가 포함된 도움말 메시지가 표시됩니다.

예시

```
nodeadm upgrade 1.32 -c file:///nodeConfig.yaml
```

```
nodeadm upgrade 1.32 -c file:///nodeConfig.yaml --timeout 20m
```

nodeadm uninstall

nodeadm uninstall 명령은 kubelet 및 Containerd를 포함하여 nodeadm install 중에 아티팩트 nodeadm 설치를 중지하고 제거합니다. 제거 명령은 클러스터에서 하이브리드 노드를 드레이닝하거나 삭제하지 않습니다. 드레이닝 및 삭제 작업을 별도로 실행해야 합니다. 자세한 내용은 [the section called “하이브리드 노드 삭제”](#) 섹션을 참조하세요. 노드에 포드가 남아 있는 경우 기본적으로 nodeadm uninstall이 진행되지 않습니다. 마찬가지로 nodeadm uninstall은 CNI 종속성 또는 클러스터에서 실행하는 다른 Kubernetes 추가 기능의 종속성을 제거하지 않습니다. 호스트에서 CNI 설치를 완전히 제거하려면 [the section called “CNI 구성”](#)의 지침을 참조하세요. 온프레미스 자격 증명 공급자로 AWS SSM 하이브리드 활성화를 사용하는 경우 nodeadm uninstall 명령은 호스트를 AWS SSM 관리형 인스턴스로 등록 취소합니다.

사용량

```
nodeadm uninstall [flags]
```

Flags

명칭	필수	설명
-s,	FALSE	건너뛸 업그레이드 단계입니다. 문제 해결에 도움이 되지 않

명칭	필수	설명
--skip		는 한 단계를 건너뛰는 것은 권장되지 않습니다. 값 pod-validation 은 대몬 세트 및 정적 포드를 제외하고 노드에서 모든 포드가 실행 중인 지 확인하는 단계를 건너뛵니다. node-validation 은 노드가 차단되었는지 확인하는 단계를 건너뛵니다. init-validation 은 제거를 실행하기 전에 노드가 성공적으로 초기화되었는지 확인하는 단계를 건너뛵니다.
-h, --help	FALSE	사용 가능한 플래그, 하위 명령, 위치 값 파라미터가 포함된 도움말 메시지가 표시됩니다.

예시

```
nodeadm uninstall
```

```
nodeadm uninstall --skip node-validation,pod-validation
```

nodeadm debug

nodeadm debug 명령을 사용하여 비정상이거나 잘못 구성된 하이브리드 노드 문제를 해결할 수 있습니다. 다음 요구 사항이 있는지 확인합니다.

- 노드는 자격 증명을 얻는 데 필요한 AWS API에 대한 네트워크 액세스 권한이 있습니다.

- 노드는 구성된 하이브리드 노드 IAM 역할에 대한 AWS 자격 증명을 가져올 수 있습니다.
- 노드에는 EKS Kubernetes API 엔드포인트에 대한 네트워크 액세스 권한과 EKS Kubernetes API 엔드포인트 인증서의 유효성이 있습니다.
- 노드는 EKS 클러스터로 인증할 수 있고, 클러스터의 자격 증명에 유효하며, 노드가 EKS 클러스터에 대해 구성된 VPC를 통해 EKS 클러스터에 액세스할 수 있는지 확인할 수 있습니다.

오류가 발견되면 명령의 출력에서 문제 해결 단계를 제안합니다. 특정 검증 단계는 하위 프로세스를 보여줍니다. 이러한 오류가 발생하면 출력이 검증 오류 아래의 stderr 섹션에 표시됩니다.

사용량

```
nodeadm debug [flags]
```

Flags

명칭	필수	설명
-c, --config-source	TRUE	nodeadm 구성 소스입니다. 하이브리드 노드의 경우 입력은 파일 체계의 URI를 따라야 합니다.
--no-color	FALSE	색상 출력을 비활성화합니다. 자동화에 유용합니다.
-h, --help	FALSE	사용 가능한 플래그, 하위 명령, 위치 값 파라미터가 포함된 도움말 메시지가 표시됩니다.

예시

```
nodeadm debug -c file:///nodeConfig.yaml
```

Nodeadm 파일 위치

nodeadm install

nodeadm install을 실행할 때 다음 파일과 파일 위치가 구성됩니다.

아티팩트	경로
IAM Roles Anywhere CLI	/usr/local/bin/aws_signing_helper
Kubelet 바이너리	/usr/bin/kubelet
Kubectl 바이너리	usr/local/bin/kubectl
ECR 자격 증명 공급자	/etc/eks/image-credential-provider/ecr-credential-provider
AWS IAM 인증자	/usr/local/bin/aws-iam-authenticator
SSM 설정 CLI	/opt/ssm/ssm-setup-cli
SSM 에이전트	Ubuntu - /snap/amazon-ssm-agent/current/amazon-ssm-agent RHEL 및 AL2023 - /usr/bin/amazon-ssm-agent
Containerd	Ubuntu 및 AL2023 - /usr/bin/containerd RHEL - /bin/containerd
Iptables	Ubuntu 및 AL2023 - /usr/sbin/iptables RHEL - /sbin/iptables
CNI 플러그인	/opt/cni/bin
설치된 아티팩트 트래커	/opt/nodeadm/tracker

nodeadm init

nodeadm init를 실행할 때 다음 파일과 파일 위치가 구성됩니다.

명칭	경로
Kubelet kubeconfig	/var/lib/kubelet/kubeconfig

명칭	경로
Kubelet 구성	/etc/kubernetes/kubelet/config.json
Kubelet systemd 단위	/etc/systemd/system/kubelet.service
이미지 자격 증명 공급자 구성	/etc/eks/image-credential-provider/config.json
Kubelet env 파일	/etc/eks/kubelet/environment
Kubelet 인증서	/etc/kubernetes/pki/ca.crt
Containerd 구성	/etc/containerd/config.toml
Containerd 커널 모듈 구성	/etc/modules-load.d/containerd.conf
AWS Config 파일	/etc/aws/hybrid/config
AWS 자격 증명 파일(자격 증명 파일을 활성화 하는 경우)	/eks-hybrid/.aws/credentials
AWS signing helper 시스템 단위	/etc/systemd/system/aws_signing_helper_update.service
Sysctl conf 파일	/etc/sysctl.d/99-nodeadm.conf
Ca-certificates	/etc/ssl/certs/ca-certificates.crt
Gpg 키 파일	/etc/apt/keyrings/docker.asc
Docker 리포지토리 소스 파일	/etc/apt/sources.list.d/docker.list

SSM 하이브리드 활성화를 위한 노드 구성

다음은 하이브리드 노드 자격 증명에 AWS SSM 하이브리드 활성화를 사용할 때의 샘플 `nodeConfig.yaml`입니다.

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
```

```

name:          # Name of the EKS cluster
region:        # AWS Region where the EKS cluster resides
hybrid:
  ssm:
    activationCode: # SSM hybrid activation code
    activationId:   # SSM hybrid activation id

```

IAM Roles Anywhere를 위한 노드 구성

다음은 하이브리드 노드 자격 증명을 위한 AWS IAM Roles Anywhere의 샘플 `nodeConfig.yaml`입니다.

AWS IAM Roles Anywhere를 온프레미스 자격 증명 공급자로 사용하는 경우 `nodeadm` 구성에서 사용하는 `nodeName`은 하이브리드 노드 IAM 역할에 대해 범위가 지정된 권한과 일치해야 합니다. 예를 들어 하이브리드 노드 IAM 역할에 대한 권한이 역할 세션 이름이 호스트 인증서의 CN과 같을 때만 AWS IAM Roles Anywhere가 역할을 수임하도록 허용하는 경우 `nodeadm` 구성의 `nodeName`은 인증서의 CN과 동일해야 합니다. 사용하는 `nodeName`은 64자를 초과할 수 없습니다. 자세한 내용은 [the section called “자격 증명 준비”](#) 섹션을 참조하세요.

```

apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:          # Name of the EKS cluster
    region:        # AWS Region where the EKS cluster resides
  hybrid:
    iamRolesAnywhere:
      nodeName:      # Name of the node
      trustAnchorArn: # ARN of the IAM Roles Anywhere trust anchor
      profileArn:     # ARN of the IAM Roles Anywhere profile
      roleArn:        # ARN of the Hybrid Nodes IAM role
      certificatePath: # Path to the certificate file to authenticate with the IAM
                       Roles Anywhere trust anchor
      privateKeyPath: # Path to the private key file for the certificate

```

kubelet 사용자 지정을 위한 노드 구성(선택 사항)

`nodeadm` 구성에서 `kubelet` 구성 및 플래그를 전달할 수 있습니다. `shutdownGracePeriod`를 30초로 설정하기 위해 노드 레이블 `abc.amazonaws.com/test-label` 및 구성을 추가하는 방법은 아래 예제를 참조하세요.

```

apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:           # Name of the EKS cluster
    region:         # AWS Region where the EKS cluster resides
  kubelet:
    config:          # Map of kubelet config and values
    shutdownGracePeriod: 30s
    flags:           # List of kubelet flags
    - --node-labels=abc.company.com/test-label=true
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id

```

Containered 사용자 지정을 위한 노드 구성(선택 사항)

nodeadm 구성에서 사용자 지정 Containered 구성을 전달할 수 있습니다. nodeadm에 대한 Containered 구성은 인라인 TOML을 허용합니다. Containered 콘텐츠 스토어에서 압축되지 않은 이미 지 계층의 삭제를 비활성화하도록 Containered를 구성하는 방법은 아래 예제를 참조하세요.

```

apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:           # Name of the EKS cluster
    region:         # AWS Region where the EKS cluster resides
  containerd:
    config: |          # Inline TOML containerd additional configuration
    [plugins."io.containerd.grpc.v1.cri".containerd]
    discard_unpacked_layers = false
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:   # SSM hybrid activation id

```

Containered 구성을 사용하여 SELinux 지원을 활성화할 수도 있습니다. Containerd에서 SELinux를 활성화한 경우 노드에 예약된 포트에 적절한 securityContext 및 selinuxOptions가 활성화되어 있는지 확인합니다. 보안 컨텍스트 구성에 대한 자세한 내용은 [Kubernetes 설명서](#)에서 확인할 수 있습니다.

Note

Red Hat Enterprise Linux(RHEL) 8 및 RHEL 9는 SELinux가 기본적으로 활성화하고 호스트에서 엄격으로 설정됩니다. Amazon Linux 2023는 SELinux가 기본적으로 활성화되고 허용 모드로 설정됩니다. SELinux가 호스트에서 허용 모드로 설정된 경우 Containerd에서 활성화하면 요청이 차단되지 않고 호스트의 SELinux 구성에 따라 로깅됩니다.

```
apiVersion: node.eks.aws/v1alpha1
kind: NodeConfig
spec:
  cluster:
    name:           # Name of the EKS cluster
    region:         # AWS Region where the EKS cluster resides
  containerd:
    config: |        # Inline TOML containerd additional configuration
      [plugins."io.containerd.grpc.v1.cri"]
      enable_selinux = true
  hybrid:
    ssm:
      activationCode: # SSM hybrid activation code
      activationId:  # SSM hybrid activation id
```

하이브리드 노드 문제 해결

이 주제에서는 Amazon EKS Hybrid Nodes를 사용하는 동안 발생할 수 있는 몇 가지 일반적 오류와 이를 해결하는 방법을 다룹니다. 기타 문제 해결 정보는 [문제 해결](#) 및 AWS re:Post의 [Amazon EKS용 지식 센터 태그](#)를 참조하세요. 문제를 해결할 수 없는 경우 AWS Support에 문의하세요.

nodeadm debug로 노드 문제 해결 하이브리드 노드에서 nodeadm debug 명령을 실행하여 네트워킹 및 자격 증명 요구 사항이 충족되는지 확인할 수 있습니다. nodeadm debug 명령에 대한 자세한 내용은 [the section called “하이브리드 노드 nodeadm”](#) 섹션을 참조하세요.

클러스터 인사이트를 통해 하이브리드 노드 관련 문제 탐지 Amazon EKS 클러스터 인사이트에는 클러스터의 EKS 하이브리드 노드 구성과 관련된 일반적인 문제를 탐지하는 인사이트 검사가 포함됩니다. AWS Management Console, AWS CLI, AWS SDK에서 모든 인사이트 검사의 결과를 볼 수 있습니다. 클러스터 인사이트에 대한 자세한 내용은 [the section called “클러스터 인사이트”](#) 섹션을 참조하세요.

하이브리드 노드 설치 문제 해결

다음 문제 해결 주제는 `nodeadm install` 명령을 사용하여 호스트에 하이브리드 노드 종속성을 설치하는 것과 관련이 있습니다.

nodeadm 명령이 **must run as root**와 함께 실패

`nodeadm install` 명령은 호스트에 root 또는 sudo 권한이 있는 사용자를 사용하여 실행해야 합니다. root 또는 sudo 권한이 없는 사용자로 `nodeadm install`을 실행하는 경우 `nodeadm` 출력에 다음 오류가 표시됩니다.

```
"msg":"Command failed","error":"must run as root"
```

종속성에 연결할 수 없음

`nodeadm install` 명령은 하이브리드 노드에 필요한 종속성을 설치합니다. 하이브리드 노드 종속성에는 `containerd`, `kubelet`, `kubect1`, AWS SSM 또는 AWS IAM Roles Anywhere 구성 요소가 포함됩니다. 이러한 종속성을 다운로드하려면 `nodeadm install`을 실행하는 위치에서 액세스할 수 있어야 합니다. 액세스가 가능해야 하는 위치 목록에 대한 자세한 내용은 [the section called “네트워킹 준비”](#) 섹션을 참조하세요. 액세스 권한이 없는 경우 `nodeadm install` 출력에 다음과 유사한 오류가 표시됩니다.

```
"msg":"Command failed","error":"failed reading file from url: ...: max retries achieved for http request"
```

패키지 관리자 업데이트 실패

`nodeadm install` 명령은 하이브리드 노드 종속성을 설치하기 전에 `apt update` 또는 `yum update` 또는 `dnf update`를 실행합니다. 이 단계가 성공하지 않으면 다음과 유사한 오류가 표시될 수 있습니다. 문제를 해결하려면 `nodeadm install` 실행 전에 `apt update` 또는 `yum update` 또는 `dnf update` 업데이트를 실행하거나 `nodeadm install`을 다시 실행해 볼 수 있습니다.

```
failed to run update using package manager
```

제한 시간 또는 컨텍스트 기한 초과

`nodeadm install`을 실행할 때 설치 프로세스의 다양한 단계에서 제한 시간 또는 컨텍스트 기한을 초과했음을 나타내는 오류가 발생하는 경우, 연결이 느려져 제한 시간이 충족되기 전에 하이브리드 노드 종속성의 설치를 방해할 수 있습니다. 이러한 문제를 해결하려면 `nodeadm`에서 `--timeout` 플래그를 사용하여 종속성을 다운로드하는 제한 시간을 연장할 수 있습니다.


```
nodeadm install K8S_VERSION --credential-provider CREDS_PROVIDER --timeout 20m0s
```

하이브리드 노드 연결 문제 해결

이 섹션의 문제 해결 주제는 `nodeadm init` 명령을 사용하여 하이브리드 노드를 EKS 클러스터에 연결하는 프로세스와 관련이 있습니다.

작업 오류 또는 지원되지 않는 체계

`nodeadm init`를 실행할 때 `operation error` 또는 `unsupported scheme`와 관련된 오류가 표시되면 `nodeConfig.yaml`을 확인하여 형식이 올바르게 지정되어 있고 `nodeadm`에 전달되었는지 확인합니다. `nodeConfig.yaml` 형식 및 옵션에 대한 자세한 내용은 [the section called “하이브리드 노드 nodeadm”](#) 섹션을 참조하세요.

```
"msg":"Command failed","error":"operation error ec2imds: GetRegion, request canceled, context deadline exceeded"
```

eks:DescribeCluster 작업에 대한 하이브리드 노드 IAM 역할 권한 누락

`nodeadm init`를 실행할 때 `nodeadm`은 클러스터 설명을 직접 호출하여 EKS 클러스터에 대한 정보를 수집하려고 시도합니다. 하이브리드 노드 IAM 역할에 `eks:DescribeCluster` 작업에 대한 권한이 없는 경우입니다. 하이브리드 노드 IAM 역할에 필요한 권한에 대한 자세한 내용은 [the section called “자격 증명 준비”](#) 섹션을 참조하세요.

```
"msg":"Command failed","error":"operation error EKS: DescribeCluster, https response error StatusCode: 403 ... AccessDeniedException"
```

노드 IP가 원격 노드 네트워크 CIDR에 없음

`nodeadm init`를 실행할 때 노드의 IP 주소가 지정된 원격 노드 네트워크 CIDR 내에 있지 않으면 오류가 발생할 수 있습니다. 오류는 다음 예와 유사합니다.

```
node IP 10.18.0.1 is not in any of the remote network CIDR blocks [10.0.0.0/16 192.168.0.0/16]
```

이 예는 원격 네트워크 CIDR가 10.0.0.0/16 및 192.168.0.0/16인 클러스터에 조인하려고 시도하는 IP 주소가 10.18.0.1인 노드를 보여줍니다. 10.18.0.1이 두 범위 중 하나에 속하지 않기 때문에 오류가 발생합니다.

모든 노드 IP 주소를 포함하도록 RemoteNodeNetworks를 올바르게 구성했는지 확인합니다. 네트워킹 구성에 대한 자세한 내용은 [the section called “네트워킹 준비”](#) 섹션을 참조하세요.

- RemoteNodeNetwork 구성을 확인하려면 클러스터가 위치한 리전에서 다음 명령을 실행합니다. 출력에 나열된 CIDR 블록에 노드의 IP 범위가 포함되어 있고 오류 메시지에 나열된 CIDR 블록과 동일한지 확인합니다. 일치하지 않는 경우 nodeConfig.yaml의 클러스터 이름 및 리전이 의도한 클러스터와 일치하는지 확인합니다.

```
aws eks describe-cluster --name CLUSTER_NAME --region REGION_NAME --query
cluster.remoteNetworkConfig.remoteNodeNetworks
```

- 의도한 노드로 작업하고 있는지 확인합니다.
 - 호스트 이름 및 IP 주소가 클러스터에 등록하려는 주소와 일치하는지 확인하여 올바른 노드에 있는지 확인합니다.
 - 이 노드가 올바른 온프레미스 네트워크(클러스터 설정 중에 CIDR 범위가 RemoteNodeNetwork로 등록된 네트워크)에 있는지 확인합니다.

노드 IP가 여전히 예상과 다를 경우 다음을 확인하세요.

- IAM Roles Anywhere를 사용하는 경우 kubelet은 IAM Roles Anywhere nodeName에서 DNS 조회를 수행하고 사용 가능한 경우 노드 이름과 연결된 IP를 사용합니다. 노드의 DNS 항목을 유지 관리하는 경우 이러한 항목이 원격 노드 네트워크 CIDR 내의 IP를 가리키는지 확인합니다.
- 노드에 여러 네트워크 인터페이스가 있는 경우 kubelet이 원격 노드 네트워크 CIDR 외부의 IP 주소가 있는 인터페이스를 기본값으로 선택할 수도 있습니다. 다른 인터페이스를 사용하려면 nodeConfig.yaml의 --node-ip kubelet 플래그를 사용하여 IP 주소를 지정합니다. 자세한 내용은 [the section called “하이브리드 노드 nodeadm”](#) 섹션을 참조하세요. 노드에서 다음 명령을 실행하여 노드의 네트워크 인터페이스 및 해당 IP 주소를 볼 수 있습니다.

```
ip addr show
```

하이브리드 노드가 EKS 클러스터에 표시되지 않음

nodeadm init를 실행했지만 하이브리드 노드가 클러스터에 표시되지 않는 경우 하이브리드 노드와 EKS 컨트롤 플레인 간의 네트워크 연결에 문제가 있거나, 필요한 보안 그룹 권한이 구성되지 않았거나, Kubernetes 역할 기반 액세스 제어(RBAC)에 필요한 하이브리드 노드 IAM 역할을 매핑하지 않았을

수 있습니다. 다음 명령으로 kubelet 및 kubelet 로그의 상태를 확인하여 디버깅 프로세스를 시작할 수 있습니다. 클러스터에 조인하지 못한 하이브리드 노드에서 다음 명령을 실행합니다.

```
systemctl status kubelet
```

```
journalctl -u kubelet -f
```

클러스터와 통신할 수 없음

하이브리드 노드가 클러스터 컨트롤 플레인과 통신할 수 없는 경우 다음과 유사한 로그가 표시될 수 있습니다.

```
"Failed to ensure lease exists, will retry" err="Get ..."
```

```
"Unable to register node with API server" err="Post ..."
```

```
Failed to contact API server when waiting for CSINode publishing ... dial tcp <ip address>: i/o timeout
```

이러한 메시지가 표시되면 다음을 확인하여 [the section called “네트워킹 준비”](#)에 자세히 설명된 하이브리드 노드 요구 사항을 충족하는지 확인합니다.

- EKS 클러스터에 전달된 VPC에 온프레미스 노드 및 선택적으로 포트 CIDR에 대한 전송 게이트웨이 (TGW) 또는 가상 프라이빗 게이트웨이(VGW)에 대한 경로가 있는지 확인합니다.
- EKS 클러스터에 대한 추가 보안 그룹에 온프레미스 노드 CIDR 및 선택적으로 포트 CIDR에 대한 인바운드 규칙이 있는지 확인합니다.
- 온프레미스 라우터가 EKS 컨트롤 플레인으로의 트래픽과 EKS 컨트롤 플레인으로부터의 트래픽을 허용하도록 구성되어 있는지 확인합니다.

권한이 없음

하이브리드 노드가 EKS 컨트롤 플레인과 통신할 수 있지만 등록할 수 없는 경우 다음과 유사한 로그가 표시될 수 있습니다. 아래 로그 메시지의 주요 차이점은 Unauthorized 오류입니다. 이는 노드에 필요한 권한이 없기 때문에 노드가 작업을 수행할 수 없음을 나타냅니다.

```
"Failed to ensure lease exists, will retry" err="Unauthorized"
```

```
"Unable to register node with API server" err="Unauthorized"
```

```
Failed to contact API server when waiting for CSINode publishing: Unauthorized
```

이러한 메시지가 표시되면 다음을 확인하여 [the section called “자격 증명 준비”](#) 및 [the section called “클러스터 액세스 준비”](#)의 하이브리드 노드 요구 사항 세부 정보를 충족하는지 확인합니다.

- 하이브리드 노드의 ID가 예상 하이브리드 노드 IAM 역할과 일치하는지 확인합니다. 하이브리드 노드에서 `sudo aws sts get-caller-identity`를 실행하여 이 작업을 수행할 수 있습니다.
- 하이브리드 노드 IAM 역할에 필요한 권한이 있는지 확인합니다.
- 클러스터에 하이브리드 노드 IAM 역할에 대한 EKS 액세스 항목이 있는지 확인하거나 `aws-auth ConfigMap`에 하이브리드 노드 IAM 역할에 대한 항목이 있는지 확인합니다. EKS 액세스 항목을 사용하는 경우 하이브리드 노드 IAM 역할에 대한 액세스 항목이 `HYBRID_LINUX` 액세스 유형인지 확인합니다. `aws-auth ConfigMap`을 사용하는 경우 하이브리드 노드 IAM 역할에 대한 항목이 [the section called “클러스터 액세스 준비”](#)에 설명된 요구 사항 및 형식을 충족하는지 확인합니다.

EKS 클러스터에 등록되었지만 **Not Ready** 상태로 표시되는 하이브리드 노드

하이브리드 노드가 EKS 클러스터에 성공적으로 등록되었지만 하이브리드 노드에 `Not Ready` 상태가 표시되는 경우 가장 먼저 확인해야 할 것은 컨테이너 네트워킹 인터페이스(CNI) 상태입니다. CNI를 설치하지 않은 경우 하이브리드 노드의 상태가 `Not Ready`일 것으로 예상됩니다. CNI가 성공적으로 설치되고 실행되면 노드가 `Ready` 상태로 전환됩니다. CNI를 설치하려고 했지만 성공적으로 실행되지 않는 경우 이 페이지의 [the section called “하이브리드 노드 CNI 문제 해결”](#) 섹션을 참조하세요.

인증서 서명 요청(CSR) 보류 중

하이브리드 노드를 EKS 클러스터에 연결한 후 하이브리드 노드에 대해 보류 중인 CSR이 있는 경우 하이브리드 노드가 자동 승인 요구 사항을 충족하지 않습니다. 하이브리드 노드에 대한 CSR은 `eks.amazonaws.com/compute-type: hybrid` 레이블이 있는 노드에서 하이브리드 노드용 CSR을 생성하고, CSR에 다음과 같은 주제 대체 이름(SAN)이 포함된 경우 자동으로 승인 및 발급됩니다. 노드 이름과 동일한 DNS SAN이 1개 이상이며 IP SAN이 원격 노드 네트워크 CIDR에 속합니다.

하이브리드 프로필 이미 있음

`nodeadm` 구성을 변경하고 새 구성으로 노드를 다시 등록하려고 하면 하이브리드 프로필이 이미 있지만 내용이 변경되었다는 오류가 표시될 수 있습니다. 구성 변경 사이에 `nodeadm init`를 실행하는 대신 `nodeadm uninstall`를 실행한 다음 `nodeadm install` 및 `nodeadm init`를 실행합니다. 이렇게 하면 구성 변경 사항을 적절히 정리할 수 있습니다.

```
"msg":"Command failed","error":"hybrid profile already exists at /etc/aws/hybrid/config but its contents do not align with the expected configuration"
```

하이브리드 노드가 프라이빗 API를 확인하지 못함

nodeadm init를 실행한 후 no such host가 있어 kubelet 로그에 EKS Kubernetes API 서버에 연결하지 못하는 오류가 표시되는 경우 온프레미스 네트워크 또는 호스트 수준에서 EKS Kubernetes API 엔드포인트의 DNS 항목을 변경해야 할 수 있습니다. AWS Route53 설명서의 [Forwarding inbound DNS queries to your VPC](#)를 참조하세요.

```
Failed to contact API server when waiting for CSINode publishing: Get ... no such host
```

EKS 콘솔에서 하이브리드 노드를 볼 수 없음

하이브리드 노드를 등록했지만 EKS 콘솔에서 볼 수 없는 경우 콘솔을 보는 데 사용 중인 IAM 보안 주체의 권한을 확인합니다. 사용 중인 IAM 보안 주체는 콘솔에서 리소스를 볼 수 있는 특정 최소 IAM 및 Kubernetes 권한이 있어야 합니다. 자세한 내용은 [the section called “클러스터 리소스 액세스”](#) 섹션을 참조하세요.

하이브리드 노드 실행 문제 해결

EKS 클러스터에 등록된 하이브리드 노드가 Ready 상태였다가 Not Ready 상태로 전환된 경우 CPU, 메모리 또는 사용 가능한 디스크 공간에 대한 리소스가 부족하거나 노드가 EKS 컨트롤 플레인에서 연결 해제되는 등 상태가 비정상일 수 있는 다양한 문제가 있습니다. 아래 단계를 사용하여 노드 문제를 해결할 수 있으며, 문제를 해결할 수 없는 경우 AWS Support에 문의하세요.

하이브리드 노드에서 nodeadm debug를 실행하여 네트워킹 및 자격 증명 요구 사항이 충족되는지 확인합니다. nodeadm debug 명령에 대한 자세한 내용은 [the section called “하이브리드 노드 nodeadm”](#) 섹션을 참조하세요.

노드 상태 가져오기

```
kubectl get nodes -o wide
```

노드 조건 및 이벤트 확인

```
kubectl describe node NODE_NAME
```

포드 상태 가져오기

```
kubectl get pods -A -o wide
```

포드 조건 및 이벤트 확인

```
kubectl describe pod POD_NAME
```

포드 로그 확인

```
kubectl logs POD_NAME
```

kubectl 로그 확인

```
systemctl status kubelet
```

```
journalctl -u kubelet -f
```

포드 활성 프로브가 실패하거나 웹후크가 작동하지 않음

하이브리드 노드에서 실행되는 애플리케이션, 추가 기능 또는 웹후크가 제대로 시작되지 않는 경우 포드에 대한 통신을 차단하는 네트워킹 문제가 있을 수 있습니다. EKS 컨트롤 플레인이 하이브리드 노드에서 실행되는 웹후크와 접촉하려면 원격 포드 네트워크로 EKS 클러스터를 구성하고 전송 게이트웨이(TGW), 가상 프라이빗 게이트웨이(VPG) 또는 VPC를 온프레미스 네트워크와 연결하는 데 사용하는 기타 게이트웨이를 대상으로 하는 VPC 라우팅 테이블의 온프레미스 포드 CIDR에 대한 경로가 있어야 합니다. 하이브리드 노드의 네트워킹 요구 사항에 대한 자세한 내용은 [the section called “네트워킹 준비”](#) 섹션을 참조하세요. 또한 온프레미스 방화벽에서 이 트래픽을 허용하고 라우터가 포드로 올바르게 라우팅될 수 있도록 해야 합니다. 하이브리드 노드에서의 웹후크 실행 요구 사항에 대한 자세한 내용은 [the section called “웹후크 구성”](#) 섹션을 참조하세요.

이 시나리오에 대한 일반적인 포드 로그 메시지는 아래에 나와 있습니다. 여기서 ip-address는 Kubernetes 서비스의 클러스터 IP입니다.

```
dial tcp <ip-address>:443: connect: no route to host
```

kubectl logs 또는 kubectl exec 명령이 작동하지 않음

다른 `kubectl exec` 명령은 성공하지만 `kubectl logs` 또는 `kubectl` 명령이 시간 초과되면 원격 네트워크 구성과 관련된 문제일 수 있습니다. 이러한 명령은 클러스터를 통해 노드의 `kubelet` 엔드포인트에 연결됩니다. 자세한 내용은 [the section called “kubelet 엔드포인트”](#)을 참조하세요.

노드 IP 및 포트 IP가 클러스터에 대해 구성된 원격 노드 네트워크 및 원격 포트 네트워크 CIDR 내에 속하는지 확인합니다. 아래 명령을 사용하여 IP 할당을 검사합니다.

```
kubectl get nodes -o wide
```

```
kubectl get pods -A -o wide
```

이러한 IP를 구성된 원격 네트워크 CIDR과 비교하여 적절한 라우팅을 보장합니다. 네트워크 구성 요구 사항에 대한 자세한 내용은 [the section called “네트워킹 준비”](#) 섹션을 참조하세요.

하이브리드 노드 CNI 문제 해결

하이브리드 노드로 Cilium 또는 Calico를 처음 시작하는 데 문제가 발생하는 경우 하이브리드 노드 또는 하이브리드 노드에서 실행되는 CNI 포트와 EKS 컨트롤 플레인 간의 네트워킹 문제로 인해 가장 자주 발생합니다. 환경이 하이브리드 노드용 네트워킹 준비의 요구 사항을 충족하는지 확인합니다. 문제를 세분화하면 도움이 됩니다.

EKS 클러스터 구성

RemoteNodeNetwork 및 RemotePodNetwork 구성이 정확합니까?

VPC 구성

전송 게이트웨이 또는 가상 프라이빗 게이트웨이가 대상인 VPC 라우팅 테이블에 RemoteNodeNetwork 및 RemotePodNetwork에 대한 경로가 있습니까?

보안 그룹 구성

RemoteNodeNetwork 및 RemotePodNetwork에 대한 인바운드 및 아웃바운드 규칙이 있습니까?

온프레미스 네트워크

EKS 컨트롤 플레인과 하이브리드 노드 및 하이브리드 노드에서 실행되는 포트 간에 경로 및 액세스 권한이 있습니까?

CNI 구성

오버레이 네트워크를 사용하는 경우 웹후크를 사용할 때 CNI의 IP 풀 구성이 EKS 클러스터에 대해 구성된 RemotePodNetwork와 일치합니까?

하이브리드 노드에 CNI가 설치되지 않고 **Ready** 상태임

하이브리드 노드에 Ready 상태가 표시되지만 클러스터에 CNI를 설치하지 않은 경우 하이브리드 노드에 이전 CNI 아티팩트가 있을 수 있습니다. 기본적으로 Helm과 같은 도구를 사용하여 Cilium 및 Calico를 제거하면 물리적 머신이나 가상 머신에서 온디스크 리소스가 제거되지 않습니다. 또한 이러한 CNI에 대한 사용자 지정 리소스 정의(CRD)가 이전 설치에서 클러스터에 계속 존재할 수 있습니다. 자세한 내용은 [the section called “CNI 구성”](#)의 Cilium 삭제 및 Calico 삭제 섹션을 참조하세요.

Cilium 문제 해결

하이브리드 노드에서 Cilium을 실행하는 데 문제가 있는 경우 Cilium 설명서의 [the troubleshooting steps](#)를 참조하세요. 아래 섹션에서는 하이브리드 노드에 Cilium을 배포하는 작업과 관련된 문제를 다룹니다.

Cilium이 시작되지 않음

각 하이브리드 노드에서 실행되는 Cilium 에이전트가 시작되지 않는 경우 Cilium 에이전트 포드의 로그에 오류가 있는지 확인합니다. Cilium 에이전트를 시작하려면 EKS Kubernetes API 엔드포인트에 연결해야 합니다. 이 연결이 올바르게 구성되지 않으면 Cilium 에이전트 시작이 실패합니다. 이 경우 Cilium 에이전트 포드 로그에 다음과 유사한 로그 메시지가 표시됩니다.

```
msg="Unable to contact k8s api-server"
level=fatal msg="failed to start: Get \"https://<k8s-cluster-ip>:443/api/v1/namespaces/kube-system\": dial tcp <k8s-cluster-ip>:443: i/o timeout"
```

Cilium 에이전트는 호스트 네트워크에서 실행됩니다. Cilium 연결을 위해 EKS 클러스터를 RemoteNodeNetwork로 구성해야 합니다. RemoteNodeNetwork에 대한 인바운드 규칙을 적용한 EKS 클러스터에 대한 추가 보안 그룹이 있고, RemoteNodeNetwork에 대한 VPC에 경로가 있으며 온프레미스 네트워크가 EKS 컨트롤 플레인에 연결할 수 있도록 올바르게 구성되어 있는지 확인합니다.

Cilium 연산자가 실행 중이고 전부가 아닌 일부 Cilium 에이전트가 실행 중인 경우 클러스터의 모든 노드에 할당할 수 있는 포트 IP가 있는지 확인합니다. Cilium 구성에서 clusterPoolIPv4PodCIDRList와 함께 클러스터 풀 IPAM을 사용할 때 할당 가능한 포트 CIDR의 크기를 구성합니다. 노드별 CIDR 크기는 Cilium 구성의 clusterPoolIPv4MaskSize 설정으로 구성됩니다. 자세한 내용은 Cilium 설명서의 [클러스터 풀 확장](#)을 참조하세요.

Cilium BGP가 작동하지 않음

Cilium BGP 컨트롤 플레인을 사용하여 포트 또는 서비스 주소를 온프레미스 네트워크에 알리는 경우 다음 Cilium CLI 명령을 사용하여 BGP가 리소스에 경로를 알리는지 확인할 수 있습니다. Cilium CLI를 설치하는 단계는 Cilium 설명서의 [Install the Cilium CLI](#)를 참조하세요.

BGP가 올바르게 작동하는 경우 출력에 세션 상태가 established인 하이브리드 노드가 있어야 합니다. 네트워킹 팀과 협력하여 환경의 로컬 AS, 피어 AS, 피어 주소에 대한 올바른 값을 식별해야 할 수 있습니다.

```
cilium bgp peers
```

```
cilium bgp routes
```

Cilium BGP를 사용하여 유형이 LoadBalancer인 서비스의 IP를 알리는 경우 CiliumLoadBalancerIPPool 및 서비스 모두에 동일한 레이블이 있어야 하며, 이는 CiliumBGPAdvertisement의 선택기에 사용해야 합니다. 예를 들면 다음과 같습니다. 참고로 Cilium BGP를 사용하여 LoadBalancer 유형의 서비스 IP를 알리는 경우 Cilium 에이전트를 다시 시작하는 동안 BGP 경로가 중단될 수 있습니다. 자세한 내용은 Cilium 설명서의 [Failure Scenarios](#)를 참조하세요.

서비스

```
kind: Service
apiVersion: v1
metadata:
  name: guestbook
  labels:
    app: guestbook
spec:
  ports:
    - port: 3000
      targetPort: http-server
  selector:
    app: guestbook
  type: LoadBalancer
```

CiliumLoadBalancerIPPool

```
apiVersion: cilium.io/v2alpha1
kind: CiliumLoadBalancerIPPool
metadata:
  name: guestbook-pool
  labels:
    app: guestbook
spec:
```

```

blocks:
- cidr: <CIDR to advertise>
serviceSelector:
  matchExpressions:
    - { key: app, operator: In, values: [ guestbook ] }

```

CiliumBGPAdvertisement

```

apiVersion: cilium.io/v2alpha1
kind: CiliumBGPAdvertisement
metadata:
  name: bgp-advertisements-guestbook
  labels:
    advertise: bgp
spec:
  advertisements:
    - advertisementType: "Service"
      service:
        addresses:
          - ExternalIP
          - LoadBalancerIP
      selector:
        matchExpressions:
          - { key: app, operator: In, values: [ guestbook ] }

```

Calico 문제 해결

하이브리드 노드에서 Calico를 실행하는 데 문제가 있는 경우 Calico 설명서의 [the troubleshooting steps](#)를 참조하세요. 아래 섹션에서는 하이브리드 노드에 Calico를 배포하는 작업과 관련된 문제를 다룹니다.

아래 표에는 Calico 구성 요소와 기본적으로 노드 또는 포드 네트워크에서 실행되는지 여부가 요약되어 있습니다. 발신 포드 트래픽에 NAT를 사용하도록 Calico를 구성한 경우 온프레미스 네트워크를 구성하여 온프레미스 노드 CIDR로 트래픽을 라우팅하고 VPC 라우팅 테이블을 전송 게이트웨이(TGW) 또는 가상 프라이빗 게이트웨이(VGW)를 대상으로 하는 온프레미스 노드 CIDR의 경로로 구성해야 합니다. 발신 포드 트래픽에 NAT를 사용하도록 Calico를 구성하지 않은 경우 온프레미스 네트워크를 구성하여 온프레미스 포드 CIDR로 트래픽을 라우팅하고 VPC 라우팅 테이블을 전송 게이트웨이(TGW) 또는 가상 프라이빗 게이트웨이(VGW)를 대상으로 하는 온프레미스 포드 CIDR의 경로로 구성해야 합니다.

구성 요소	Network
Calico API 서버	노드
Kubernetes용 Calico 컨트롤러	Pod
Calico 노드 에이전트	노드
Calico typha	노드
Calico CSI 노드 드라이버	Pod
Calico 연산자	노드

Calico 리소스가 차단된 노드에서 예약 또는 실행됨

DaemonSet로 실행되지 않는 Calico 리소스에는 기본적으로 유연한 허용 오차가 있어 포드를 예약하거나 실행할 준비가 되지 않은 차단된 노드에서 예약할 수 있습니다. 다음을 포함하도록 연산자 설치를 변경하여 DaemonSet Calico가 아닌 리소스에 대한 허용치를 강화할 수 있습니다.

```

installation:
  ...
  controlPlaneTolerations:
    - effect: NoExecute
      key: node.kubernetes.io/unreachable
      operator: Exists
      tolerationSeconds: 300
    - effect: NoExecute
      key: node.kubernetes.io/not-ready
      operator: Exists
      tolerationSeconds: 300
  calicoKubeControllersDeployment:
    spec:
      template:
        spec:
          tolerations:
            - effect: NoExecute
              key: node.kubernetes.io/unreachable
              operator: Exists
              tolerationSeconds: 300
            - effect: NoExecute

```

```

        key: node.kubernetes.io/not-ready
        operator: Exists
        tolerationSeconds: 300
typhaDeployment:
  spec:
    template:
      spec:
        tolerations:
          - effect: NoExecute
            key: node.kubernetes.io/unreachable
            operator: Exists
            tolerationSeconds: 300
          - effect: NoExecute
            key: node.kubernetes.io/not-ready
            operator: Exists
            tolerationSeconds: 300

```

자격 증명 문제 해결

AWS SSM 하이브리드 활성화와 AWS IAM Roles Anywhere 모두에 대해 하이브리드 노드에서 다음 명령을 실행하여 하이브리드 노드 IAM 역할에 대한 자격 증명이 하이브리드 노드에 올바르게 구성되어 있는지 검증할 수 있습니다. 노드 이름과 하이브리드 노드 IAM 역할 이름이 예상대로인지 확인합니다.

```
sudo aws sts get-caller-identity
```

```

{
  "UserId": "ABCDEFGHJKLM12345678910:<node-name>",
  "Account": "<aws-account-id>",
  "Arn": "arn:aws:sts::<aws-account-id>:assumed-role/<hybrid-nodes-iam-role/<node-name>"
}

```

AWS Systems Manager(SSM) 문제 해결

하이브리드 노드 자격 증명에 AWS SSM 하이브리드 활성화를 사용하는 경우 nodeadm이 하이브리드 노드에 설치하는 다음 SSM 디렉터리 및 아티팩트에 유의합니다. SSM 에이전트에 대한 자세한 내용은 AWS Systems Manager 사용 설명서의 [Working with the SSM agent](#)를 참조하세요.

설명	위치
SSM 에이전트	Ubuntu-/snap/amazon-ssm-agent/ current/amazon-ssm-agent RHEL & AL2023-/usr/bin/amazon-ssm-agent
SSM Agent 로그	/var/log/amazon/ssm
AWS 보안 인증	/root/.aws/credentials
SSM 설정 CLI	/opt/ssm/ssm-setup-cli

SSM 에이전트 다시 시작

일부 문제는 SSM 에이전트를 다시 시작하여 해결할 수 있습니다. 아래 명령을 사용하여 다시 시작할 수 있습니다.

AL2023 및 기타 운영 체제

```
systemctl restart amazon-ssm-agent
```

Ubuntu

```
systemctl restart snap.amazon-ssm-agent.amazon-ssm-agent
```

SSM 엔드포인트에 대한 연결 확인

하이브리드 노드에서 SSM 엔드포인트에 연결할 수 있는지 확인합니다. SSM 엔드포인트 목록은 [AWS Systems Manager endpoints and quotas](#)를 참조하세요. 아래 명령에서 us-west-2를 AWS SSM 하이브리드 활성화를 위한 AWS 리전으로 바꿉니다.

```
ping ssm.us-west-2.amazonaws.com
```

등록된 SSM 인스턴스의 연결 상태 보기

다음 AWS CLI 명령을 사용하여 SSM 하이브리드 활성화에 등록된 인스턴스의 연결 상태를 확인할 수 있습니다. 시스템 ID를 인스턴스의 시스템 ID로 바꿉니다.

```
aws ssm get-connection-status --target mi-012345678abcdefgh
```

SSM 설정 CLI 체크섬 불일치

`nodeadm install`을 실행할 때 `ssm-setup-cli` 체크섬 불일치 문제가 발생하면 호스트에 기존 SSM 설치가 없는지 확인해야 합니다. 호스트에 이전 SSM 설치가 있는 경우 이를 제거하고 `nodeadm install`을 다시 실행하여 문제를 해결합니다.

```
Failed to perform agent-installation/on-prem registration: error while verifying
installed ssm-setup-cli checksum: checksum mismatch with latest ssm-setup-cli.
```

SSM InvalidActivation

인스턴스를 AWS SSM에 등록하는 동안 오류가 발생하면 `nodeConfig.yaml`의 `region`, `activationCode`, `activationId`가 올바른지 확인합니다. EKS 클러스터의 AWS 리전은 SSM 하이브리드 활성화의 리전과 일치해야 합니다. 이러한 값이 잘못 구성된 경우 다음과 유사한 오류가 표시될 수 있습니다.

```
ERROR Registration failed due to error registering the instance with AWS SSM.
InvalidActivation
```

SSM ExpiredTokenException: 요청에 포함된 보안 토큰이 만료됨

SSM 에이전트가 자격 증명을 새로 고칠 수 없는 경우 `ExpiredTokenException`이 표시될 수 있습니다. 이 시나리오에서는 하이브리드 노드에서 SSM 엔드포인트에 연결할 수 있는 경우 SSM 에이전트를 다시 시작하여 자격 증명 새로 고침을 강제로 수행해야 할 수 있습니다.

```
"msg":"Command failed","error":"operation error SSM: DescribeInstanceInformation, https
response error StatusCode: 400, RequestID: eee03a9e-f7cc-470a-9647-73d47e4cf0be, api
error ExpiredTokenException: The security token included in the request is expired"
```

시스템 등록 명령을 실행하는 중 SSM 오류 발생

시스템을 SSM에 등록하는 동안 오류가 발생하면 `nodeadm install`을 다시 실행하여 모든 SSM 종속성이 올바르게 설치되었는지 확인해야 할 수 있습니다.

```
"error":"running register machine command: , error: fork/exec /opt/aws/ssm-setup-cli:
no such file or directory"
```

SSM ActivationExpired

nodeadm init를 실행할 때 만료된 활성화로 인해 인스턴스를 SSM에 등록하는 동안 오류가 발생하면 새 SSM 하이브리드 활성화를 생성하고, 새 SSM 하이브리드 활성화의 activationCode 및 activationId로 nodeConfig.yaml을 업데이트하고, nodeadm init를 다시 실행해야 합니다.

```
"msg":"Command failed","error":"SSM activation expired. Please use a valid activation"
```

```
ERROR Registration failed due to error registering the instance with AWS SSM.
ActivationExpired
```

SSM이 캐시된 자격 증명을 새로 고치지 못함

캐시된 자격 증명을 새로 고치지 못하면 호스트에서 /root/.aws/credentials 파일이 삭제되었을 수 있습니다. 먼저 SSM 하이브리드 활성화를 확인하고, 활성화가 활성 상태이며 하이브리드 노드가 활성화를 사용하도록 올바르게 구성되어 있는지 확인합니다. /var/log/amazon/ssm에서 SSM 에이전트 로그를 확인하고 SSM 측에서 문제를 해결한 후 nodeadm init 명령을 다시 실행합니다.

```
"Command failed","error":"operation error SSM: DescribeInstanceInformation, get
identity: get credentials: failed to refresh cached credentials"
```

SSM 정리

호스트에서 SSM 에이전트를 제거하려면 다음 명령을 실행할 수 있습니다.

```
dnf remove -y amazon-ssm-agent
sudo apt remove --purge amazon-ssm-agent
snap remove amazon-ssm-agent
rm -rf /var/lib/amazon/ssm/Vault/Store/RegistrationKey
```

AWS IAM Roles Anywhere 문제 해결

하이브리드 노드 자격 증명에 AWS IAM Roles Anywhere를 사용하는 경우 nodeadm에서 하이브리드 노드에 설치하는 다음 디렉터리 및 아티팩트에 유의합니다. IAM Roles Anywhere 문제 해결에 대한 자세한 내용은 AWS IAM Roles Anywhere 사용 설명서의 [Troubleshooting AWS IAM Roles Anywhere identity and access](#)을 참조하세요.

설명	위치
IAM Roles Anywhere CLI	/usr/local/bin/aws_signing_helper

설명	위치
기본 인증서 위치 및 이름	/etc/iam/pki/server.pem
기본 키 위치 및 이름	/etc/iam/pki/server.key

IAM Roles Anywhere가 캐시된 자격 증명을 새로 고치지 못함

캐시된 자격 증명을 새로 고치지 못한 경우 /etc/aws/hybrid/config의 내용을 검토하고 nodeadm 구성에서 IAM Roles Anywhere가 올바르게 구성되었는지 확인합니다. /etc/iam/pki가 존재하는지 확인합니다. 각 노드에는 고유한 인증서와 키가 있어야 합니다. 기본적으로 IAM Roles Anywhere를 자격 증명 공급자로 사용하는 경우 nodeadm은 인증서 위치 및 이름에 /etc/iam/pki/server.pem을 사용하고 프라이빗 키에 /etc/iam/pki/server.key를 사용합니다. `sudo mkdir -p /etc/iam/pki`를 사용하여 디렉터리에 인증서와 키를 배치하기 전에 디렉터리를 생성해야 할 수 있습니다. 아래 명령을 사용하여 인증서의 내용을 확인할 수 있습니다.

```
openssl x509 -text -noout -in server.pem
```

```
open /etc/iam/pki/server.pem: no such file or directory
could not parse PEM data
Command failed {"error": "... get identity: get credentials: failed to refresh cached
credentials, process provider error: error in credential_process: exit status 1"}
```

IAM Roles Anywhere에 **sts:AssumeRole** 수행 권한 없음

kubelet 로그에서 IAM Roles Anywhere를 사용할 때 sts:AssumeRole 작업에 대한 액세스 거부 문제가 표시되면 하이브리드 노드 IAM 역할의 신뢰 정책을 확인하여 IAM Roles Anywhere 서비스 보안 주체가 하이브리드 노드 IAM 역할을 수임할 수 있는지 확인합니다. 또한 하이브리드 노드 IAM 역할 신뢰 정책에서 트러스트 앵커 ARN이 올바르게 구성되었는지 그리고 하이브리드 노드 IAM 역할이 IAM Roles Anywhere 프로필에 추가되었는지 확인합니다.

```
could not get token: AccessDenied: User: ... is not authorized to perform:
sts:AssumeRole on resource: ...
```

IAM Roles Anywhere에 **roleSessionName** 설정 권한 없음

kubelet 로그에서 roleSessionName 설정을 위한 액세스 거부 문제가 표시되면 IAM Roles Anywhere 프로필에 대해 acceptRoleSessionName을 true로 설정했는지 확인합니다.


```
AccessDeniedException: Not authorized to set roleSessionName
```

운영 체제 문제 해결

RHEL

권한 또는 구독 관리자 등록 실패

nodeadm install을 실행 중이고 권한 등록 문제로 인해 하이브리드 노드 종속성을 설치하지 못한 경우 호스트에 Red Hat 사용자 이름과 암호를 올바르게 설정했는지 확인합니다.

```
This system is not registered with an entitlement server
```

Ubuntu

GLIBC를 찾을 수 없음

운영 체제에 Ubuntu를 사용하고 하이브리드 노드가 있는 자격 증명 공급자에 IAM Roles Anywhere를 사용하고 있는데 GLIBC에서 문제를 찾을 수 없는 경우 해당 종속성을 수동으로 설치하여 문제를 해결할 수 있습니다.

```
GLIBC_2.32 not found (required by /usr/local/bin/aws_signing_helper)
```

다음 명령을 실행하여 종속성을 설치합니다.

```
ldd --version
sudo apt update && apt install libc6
sudo apt install glibc-source
```

Bottlerocket

Bottlerocket 관리자 컨테이너를 활성화한 경우 SSH로 해당 컨테이너에 액세스하여 승격된 권한으로 고급 디버깅 및 문제 해결을 수행할 수 있습니다. 다음 섹션에는 Bottlerocket 호스트의 컨텍스트에서 실행해야 하는 명령이 포함되어 있습니다. 관리자 컨테이너에 있으면 shelvie를 실행하여 Bottlerocket 호스트에 전체 루트 셸을 가져올 수 있습니다.

```
shelvie
```

각 명령에 `sudo chroot /.bottlerocket/rootfs`로 접두사를 지정하여 관리자 컨테이너 셸에서 다음 섹션의 명령을 실행할 수도 있습니다.

```
sudo chroot /.bottlerocket/rootfs <command>
```

로그 수집에 logdog 사용

Bottlerocket은 문제 해결을 위해 로그 및 시스템 정보를 효율적으로 수집하는 logdog 유틸리티를 제공합니다.

```
logdog
```

logdog 유틸리티는 Bottlerocket 호스트의 다양한 위치에서 로그를 수집하여 tarball로 결합합니다. 기본적으로 tarball은 `/var/log/support/bottlerocket-logs.tar.gz`에서 생성되며 `/.bottlerocket/support/bottlerocket-logs.tar.gz`의 호스트 컨테이너에서 액세스할 수 있습니다.

journalctl을 사용하여 시스템 로그에 액세스

다음 명령을 사용하여 kubelet, containerd 등과 같은 다양한 시스템 서비스의 상태를 확인하고 해당 로그를 볼 수 있습니다. -f 플래그는 로그를 실시간으로 따릅니다.

kubelet 서비스 상태를 확인하고 kubelet 로그를 검색하기 위해 다음을 실행할 수 있습니다.

```
systemctl status kubelet  
journalctl -u kubelet -f
```

오케스트레이션된 containerd 인스턴스에 대한 containerd 서비스 상태를 확인하고 로그를 검색하기 위해 다음을 실행할 수 있습니다.

```
systemctl status containerd  
journalctl -u containerd -f
```

호스트 containerd 인스턴스에 대한 host-containerd 서비스 상태를 확인하고 로그를 검색하기 위해 다음을 실행할 수 있습니다.

```
systemctl status host-containerd  
journalctl -u host-containerd -f
```

부트스트랩 컨테이너 및 호스트 컨테이너에 대한 로그를 검색하기 위해 다음을 실행할 수 있습니다.

```
journalctl _COMM=host-ctr -f
```

클러스터에 애플리케이션 데이터 스토리지 사용

애플리케이션의 스토리지 요구 사항에 맞게 Amazon EKS와 함께 다양한 AWS 스토리지 서비스를 사용할 수 있습니다. AWS에서 지원하는 광범위한 컨테이너 스토리지 인터페이스(CSI) 드라이버를 통해 Amazon EKS에서 실행되는 애플리케이션의 스토리지 요구 사항에 따라 Amazon EBS, Amazon S3, Amazon EFS, Amazon FSX, Amazon File Cache를 쉽게 사용할 수 있습니다.

이 장에서는 Amazon EKS 클러스터를 위한 스토리지 옵션에 관해 다룹니다.

주제

- [Amazon EBS와 함께 Kubernetes 볼륨 스토리지 사용](#)
- [Amazon EFS와 함께 탄력적 파일 시스템 스토리지 사용](#)
- [Amazon FSx for Lustre와 함께 고성능 앱 스토리지 사용](#)
- [FSx for NetApp ONTAP과 함께 고성능 앱 스토리지 사용](#)
- [Amazon FSx for OpenZFS와 함께 데이터 스토리지 사용](#)
- [Amazon File Cache를 사용한 지연 시간 최소화](#)
- [Mountpoint for Amazon S3 CSI 드라이버를 통해 Amazon S3 객체에 액세스](#)
- [CSI 볼륨의 스냅샷 기능 활성화](#)

Amazon EBS와 함께 Kubernetes 볼륨 스토리지 사용

Note

신규: Amazon EKS Auto Mode는 블록 스토리지에 대한 일상적인 작업을 자동화합니다. [the section called “상태 저장 워크로드 배포”](#) 방법에 대해 알아보십시오.

[Amazon Elastic Block Store\(Amazon EBS\) CSI\(Container Storage Interface\) 드라이버](#)에서는 Amazon EBS 볼륨의 수명 주기를 사용자가 생성하는 Kubernetes 볼륨의 스토리지로 관리합니다. Amazon EBS CSI 드라이버는 Amazon EBS 볼륨을 Kubernetes 볼륨 유형인 일반 [임시 볼륨](#) 및 [영구 볼륨](#)에 사용할 수 있도록 만듭니다.

고려 사항

- Amazon EBS CSI 컨트롤러를 EKS Auto Mode 클러스터에 설치할 필요가 없습니다.

- Amazon EBS 볼륨을 Fargate 포드에 탑재할 수 없습니다.
- Amazon EBS CSI 컨트롤러는 Fargate 노드에서 실행할 수 있지만 Amazon EBS CSI 노드 DaemonSet은(는) Amazon EC2 인스턴스에서만 실행할 수 있습니다.
- Amazon EBS 볼륨 및 Amazon EBS CSI 드라이버는 Amazon EKS Hybrid Nodes와 호환되지 않습니다.
- 최신 추가 기능 버전과 하나의 이전 버전에 대한 지원이 제공됩니다. 최신 버전에서 발견된 버그나 취약성은 새 마이너 버전의 이전 릴리스로 백포트됩니다.
- EKS 자동 모드를 사용하려면 스토리지 클래스에서 `ebs.csi.eks.amazonaws.com`을 프로비저너로 사용해야 합니다. 표준 Amazon EBS CSI 드라이버(`ebs.csi.aws.com`)는 자체 볼륨을 별도로 관리합니다. EKS 자동 모드에서 기존 볼륨을 사용하려면 볼륨 스냅샷을 사용하여 자동 모드 프로비저너를 사용하는 스토리지 클래스로 마이그레이션합니다.

Important

Amazon EBS CSI 드라이버의 스냅샷 기능을 사용하려면 먼저 CSI 스냅샷 컨트롤러를 설치해야 합니다. 자세한 내용은 [the section called “CSI 스냅샷 컨트롤러”](#) 섹션을 참조하세요.

사전 조건

- 기존 클러스터가 있어야 합니다. 필요한 플랫폼 버전을 확인하려면 다음 명령을 실행합니다.

```
aws eks describe-addon-versions --addon-name aws-ebs-csi-driver
```

- EBS CSI 드라이버에는 AWS IAM 권한이 필요합니다.
 - AWS는 EKS Pod Identity 사용을 제안합니다. 자세한 내용은 [the section called “EKS Pod Identity 설정 개요”](#) 섹션을 참조하세요.
 - 서비스 계정에 대한 IAM 역할에 대한 자세한 내용은 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요.

1단계-IAM 역할 생성

Amazon EBS CSI 플러그 인이 사용자를 대신하여 AWS API를 직접 호출하려면 IAM 권한이 필요합니다. 이 단계를 수행하지 않는 경우 추가 기능을 설치하려고 시도하고 `kubectl describe pvc`를 실행하면 `failed to provision volume with StorageClass 오류`와 함께 `could not`

create volume in EC2: UnauthorizedOperation가 표시됩니다. 자세한 내용은 GitHub의 [Set up driver permission](#)을 참조하세요.

Note

IMDS에 대한 액세스 권한을 차단하지 않는 한 포드는 IAM 역할에 할당된 권한에 대한 액세스 권한이 있습니다. 자세한 내용은 [the section called “모범 사례”](#) 섹션을 참조하세요.

다음 절차에서는 IAM 역할을 생성하고 AWS 관리형 정책을 연결하는 방법을 보여줍니다. 이 절차를 구현하려면 다음 도구 중 하나를 사용할 수 있습니다.

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)
- [the section called “AWS CLI”](#)

Note

이 절차의 특정 단계는 드라이버를 Amazon EKS 추가 기능으로 사용하기 위해 작성되었습니다. 드라이버를 자체 관리형 추가 기능으로 드라이버를 사용하려면 여러 단계가 필요합니다. 자세한 내용은 GitHub의 [Set up driver permissions](#)를 참조하세요.

eksctl

1. IAM 역할을 생성하여 정책을 연결합니다. AWS은(는) AWS 관리형 정책을 유지 관리하거나 자체 사용자 지정 정책을 생성할 수 있습니다. IAM 역할을 생성하고 다음 명령을 사용하여 AWS 관리형 정책을 연결할 수 있습니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다. 이 명령은 IAM 역할을 생성하고 IAM 정책을 연결하는 AWS CloudFormation 스택을 배포합니다.

```
eksctl create iamserviceaccount \
    --name ebs-csi-controller-sa \
    --namespace kube-system \
    --cluster my-cluster \
    --role-name AmazonEKS_EBS_CSI_DriverRole \
    --role-only \
    --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
```

```
--approve
```

2. 사용자 지정 [KMS 키](#)를 사용하지 않는 경우 이 단계를 건너뛸 수 있습니다. Amazon EBS 볼륨의 암호화에 사용자 지정 KMS 키를 사용하는 경우 필요에 따라 IAM 역할을 사용자 지정합니다. 예를 들어, 다음을 수행합니다.

- a. 다음 코드를 복사하여 새로운 kms-key-for-encryption-on-ebs.json 파일에 붙여 넣습니다. *custom-key-arn*을 사용자 지정 [KMS 키 ARN](#)으로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": ["custom-key-arn"],
      "Condition": {
        "Bool": {
          "kms:GrantIsForAWSResource": "true"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": ["custom-key-arn"]
    }
  ]
}
```

- b. 정책을 생성합니다. *KMS_Key_For_Encryption_On_EBS_Policy*를 다른 이름으로 변경할 수 있습니다. 그러나 이렇게 하는 경우 이후 단계에서도 변경해야 합니다.

```
aws iam create-policy \  
  --policy-name KMS_Key_For_Encryption_On_EBS_Policy \  
  --policy-document file://kms-key-for-encryption-on-ebs.json
```

- c. 다음 명령을 사용하여 IAM 정책을 역할에 연결합니다. **111122223333**을 계정 ID로 바꿉니다.

```
aws iam attach-role-policy \  
  --policy-arn arn:aws:iam::111122223333:policy/  
KMS_Key_For_Encryption_On_EBS_Policy \  
  --role-name AmazonEKS_EBS_CSI_DriverRole
```

AWS Management Console

1. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 역할을 선택합니다.
3. 역할 페이지에서 역할 생성을 선택합니다.
4. 신뢰할 수 있는 엔터티 선택 페이지에서 다음을 수행합니다.
 - a. 신뢰할 수 있는 엔터티 유형 섹션에서 웹 자격 증명을 선택합니다.
 - b. 아이덴티티 제공업체(Identity provider)의 경우 클러스터에 대해 OpenID Connect 공급자 URL(OpenID Connect provider URL)을 선택합니다(Amazon EKS의 개요에 표시된 대로).
 - c. 대상에서 `sts.amazonaws.com`을 입력합니다.
 - d. 다음을 선택합니다.
5. 권한 추가 페이지에서 다음을 수행합니다.
 - a. 필터 정책 상자에 AmazonEBSCSIDriverPolicy를 입력합니다.
 - b. 검색에서 반환된 AmazonEBSCSIDriverPolicy 왼쪽에 있는 확인란을 선택합니다.
 - c. 다음을 선택합니다.
6. 이름, 검토 및 생성 페이지에서 다음을 수행합니다.
 - a. 역할 이름에 역할의 고유한 이름(예: **AmazonEKS_EBS_CSI_DriverRole**)을 입력합니다.
 - b. 태그 추가(선택사항)에서 태그를 키-값 페어로 연결하여 메타데이터를 역할에 추가합니다. IAM에서 태그 사용에 대한 자세한 내용을 알아보려면 IAM 사용 설명서의 [IAM 리소스에 태깅](#)을 참조하세요.
 - c. 역할 생성을 선택합니다.
7. 역할을 생성한 후 편집할 수 있도록 콘솔에서 이 역할을 선택하여 엽니다.

8. 신뢰 관계 탭을 선택한 후 신뢰 정책 편집을 선택합니다.
9. 다음과 비슷한 줄을 찾습니다.

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud":
  "sts.amazonaws.com"
```

이전 줄의 끝에 쉼표를 추가한 다음 이전 줄 뒤에 다음 줄을 추가합니다. *region-code*를 클러스터가 있는 AWS 리전으로 바꿉니다. *EXAMPLED539D4633E53DE1B71EXAMPLE*을 클러스터의 OIDC 공급자 ID로 바꿉니다.

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":
  "system:serviceaccount:kube-system:ebs-csi-controller-sa"
```

10. 신뢰 정책 업데이트를 선택하여 종료합니다.
11. Amazon EBS 볼륨의 암호화에 사용자 지정 [KMS 키](#)를 사용하는 경우 필요에 따라 IAM 역할을 사용자 지정합니다. 예를 들어, 다음을 수행합니다.
 - a. 왼쪽 탐색 창에서 정책을 선택합니다.
 - b. 정책 페이지에서 정책 생성을 선택합니다.
 - c. 정책 생성 페이지에서 JSON 탭을 선택합니다.
 - d. 다음 코드를 복사하여 편집기에 붙여 넣어 *custom-key-arn*을 사용자 지정 [KMS 키 ARN](#)으로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": ["custom-key-arn"],
      "Condition": {
        "Bool": {
          "kms:GrantIsForAWSResource": "true"
        }
      }
    }
  ],
}
```



```
{
  "Effect": "Allow",
  "Action": [
    "kms:Encrypt",
    "kms:Decrypt",
    "kms:ReEncrypt*",
    "kms:GenerateDataKey*",
    "kms:DescribeKey"
  ],
  "Resource": ["custom-key-arn"]
}
```

- e. 다음: 태그를 선택합니다.
- f. 태그 추가(선택사항) 페이지에서 다음: 검토를 선택합니다.
- g. 이름에 정책의 고유한 이름(예: *KMS_Key_For_Encryption_On_EBS_Policy*)을 입력합니다.
- h. 정책 생성을 선택합니다.
- i. 왼쪽 탐색 창에서 역할을 선택합니다.
- j. 콘솔에서 *AmazonEKS_EBS_CSI_DriverRole* 을 선택하여 열고 편집합니다.
- k. 권한 추가 드롭다운 목록에서 정책 연결을 선택합니다.
- l. 정책 필터링 상자에 *KMS_Key_For_Encryption_On_EBS_Policy*를 입력합니다.
- m. 검색에서 반환된 *KMS_Key_For_Encryption_On_EBS_Policy* 왼쪽의 확인란을 선택합니다.
- n. 정책 연결을 선택합니다.

AWS CLI

1. 클러스터의 OIDC 공급자 URL을 확인합니다. *<cluster-name>*을 클러스터 이름으로 바꿉니다. 명령의 출력이 None인 경우 사전 요구 사항을 검토합니다.

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer" --output text
```

예제 출력은 다음과 같습니다.

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

2. IAM 역할을 생성하여 AssumeRoleWithWebIdentity 작업을 부여합니다.

- a. 다음 콘텐츠를 `aws-ebs-csi-driver-trust-policy.json`이라는 파일에 복사합니다. **111122223333**을 계정 ID로 바꿉니다. **EXAMPLED539D4633E53DE1B71EXAMPLE** 및 **region-code**를 이전 단계에서 반환된 값으로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
          "oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:ebs-csi-controller-sa"
        }
      }
    }
  ]
}
```

- b. 역할을 생성합니다. **AmazonEKS_EBS_CSI_DriverRole**을 다른 이름으로 변경할 수 있습니다. 변경하는 경우 이후 단계에서 변경해야 합니다.

```
aws iam create-role \
  --role-name AmazonEKS_EBS_CSI_DriverRole \
  --assume-role-policy-document file://aws-ebs-csi-driver-trust-policy.json
```

3. 정책을 연결합니다. AWS은(는) AWS 관리형 정책을 유지 관리하거나 자체 사용자 지정 정책을 생성할 수 있습니다. 다음 명령으로 AWS 관리형 정책을 역할에 연결합니다.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
  --role-name AmazonEKS_EBS_CSI_DriverRole
```

4. Amazon EBS 볼륨의 암호화에 사용자 지정 [KMS 키](#)를 사용하는 경우 필요에 따라 IAM 역할을 사용자 지정합니다. 예를 들어, 다음을 수행합니다.

- a. 다음 코드를 복사하여 새로운 kms-key-for-encryption-on-ebs.json 파일에 붙여 넣습니다. *custom-key-arn*을 사용자 지정 [KMS 키 ARN](#)으로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:CreateGrant",
        "kms:ListGrants",
        "kms:RevokeGrant"
      ],
      "Resource": ["custom-key-arn"],
      "Condition": {
        "Bool": {
          "kms:GrantIsForAWSResource": "true"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
      ],
      "Resource": ["custom-key-arn"]
    }
  ]
}
```

- b. 정책을 생성합니다. *KMS_Key_For_Encryption_On_EBS_Policy*를 다른 이름으로 변경할 수 있습니다. 그러나 이렇게 하는 경우 이후 단계에서도 변경해야 합니다.

```
aws iam create-policy \
  --policy-name KMS_Key_For_Encryption_On_EBS_Policy \
  --policy-document file://kms-key-for-encryption-on-ebs.json
```

- c. 다음 명령을 사용하여 IAM 정책을 역할에 연결합니다. **111122223333**을 계정 ID로 바꿉니다.

```
aws iam attach-role-policy \
    --policy-arn arn:aws:iam::111122223333:policy/
KMS_Key_For_Encryption_On_EBS_Policy \
    --role-name AmazonEKS_EBS_CSI_DriverRole
```

이제 Amazon EBS CSI 드라이버 IAM 역할을 생성했으므로 다음 섹션을 계속 진행할 수 있습니다. 이 IAM 역할을 사용하여 추가 기능을 배포하면 ebs-csi-controller-sa라는 서비스 계정을 생성하고 사용하도록 구성됩니다. 이 서비스 계정은 필요한 Kubernetes 권한이 할당된 Kubernetes clusterrole에 바인딩되어 있습니다.

2단계: Amazon EBS CSI 드라이버 받기

보안을 강화하고 작업량을 줄이려면 Amazon EKS 추가 기능을 통해 Amazon EBS CSI 드라이버를 설치하는 것이 좋습니다. Amazon EKS 추가 기능을 클러스터에 추가하려면 [the section called “추가 기능 생성”](#) 섹션을 참조하세요. Amazon EKS 추가 기능에 대한 자세한 내용은 [the section called “Amazon EKS 추가 기능”](#) 섹션을 참조하세요.

Important

Amazon EBS 드라이버를 Amazon EKS 추가 기능으로 추가하기 전에 클러스터에 자체 관리형 버전의 드라이버가 설치되어 있지 않은지 확인하세요. 설치되어 있다면 GitHub의 [Uninstalling a self-managed Amazon EBS CSI driver](#)를 참조하세요.

대신 Amazon EBS CSI 드라이버의 자체 관리형 설치를 원하는 경우 GitHub의 [Installation](#)을 참조하세요.

3단계: 샘플 애플리케이션 배포

다양한 샘플 앱을 배포하고 필요에 따라 수정할 수 있습니다. 자세한 내용은 GitHub의 [Kubernetes Examples](#)를 참조하세요.

Amazon EFS와 함께 탄력적 파일 시스템 스토리지 사용

[Amazon Elastic File System](#)(Amazon EFS)은 완전히 탄력적인 서버리스 파일 스토리지를 제공하므로 스토리지 용량과 성능을 프로비저닝하거나 관리하지 않고도 파일 데이터를 공유할 수 있습니다.

[Amazon EBS 컨테이너 스토리지 인터페이스\(CSI\) 드라이버](#)는 AWS에서는 Kubernetes 클러스터가 Amazon EFS 파일 시스템의 수명 주기를 관리할 수 있게 해주는 CSI 인터페이스를 제공합니다. 이 주제에서는 Amazon EFS CSI 드라이버를 Amazon EKS 클러스터에 배포하는 방법을 설명합니다.

고려 사항

- Amazon EFS CSI 드라이버는 Windows 기반 컨테이너 이미지와 호환되지 않습니다.
- Fargate 노드에는 영구 볼륨에 대해 [동적 프로비저닝](#)을 사용할 수 없지만 [정적 프로비저닝](#)은 사용할 수 있습니다.
- [동적 프로비저닝](#)은 [1.2](#) 이상의 드라이버가 필요합니다. 모든 지원되는 Amazon EKS 클러스터 버전에서 드라이버 버전 1.1을 사용하여 영구 볼륨에 대해 [정적 프로비저닝](#)을 사용할 수 있습니다. ([the section called “Kubernetes 버전”](#) 참고)
- 이 드라이버의 버전 [1.3.2](#) 이상은 Amazon EC2 Graviton 기반 인스턴스를 포함하여 Arm64 아키텍처를 지원합니다.
- 이 드라이버의 버전 [1.4.2](#) 이상은 파일 시스템 탑재에 FIPS 사용을 지원합니다.
- Amazon EFS의 리소스 할당량을 기록해 둡니다. 예를 들어, 각 Amazon EFS 파일 시스템에 대해 생성할 수 있는 액세스 포인트 할당량은 1,000개입니다. 자세한 내용은 [변경할 수 없는 Amazon EFS 리소스 할당량](#)을 참조하세요.
- 버전 [2.0.0](#)부터 이 드라이버는 TLS 연결에 stunnel을 사용하던 것을 efs-proxy로 전환했습니다. efs-proxy를 사용하면 실행 중인 노드의 코어 수에 1을 더한 것과 같은 수의 스레드가 열립니다.
- Amazon EFS CSI 드라이버는 Amazon EKS Hybrid Nodes와 호환되지 않습니다.

사전 조건

- Amazon EFS CSI 드라이버에는 AWS Identity and Access Management(IAM) 권한이 필요합니다.
 - AWS는 EKS Pod Identity 사용을 제안합니다. 자세한 내용은 [the section called “EKS Pod Identity 설정 개요”](#) 섹션을 참조하세요.
 - 서비스 계정의 IAM 역할 및 클러스터의 IAM OpenID Connect(OIDC) 공급자 설정에 대한 자세한 내용은 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요.
- 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다.

다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.

- kubectl 명령줄 도구는 장치 또는 AWS CloudShell에 설치됩니다. 버전은 클러스터의 Kubernetes 버전과 동일하거나 최대 하나 이전 또는 이후의 마이너 버전일 수 있습니다. 예를 들어, 클러스터 버전이 1.29인 경우 kubectl 버전 1.28, 1.29 또는 1.30를 함께 사용할 수 있습니다. kubectl을 설치하거나 업그레이드하려면 [the section called “kubectl 및 eksctl 설정”](#) 부분을 참조하세요.

Note

Fargate에서 실행되는 포드는 수동 드라이버 설치 단계 없이 Amazon EFS 파일 시스템을 자동으로 탑재합니다.

1단계-IAM 역할 생성

Amazon EFS CSI 드라이버가 파일 시스템과 상호 작용하려면 IAM 권한이 필요합니다. IAM 역할을 생성하고 필요한 AWS 관리형 정책을 연결합니다. 이 절차를 구현하려면 다음 도구 중 하나를 사용할 수 있습니다.

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)
- [the section called “AWS CLI”](#)

Note

이 절차의 특정 단계는 드라이버를 Amazon EKS 추가 기능으로 사용하기 위해 작성되었습니다. 자체 관리형 설치에 대한 자세한 내용은 GitHub의 [Set up driver permission](#)을 참조하세요.

eksctl

Pod Identity를 사용하는 경우

다음 명령을 실행하여 eksctl로 IAM 역할과 Pod Identity 연결을 생성할 수 있습니다. my-cluster을 클러스터 이름으로 교체합니다. 또한 AmazonEKS_EFS_CSI_DriverRole을 다른 이름으로 바꿀 수 있습니다.

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
eksctl create podidentityassociation \
  --service-account-name efs-csi-controller-sa \
  --namespace kube-system \
  --cluster $cluster_name \
  --role-name $role_name \
  --permission-policy-arns arn:aws:iam::aws:policy/service-role/
AmazonEFSCSIDriverPolicy \
  --approve
```

서비스 계정에 IAM 역할을 사용하는 경우

다음 명령을 실행하여 eksctl로 IAM 역할을 만들 수 있습니다. my-cluster를 클러스터 이름으로 교체합니다. 또한 AmazonEKS_EFS_CSI_DriverRole을 다른 이름으로 바꿀 수 있습니다.

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
eksctl create iamserviceaccount \
  --name efs-csi-controller-sa \
  --namespace kube-system \
  --cluster $cluster_name \
  --role-name $role_name \
  --role-only \
  --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
  --approve
TRUST_POLICY=$(aws iam get-role --output json --role-name $role_name --query
  'Role.AssumeRolePolicyDocument' | \
  sed -e 's/efs-csi-controller-sa/efs-csi-*/' -e 's/StringEquals/StringLike/')
aws iam update-assume-role-policy --role-name $role_name --policy-document
"$TRUST_POLICY"
```

AWS Management Console

다음을 실행하여 AWS Management Console를 사용하여 IAM 역할을 만듭니다.

1. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 역할을 선택합니다.
3. 역할 페이지에서 역할 생성을 선택합니다.
4. 신뢰할 수 있는 엔터티 선택 페이지에서 다음을 수행합니다.

- a. EKS Pod Identity를 사용하는 경우
 - i. 신뢰할 수 있는 엔터티 유형 섹션에서 AWS 서비스를 선택합니다.
 - ii. 서비스 또는 사용 사례 드롭다운에서 EKS를 선택합니다.
 - iii. 사용 사례 섹션에서 EKS - Pod Identity를 선택합니다.
 - iv. 다음을 선택합니다.
- b. 서비스 계정에 IAM 역할을 사용하는 경우
 - i. 신뢰할 수 있는 엔터티 유형 섹션에서 웹 자격 증명을 선택합니다.
 - ii. 아이덴티티 제공업체(Identity provider)의 경우 클러스터에 대해 OpenID Connect 공급자 URL(OpenID Connect provider URL)을 선택합니다(Amazon EKS의 개요에 표시된 대로).
 - iii. 대상에서 sts.amazonaws.com을 입력합니다.
 - iv. 다음을 선택합니다.
5. 권한 추가 페이지에서 다음을 수행합니다.
 - a. 필터 정책 상자에 AmazonEFSCSIDriverPolicy를 입력합니다.
 - b. 검색에서 반환된 AmazonEFSCSIDriverPolicy 왼쪽에 있는 확인란을 선택합니다.
 - c. 다음을 선택합니다.
6. 이름, 검토 및 생성 페이지에서 다음을 수행합니다.
 - a. 역할 이름에 역할의 고유한 이름(예: AmazonEKS_EFS_CSI_DriverRole)을 입력합니다.
 - b. 태그 추가(선택사항)에서 태그를 키-값 페어로 연결하여 메타데이터를 역할에 추가합니다. IAM에서 태그 사용에 대한 자세한 내용을 알아보려면 IAM 사용 설명서의 [IAM 리소스에 태깅](#)을 참조하세요.
 - c. 역할 생성을 선택합니다.
7. 역할이 생성된 후
 - a. EKS Pod Identity를 사용하는 경우
 - i. [Amazon EKS 콘솔](#)을 엽니다.
 - ii. 왼쪽 탐색 창에서 클러스터를 선택한 다음 EKS Pod Identity 연결을 구성할 클러스터의 이름을 선택합니다.
 - iii. 액세스 탭을 선택합니다.
 - iv. Pod Identity 연결에서 생성을 선택합니다.
 - v. IAM 역할 드롭다운을 선택하고 새로 생성된 역할을 선택합니다.
 - vi. [Kubernetes 네임스페이스 필드](#)를 선택하고 kube-system을 입력합니다.
 - vii. Kubernetes 서비스 계정 필드를 선택하고 efs-csi-controller-sa를 입력합니다.

viii. 생성(Create)을 선택합니다.

ix. Pod Identity 연결 생성에 대한 자세한 내용은 [the section called “pod identity 연결 생성\(AWS 콘솔\)”](#) 섹션을 참조하세요.

b. 서비스 계정에 IAM 역할을 사용하는 경우

i. 역할을 선택하여 편집을 위해 엽니다.

ii. 신뢰 관계 탭을 선택한 후 신뢰 정책 편집을 선택합니다.

iii. 다음과 비슷한 줄을 찾습니다.

```
"oidc.eks.region-code.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>:aud":
"sts.amazonaws.com"
```

이전 줄 위에 다음 줄을 추가합니다. <region-code>를 클러스터가 있는 AWS 리전으로 바꿉니다. <EXAMPLED539D4633E53DE1B71EXAMPLE>을 클러스터의 OIDC 공급자 ID로 바꿉니다.

```
"oidc.eks.<region-code>.amazonaws.com/id/
<EXAMPLED539D4633E53DE1B71EXAMPLE>:sub": "system:serviceaccount:kube-system:efs-
csi-*",
```

iv. Condition 운영자를 "StringEquals"에서 "StringLike"(으)로 수정합니다.

v. 신뢰 정책 업데이트를 선택하여 종료합니다.

AWS CLI

다음 명령을 실행하여 AWS CLI로 IAM 역할을 만들 수 있습니다.

Pod Identity를 사용하는 경우

1. pods.eks.amazonaws.com 서비스에 AssumeRole 및 TagSession 작업을 부여하는 IAM 역할을 생성합니다.

a. 다음 콘텐츠를 aws-efs-csi-driver-trust-policy-pod-identity.json라는 파일에 복사합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```

        "Sid": "AllowEksAuthToAssumeRoleForPodIdentity",
        "Effect": "Allow",
        "Principal": {
            "Service": "pods.eks.amazonaws.com"
        },
        "Action": [
            "sts:AssumeRole",
            "sts:TagSession"
        ]
    }
]
}

```

- b. 역할을 생성합니다. `my-cluster`을 클러스터 이름으로 교체합니다. 또한 `AmazonEKS_EFS_CSI_DriverRole`을 다른 이름으로 바꿀 수 있습니다.

```

export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
aws iam create-role \
    --role-name $role_name \
    --assume-role-policy-document file://"aws-efs-csi-driver-trust-policy-pod-identity.json"

```

2. 다음 명령어로 필요한 AWS 관리형 정책을 역할에 연결합니다.

```

aws iam attach-role-policy \
    --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
    --role-name $role_name

```

3. 다음 명령을 실행하여 Pod Identity 연결을 생성합니다.

`arn:aws:iam::<111122223333>:role/my-role`을 이전 단계에서 생성한 역할로 바꿉니다.

```

aws eks create-pod-identity-association --cluster-name $cluster_name --role-arn {arn-aws}iam::<111122223333>:role/my-role --namespace kube-system --service-account efs-csi-controller-sa

```

4. Pod Identity 연결 생성에 대한 자세한 내용은 [the section called “pod identity 연결 생성\(AWS 콘솔\)”](#) 섹션을 참조하세요.

서비스 계정에 IAM 역할을 사용하는 경우

1. 클러스터의 OIDC 공급자 URL을 확인합니다. `my-cluster`을 클러스터 이름으로 교체합니다. 또한 `AmazonEKS_EFS_CSI_DriverRole`을 다른 이름으로 바꿀 수 있습니다.

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
aws eks describe-cluster --name $cluster_name --query "cluster.identity.oidc.issuer"
--output text
```

예제 출력은 다음과 같습니다.

```
https://oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>
```

명령의 출력이 `None`인 경우 사전 요구 사항을 검토합니다.

2. `AssumeRoleWithWebIdentity` 작업을 부여하는 IAM 역할을 생성합니다.
 - a. 다음 콘텐츠를 `aws-efs-csi-driver-trust-policy.json`라는 파일에 복사합니다. `<111122223333>`을 계정 ID로 바꿉니다. `<EXAMPLED539D4633E53DE1B71EXAMPLE>` 및 `<region-code>`를 이전 단계에서 반환된 값으로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::<111122223333>:oidc-provider/oidc.eks.<region-code>.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringLike": {
          "oidc.eks.region-code.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>:sub": "system:serviceaccount:kube-system:efs-csi-*",
          "oidc.eks.region-code.amazonaws.com/id/<EXAMPLED539D4633E53DE1B71EXAMPLE>:aud": "sts.amazonaws.com"
        }
      }
    }
  ]
}
```

```
]
}
```

b. 역할을 생성합니다.

```
aws iam create-role \
  --role-name $role_name \
  --assume-role-policy-document file:///aws-efs-csi-driver-trust-policy.json"
```

3. 다음 명령어로 필요한 AWS 관리형 정책을 역할에 연결합니다.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
  --role-name $role_name
```

2단계: Amazon EFS CSI 드라이버 받기

Amazon EKS 추가 기능을 통해 Amazon EFS CSI 드라이버를 설치하는 것이 좋습니다. Amazon EKS 추가 기능을 클러스터에 추가하려면 [the section called “추가 기능 생성”](#) 섹션을 참조하세요. Amazon EKS 추가 기능에 대한 자세한 내용은 [the section called “Amazon EKS 추가 기능”](#) 섹션을 참조하세요. Amazon EKS 추가 기능을 사용할 수 없는 경우, 사용할 수 없는 이유에 대한 문제를 [컨테이너 로드맵 GitHub 리포지토리에](#) 제출하는 것이 좋습니다.

Important

Amazon EFS 드라이버를 Amazon EKS 추가 기능으로 추가하기 전에 클러스터에 자체 관리형 버전의 드라이버가 설치되어 있지 않은지 확인하세요. 그러한 경우 GitHub에서 [Amazon EFS CSI 드라이버 제거](#)를 참조하세요.

대신 Amazon EFS CSI 드라이버의 자체 관리형 설치를 원하는 경우 GitHub의 [Installation](#)을 참조하세요.

3단계: Amazon EFS 파일 시스템 생성

Amazon EFS 파일 시스템을 생성하려면 GitHub의 [Create an Amazon EFS file system for Amazon EKS](#)를 참조하세요.

4단계: 샘플 애플리케이션 배포

다양한 샘플 앱을 배포하고 필요에 따라 수정할 수 있습니다. 자세한 내용은 GitHub의 [Examples](#)를 참조하세요.

Amazon FSx for Lustre와 함께 고성능 앱 스토리지 사용

[Amazon FSx for Lustre 컨테이너 스토리지\(CSI\) 드라이버](#)에서는 Amazon EKS 클러스터가 Amazon FSx for Lustre 파일 시스템의 수명 주기를 관리할 수 있게 해주는 CSI 인터페이스를 제공합니다. 자세한 내용을 알아보려면 [Amazon FSx for Lustre 사용 설명서](#)를 참조하세요.

Amazon FSx for Lustre CSI 드라이버를 Amazon EKS 클러스터에 배포하고 제대로 작동하는지 확인하는 방법에 대한 자세한 내용은 [the section called “드라이버 배포”](#) 섹션을 참조하세요.

FSx for Lustre CSI 드라이버 배포

이 주제에서는 [FSx for Lustre CSI 드라이버](#)를 Amazon EKS 클러스터에 배포하고 제대로 작동하는지 확인하는 방법에 대해 설명합니다. 드라이버는 최신 버전을 사용하는 것이 좋습니다. 사용 가능한 버전은 GitHub의 [CSI Specification Compatibility Matrix](#)를 참조하세요.

Note

드라이버는 Fargate 또는 Amazon EKS Hybrid Nodes에서 지원되지 않습니다.

사용할 수 있는 파라미터에 대한 자세한 설명과 드라이버의 기능을 보여주는 전체 예제는 GitHub의 [FSx for Lustre Container Storage Interface \(CSI\) driver](#) 프로젝트를 참조하세요.

사전 조건

- 기존 클러스터가 있어야 합니다.
- Amazon FSx CSI 드라이버 EKS 추가 기능을 사용하려면 인증을 위해 EKS Pod Identity 에이전트가 필요합니다. 이 구성 요소가 없으면 Amazon EKS Pod Identity agent is not installed in the cluster 오류와 함께 추가 기능이 실패하여 볼륨 작업을 수행하지 못합니다. FSx CSI 드라이버 추가 기능을 배포하기 전 또는 후에 Pod Identity 에이전트를 설치합니다. 자세한 내용은 [the section called “에이전트 설정”](#) 섹션을 참조하세요.
- 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지

지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.

- 장치에 설치된 eksctl 명령줄 도구의 버전 0.210.0 이상 또는 AWS CloudShell이 필요합니다. eksctl을 설치 또는 업그레이드하려면 eksctl 설명서에서 [설치](#)를 참조하세요.
- kubectl 명령줄 도구는 장치 또는 AWS CloudShell에 설치됩니다. 버전은 클러스터의 Kubernetes 버전과 동일하거나 최대 하나 이전 또는 이후의 마이너 버전일 수 있습니다. 예를 들어, 클러스터 버전이 1.29인 경우 kubectl 버전 1.28, 1.29 또는 1.30를 함께 사용할 수 있습니다. kubectl을 설치하거나 업그레이드하려면 [the section called “kubectl 및 eksctl 설정”](#) 부분을 참조하세요.

1단계-IAM 역할 생성

Amazon FSx CSI 플러그인이 사용자를 대신하여 AWS API를 직접적으로 호출하려면 IAM 권한이 필요합니다.

Note

IMDS에 대한 액세스 권한을 차단하지 않는 한 포드는 IAM 역할에 할당된 권한에 대한 액세스 권한이 있습니다. 자세한 내용은 [the section called “모범 사례”](#) 섹션을 참조하세요.

다음 절차에서는 IAM 역할을 생성하고 AWS 관리형 정책을 연결하는 방법을 보여줍니다.

1. 다음 명령을 사용하여 IAM 역할을 생성하고 AWS 관리형 정책을 연결합니다. my-cluster를 사용하려는 클러스터의 이름으로 바꿉니다. 이 명령은 IAM 역할을 생성하고 IAM 정책을 연결하는 AWS CloudFormation 스택을 배포합니다.

```
eksctl create iamserviceaccount \
  --name fsx-csi-controller-sa \
  --namespace kube-system \
  --cluster my-cluster \
  --role-name AmazonEKS_FSx_CSI_DriverRole \
  --role-only \
  --attach-policy-arn arn:aws:iam::aws:policy/AmazonFSxFullAccess \
  --approve
```

서비스 계정이 생성되면 여러 줄의 출력이 표시됩니다. 출력의 마지막 줄은 다음과 유사합니다.

```
[#] 1 task: {
    2 sequential sub-tasks: {
        create IAM role for serviceaccount "kube-system/fsx-csi-controller-sa",
        create serviceaccount "kube-system/fsx-csi-controller-sa",
    } }
[#] building iamserviceaccount stack "eksctl-my-cluster-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa"
[#] deploying stack "eksctl-my-cluster-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa"
[#] waiting for CloudFormation stack "eksctl-my-cluster-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa"
[#] created serviceaccount "kube-system/fsx-csi-controller-sa"
```

배포된 AWS CloudFormation 스택의 이름을 기록합니다. 이전 예제 출력에서 스택의 이름이 `eksctl-my-cluster-addon-iamserviceaccount-kube-system-fsx-csi-controller-sa`로 지정되어 있습니다.

이제 Amazon FSx CSI 드라이버 IAM 역할을 생성했으므로 다음 섹션을 계속 진행할 수 있습니다. 이 IAM 역할을 사용하여 추가 기능을 배포하면 `fsx-csi-controller-sa`라는 서비스 계정을 생성하고 사용하도록 구성됩니다. 이 서비스 계정은 필요한 Kubernetes 권한이 할당된 Kubernetes `clusterrole`에 바인딩되어 있습니다.

2단계: Amazon FSx CSI 드라이버 설치

보안을 강화하고 작업량을 줄이려면 Amazon FSx 추가 기능을 통해 Amazon EBS CSI 드라이버를 설치하는 것이 좋습니다. Amazon EKS 추가 기능을 클러스터에 추가하려면 [the section called “추가 기능 생성”](#) 섹션을 참조하세요. Amazon EKS 추가 기능에 대한 자세한 내용은 [the section called “Amazon EKS 추가 기능”](#) 섹션을 참조하세요.

Important

클러스터에 기존 Amazon FSx CSI 드라이버를 설치하면 추가 기능 설치에 실패할 수 있습니다. 비 EKS FSx CSI 드라이버가 있는 동안 Amazon EKS 추가 기능 버전을 설치하려고 하면 리소스 충돌로 인해 설치가 실패합니다. 설치 중 OVERWRITE 플래그를 사용하여 이 문제를 해결합니다.

```
aws eks create-addon --addon-name aws-fsx-csi-driver --cluster-name my-cluster
--resolve-conflicts OVERWRITE
```

대신 Amazon FSx CSI 드라이버의 자체 관리형 설치를 원하는 경우 GitHub의 [Installation](#)을 참조하세요.

3단계: 스토리지 클래스, 영구 볼륨 클레임 및 샘플 앱 배포

이 절차에서는 [FSx for Lustre Container Storage Interface\(CSI\) 드라이버](#) GitHub 리포지토리를 사용하여 동적으로 프로비저닝된 FSx for Lustre 볼륨을 사용합니다.

1. 클러스터의 보안 그룹을 기록합니다. 이는 네트워킹 섹션 아래의 AWS Management Console에서 또는 다음을 AWS CLI 명령을 사용하여 볼 수 있습니다. my-cluster를 사용하려는 클러스터의 이름으로 바꿉니다.

```
aws eks describe-cluster --name my-cluster --query
cluster.resourcesVpcConfig.clusterSecurityGroupId
```

2. Amazon FSx for Lustre 사용 설명서의 [Amazon VPC 보안 그룹](#)에 표시된 기준에 따라 Amazon FSx 파일 시스템에 대한 보안 그룹을 생성합니다. VPC의 경우 네트워킹 섹션 아래 표시된 대로 클러스터의 VPC를 선택합니다. “Lustre 클라이언트와 연결된 보안 그룹”의 경우 클러스터 보안 그룹을 사용합니다. 아웃바운드 규칙을 그대로 두어 모든 트래픽을 허용할 수 있습니다.
3. 다음 명령을 사용하여 스토리지 클래스 매니페스트를 다운로드합니다.

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/
examples/kubernetes/dynamic_provisioning/specs/storageclass.yaml
```

4. storageclass.yaml 파일의 파라미터 섹션을 편집합니다. 모든 예제 값을 자신의 값으로 바꾸세요.

```
parameters:
  subnetId: subnet-0eabfaa81fb22bcaf
  securityGroupIds: sg-068000ccf82dfba88
  deploymentType: PERSISTENT_1
  automaticBackupRetentionDays: "1"
  dailyAutomaticBackupStartTime: "00:00"
  copyTagsToBackups: "true"
  perUnitStorageThroughput: "200"
```



```
dataCompressionType: "NONE"
weeklyMaintenanceStartTime: "7:09:00"
fileSystemTypeVersion: "2.12"
```

- **subnetId** - Amazon FSx for Lustre 파일 시스템을 생성해야 하는 서브넷 ID입니다. Amazon FSx for Lustre는 일부 가용 영역에서 지원되지 않습니다. <https://console.aws.amazon.com/fsx/>에서 Amazon FSx for Lustre 콘솔을 열어 사용하려는 서브넷이 지원되는 가용 영역에 있는지 확인합니다. 서브넷은 노드를 포함할 수도 있고 다른 서브넷이나 VPC일 수도 있습니다.
- 컴퓨팅 섹션에서 노드 그룹을 선택하여 AWS Management Console에서 노드 서브넷을 확인할 수 있습니다.
- 지정한 서브넷이 노드가 있는 서브넷과 다른 경우 VPC가 [연결](#)되어 있어야 하며 보안 그룹에 필요한 포트가 열려 있는지 확인해야 합니다.
- **securityGroupIds** - 파일 시스템에 대해 생성된 보안 그룹의 ID.
- **deploymentType** (선택사항) - 파일 시스템 배포 유형입니다. 유효한 값은 SCRATCH_1, SCRATCH_2, PERSISTENT_1 및 PERSISTENT_2입니다. 배포 유형에 대한 자세한 내용은 [Amazon FSx for Lustre 파일 시스템 생성](#)을 참조하세요.
- 기타 파라미터(선택 사항) - 기타 파라미터에 대한 자세한 내용을 알아보려면 GitHub의 [스토리지 클래스 편집](#)을 참조하세요.

5. 스토리지 클래스 매니페스트를 생성합니다.

```
kubectl apply -f storageclass.yaml
```

예제 출력은 다음과 같습니다.

```
storageclass.storage.k8s.io/fsx-sc created
```

6. 영구 볼륨 클레임 매니페스트를 다운로드합니다.

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/examples/kubernetes/dynamic_provisioning/specs/claim.yaml
```

7. (선택사항) claim.yaml 파일을 편집합니다. 스토리지 요구 사항과 이전 단계에서 선택한 deploymentType에 따라 1200Gi를 다음 중분 값 중 하나로 변경합니다.

```
storage: 1200Gi
```

- **SCRATCH_2 및 PERSISTENT - 1.2 TiB, 2.4 TiB 또는 2.4TiB 대비 2.4TiB씩 증가합니다.**

- SCRATCH_1 - 1.2 TiB, 2.4 TiB, 3.6 TiB 또는 3.6TiB 대비 3.6TiB씩 증가합니다.

8. 영구 볼륨 클레임을 생성합니다.

```
kubectl apply -f claim.yaml
```

예제 출력은 다음과 같습니다.

```
persistentvolumeclaim/fsx-claim created
```

9. 파일 시스템이 프로비저닝되었는지 확인합니다.

```
kubectl describe pvc
```

예제 출력은 다음과 같습니다.

```
Name:          fsx-claim
Namespace:     default
StorageClass:  fsx-sc
Status:        Bound
[...]
```

Note

Status는 Bound로 변경되기 전에 5~10분 동안 Pending으로 표시될 수 있습니다. Status가 Bound가 될 때까지 다음 단계를 계속하지 마세요. Status에 10분 이상 동안 Pending이 표시되는 경우 모든 문제를 해결하기 위한 참조로 Events의 경고 메시지를 사용합니다.

10. 샘플 애플리케이션을 배포합니다.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-fsx-csi-driver/master/examples/kubernetes/dynamic_provisioning/specs/pod.yaml
```

11. 샘플 애플리케이션이 실행 중인지 확인합니다.

```
kubectl get pods
```

예제 출력은 다음과 같습니다.

NAME	READY	STATUS	RESTARTS	AGE
fsx-app	1/1	Running	0	8s

12.파일 시스템이 애플리케이션에 의해 올바르게 탑재되었는지 확인합니다.

```
kubectl exec -ti fsx-app -- df -h
```

예제 출력은 다음과 같습니다.

```
Filesystem                Size      Used Avail Use% Mounted on
overlay                  80G      4.0G   77G   5% /
tmpfs                    64M         0   64M   0% /dev
tmpfs                    3.8G         0   3.8G   0% /sys/fs/cgroup
192.0.2.0@tcp:/abcdef01   1.1T      7.8M   1.1T   1% /data
/dev/nvme0n1p1           80G      4.0G   77G   5% /etc/hosts
shm                      64M         0   64M   0% /dev/shm
tmpfs                    6.9G      12K   6.9G   1% /run/secrets/kubernetes.io/
serviceaccount
tmpfs                    3.8G         0   3.8G   0% /proc/acpi
tmpfs                    3.8G         0   3.8G   0% /sys/firmware
```

13.샘플 앱에서 데이터가 FSx for Lustre 파일 시스템에 작성되었는지 확인합니다.

```
kubectl exec -it fsx-app -- ls /data
```

예제 출력은 다음과 같습니다.

```
out.txt
```

이 출력 예는 샘플 앱이 파일 시스템에 out.txt 파일을 성공적으로 기록했다는 것을 보여줍니다.

Note

클러스터를 삭제하기 전에 FSx for Lustre 파일 시스템을 삭제해야 합니다. 자세한 내용은 FSx for Lustre 사용 설명서의 [리소스 정리](#)를 참조하세요.

FSx for Lustre의 성능 튜닝

Amazon EKS와 함께 FSx for Lustre를 사용하는 경우 노드 초기화 중 Lustre 튜닝을 적용하여 성능을 최적화할 수 있습니다. 권장 접근 방식은 시작 템플릿 사용자 데이터를 사용하여 모든 노드에서 일관된 구성을 보장하는 것입니다.

이러한 튜닝에는 다음이 포함됩니다.

- 네트워크 및 RPC 최적화
- Lustre 모듈 관리
- LRU(리소스 잠금 단위) 튜닝
- 클라이언트 캐시 제어 설정
- OST 및 MDC에 대한 RPC 제어

이러한 성능 튜닝을 실행하는 방법에 대한 자세한 지침은 다음과 같습니다.

- 표준 노드(비 EFA)의 성능을 최적화하려면 [the section called “최적화\(비 EFA\)”](#)에서 시작 템플릿 사용자 데이터에 추가할 수 있는 전체 스크립트를 참조하세요.
- EFA 지원 노드의 성능 최적화는 [the section called “최적화\(EFA\)”](#) 섹션을 참조하세요.

노드에서 Amazon FSx for Lustre 성능 최적화(EFA)

이 주제에서는 Amazon EKS 및 Amazon FSx for Lustre를 사용하여 Elastic Fabric Adapter(EFA) 튜닝을 설정하는 방법을 설명합니다.

Note

- FSx for Lustre CSI 드라이버 생성 및 배포에 대한 자세한 내용은 [the section called “드라이버 배포”](#) 섹션을 참조하세요.
- EFA 없이 표준 노드를 최적화하려면 [the section called “최적화\(비 EFA\)”](#) 섹션을 참조하세요.

1단계. EKS 클러스터 생성

제공된 구성 파일을 사용하여 클러스터를 생성합니다.

```
# Create cluster using efa-cluster.yaml
eksctl create cluster -f efa-cluster.yaml
```

예) efa-cluster.yaml:

```
#efa-cluster.yaml

apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: csi-fsx
  region: us-east-1
  version: "1.30"

iam:
  withOIDC: true

availabilityZones: ["us-east-1a", "us-east-1d"]

managedNodeGroups:
  - name: my-efa-ng
    instanceType: c6gn.16xlarge
    minSize: 1
    desiredCapacity: 1
    maxSize: 1
    availabilityZones: ["us-east-1b"]
    volumeSize: 300
    privateNetworking: true
    amiFamily: Ubuntu2204
    efaEnabled: true
    preBootstrapCommands:
      - |
        #!/bin/bash
        eth_intf="$(/sbin/ip -br -4 a sh | grep $(hostname -i)/ | awk '{print $1}')"
        efa_version=$(modinfo efa | awk '/^version:/ {print $2}' | sed 's/^[^0-9.]/g')
        min_efa_version="2.12.1"

        if [[ "$(printf '%s\n' "$min_efa_version" "$efa_version" | sort -V | head -
n1)" != "$min_efa_version" ]]; then
            sudo curl -O https://efa-installer.amazonaws.com/aws-efa-
installer-1.37.0.tar.gz
            tar -xf aws-efa-installer-1.37.0.tar.gz && cd aws-efa-installer
```

```

        echo "Installing EFA driver"
        sudo apt-get update && apt-get upgrade -y
        sudo apt install -y pciutils environment-modules libnl-3-dev libnl-
route-3-200 libnl-route-3-dev dkms
        sudo ./efa_installer.sh -y
        modinfo efa
    else
        echo "Using EFA driver version $efa_version"
    fi

    echo "Installing Lustre client"
    sudo wget -O - https://fsx-lustre-client-repo-public-keys.s3.amazonaws.com/fsx-
ubuntu-public-key.asc | gpg --dearmor | sudo tee /usr/share/keyrings/fsx-ubuntu-public-
key.gpg > /dev/null
    sudo echo "deb [signed-by=/usr/share/keyrings/fsx-ubuntu-public-key.gpg]
https://fsx-lustre-client-repo.s3.amazonaws.com/ubuntu jammy main" > /etc/apt/
sources.list.d/fsxlustreclientrepo.list
    sudo apt update | tail
    sudo apt install -y lustre-client-modules-$(uname -r) amazon-ec2-utils | tail
    modinfo lustre

    echo "Loading Lustre/EFA modules..."
    sudo /sbin/modprobe lnet
    sudo /sbin/modprobe kefalnd ipif_name="$eth_intf"
    sudo /sbin/modprobe ksocklnd
    sudo lnetctl lnet configure

    echo "Configuring TCP interface..."
    sudo lnetctl net del --net tcp 2> /dev/null
    sudo lnetctl net add --net tcp --if $eth_intf

    # For P5 instance type which supports 32 network cards,
    # by default add 8 EFA interfaces selecting every 4th device (1 per PCI bus)
    echo "Configuring EFA interface(s)..."
    instance_type="$(ec2-metadata --instance-type | awk '{ print $2 }')"
    num_efa_devices="$(ls -1 /sys/class/infiniband | wc -l)"
    echo "Found $num_efa_devices available EFA device(s)"

    if [[ "$instance_type" == "p5.48xlarge" || "$instance_type" ==
"p5e.48xlarge" ]]; then
        for intf in $(ls -1 /sys/class/infiniband | awk 'NR % 4 == 1'); do
            sudo lnetctl net add --net efa --if $intf --peer-credits 32
        done
    else

```

```
# Other instances: Configure 2 EFA interfaces by default if the instance
supports multiple network cards,
# or 1 interface for single network card instances
# Can be modified to add more interfaces if instance type supports it
sudo lnctl net add --net efa --if $(ls -l /sys/class/infiniband | head -
n1) --peer-credits 32
if [[ $num_efa_devices -gt 1 ]]; then
    sudo lnctl net add --net efa --if $(ls -l /sys/class/infiniband | tail
-n1) --peer-credits 32
fi
fi

echo "Setting discovery and UDSP rule"
sudo lnctl set discovery 1
sudo lnctl udsp add --src efa --priority 0
sudo /sbin/modprobe lustre

sudo lnctl net show
echo "Added $(sudo lnctl net show | grep -c '@efa') EFA interface(s)"
```

2단계. 노드 그룹 생성

EFA 지원 노드 그룹을 생성합니다.

```
# Create node group using efa-ng.yaml
eksctl create nodegroup -f efa-ng.yaml
```

Important

=== # 5. Mount FSx filesystem 섹션에서 환경에 맞게 이러한 값을 조정합니다.

```
FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
MOUNT_POINT="</your/mount/point>" # Needs to be adjusted.
```

===

예) efa-ng.yaml:

```
apiVersion: eksctl.io/v1alpha5
```

```

kind: ClusterConfig

metadata:
  name: final-efa
  region: us-east-1

managedNodeGroups:
  - name: ng-1
    instanceType: c6gn.16xlarge
    minSize: 1
    desiredCapacity: 1
    maxSize: 1
    availabilityZones: ["us-east-1a"]
    volumeSize: 300
    privateNetworking: true
    amiFamily: Ubuntu2204
    efaEnabled: true
    preBootstrapCommands:
      - |
        #!/bin/bash
        exec 1> >(logger -s -t $(basename $0)) 2>&1

#####
#                                     Configuration Section
#
#####

# File System Configuration
FSX_DNS="" # Needs to be adjusted.
MOUNT_NAME="" # Needs to be adjusted.
MOUNT_POINT="" # Needs to be adjusted.

# Lustre Tuning Parameters
LUSTRE_LRU_MAX_AGE=600000
LUSTRE_MAX_CACHED_MB=64
LUSTRE_OST_MAX_RPC=32
LUSTRE_MDC_MAX_RPC=64
LUSTRE_MDC_MOD_RPC=50

# File paths
FUNCTIONS_SCRIPT="/usr/local/bin/lustre_functions.sh"
TUNINGS_SCRIPT="/usr/local/bin/apply_lustre_tunings.sh"

```



```

SERVICE_FILE="/etc/systemd/system/lustre-tunings.service"

#EFA
MIN_EFA_VERSION="2.12.1"

# Function to check if a command was successful
check_success() {
    if [ $? -eq 0 ]; then
        echo "SUCCESS: $1"
    else
        echo "FAILED: $1"
        return 1
    fi
}

echo "*****Starting FSx for Lustre configuration*****"

# 1. Install Lustre client
if grep -q '^ID=ubuntu' /etc/os-release; then
    echo "Detected Ubuntu, proceeding with Lustre setup..."
    # Add Lustre repository
    sudo wget -O - https://fsx-lustre-client-repo-public-keys.s3.amazonaws.com/
fsx-ubuntu-public-key.asc | sudo gpg --dearmor | sudo tee /usr/share/keyrings/fsx-
ubuntu-public-key.gpg > /dev/null

    echo "deb [signed-by=/usr/share/keyrings/fsx-ubuntu-public-key.gpg]
https://fsx-lustre-client-repo.s3.amazonaws.com/ubuntu jammy main" | sudo tee /etc/
apt/sources.list.d/fsxlustreclientrepo.list

    sudo apt-get update
    sudo apt-get install -y lustre-client-modules-$(uname -r)
    sudo apt-get install -y lustre-client
else
    echo "Not Ubuntu, exiting"
    exit 1
fi

check_success "Install Lustre client"

# Ensure Lustre tools are in the PATH
export PATH=$PATH:/usr/sbin

# 2. Apply network and RPC tunings
echo "*****Applying network and RPC tunings*****"

```

```

    if ! grep -q "options ptlrpc ptlrpcd_per_cpt_max" /etc/modprobe.d/
modprobe.conf; then
        echo "options ptlrpc ptlrpcd_per_cpt_max=64" | sudo tee -a /etc/modprobe.d/
modprobe.conf
        check_success "Set ptlrpcd_per_cpt_max"
    else
        echo "ptlrpcd_per_cpt_max already set in modprobe.conf"
    fi

    if ! grep -q "options ksocklnd credits" /etc/modprobe.d/modprobe.conf; then
        echo "options ksocklnd credits=2560" | sudo tee -a /etc/modprobe.d/
modprobe.conf
        check_success "Set ksocklnd credits"
    else
        echo "ksocklnd credits already set in modprobe.conf"
    fi

# 3. Load Lustre modules
manage_lustre_modules() {
    echo "Checking for existing Lustre modules..."
    if lsmod | grep -q lustre; then
        echo "Existing Lustre modules found."

        # Check for mounted Lustre filesystems
        echo "Checking for mounted Lustre filesystems..."
        if mount | grep -q "type lustre"; then
            echo "Found mounted Lustre filesystems. Attempting to unmount..."
            mounted_fs=$(mount | grep "type lustre" | awk '{print $3}')
            for fs in $mounted_fs; do
                echo "Unmounting $fs"
                sudo umount $fs
                check_success "Unmount filesystem $fs"
            done
        else
            echo "No Lustre filesystems mounted."
        fi

        # After unmounting, try to remove modules
        echo "Attempting to remove Lustre modules..."
        sudo lustre_rmmod
        if [ $? -eq 0 ]; then
            echo "SUCCESS: Removed existing Lustre modules"
        else

```

```

        echo "WARNING: Could not remove Lustre modules. They may still be
in use."

        echo "Please check for any remaining Lustre processes or mounts."
        return 1
    fi
else
    echo "No existing Lustre modules found."
fi

echo "Loading Lustre modules..."
sudo modprobe lustre
check_success "Load Lustre modules" || exit 1

echo "Checking loaded Lustre modules:"
lsmod | grep lustre
}

# Managing Lustre kernel modules
echo "*****Managing Lustre kernel modules*****"
manage_lustre_modules

# 4. Initializing Lustre networking
echo "*****Initializing Lustre networking*****"
sudo lctl network up
check_success "Initialize Lustre networking" || exit 1

# 4.5 EFA Setup and Configuration
setup_efa() {
    echo "*****Starting EFA Setup*****"

    # Get interface and version information
    eth_intf="$(/sbin/ip -br -4 a sh | grep $(hostname -i)/ | awk '{print
$1}')"
    efa_version=$(modinfo efa | awk '/^version:/ {print $2}' | sed 's/[^0-9.]//
g')
    min_efa_version=$MIN_EFA_VERSION

    # Install or verify EFA driver
    if [[ "$(printf '%s\n' "$min_efa_version" "$efa_version" | sort -V | head -
n1)" != "$min_efa_version" ]]; then
        echo "Installing EFA driver..."
        sudo curl -O https://efa-installer.amazonaws.com/aws-efa-
installer-1.37.0.tar.gz
        tar -xf aws-efa-installer-1.37.0.tar.gz && cd aws-efa-installer
    fi
}

```

```

        # Install dependencies
        sudo apt-get update && apt-get upgrade -y
        sudo apt install -y pciutils environment-modules libnl-3-dev libnl-
route-3-200 libnl-route-3-dev dkms

        # Install EFA
        sudo ./efa_installer.sh -y
        modinfo efa
    else
        echo "Using existing EFA driver version $efa_version"
    fi
}

configure_efa_network() {
    echo "*****Configuring EFA Network*****"

    # Load required modules
    echo "Loading network modules..."
    sudo /sbin/modprobe lnet
    sudo /sbin/modprobe kefalnd ipif_name="$eth_intf"
    sudo /sbin/modprobe ksocklnd

    # Initialize LNet
    echo "Initializing LNet..."
    sudo lnctl lnet configure

    # Configure TCP interface
    echo "Configuring TCP interface..."
    sudo lnctl net del --net tcp 2> /dev/null
    sudo lnctl net add --net tcp --if $eth_intf

    # For P5 instance type which supports 32 network cards,
    # by default add 8 EFA interfaces selecting every 4th device (1 per PCI
bus)

    echo "Configuring EFA interface(s)..."
    instance_type="$(ec2-metadata --instance-type | awk '{ print $2 }')"
    num_efa_devices="$(ls -1 /sys/class/infiniband | wc -l)"
    echo "Found $num_efa_devices available EFA device(s)"

    if [[ "$instance_type" == "p5.48xlarge" || "$instance_type" ==
    "p5e.48xlarge" ]]; then
        # P5 instance configuration
        for intf in $(ls -1 /sys/class/infiniband | awk 'NR % 4 == 1'); do

```

```

        sudo lnctl net add --net efa --if $intf --peer-credits 32
    done
else
    # Standard configuration
    # Other instances: Configure 2 EFA interfaces by default if the
instance supports multiple network cards,
    # or 1 interface for single network card instances
    # Can be modified to add more interfaces if instance type supports it
    sudo lnctl net add --net efa --if $(ls -1 /sys/class/infiniband |
head -n1) --peer-credits 32
    if [[ $num_efa_devices -gt 1 ]]; then
        sudo lnctl net add --net efa --if $(ls -1 /sys/class/infiniband |
tail -n1) --peer-credits 32
    fi
fi

# Configure discovery and UDSP
echo "Setting up discovery and UDSP rules..."
sudo lnctl set discovery 1
sudo lnctl udsp add --src efa --priority 0
sudo /sbin/modprobe lustre

# Verify configuration
echo "Verifying EFA network configuration..."
sudo lnctl net show
echo "Added $(sudo lnctl net show | grep -c '@efa') EFA interface(s)"
}

# Main execution
setup_efa
configure_efa_network

# 5. Mount FSx filesystem
if [ ! -z "$FSX_DNS" ] && [ ! -z "$MOUNT_NAME" ]; then
    echo "*****Creating mount point*****"
    sudo mkdir -p $MOUNT_POINT
    check_success "Create mount point"

    echo "Mounting FSx filesystem..."
    sudo mount -t lustre ${FSX_DNS}@tcp:${MOUNT_NAME} ${MOUNT_POINT}
    check_success "Mount FSx filesystem"
else
    echo "Skipping FSx mount as DNS or mount name is not provided"
fi

```

```
# 6. Applying Lustre performance tunings
echo "*****Applying Lustre performance tunings*****"

# Get number of CPUs
NUM_CPUS=$(nproc)

# Calculate LRU size (100 * number of CPUs)
LRU_SIZE=$((100 * NUM_CPUS))

#Apply LRU tunings
echo "Apply LRU tunings"
sudo lctl set_param ldlm.namespaces.*.lru_max_age=${LUSTRE_LRU_MAX_AGE}
check_success "Set lru_max_age"
sudo lctl set_param ldlm.namespaces.*.lru_size=$LRU_SIZE
check_success "Set lru_size"

# Client Cache Control
sudo lctl set_param llite.*.max_cached_mb=${LUSTRE_MAX_CACHED_MB}
check_success "Set max_cached_mb"

# RPC Controls
sudo lctl set_param osc.*OST*.max_rpcs_in_flight=${LUSTRE_OST_MAX_RPC}
check_success "Set OST max_rpcs_in_flight"

sudo lctl set_param mdc.*.max_rpcs_in_flight=${LUSTRE_MDC_MAX_RPC}
check_success "Set MDC max_rpcs_in_flight"

sudo lctl set_param mdc.*.max_mod_rpcs_in_flight=${LUSTRE_MDC_MOD_RPC}
check_success "Set MDC max_mod_rpcs_in_flight"

# 7. Verify all tunings
echo "*****Verifying all tunings*****"

# Function to verify parameter value
verify_param() {
    local param=$1
    local expected=$2
    local actual=$3

    if [ "$actual" == "$expected" ]; then
        echo "SUCCESS: $param is correctly set to $expected"
    else
        echo "WARNING: $param is set to $actual (expected $expected)"
    fi
}
```

```

    fi
}

echo "Verifying all parameters:"

# LRU tunings
actual_lru_max_age=$(lctl get_param -n ldlm.namespaces.*.lru_max_age | head -1)
verify_param "lru_max_age" "600000" "$actual_lru_max_age"

actual_lru_size=$(lctl get_param -n ldlm.namespaces.*.lru_size | head -1)
verify_param "lru_size" "$LRU_SIZE" "$actual_lru_size"

# Client Cache
actual_max_cached_mb=$(lctl get_param -n llite.*.max_cached_mb | grep
"max_cached_mb:" | awk '{print $2}')
verify_param "max_cached_mb" "64" "$actual_max_cached_mb"

# RPC Controls
actual_ost_rpcs=$(lctl get_param -n osc.*OST*.max_rpcs_in_flight | head -1)
verify_param "OST max_rpcs_in_flight" "32" "$actual_ost_rpcs"

actual_mdc_rpcs=$(lctl get_param -n mdc.*.max_rpcs_in_flight | head -1)
verify_param "MDC max_rpcs_in_flight" "64" "$actual_mdc_rpcs"

actual_mdc_mod_rpcs=$(lctl get_param -n mdc.*.max_mod_rpcs_in_flight | head -1)
verify_param "MDC max_mod_rpcs_in_flight" "50" "$actual_mdc_mod_rpcs"

# Network and RPC configurations from modprobe.conf
actual_ptlrpc=$(grep "ptlrpc ptlrpcd_per_cpt_max" /etc/modprobe.d/modprobe.conf
| awk '{print $3}')
verify_param "ptlrpcd_per_cpt_max" "ptlrpcd_per_cpt_max=64" "$actual_ptlrpc"

actual_ksocklnd=$(grep "ksocklnd credits" /etc/modprobe.d/modprobe.conf | awk
'{print $3}')
verify_param "ksocklnd credits" "credits=2560" "$actual_ksocklnd"

# 8. Setup persistence
setup_persistence() {
    # Create functions file
    cat << EOF > $FUNCTIONS_SCRIPT
#!/bin/bash

apply_lustre_tunings() {
    local NUM_CPUS=$(nproc)

```

```

    local LRU_SIZE=$((100 * NUM_CPUS))

    echo "Applying Lustre performance tunings..."
    lctl set_param ldlm.namespaces.*.lru_max_age=$LUSTRE_LRU_MAX_AGE
    lctl set_param ldlm.namespaces.*.lru_size=$LRU_SIZE
    lctl set_param llite.*.max_cached_mb=$LUSTRE_MAX_CACHED_MB
    lctl set_param osc.*OST*.max_rpcs_in_flight=$LUSTRE_OST_MAX_RPC
    lctl set_param mdc.*.max_rpcs_in_flight=$LUSTRE_MDC_MAX_RPC
    lctl set_param mdc.*.max_mod_rpcs_in_flight=$LUSTRE_MDC_MOD_RPC
}
EOF

# Create tuning script
cat << EOF > $TUNINGS_SCRIPT
#!/bin/bash
exec 1> >(logger -s -t \$(basename \${0})) 2>&1

source $FUNCTIONS_SCRIPT

# Function to check if Lustre is mounted
is_lustre_mounted() {
    mount | grep -q "type lustre"
}

# Function to mount Lustre
mount_lustre() {
    echo "Mounting Lustre filesystem..."
    mkdir -p $MOUNT_POINT
    mount -t lustre ${FSX_DNS}@tcp:/${MOUNT_NAME} $MOUNT_POINT
    return $?
}

# Main execution
# Try to mount if not already mounted
if ! is_lustre_mounted; then
    echo "Lustre filesystem not mounted, attempting to mount..."
    mount_lustre
fi

# Wait for successful mount (up to 5 minutes)
for i in {1..30}; do
    if is_lustre_mounted; then
        echo "Lustre filesystem mounted, applying tunings..."
        apply_lustre_tunings
    fi
done

```



```
        exit 0
    fi
    echo "Waiting for Lustre filesystem to be mounted... (attempt $i/30)"
    sleep 10
done

echo "Timeout waiting for Lustre filesystem mount"
exit 1
EOF

# Create systemd service

# Create systemd directory if it doesn't exist
sudo mkdir -p /etc/systemd/system/

    # Create service file directly for Ubuntu
    cat << EOF > $SERVICE_FILE
[Unit]
Description=Apply Lustre Performance Tunings
After=network.target remote-fs.target

[Service]
Type=oneshot
ExecStart=/bin/bash -c 'source $FUNCTIONS_SCRIPT && $TUNINGS_SCRIPT'
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
EOF

    # Make scripts executable and enable service
    sudo chmod +x $FUNCTIONS_SCRIPT
    sudo chmod +x $TUNINGS_SCRIPT
    systemctl enable lustre-tunings.service
    systemctl start lustre-tunings.service
}

echo "*****Setting up persistent tuning*****"
setup_persistence

echo "FSx for Lustre configuration completed."
```

(선택 사항) 3단계. EFA 설정 확인

노드에 SSH로 연결합니다.

```
# Get instance ID from EKS console or AWS CLI
ssh -i /path/to/your-key.pem ec2-user@<node-internal-ip>
```

EFA 구성을 확인합니다.

```
sudo lnctl net show
```

설정 로그를 확인합니다.

```
sudo cat /var/log/cloud-init-output.log
```

다음은 `lnctl net show`에 대한 예상 출력의 예입니다.

```
net:
  - net type: tcp
    ...
  - net type: efa
    local NI(s):
      - nid: xxx.xxx.xxx.xxx@efa
        status: up
```

배포 예제

a. claim.yaml 생성

```
#claim.yaml

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: fsx-claim-efa
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
```

```
storage: 4800Gi
volumeName: fsx-pv
```

클레임을 적용합니다.

```
kubectl apply -f claim.yaml
```

b. pv.yaml 생성

<replaceable-placeholders>를 업데이트합니다.

```
#pv.yaml

apiVersion: v1
kind: PersistentVolume
metadata:
  name: fsx-pv
spec:
  capacity:
    storage: 4800Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteMany
  mountOptions:
    - flock
  persistentVolumeReclaimPolicy: Recycle
  csi:
    driver: fsx.csi.aws.com
    volumeHandle: fs-<1234567890abcdef0>
    volumeAttributes:
      dnsname: fs-<1234567890abcdef0>.fsx.us-east-1.amazonaws.com
      mountname: <abcdef01>
```

영구 볼륨을 적용합니다.

```
kubectl apply -f pv.yaml
```

c. pod.yaml 생성

```
#pod.yaml

apiVersion: v1
```

```
kind: Pod
metadata:
  name: fsx-efa-app
spec:
  containers:
    - name: app
      image: amazonlinux:2
      command: ["/bin/sh"]
      args: ["-c", "while true; do dd if=/dev/urandom bs=100M count=20 > data/test_file;
sleep 10; done"]
      resources:
        requests:
          vpc.amazonaws.com/efa: 1
        limits:
          vpc.amazonaws.com/efa: 1
      volumeMounts:
        - name: persistent-storage
          mountPath: /data
  volumes:
    - name: persistent-storage
      persistentVolumeClaim:
        claimName: fsx-claim-efa
```

포드를 적용합니다.

```
kubectl apply -f pod.yaml
```

추가 확인 명령

파일 시스템에 포드 탑재 및 쓰기를 확인합니다.

```
kubectl exec -ti fsx-efa-app -- df -h | grep data
# Expected output:
# <192.0.2.0>@tcp:/<abcdef01> 4.5T 1.2G 4.5T 1% /data

kubectl exec -ti fsx-efa-app -- ls /data
# Expected output:
# test_file
```

노드에 SSH로 연결하여 트래픽이 EFA를 통과하는지 확인합니다.

```
sudo lnctl net show -v
```

예상 출력에는 트래픽 통계가 포함된 EFA 인터페이스가 표시됩니다.

관련 정보

- [the section called “드라이버 배포”](#)
- [the section called “최적화\(비 EFA\)”](#)
- [Amazon FSx for Lustre Performance](#)
- [Elastic Fabric Adapter\(EFA\)](#)

노드에서 Amazon FSx for Lustre 성능 최적화(비 EFA)

시작 템플릿 사용자 데이터를 사용한 노드 초기화 중에 튜닝 파라미터를 적용하여 Amazon FSx for Lustre 성능을 최적화할 수 있습니다.

Note

- FSx for Lustre CSI 드라이버 생성 및 배포에 대한 자세한 내용은 [the section called “드라이버 배포”](#) 섹션을 참조하세요. EFA 지원 노드를 사용하여 성능을 최적화하려면 [the section called “최적화\(EFA\)”](#) 섹션을 참조하세요.

시작 템플릿 사용자 데이터를 사용하는 이유는 무엇인가요?

- 노드 초기화 중에 튜닝을 자동으로 적용합니다.
- 모든 노드에서 일관된 구성을 보장합니다.
- 수동 노드 구성이 필요하지 않습니다.

예제 스크립트 개요

이 주제에 정의된 예제 스크립트는 다음 작업을 수행합니다.

1. Install Lustre client

- OS 버전 Amazon Linux 2(AL2) 또는 Amazon Linux 2023(AL2023)을 자동으로 감지합니다.
- 적절한 Lustre 클라이언트 패키지를 설치합니다.

2. Apply network and RPC tunings

- 병렬 RPC 처리를 위해 `ptlrpcd_per_cpt_max=64`를 설정합니다.
- 네트워크 버퍼를 최적화하도록 `ksocklnd credits=2560`을 구성합니다.

3. Load Lustre modules

- 기존 Lustre 모듈을 안전하게 제거합니다(있는 경우).
- 기존 파일 시스템의 탑재 해제를 처리합니다.
- 새 Lustre 모듈을 로드합니다.

4. Lustre Network Initialization

- Lustre 네트워킹 구성을 초기화합니다.
- 필요한 네트워크 파라미터를 설정합니다.

5. Mount FSx filesystem

- 이 섹션에서는 환경의 값을 조정해야 합니다.

6. Apply tunings

- LRU(리소스 잠금 단위) 튜닝:
 - `lru_max_age=600000`
 - CPU 수를 기준으로 계산된 `lru_size`
- 클라이언트 캐시 제어: `max_cached_mb=64`
- RPC 제어:
 - OST `max_rpcs_in_flight=32`
 - MDC `max_rpcs_in_flight=64`
 - MDC `max_mod_rpcs_in_flight=50`

7. Verify tunings

- 적용된 튜닝을 모두 확인합니다.

- 각 파라미터에 대해 성공 또는 경고를 보고합니다.

8. Setup persistence

- 이 섹션에서도 환경의 값을 조정해야 합니다.
- OS 버전(AL2 또는 AL2023)을 자동으로 감지하여 적용할 Systemd 서비스를 결정합니다.
- 시스템을 시작합니다.
- Systemd가 lustre-tunings 서비스를 시작합니다(WantedBy=multi-user.target 때문).
- 서비스는 다음 작업을 수행하는 `apply_lustre_tunings.sh`를 실행합니다.
 - 파일 시스템이 탑재되었는지 확인합니다.
 - 탑재되지 않은 경우 파일 시스템을 탑재합니다.
 - 탑재가 성공할 때까지 기다립니다(최대 5분).
 - 탑재 성공 후 튜닝 파라미터를 적용합니다.
- 설정은 재부팅할 때까지 활성 상태로 유지됩니다.
- 스크립트 완료 후 서비스가 종료됩니다.
 - Systemd는 서비스를 '활성(종료됨)'으로 표시합니다.
- 다음 재부팅 시 프로세스가 반복됩니다.

시작 템플릿 생성

1. <https://console.aws.amazon.com/ec2/>에서 Amazon EC2 콘솔을 엽니다.
2. 시작 템플릿을 선택합니다.
3. Create launch template(시작 템플릿 생성)을 선택합니다.
4. 고급 세부 정보에서 사용자 데이터 섹션을 찾습니다.
5. 아래 스크립트를 붙여넣고 필요한 항목을 업데이트합니다.

Important

5. Mount FSx filesystem 섹션에서 그리고 # 8. Setup persistence 섹션 내 `apply_lustre_tunings.sh`의 `setup_persistence()` 함수에서 다음 값을 환경에 맞게 조정합니다.

```
FSX_DNS="<your-fsx-filesystem-dns>" # Needs to be adjusted.
MOUNT_NAME="<your-mount-name>" # Needs to be adjusted.
```

```
MOUNT_POINT="/your/mount/point" # Needs to be adjusted.
```

```
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=="MYBOUNDARY=="
--==MYBOUNDARY==
Content-Type: text/x-shellscript; charset="us-ascii"
#!/bin/bash
exec 1> >(logger -s -t $(basename $0)) 2>&1
# Function definitions
check_success() {
    if [ $? -eq 0 ]; then
        echo "SUCCESS: $1"
    else
        echo "FAILED: $1"
        return 1
    fi
}
apply_tunings() {
    local NUM_CPUS=$(nproc)
    local LRU_SIZE=$((100 * NUM_CPUS))
    local params=(
        "ldlm.namespaces.*.lru_max_age=6000000"
        "ldlm.namespaces.*.lru_size=$LRU_SIZE"
        "llite.*.max_cached_mb=64"
        "osc.*OST*.max_rpcs_in_flight=32"
        "mdc.*.max_rpcs_in_flight=64"
        "mdc.*.max_mod_rpcs_in_flight=50"
    )
    for param in "${params[@]"; do
        lctl set_param $param
        check_success "Set ${param%%=*}"
    done
}
verify_param() {
    local param=$1
    local expected=$2
    local actual=$3

    if [ "$actual" == "$expected" ]; then
        echo "SUCCESS: $param is correctly set to $expected"
    else
```



```

        echo "WARNING: $param is set to $actual (expected $expected)"
    fi
}
verify_tunings() {
    local NUM_CPUS=$(nproc)
    local LRU_SIZE=$((100 * NUM_CPUS))
    local params=(
        "ldlm.namespaces.*.lru_max_age:600000"
        "ldlm.namespaces.*.lru_size:$LRU_SIZE"
        "llite.*.max_cached_mb:64"
        "osc.*OST*.max_rpcs_in_flight:32"
        "mdc.*.max_rpcs_in_flight:64"
        "mdc.*.max_mod_rpcs_in_flight:50"
    )
    echo "Verifying all parameters:"
    for param in "${params[@]}; do
        name="${param%%:*}"
        expected="${param#*:}"
        actual=$(lctl get_param -n $name | head -1)
        verify_param "${name##*.*}" "$expected" "$actual"
    done
}
setup_persistence() {
    # Create functions file
    cat << 'EOF' > /usr/local/bin/lustre_functions.sh
#!/bin/bash
apply_lustre_tunings() {
    local NUM_CPUS=$(nproc)
    local LRU_SIZE=$((100 * NUM_CPUS))

    echo "Applying Lustre performance tunings..."
    lctl set_param ldlm.namespaces.*.lru_max_age=600000
    lctl set_param ldlm.namespaces.*.lru_size=$LRU_SIZE
    lctl set_param llite.*.max_cached_mb=64
    lctl set_param osc.*OST*.max_rpcs_in_flight=32
    lctl set_param mdc.*.max_rpcs_in_flight=64
    lctl set_param mdc.*.max_mod_rpcs_in_flight=50
}
EOF
    # Create tuning script
    cat << 'EOF' > /usr/local/bin/apply_lustre_tunings.sh
#!/bin/bash
exec 1> >(logger -s -t $(basename $0)) 2>&1
# Source the functions

```

```

source /usr/local/bin/lustre_functions.sh
# FSx details
FSX_DNS="<<your-fsx-filesystem-dns>" # Needs to be adjusted.
MOUNT_NAME="<<your-mount-name>" # Needs to be adjusted.
MOUNT_POINT="<</your/mount/point>" # Needs to be adjusted.
# Function to check if Lustre is mounted
is_lustre_mounted() {
    mount | grep -q "type lustre"
}
# Function to mount Lustre
mount_lustre() {
    echo "Mounting Lustre filesystem..."
    mkdir -p ${MOUNT_POINT}
    mount -t lustre ${FSX_DNS}@tcp:${MOUNT_NAME} ${MOUNT_POINT}
    return $?
}
# Main execution
# Try to mount if not already mounted
if ! is_lustre_mounted; then
    echo "Lustre filesystem not mounted, attempting to mount..."
    mount_lustre
fi
# Wait for successful mount (up to 5 minutes)
for i in {1..30}; do
    if is_lustre_mounted; then
        echo "Lustre filesystem mounted, applying tunings..."
        apply_lustre_tunings
        exit 0
    fi
    echo "Waiting for Lustre filesystem to be mounted... (attempt $i/30)"
    sleep 10
done
echo "Timeout waiting for Lustre filesystem mount"
exit 1
EOF
# Create systemd service
cat << 'EOF' > /etc/systemd/system/lustre-tunings.service
[Unit]
Description=Apply Lustre Performance Tunings
After=network.target remote-fs.target
StartLimitIntervalSec=0
[Service]
Type=oneshot
ExecStart=/usr/local/bin/apply_lustre_tunings.sh

```

```

RemainAfterExit=yes
Restart=on-failure
RestartSec=30
[Install]
WantedBy=multi-user.target
EOF
    chmod +x /usr/local/bin/lustre_functions.sh
    chmod +x /usr/local/bin/apply_lustre_tunings.sh
    systemctl enable lustre-tunings.service
    systemctl start lustre-tunings.service
}
echo "Starting FSx for Lustre configuration..."
# 1. Install Lustre client
if grep -q 'VERSION="2"' /etc/os-release; then
    amazon-linux-extras install -y lustre
elif grep -q 'VERSION="2023"' /etc/os-release; then
    dnf install -y lustre-client
fi
check_success "Install Lustre client"
# 2. Apply network and RPC tunings
export PATH=$PATH:/usr/sbin
echo "Applying network and RPC tunings..."
if ! grep -q "options ptlrpc ptlrpcd_per_cpt_max" /etc/modprobe.d/modprobe.conf; then
    echo "options ptlrpc ptlrpcd_per_cpt_max=64" | tee -a /etc/modprobe.d/
modprobe.conf
    echo "options ksocklnd credits=2560" | tee -a /etc/modprobe.d/modprobe.conf
fi
# 3. Load Lustre modules
modprobe lustre
check_success "Load Lustre modules" || exit 1
# 4. Lustre Network Initialization
lctl network up
check_success "Initialize Lustre networking" || exit 1
# 5. Mount FSx filesystem
FSX_DNS="" # Needs to be adjusted.
MOUNT_NAME="" # Needs to be adjusted.
MOUNT_POINT="" # Needs to be adjusted.
if [ ! -z "$FSX_DNS" ] && [ ! -z "$MOUNT_NAME" ]; then
    mkdir -p $MOUNT_POINT
    mount -t lustre ${FSX_DNS}@tcp:${MOUNT_NAME} ${MOUNT_POINT}
    check_success "Mount FSx filesystem"
fi
# 6. Apply tunings
apply_tunings

```

```
# 7. Verify tunings
verify_tunings
# 8. Setup persistence
setup_persistence
echo "FSx for Lustre configuration completed."
--==MYBOUNDARY==--
```

6. Amazon EKS 노드 그룹을 생성할 때 이 시작 템플릿을 선택합니다. 자세한 내용은 [the section called “생성”](#) 섹션을 참조하세요.

관련 정보

- [the section called “드라이버 배포”](#)
- [the section called “최적화\(EFA\)”](#)
- [Amazon FSx for Lustre Performance](#)

FSx for NetApp ONTAP과 함께 고성능 앱 스토리지 사용

NetApp Trident에서는 Container Storage Interface(CSI) 호환 드라이버를 사용하여 동적 스토리지 오케스트레이션을 제공합니다. 이를 사용하면 Amazon FSx for NetApp ONTAP 파일 시스템에서 지원하는 영구 볼륨(PV)의 수명 주기를 Amazon EKS 클러스터에서 관리할 수 있습니다. Amazon FSx for NetApp ONTAP CSI 드라이버는 Amazon EKS Hybrid Nodes와 호환되지 않습니다. 시작하려면 NetApp Trident 설명서의 [Amazon FSx for NetApp ONTAP에서 Trident 사용](#)을 참조하세요.

Amazon FSx for NetApp ONTAP은 클라우드에서 완전관리형 ONTAP 파일 시스템을 시작하고 실행할 수 있는 스토리지 서비스입니다. ONTAP은 널리 채택된 데이터 액세스 및 데이터 관리 기능의 집합을 제공하는 NetApp의 파일 시스템 기술입니다. FSx for ONTAP은 완전 관리형 AWS 서비스의 민첩성, 확장성 및 단순성을 온프레미스 NetApp 파일 시스템의 기능, 성능 및 API에 제공합니다. 자세한 내용은 [FSx for ONTAP 사용 설명서](#)를 참조하세요.

Important

Amazon EBS CSI 드라이버와 함께 Amazon FSx for NetApp ONTAP을 사용하여 EBS 볼륨을 프로비저닝하는 경우 `multipath.conf` 파일에서 EBS 디바이스를 사용하지 않도록 지정해야 합니다. 지원되는 방법은 [구성 파일 블랙리스트](#)를 참조하세요. 다음 예를 참고하세요

```
defaults {
    user_friendly_names yes
```

```
    find_multipaths no
  }
  blacklist {
    device {
      vendor "NVME"
      product "Amazon Elastic Block Store"
    }
  }
}
```

Amazon FSx for OpenZFS와 함께 데이터 스토리지 사용

Amazon FSx for OpenZFS는 온프레미스 ZFS 또는 기타 Linux 기반 파일 서버에서 AWS로 데이터를 손쉽게 이동할 수 있는 완전관리형 파일 스토리지 서비스입니다. 애플리케이션 코드나 데이터 관리 방법을 변경하지 않고도 이 작업을 수행할 수 있습니다. 또한 Amazon FSx for OpenZFS는 오픈 소스 OpenZFS 파일 시스템에 구축된 매우 안정적이고 확장 가능하며 효율적이고 기능이 풍부한 파일 스토리지를 제공합니다. 이러한 기능을 완전관리형 AWS 서비스의 민첩성, 확장성 및 단순성과 결합합니다. 자세한 내용은 [Amazon FSx for OpenZFS User Guide](#)를 참조하세요.

FSx for OpenZFS 컨테이너 스토리지(CSI) 드라이버에서는 Amazon EKS 클러스터가 FSx for OpenZFS 볼륨의 수명 주기를 관리할 수 있도록 하는 CSI 인터페이스를 제공합니다. Amazon FSx for OpenZFS CSI 드라이버는 Amazon EKS Hybrid Nodes와 호환되지 않습니다. FSx for OpenZFS CSI 드라이버를 Amazon EKS 클러스터에 배포하려면 GitHub의 [aws-fsx-openzfs-csi-driver](#) 섹션을 참조하세요.

Amazon File Cache를 사용한 지연 시간 최소화

Amazon 파일 캐시는 데이터가 저장된 위치에 관계없이 파일 데이터를 처리하는 데 사용되는 AWS의 완전 관리형 고속 캐시입니다. Amazon File Cache는 처음 액세스하면 캐시에 데이터를 자동으로 로드하고 사용하지 않을 때는 데이터를 릴리스합니다. 자세한 내용은 [Amazon File Cache 사용 설명서](#)를 참조하세요.

Amazon EBS Container Storage Interface(CSI) 드라이버는 클러스터가 Amazon EFS 파일 시스템의 수명 주기를 관리할 수 있게 해주는 CSI 인터페이스를 제공합니다. Amazon File Cache CSI 드라이버는 Amazon EKS Hybrid Nodes와 호환되지 않습니다. Amazon 파일 캐시 CSI 드라이버를 Amazon EKS 클러스터에 배포하려면 GitHub에서 [aws-file-cache-csi-driver](#)를 참조하세요.

Mountpoint for Amazon S3 CSI 드라이버를 통해 Amazon S3 객체에 액세스

[Mountpoint for Amazon S3 컨테이너 스토리지 인터페이스\(CSI\) 드라이버](#)를 사용하면 Kubernetes 애플리케이션이 파일 시스템 인터페이스를 통해 Amazon S3 객체에 액세스하여 Kubernetes 애플리케이션 코드를 변경하지 않고도 높은 총 처리량을 달성할 수 있습니다. [Mountpoint for Amazon S3](#)를 기반으로 구축된 CSI 드라이버는 Amazon EKS 및 자체 관리형 Kubernetes 클러스터의 컨테이너가 액세스할 수 있는 볼륨으로 Amazon S3 버킷을 제공합니다.

고려 사항

- Mountpoint for Amazon S3 CSI 드라이버는 현재 Windows 기반 컨테이너 이미지와 호환되지 않습니다.
- Mountpoint for Amazon S3 CSI 드라이버는 현재 Amazon EKS Hybrid Nodes와 호환되지 않습니다.
- Mountpoint for Amazon S3 CSI 드라이버는 AWS Fargate를 지원하지 않습니다. 하지만 Amazon EC2에서 실행되는 컨테이너(Amazon EKS 또는 사용자 지정 Kubernetes 설치)는 지원됩니다.
- Mountpoint for Amazon S3 CSI 드라이버는 정적 프로비저닝만 지원합니다. 동적 프로비저닝 또는 새 버킷의 생성은 지원되지 않습니다.

Note

정적 프로비저닝이란 PersistentVolume 객체의 volumeAttributes에서 bucketName으로 지정된 기존 Amazon S3 버킷을 사용하는 것을 말합니다. 자세한 내용은 GitHub의 [Static Provisioning](#)을 참조하세요.

- Mountpoint for Amazon S3 CSI 드라이버가 탑재된 볼륨은 모든 POSIX 파일 시스템 기능을 지원하지는 않습니다. 파일 시스템 동작에 대한 자세한 내용은 GitHub의 [Mountpoint for Amazon S3 file system behavior](#)를 참조하세요.

드라이버 배포에 대한 자세한 내용은 [the section called “드라이버 배포”](#) 섹션을 참조하세요. 드라이버 제거에 대한 자세한 내용은 [the section called “드라이버 제거”](#) 섹션을 참조하세요.

Mountpoint for Amazon S3 드라이버 배포

[Mountpoint for Amazon S3 컨테이너 스토리지 인터페이스\(CSI\) 드라이버](#)를 사용하면 Kubernetes 애플리케이션이 파일 시스템 인터페이스를 통해 Amazon S3 객체에 액세스하여 Kubernetes 애플리케이션 코드를 변경하지 않고도 높은 총 처리량을 달성할 수 있습니다.

이 절차에서는 [Mountpoint for Amazon S3 CSI Amazon EKS 드라이버](#)를 배포하는 방법을 보여줍니다. 계속하기 전에 [고려 사항](#)을 검토하세요.

사전 조건

- 클러스터에 대한 기존 AWS Identity and Access Management(IAM) OpenID Connect(OIDC) 제공자입니다. 이미 있는지 아니면 생성해야 하는지 확인하려면 [the section called “IAM OIDC 제공자”](#) 부분을 참조하세요.
- 디바이스 또는 AWS CloudShell에 AWS CLI 버전 2.12.3 이후 버전이 설치 및 구성됩니다.
- kubectl 명령줄 도구는 장치 또는 AWS CloudShell에 설치됩니다. 버전은 클러스터의 Kubernetes 버전과 동일하거나 최대 하나 이전 또는 이후의 마이너 버전일 수 있습니다. 예를 들어, 클러스터 버전이 1.29인 경우 kubectl 버전 1.28, 1.29 또는 1.30를 함께 사용할 수 있습니다. kubectl을 설치하거나 업그레이드하려면 [the section called “kubectl 및 eksctl 설정”](#) 부분을 참조하세요.

1단계: IAM 정책 생성

Mountpoint for Amazon S3 CSI 드라이버가 파일 시스템과 상호 작용하려면 Amazon S3 권한이 필요합니다. 이 섹션에서는 필요한 권한을 부여하는 IAM 정책을 생성하는 방법을 소개합니다.

다음 정책 예시는 Mountpoint에 대한 IAM 권한 권장 사항을 따릅니다. 또는 AWS 관리형 정책 [AmazonS3FullAccess](#)를 사용할 수 있지만 이 관리형 정책은 Mountpoint에 필요한 것보다 더 많은 권한을 부여합니다.

Mountpoint에 대해 권장되는 권한에 대한 자세한 내용은 GitHub의 [Mountpoint IAM permissions](#)를 참조하세요.

- <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
- 왼쪽 탐색 창에서 정책을 선택합니다.
- 정책 페이지에서 정책 생성을 선택합니다.
- 정책 편집기에서 JSON을 선택합니다.
- 정책 편집기에서 다음을 복사하고 붙여 넣습니다.

Important

amzn-s3-demo-bucket1을 사용자의 Amazon S3 버킷 이름으로 대체합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "MountpointFullBucketAccess",
      "Effect": "Allow",
      "Action": [
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket1"
      ]
    },
    {
      "Sid": "MountpointFullObjectAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:AbortMultipartUpload",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket1/*"
      ]
    }
  ]
}
```

Amazon S3 Express One Zone 스토리지 클래스에 도입된 디렉터리 버킷은 범용 버킷과 다른 인증 메커니즘을 사용합니다. `s3:*` 작업을 사용하는 대신 `s3express:CreateSession` 작업을 사용해야 합니다. 디렉터리 버킷에 대한 자세한 내용은 Amazon S3 사용 설명서의 [디렉터리 버킷](#)을 참조하세요.

다음은 디렉터리 버킷에 사용할 최소 권한 정책의 예입니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```



```

        "Effect": "Allow",
        "Action": "s3express:CreateSession",
        "Resource": "arn:aws:s3express:us-west-2:111122223333:bucket/amzn-s3-
demo-bucket1--usw2-az1--x-s3"
    }
]
}

```

6. 다음을 선택합니다.
7. 검토 및 생성 페이지에서 정책의 이름을 지정합니다. 이 예시 연습에서는 AmazonS3CSIDriverPolicy라는 이름을 사용합니다.
8. 정책 생성을 선택합니다.

2단계 - IAM 역할 생성

Mountpoint for Amazon S3 CSI 드라이버가 파일 시스템과 상호 작용하려면 Amazon S3 권한이 필요합니다. 이 섹션에서는 IAM 역할을 생성하여 이러한 권한을 위임하는 방법을 설명합니다. 이 역할을 생성하려면 다음 도구 중 하나를 사용할 수 있습니다.

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)
- [the section called “AWS CLI”](#)

Note

IAM 정책 AmazonS3CSIDriverPolicy는 이전 섹션에서 생성되었습니다.

eksctl

eksctl을 사용하여 Mountpoint for Amazon S3 CSI 드라이버 IAM 역할 생성

IAM 역할 및 Kubernetes 서비스 계정을 생성하려면 다음 명령을 실행합니다. 또한 이러한 명령은 AmazonS3CSIDriverPolicy IAM 정책을 역할에 연결하고, Kubernetes 서비스 계정(s3-csi-controller-sa)에 IAM 역할 Amazon 리소스 이름(ARN)을 주석으로 추가하고, Kubernetes 서비스 계정 이름을 IAM 역할에 대한 신뢰 정책에 추가합니다.

```
CLUSTER_NAME=my-cluster
```

```

REGION=region-code
ROLE_NAME=AmazonEKS_S3_CSI_DriverRole
POLICY_ARN=AmazonEKS_S3_CSI_DriverRole_ARN
eksctl create iamserviceaccount \
  --name s3-csi-driver-sa \
  --namespace kube-system \
  --cluster $CLUSTER_NAME \
  --attach-policy-arn $POLICY_ARN \
  --approve \
  --role-name $ROLE_NAME \
  --region $REGION \
  --role-only

```

AWS Management Console

1. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 역할을 선택합니다.
3. 역할 페이지에서 역할 생성을 선택합니다.
4. 신뢰할 수 있는 엔터티 선택 페이지에서 다음을 수행합니다.
 - a. 신뢰할 수 있는 엔터티 유형 섹션에서 웹 자격 증명을 선택합니다.
 - b. 아이덴티티 제공업체(Identity provider)의 경우 클러스터에 대해 OpenID Connect 공급자 URL(OpenID Connect provider URL)을 선택합니다(Amazon EKS의 개요에 표시된 대로).

URL이 표시되지 않으면 [필수 요건](#)을 검토하세요.

- c. 대상에서 sts.amazonaws.com을 입력합니다.
 - d. 다음을 선택합니다.
5. 권한 추가 페이지에서 다음을 수행합니다.
 - a. 필터 정책 상자에 AmazonS3CSIDriverPolicy를 입력합니다.



Note

이 정책은 이전 섹션에서 생성되었습니다.

- b. 검색에서 반환된 AmazonS3CSIDriverPolicy 결과 왼쪽에 있는 확인란을 선택합니다.
 - c. 다음을 선택합니다.
6. 이름, 검토 및 생성 페이지에서 다음을 수행합니다.
 - a. 역할 이름에 역할의 고유한 이름(예: AmazonEKS_S3_CSI_DriverRole)을 입력합니다.

- b. 태그 추가(선택사항)에서 태그를 키-값 페어로 연결하여 메타데이터를 역할에 추가합니다. IAM에서 태그 사용에 대한 자세한 내용을 알아보려면 IAM 사용 설명서의 [IAM 리소스에 태깅](#)을 참조하세요.
 - c. 역할 생성을 선택합니다.
7. 역할을 생성한 후 편집할 수 있도록 콘솔에서 이 역할을 선택하여 엽니다.
 8. 신뢰 관계 탭을 선택한 후 신뢰 정책 편집을 선택합니다.
 9. 다음과 비슷한 줄을 찾습니다.

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud":
"sts.amazonaws.com"
```

이전 줄의 끝에 쉼표를 추가한 다음 이전 줄 뒤에 다음 줄을 추가합니다. *region-code*를 클러스터가 있는 AWS 리전으로 바꿉니다. *EXAMPLED539D4633E53DE1B71EXAMPLE*을 클러스터의 OIDC 공급자 ID로 바꿉니다.

```
"oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub":
"system:serviceaccount:kube-system:s3-csi-driver-sa"
```

10. Condition 연산자가 "StringEquals"로 설정되어 있는지 확인합니다.

11. 신뢰 정책 업데이트를 선택하여 종료합니다.

AWS CLI

1. 클러스터의 OIDC 공급자 URL을 확인합니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다. 명령의 출력이 None인 경우 [사전 요구 사항](#)을 검토합니다.

```
aws eks describe-cluster --name my-cluster --query "cluster.identity.oidc.issuer" --
output text
```

예제 출력은 다음과 같습니다.

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

2. IAM 역할을 생성하여 Kubernetes 서비스 계정에 AssumeRoleWithWebIdentity 작업 권한을 부여합니다.

- a. 다음 콘텐츠를 `aws-s3-csi-driver-trust-policy.json`라는 파일에 복사합니다.
111122223333을 계정 ID로 바꿉니다. **EXAMPLED539D4633E53DE1B71EXAMPLE** 및 **region-code**를 이전 단계에서 반환된 값으로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.region-
code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:s3-csi-
driver-sa",
          "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
        }
      }
    }
  ]
}
```

- b. 역할을 생성합니다. **AmazonEKS_S3_CSI_DriverRole**를 다른 이름으로 변경할 수 있지만, 그
 렇게 할 경우 이후 단계에서도 변경해야 합니다.

```
aws iam create-role \
  --role-name AmazonEKS_S3_CSI_DriverRole \
  --assume-role-policy-document file://"aws-s3-csi-driver-trust-policy.json"
```

3. 다음 명령을 사용하여 이전에 생성한 IAM 정책을 역할에 연결합니다.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonS3CSIDriverPolicy \
  --role-name AmazonEKS_S3_CSI_DriverRole
```

Note

IAM 정책 AmazonS3CSIDriverPolicy는 이전 섹션에서 생성되었습니다.

4. 드라이버를 Amazon EKS 추가 기능으로 설치하는 경우에는 이 단계를 생략합니다. 자체 관리형 드라이버 설치의 경우 생성한 IAM 역할의 ARN으로 주석이 달린 Kubernetes 서비스 계정을 생성합니다.
 - a. 다음 콘텐츠를 `mountpoint-s3-service-account.yaml`이라는 파일에 저장합니다.
`111122223333`을 계정 ID로 바꿉니다.

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/name: aws-mountpoint-s3-csi-driver
  name: mountpoint-s3-csi-controller-sa
  namespace: kube-system
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/
    AmazonEKS_S3_CSI_DriverRole
```

- b. 클러스터에 Kubernetes 서비스 계정을 생성합니다. Kubernetes 서비스 계정(`mountpoint-s3-csi-controller-sa`)에는 `AmazonEKS_S3_CSI_DriverRole`이라는 이름으로 생성한 IAM 역할이 주석으로 지정되어 있습니다.

```
kubectl apply -f mountpoint-s3-service-account.yaml
```

Note

해당 절차에서 플러그인이 배포되면 `s3-csi-driver-sa`라는 서비스 계정을 생성하고 해당 계정을 사용하도록 구성됩니다.

3단계: Mountpoint for Amazon S3 CSI 드라이버 설치

Amazon EKS 추가 기능을 통해 Mountpoint for Amazon S3 CSI 드라이버를 설치할 수 있습니다. 다음 도구를 사용하여 클러스터에 애드온을 추가할 수 있습니다.

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)
- [the section called “AWS CLI”](#)

또는 Mountpoint for Amazon S3 CSI 드라이버를 자체 관리형 설치로 설치할 수 있습니다. 자체 관리형 설치 방법에 대한 지침은 GitHub의 [Installation](#)을 참조하세요.

v1.8.0부터 CSI 드라이버의 포드에 대해 허용하도록 테인트를 구성할 수 있습니다. 이렇게 하려면 `node.tolerations`를 사용하여 사용자 지정 테인트 세트를 지정하거나 `node.tolerateAllTaints`를 사용하여 모든 테인트를 허용하도록 지정하세요. 자세한 내용은 Kubernetes 설명서의 [테인트와 허용 오차](#)를 참조하세요.

eksctl

eksctl을 사용하여 Amazon S3 CSI 추가 기능 추가

다음 명령을 실행합니다. *my-cluster*를 클러스터 이름으로, *111122223333*을 계정 ID로, *AmazonEKS_S3_CSI_DriverRole*을 [앞서 만든 IAM 역할의 이름](#)으로 바꿉니다.

```
eksctl create addon --name aws-mountpoint-s3-csi-driver --cluster my-cluster \
  --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole
--force
```

기존 설정과 충돌하는 *--force* 옵션과 Amazon EKS 추가 기능 설정을 제거할 경우, Amazon EKS 추가 기능의 업데이트 작업이 실패하고 충돌을 해결하는 데 도움이 되는 오류 메시지가 표시됩니다. 이 옵션을 사용하여 이러한 설정을 덮어쓰기 때문에 이 옵션을 지정하기 전에 Amazon EKS 추가 기능이 관리해야 하는 설정을 관리하지 않는지 확인합니다. 이 설정의 기타 옵션에 대한 자세한 내용은 eksctl 설명서의 [추가 기능](#)을 참조하세요. Amazon EKS Kubernetes 필드 관리에 대한 자세한 내용은 [the section called “사용자 지정할 수 있는 필드”](#) 섹션을 참조하세요.

구성 파일을 통해 eksctl을 사용자 지정할 수 있습니다. 자세한 내용은 eksctl 설명서의 [구성 값으로 작업하기](#)를 참조하세요. 다음 예제에서는 모든 테인트를 허용하는 방법을 보여줍니다.

```
# config.yaml
...

addons:
- name: aws-mountpoint-s3-csi-driver
  serviceAccountRoleARN: arn:aws:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole
```

```
configurationValues: |-
  node:
    tolerateAllTaints: true
```

AWS Management Console

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 좌측 탐색 창에서 클러스터를 선택합니다.
3. Mountpoint for Amazon S3 CSI 추가 기능을 구성하려는 클러스터의 이름을 선택합니다.
4. 추가 기능 탭을 선택합니다.
5. 추가 기능 더 가져오기를 선택합니다.
6. 추가 기능 선택 페이지에서 다음을 수행합니다.
 - a. Amazon EKS 추가 기능 섹션에서 Mountpoint for Amazon S3 CSI 드라이버 확인란을 선택합니다.
 - b. 다음을 선택합니다.
7. 선택한 추가 기능 설정 구성 페이지에서 다음을 수행합니다.
 - a. 사용할 버전을 선택합니다.
 - b. IAM 역할 선택에서 Mountpoint for Amazon S3 CSI 드라이버 IAM 정책을 연결한 IAM 역할의 이름을 선택합니다.
 - c. (선택사항) 선택적 구성 설정을 확장한 후 충돌 해결 방법을 업데이트하세요. 재정의의 선택한 경우 기존 추가 기능에 대한 하나 이상의 설정을 Amazon EKS 추가 기능의 설정으로 덮어쓸 수 있습니다. 이 옵션을 사용 설정하지 않고 기존 설정과 충돌이 있는 경우 작업이 실패합니다. 결과 오류 메시지를 사용하여 충돌을 해결할 수 있습니다. 이 옵션을 선택하기 전에 Amazon EKS 추가 기능이 사용자가 자체 관리해야 하는 설정을 관리하지 않는지 확인하세요.
 - d. (선택사항) 선택적 구성 설정을 확장한 후 구성 값 필드에서 톨러레이션을 구성합니다.
 - e. 다음을 선택합니다.
8. 검토 및 추가 페이지에서 생성을 선택합니다. 추가 기능 설치가 완료되면 설치한 추가 기능이 표시 됩니다.

AWS CLI

AWS CLI를 사용하여 Mountpoint for Amazon S3 CSI 추가 기능 추가

다음 명령을 실행합니다. *my-cluster*를 클러스터 이름으로, *111122223333*을 계정 ID로, *AmazonEKS_S3_CSI_DriverRole*을 앞서 만든 역할의 이름으로 바꿉니다.

```
aws eks create-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-driver \
  --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole
```

--configuration-values 플래그와 함께 명령을 사용자 지정할 수 있습니다. 다음 대체 예제는 모든 테인트를 허용하는 방법을 보여줍니다.

```
aws eks create-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-driver \
  --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKS_S3_CSI_DriverRole \
  --configuration-values '{"node":{"tolerateAllTaints":true}}'
```

4단계: Mountpoint for Amazon S3 구성

대부분의 경우 버킷 이름만 사용하여 Mountpoint for Amazon S3를 구성할 수 있습니다. Mountpoint for Amazon S3 구성에 대한 지침은 GitHub의 [Configuring Mountpoint for Amazon S3](#)를 참조하세요.

5단계: 샘플 애플리케이션 배포

기존 Amazon S3 버킷의 드라이버에 정적 프로비저닝을 배포할 수 있습니다. 자세한 내용은 GitHub의 [Static provisioning](#)을 참조하세요.

Mountpoint for Amazon S3 Amazon EKS 추가 기능 제거

[Mountpoint for Amazon S3 CSI 드라이버](#)를 제거하는 데는 두 가지 옵션이 있습니다.

- 클러스터에 추가 기능 소프트웨어 보존 – 이 옵션은 모든 설정에 대한 Amazon EKS 관리를 제거합니다. 또한 업데이트를 시작한 후 Amazon EKS가 업데이트를 알리고 Amazon EKS 추가 기능을 자동으로 업데이트하는 기능을 제거합니다. 그러나 클러스터에 추가 기능 소프트웨어는 유지됩니다. 이 옵션을 사용하면 추가 기능을 Amazon EKS 추가 기능이 아닌 자체 관리형 설치 기능으로 만들 수 있습니다. 이 옵션을 사용하면 추가 기능에 대한 가동 중지 시간이 없습니다. 이 절차의 명령은 이 옵션을 사용합니다.
- 클러스터에서 추가 기능 소프트웨어 완전히 제거(Remove the add-on software entirely from your cluster) - 클러스터에 종속된 리소스가 없는 경우,에만 클러스터에서 Amazon EKS 추가 기능을 제거하는 것이 좋습니다. 이 옵션을 수행하려면 이 절차에서 사용하는 명령에서 --preserve를 삭제합니다.

추가 기능에 연결된 IAM 계정이 있는 경우 IAM 계정은 제거되지 않습니다.

다음 도구를 사용하여 Amazon S3 CSI 애드온을 제거할 수 있습니다:

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)
- [the section called “AWS CLI”](#)

eksctl

eksctl을 사용하여 Amazon S3 CSI 추가 기능 제거

*my-cluster*를 클러스터 이름으로 바꾸고 다음 명령을 실행합니다.

```
eksctl delete addon --cluster my-cluster --name aws-mountpoint-s3-csi-driver --preserve
```

AWS Management Console

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 좌측 탐색 창에서 클러스터를 선택합니다.
3. Amazon EBS CSI 추가 기능을 제거할 클러스터의 이름을 선택합니다.
4. 추가 기능 탭을 선택합니다.
5. Mountpoint for Amazon S3 CSI 드라이버를 선택합니다.
6. 제거를 선택합니다.
7. 제거: aws-mountpoint-s3-csi-driver 확인 대화 상자에서 다음을 수행합니다.
 - a. 추가 기능에 대한 Amazon EKS의 설정 관리를 중지하려면 클러스터에 보존을 선택합니다. 클러스터에 추가 소프트웨어를 유지하려면 이렇게 합니다. 이는 추가 기능의 모든 설정을 스스로 관리할 수 있도록 하기 위한 것입니다.
 - b. aws-mountpoint-s3-csi-driver을 입력합니다.
 - c. 제거를 선택합니다.

AWS CLI

AWS CLI을 사용하여 Amazon S3 CSI 추가 기능 제거

*my-cluster*를 클러스터 이름으로 바꾸고 다음 명령을 실행합니다.

```
aws eks delete-addon --cluster-name my-cluster --addon-name aws-mountpoint-s3-csi-driver --preserve
```

CSI 볼륨의 스냅샷 기능 활성화

스냅샷 기능을 사용하면 데이터의 특정 시점 사본을 사용할 수 있습니다. Kubernetes에서 이 기능이 작동하려면 스냅샷을 지원하는 CSI 드라이버(예: Amazon EBS CSI 드라이버)와 CSI 스냅샷 컨트롤러가 모두 필요합니다. 스냅샷 컨트롤러는 Amazon EKS 관리형 추가 기능 또는 자체 관리형 설치로 사용할 수 있습니다.

CSI 스냅샷 컨트롤러 사용 시 고려할 사항이 몇 가지 있습니다.

- 스냅샷 컨트롤러는 스냅샷 기능이 있는 CSI 드라이버와 함께 설치해야 합니다. Amazon EBS CSI 드라이버 설치 지침은 [the section called “Amazon EBS”](#)을 참조하세요.
- Kubernetes는 프로비저너 kubernetes.io/aws-ebs에 StorageClass를 사용하는 Amazon EBS와 같이 CSI 마이그레이션을 통해 제공되는 볼륨의 스냅샷을 지원하지 않습니다. CSI 드라이버 프로비저너 ebs.csi.aws.com를 참조하는 StorageClass로 볼륨을 생성해야 합니다.
- Amazon EKS Auto Mode에는 스냅샷 컨트롤러가 포함되지 않습니다. EKS Auto Mode의 스토리지 기능은 스냅샷 컨트롤러와 호환됩니다.

Amazon EKS 관리형 추가 기능을 통해 CSI 스냅샷 컨트롤러를 설치하는 것이 좋습니다. 이 추가 기능에는 Amazon EKS에서 스냅샷을 생성하고 관리하는 데 필요한 사용자 지정 리소스 정의(CRDs)가 포함되어 있습니다. Amazon EKS 추가 기능을 클러스터에 추가하려면 [the section called “추가 기능 생성”](#) 섹션을 참조하세요. Amazon EKS 추가 기능에 대한 자세한 내용은 [the section called “Amazon EKS 추가 기능”](#) 섹션을 참조하세요.

또는 CSI 스냅샷 컨트롤러의 자체 관리형 설치를 원하는 경우 GitHub의 업스트림 Kubernetes [external-snapshotter](#)에서 [Usage](#)를 참조하세요.

Amazon EKS 클러스터에 대한 네트워킹 구성

Amazon EKS 클러스터는 VPC에서 생성됩니다. 포드 네트워킹은 AWS 인프라에서 실행되는 노드에 대해 Amazon VPC 컨테이너 네트워크 인터페이스(CNI) 플러그 인을 통해 제공됩니다. 자체 인프라에서 노드를 실행하는 경우 [the section called “CNI 구성”](#) 섹션을 참조하세요. 이 장에서는 클러스터의 네트워킹에 대해 자세히 알아보기 위해 다음 주제를 다룹니다.

주제

- [관리 콘솔에서 Amazon EKS 클러스터에 기존 VPC 서브넷 추가](#)
- [VPC 및 서브넷에 대한 Amazon EKS 네트워킹 요구 사항 보기](#)
- [Amazon EKS 클러스터에 대한 Amazon VPC 생성](#)
- [클러스터에 대한 Amazon EKS 보안 그룹 요구 사항 보기](#)
- [Amazon EKS 클러스터에 대한 네트워킹 추가 기능 관리](#)

관리 콘솔에서 Amazon EKS 클러스터에 기존 VPC 서브넷 추가

1. 관리 콘솔에서 클러스터로 이동합니다.
2. 네트워킹 탭에서 VPC 리소스 관리를 선택합니다.
3. 서브넷 드롭다운에서 클러스터의 VPC로부터 추가 서브넷을 선택합니다.

새 VPC 서브넷 생성:

- [EKS 서브넷 요구 사항 검토](#)
- Amazon Virtual Private Cloud 사용 설명서의 [서브넷 생성](#)을 참조하세요.

VPC 및 서브넷에 대한 Amazon EKS 네트워킹 요구 사항 보기

클러스터를 생성하는 경우 서로 다른 가용 영역에 있는 [VPC](#) 및 두 개 이상의 서브넷을 지정합니다. 이 주제에서는 클러스터에서 사용하는 VPC 및 서브넷에 대한 Amazon EKS별 요구 사항과 고려 사항을 간략하게 살펴봅니다. Amazon EKS에 사용할 VPC가 없는 경우 [the section called “VPC 생성”](#) 섹션을 참조하세요. AWS Outposts에서 로컬 또는 확장 클러스터를 생성하는 경우, 이 주제 대신 [the section called “VPC 및 서브넷 생성”](#)(Amazon EKS 로컬 클러스터 VPC 및 서브넷 요구 사항 및 고려 사항)를 참조하세요. 이 주제의 내용은 하이브리드 노드가 있는 Amazon EKS 클러스터에 적용됩니다. 하이브리드 노드에 대한 추가 네트워킹 요구 사항은 [the section called “네트워킹 준비”](#) 섹션을 참조하세요.

VPC 요구 사항 및 고려 사항

클러스터를 생성하는 경우 지정하는 VPC는 다음 요구 사항 및 고려 사항을 충족해야 합니다.

- VPC에는 생성하려는 클러스터, 노드 및 기타 Kubernetes 리소스에 사용 가능한 IP 주소가 충분해야 합니다. 사용하려는 VPC에 IP 주소가 충분하지 않은 경우 사용 가능한 IP 주소 수를 늘립니다.

클러스터 구성을 업데이트하여 클러스터에서 사용하는 서브넷 및 보안 그룹을 변경하면 이 작업을 수행할 수 있습니다. AWS Management Console, 최신 버전의 AWS CLI, AWS CloudFormation 및 eksctl 버전 v0.164.0-rc.0 이상에서 업데이트할 수 있습니다. 서브넷에 사용 가능한 IP 주소를 더 많이 제공하여 클러스터 버전을 성공적으로 업그레이드하려면 이 작업을 수행해야 할 수도 있습니다.

Important

추가하는 모든 서브넷은 클러스터를 생성할 때 원래 제공한 것과 동일한 AZ 세트에 있어야 합니다. 새 서브넷은 다른 모든 요구 사항을 충족해야 합니다. 예를 들어, 충분한 IP 주소가 있어야 합니다.

예를 들어, 클러스터를 만들고 서브넷 4개를 지정했다고 가정합니다. 지정한 순서대로 첫 번째 서브넷은 us-west-2a 가용 영역에, 두 번째와 세 번째 서브넷은 us-west-2b 가용 영역에, 네 번째 서브넷은 us-west-2c 가용 영역에 있습니다. 서브넷을 변경하려면 세 가용 영역 각각에 하나 이상의 서브넷을 제공해야 하며, 서브넷은 원래 서브넷과 동일한 VPC에 있어야 합니다.

VPC의 CIDR 블록보다 많은 IP 주소가 필요한 경우, 추가적인 [CIDR\(Classless Inter-Domain Routing\) 블록을 VPC에 연결](#)하여 CIDR 블록을 추가할 수 있습니다. 클러스터를 생성하기 전이나 후에 프라이빗(RFC 1918) 및 퍼블릭(비RFC 1918) CIDR 블록을 VPC에 연결할 수 있습니다.

새 CIDR 블록을 추가한 후 바로 사용할 수 있습니다. 그러나 조정이 완료된 후에만 컨트롤 플레인에서 새 CIDR 블록을 인식하므로 VPC와 연결한 CIDR 블록을 인식하는 데 5시간까지 걸릴 수 있습니다. 그런 다음 kubectl exec 및 kubectl logs 명령을 실행할 수 있습니다. 또한 웹훅 백엔드로 작동하는 포드가 있는 경우 컨트롤 플레인 조정이 완료될 때까지 기다려야 합니다.

공유 서비스 VPC를 통해 전송 게이트웨이를 사용하여 IP 주소 사용을 절약할 수 있습니다. 자세한 내용은 [공유 서비스를 사용하여 격리된 VPC](#) 및 [하이브리드 네트워크의 Amazon EKS VPC 라우팅 가능 IP 주소 보존 패턴](#)을 참조하세요.

- Kubernetes가 IPv6 주소를 포드와 서비스에 할당하게 하려는 경우 IPv6 CIDR 블록을 VPC와 연결합니다. 자세한 내용은 Amazon VPC 사용 설명서의 [IPv6 CIDR 블록을 VPC와 연결](#)을 참조하세요. 하이브리드 노드에서 실행되는 포드 및 서비스에서는 IPv6 주소를 사용할 수 없으며 IPv6 IP 주소 패밀리로 구성된 클러스터에서는 하이브리드 노드를 사용할 수 없습니다.
- VPC는 DNS 호스트 이름과 DNS 확인을 모두 지원해야 합니다. 그렇지 않으면 노드가 클러스터에 등록할 수 없습니다. 자세한 내용을 알아보려면 Amazon VPC 사용 설명서의 [VPC에 대한 DNS 속성](#)을 참조하세요.
- VPC에는 AWS 프라이빗 링크를 사용하는 VPC 엔드포인트가 필요할 수 있습니다. 자세한 내용은 [the section called “서브넷 요구 사항 및 고려 사항”](#) 섹션을 참조하세요.

Kubernetes 1.14 이전 버전을 사용하여 클러스터를 생성한 경우 Amazon EKS에서는 VPC에 다음 태그를 추가했습니다.

키	값
kubernetes.io/cluster/ <i>my-cluster</i>	owned

이 태그는 Amazon EKS에서만 사용되었습니다. 서비스에 영향을 주지 않고 태그를 제거할 수 있습니다. 버전 1.15 이상인 클러스터에서는 사용되지 않습니다.

서브넷 요구 사항 및 고려 사항

클러스터를 생성하는 경우 Amazon EKS에서는 지정한 서브넷에 2~4개의 [탄력적 네트워크 인터페이스](#)를 생성합니다. 이러한 네트워크 인터페이스를 사용하면 클러스터와 VPC 간에 통신할 수 있습니다. 또한 이러한 네트워크 인터페이스를 통해 `kubectl exec` 및 `kubectl logs`와 같은 Kubernetes 기능을 사용 설정할 수 있습니다. 각 Amazon EKS 생성 네트워크 인터페이스의 설명에는 Amazon EKS *cluster-name* 이 포함되어 있습니다.

클러스터를 생성하는 경우 Amazon EKS에서는 지정한 서브넷에 해당 네트워크 인터페이스를 생성합니다. 클러스터가 생성되면 Amazon EKS가 네트워크 인터페이스를 생성하는 서브넷을 변경할 수 있습니다. Kubernetes 버전의 클러스터를 업데이트하면 Amazon EKS에서는 클러스터가 생성한 원래 네트워크 인터페이스를 삭제하고 새로운 네트워크 인터페이스를 생성합니다. 이러한 네트워크 인터페이스는 원래 네트워크 인터페이스와 동일한 서브넷 또는 원래 네트워크 인터페이스와 다른 서브넷에서 생성될 수 있습니다. 네트워크 인터페이스가 생성되는 서브넷을 제어하려면 클러스터를 생성하거나 클러스터를 생성한 후 서브넷을 업데이트할 때 지정하는 서브넷 수를 2개로 제한할 수 있습니다.

클러스터의 서브넷 요구 사항

클러스터를 생성하거나 업데이트하는 경우 지정하는 [서브넷](#)은 다음 요구 사항을 충족해야 합니다.

- 각 서브넷에는 Amazon EKS에서 사용할 IP 주소가 6개 이상 있어야 합니다. 하지만 16개 이상의 IP 주소를 사용하는 것이 좋습니다.
- 서브넷이 서로 다른 가용 영역에 있어야 합니다.
- 서브넷은 AWS Outposts 또는 AWS Wavelength에 상주할 수 없습니다. 그러나 VPC에 있는 경우 자체 관리형 노드 및 Kubernetes 리소스를 이러한 유형의 서브넷에 배포할 수 있습니다. 자체 관리형 노드에 대한 자세한 내용은 [the section called “자체 관리형 노드”](#) 섹션을 참조하세요.
- 서브넷은 퍼블릭 또는 프라이빗일 수 있습니다. 그러나 가능한 경우 프라이빗 서브넷을 지정하는 것이 좋습니다. 퍼블릭 서브넷은 [인터넷 게이트웨이](#)로 이어지는 경로가 포함된 라우팅 테이블을 사용한 서브넷인 반면 프라이빗 서브넷은 인터넷 게이트웨이로 이어지는 경로가 포함되지 않은 라우팅 테이블을 사용한 서브넷입니다.
- 서브넷은 다음 가용 영역에 상주할 수 없습니다.

AWS 리전	리전 이름	허용되지 않는 가용 영역 ID
us-east-1	미국 동부(버지니아 북부)	use1-az3
us-west-1	미국 서부(캘리포니아 북부)	usw1-az2
ca-central-1	캐나다(중부)	cac1-az3

구성 요소별 IP 주소 패밀리 사용

다음 표에는 Amazon EKS의 각 구성 요소에서 사용하는 IP 주소 패밀리가 나와 있습니다. Network Address Translation(NAT) 또는 기타 호환성 시스템을 사용하여 테이블 항목 ‘No’ 값이 있는 패밀리의 소스 IP 주소에서 이러한 구성 요소에 연결할 수 있습니다.

기능은 클러스터의 IP 패밀리(ipFamily) 설정에 따라 다를 수 있습니다. 이 설정은 Kubernetes에서 서비스에 할당하는 CIDR 블록에 사용되는 IP 주소 유형을 변경합니다. 설정 값이 IPv4인 클러스터를 IPv4 클러스터라고 하고, 설정 값이 IPv6인 클러스터를 IPv6 클러스터라고 합니다.

구성 요소	IPv4 주소	IPv6 주소	이중 스택 주소
EKS API 퍼블릭 엔드포인트	예 ^{1,3}	예 ^{1,3}	예 ^{1,3}
EKS API VPC 엔드포인트	예	아니요	아니요
EKS Auth API 퍼블릭 엔드포인트(EKS Pod Identity)	예 ¹	예 ¹	예 ¹
EKS Auth API VPC 엔드포인트(EKS Pod Identity)	예 ¹	예 ¹	예 ¹
IPv4 Kubernetes 클러스터 퍼블릭 엔드포인트 ²	예	아니요	아니요
IPv4 Kubernetes 클러스터 프라이빗 엔드포인트 ²	예	아니요	아니요
IPv6 Kubernetes 클러스터 퍼블릭 엔드포인트 ²	예 ^{1,4}	예 ^{1,4}	예 ⁴
IPv6 Kubernetes 클러스터 프라이빗 엔드포인트 ²	예 ^{1,4}	예 ^{1,4}	예 ⁴
Kubernetes 클러스터 서브넷	예 ²	아니요	예 ²
노드 기본 IP 주소	예 ²	아니요	예 ²
서비스 IP 주소의 클러스터 CIDR 범위	예 ²	예 ²	아니요

구성 요소	IPv4 주소	IPv6 주소	이중 스택 주소
VPC CNI의 포트 IP 주소	예 ²	예 ²	아니요
IRSA OIDC 발급자 URL	예 ^{1,3}	예 ^{1,3}	예 ^{1,3}

Note

¹ 엔드포인트는 IPv4 및 IPv6 주소가 모두 포함된 이중 스택입니다. AWS 외부의 애플리케이션, 클러스터의 노드 및 클러스터 내부의 포트는 IPv4 또는 IPv6 중 하나를 통해 이 엔드포인트에 도달할 수 있습니다.

² 클러스터를 생성할 때 클러스터의 IP 패밀리(ipFamily) 설정에서 IPv4 클러스터와 IPv6 클러스터 중 하나를 선택하며, 이는 변경할 수 없습니다. 대신 다른 클러스터를 생성하고 워크로드를 마이그레이션할 때 다른 설정을 선택해야 합니다.

³ 이중 스택 엔드포인트는 2024년 8월에 도입되었습니다. AWS CLI와 함께 듀얼 스택 엔드포인트를 사용하려면 AWS SDK 및 도구 참조 가이드에서 [듀얼 스택 및 FIPS 엔드포인트 구성](#)을 참조하세요. 다음은 새 엔드포인트를 나열합니다.

EKS API 퍼블릭 엔드포인트

`eks.region.api.aws`

IRSA OIDC 발급자 URL

`oidc-eks.region.api.aws`

⁴ 이중 스택 클러스터 엔드포인트는 2024년 10월에 도입되었습니다. EKS는 이 날짜 이후에 만들어지고 클러스터의 IP 패밀리(ipFamily) 설정에서 IPv6를 선택하는 새 클러스터에 대해 다음과 같은 엔드포인트를 생성합니다.

EKS 클러스터 퍼블릭/프라이빗 엔드포인트

`eks-cluster.region.api.aws`

노드의 서브넷 요구 사항

클러스터를 생성할 때 지정하는 것과 동일한 서브넷에 노드 및 Kubernetes 리소스를 배포할 수 있습니다. 그러나 이 작업은 필요하지 않습니다. 클러스터를 생성할 때 지정하지 않은 서브넷에 노드 및 Kubernetes 리소스를 배포할 수 있기 때문입니다. 노드를 다른 서브넷에 배포하는 경우 Amazon EKS에서는 해당 서브넷에서 클러스터 네트워크 인터페이스를 생성하지 않습니다. 노드 및 Kubernetes 리소스를 배포하는 모든 서브넷은 다음 요구 사항을 충족해야 합니다.

- 서브넷에는 모든 노드와 Kubernetes 리소스를 배포할 수 있는 IP 주소가 충분해야 합니다.
- Kubernetes가 IPv6 주소를 포드와 서비스에 할당하도록 하려는 경우 서브넷에 연결된 IPv6 CIDR 블록 하나와 IPv4 CIDR 블록 하나가 있어야 합니다. 자세한 내용을 알아보려면 Amazon VPC 사용 설명서의 [IPv6 CIDR 블록을 서브넷과 연결](#)을 참조하세요. 서브넷과 연결된 라우팅 테이블에는 IPv4 및 IPv6 주소로 이어지는 경로가 포함되어야 합니다. 자세한 내용을 알아보려면 Amazon VPC 사용 설명서의 [경로](#)를 참조하세요. 포드는 IPv6 주소에만 할당됩니다. 그러나 Amazon EKS가 클러스터와 노드에 대해 생성하는 네트워크 인터페이스에는 IPv4 및 IPv6 주소가 할당됩니다.
- 인터넷에서 포드로 인바운드 액세스가 필요한 경우 로드 밸런서와 수신을 배포할 수 있는 IP 주소가 충분한 퍼블릭 서브넷이 하나 이상 있어야 합니다. 로드 밸런서를 퍼블릭 서브넷에 배포할 수 있습니다. 로드 밸런서는 프라이빗 또는 퍼블릭 서브넷의 포드에 로드 밸런싱할 수 있습니다. 가능하면 프라이빗 서브넷에 노드를 배포하는 것이 좋습니다.
- 노드를 퍼블릭 서브넷에 배포하려는 경우 서브넷이 IPv4 퍼블릭 주소 또는 IPv6 주소를 자동 할당해야 합니다. 연결된 IPv6 CIDR 블록이 있는 프라이빗 서브넷에 노드를 배포하는 경우 프라이빗 서브넷도 IPv6 주소를 자동 할당해야 합니다. 2020년 3월 26일 이후에 Amazon EKS에서 제공한 AWS CloudFormation 템플릿을 사용하여 VPC를 배포한 경우 이 설정이 활성화됩니다. 템플릿을 사용하여 이 날짜 이전에 VPC를 배포했거나 자체 VPC를 사용하는 경우 이 설정을 수동으로 사용 설정해야 합니다. 템플릿은 [the section called “VPC 생성”](#) 섹션을 참조하세요. 자세한 내용은 [Amazon VPC 사용 설명서](#)에서 [서브넷의 공용 IPv4 주소 지정 속성 수정](#) 및 [서브넷의 IPv6 주소 지정 속성 수정](#)을 참조하세요.
- 노드를 배포하는 서브넷이 프라이빗 서브넷이고 해당 라우팅 테이블에 Network Address Translation([NAT 디바이스](#)) (IPv4) 또는 [발신 전용 게이트웨이](#) (IPv6)로 이어지는 경로가 포함되지 않은 경우, AWS를 사용하는 VPC 엔드포인트를 사용자 VPC에 추가합니다. VPC 엔드포인트는 노드 및 포드와 통신해야 하는 모든 AWS 서비스에 필요합니다. 예를 들면 Amazon ECR, Elastic Load Balancing, Amazon CloudWatch, AWS 보안 토큰 서비스 및 Amazon Simple Storage Service(Amazon S3)가 있습니다. 엔드포인트에는 노드가 있는 서브넷이 포함되어야 합니다. 일부 AWS 서비스가 VPC 엔드포인트를 지원합니다. 자세한 내용은 [AWS 프라이빗 링크란 무엇인가요?](#) 및 [AWS 프라이빗 링크와 통합되는 AWS 서비스](#)를 참조하세요. Amazon EKS 추가 요구 사항 목록은 [the section called “프라이빗 클러스터”](#) 섹션을 참조하세요.

- 서브넷에 로드 밸런서를 배포하려는 경우 서브넷에는 다음 태그가 있어야 합니다.
 - 프라이빗 서브넷

키	값
kubernetes.io/role/internal-elb	1

- 퍼블릭 서브넷

키	값
kubernetes.io/role/elb	1

이전에는 버전이 1.18 이하인 Kubernetes 클러스터가 생성되면 Amazon EKS에서 지정된 모든 서브넷에 다음 태그를 추가했습니다.

키	값
kubernetes.io/cluster/ <i>my-cluster</i>	shared

지금은 새 Kubernetes 클러스터를 생성하면 Amazon EKS에서 서브넷에 태그를 추가하지 않습니다. 태그가 이전에 1.19 이하 버전인 클러스터에서 사용된 서브넷에 있는 경우 클러스터가 최신 버전으로 업데이트될 때 태그가 서브넷에서 자동으로 제거되지 않았습니다. AWS 로드 밸런서 컨트롤러 버전 2.1.1 이하에는 이 태그가 필요합니다. 최신 버전의 로드 밸런서 컨트롤러를 사용하는 경우 서비스를 중단하지 않고 태그를 제거할 수 있습니다. 컨트롤러에 대한 자세한 내용은 [the section called “AWS 로드 밸런서 컨트롤러”](#) 섹션을 참조하세요.

eksctl 또는 Amazon EKS AWS CloudFormation VPC 템플릿 중 하나를 사용하여 VPC를 배포한 경우 다음이 적용됩니다.

- 2020년 3월 26일 또는 그 이후 - 퍼블릭 서브넷에서 퍼블릭 서브넷에 배포된 새 노드에 퍼블릭 IPv4 주소가 자동으로 할당됩니다.
- 2020년 3월 26일 이전 - 퍼블릭 서브넷에서 퍼블릭 서브넷에 배포된 새 노드에 퍼블릭 IPv4 주소가 자동으로 할당되지 않습니다.

이 변경은 다음과 같은 방식으로 퍼블릭 서브넷에 배포된 새 노드 그룹에 영향을 줍니다.

- [관리형 노드 그룹](#) -2020년 4월 22일 또는 그 이후에 노드 그룹이 퍼블릭 서브넷에 배포된 경우 퍼블릭 서브넷에 퍼블릭 IP 주소가 자동으로 할당되도록 사용 설정되어 있어야 합니다. 자세한 내용은 [서브넷의 퍼블릭 IPv4 주소 지정 속성 수정](#) 섹션을 참조하세요.
- [Linux](#), [Windows](#) 또는 [Arm](#) 자체 관리형 노드 그룹-2020년 3월 26일 또는 그 이후에 노드 그룹이 퍼블릭 서브넷에 배포된 경우 퍼블릭 서브넷에 퍼블릭 IP 주소가 자동으로 할당되도록 사용 설정되어 있어야 합니다. 그렇지 않으면 대신에 공용 IP 주소로 노드를 시작해야 합니다. 자세한 내용은 [서브넷에 대한 퍼블릭 IPv4 주소 지정 속성 수정](#) 또는 [인스턴스 시작 시 퍼블릭 IPv4 주소 할당](#)을 참조하세요.

공유 서브넷 요구 사항 및 고려 사항

VPC 공유를 사용하여 서브넷을 동일한 AWS 조직내의 다른 AWS 계정과 공유할 수 있습니다. 다음 사항을 고려하여 공유 서브넷에서 Amazon EKS 클러스터를 생성할 수 있습니다.

- VPC 서브넷의 소유자가 참여자 계정과 서브넷을 공유해야 해당 계정에서 Amazon EKS 클러스터를 생성할 수 있습니다.
- 참여자는 VPC의 기본 보안 그룹을 사용하여 리소스를 시작할 수 없습니다. 보안 그룹은 소유자에게 속해 있기 때문입니다. 추가로 참여자는 소유자 또는 다른 참여자가 소유한 보안 그룹을 사용하여 리소스를 시작할 수 없습니다.
- 공유 서브넷에서는 참여자와 소유자가 각 계정 내의 보안 그룹을 별도로 제어합니다. 서브넷 소유자는 참여자가 생성한 보안 그룹을 볼 수 있지만 이 보안 그룹에 대해 조치를 취할 수는 없습니다. 서브넷 소유자가 보안 그룹을 제거하거나 수정하고자 하는 경우 보안 그룹을 생성한 참가자가 조치를 취해야 합니다.
- 참여자가 클러스터를 생성한 경우 다음 고려 사항이 적용됩니다.
 - 클러스터 IAM 역할 및 노드 IAM 역할을 해당 계정에서 생성해야 합니다. 자세한 내용은 [the section called “클러스터 IAM 역할”](#) 및 [the section called “노드 IAM 역할”](#) 섹션을 참조하세요.
 - 관리형 노드 그룹을 포함하여 모든 노드는 동일한 참여자가 생성해야 합니다.
- 공유 VPC 소유자는 참여자가 공유 서브넷에서 생성한 클러스터의 보기, 업데이트 또는 삭제가 불가능합니다. 여기에 더해 계정마다 액세스 권한이 다른 VPC 리소스가 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 [소유자 및 참여자에 대한 책임 및 권한](#)을 참조하세요.
- Kubernetes용 Amazon VPC CNI 플러그인의 사용자 지정 네트워킹 기능을 사용하는 경우 소유자 계정에 나열된 가용 영역 ID 매핑을 사용하여 각 ENIConfig를 생성해야 합니다. 자세한 내용은 [the section called “사용자 지정 네트워킹”](#) 섹션을 참조하세요.

VPC 서브넷 공유에 관한 자세한 내용은 Amazon VPC 사용 설명서의 [다른 계정과 VPC 공유](#)를 참조하세요.

Amazon EKS 클러스터에 대한 Amazon VPC 생성

Amazon Virtual Private Cloud(Amazon VPC)를 사용하여 사용자가 정의한 가상 네트워크로 AWS 리소스를 시작할 수 있습니다. 이 가상 네트워크는 사용자의 자체 데이터 센터에서 운영하는 기존 네트워크와 매우 유사합니다. 그러나 Amazon Web Services Services의 확장 가능한 인프라를 사용하면 얻을 수 있는 이점이 있습니다. 프로덕션 Amazon EKS 클러스터를 배포하기 전에 Amazon VPC 서비스를 충분히 이해하는 것이 좋습니다. 자세한 내용은 [Amazon VPC 사용 설명서](#)를 참조하세요.

Amazon EKS 클러스터, 노드 및 Kubernetes 리소스는 VPC에 배포됩니다. Amazon EKS에서 기존 VPC를 사용하려는 경우 해당 VPC는 [the section called “VPC 및 서브넷 요구 사항”](#)에서 설명하는 요구 사항을 충족해야 합니다. 이 주제에서는 Amazon EKS 제공 AWS CloudFormation 템플릿을 사용하여 Amazon EKS 요구 사항을 충족하는 VPC를 생성하는 방법을 설명합니다. 템플릿을 배포했다면 템플릿에서 생성된 리소스를 보면서 생성된 리소스와 해당 리소스의 구성을 정확히 알 수 있습니다. 하이브리드 노드를 사용하는 경우 VPC의 라우팅 테이블에 온프레미스 네트워크의 경로가 있어야 합니다. 하이브리드 노드의 네트워크 요구 사항에 대한 자세한 내용은 [the section called “네트워킹 준비”](#) 섹션을 참조하세요.

사전 조건

Amazon EKS용 VPC를 생성하려면 Amazon VPC 리소스를 생성하는 데 필요한 IAM 권한이 있어야 합니다. 이러한 리소스는 VPC, 서브넷, 보안 그룹, 라우팅 테이블 및 경로, 인터넷 및 NAT 게이트웨이입니다. 자세한 내용을 알아보려면 Amazon VPC 사용 설명서의 [퍼블릭 서브넷에 정책을 사용하여 VPC 생성](#) 및 [서비스 승인 참조](#)의 [작업](#)의 전체 목록을 참조하세요.

퍼블릭 및 프라이빗 서브넷을 사용하거나 퍼블릭 서브넷만 또는 프라이빗 서브넷만 있는 VPC를 생성할 수 있습니다.

퍼블릭 및 프라이빗 서브넷

이 VPC에는 2개의 퍼블릭 서브넷과 2개의 프라이빗 서브넷이 있습니다. 퍼블릭 서브넷은 인터넷 게이트웨이로 이어지는 라우팅이 있는 라우팅 테이블과 연결된 서브넷입니다. 그러나 프라이빗 서브넷의 라우팅 테이블에는 인터넷 게이트웨이로 이어지는 라우팅이 없습니다. 1개의 퍼블릭 서브넷과 1개의 프라이빗 서브넷이 동일한 가용 영역에 배포됩니다. 다른 퍼블릭 서브넷과 프라이빗 서브넷은 동일한 AWS 리전의 두 번째 가용 영역에 배포됩니다. 대부분 배포에 이 옵션을 사용하는 것이 좋습니다.

이 옵션을 사용하면 노드를 프라이빗 서브넷에 배포할 수 있습니다. 이 옵션을 사용하면 Kubernetes가 프라이빗 서브넷의 노드에서 실행되는 포드로 트래픽을 로드 밸런싱할 수 있는 퍼블릭 서브넷에 로드 밸런서를 배포할 수 있습니다. 퍼블릭 IPv4 주소는 퍼블릭 서브넷에 배포된 노드에 자동으로 할당되지만 퍼블릭 IPv4 주소는 프라이빗 서브넷에 배포된 노드에는 할당되지 않습니다.

퍼블릭 및 프라이빗 서브넷의 노드에 IPv6 주소를 할당할 수도 있습니다. 프라이빗 서브넷의 노드는 클러스터 및 기타 AWS 서비스와 통신할 수 있습니다. 포드는 IPv4 주소를 사용하여 NAT 게이트웨이를 통하거나 각 가용 영역에 배포된 IPv6 주소를 사용하여 아웃바운드 전용 인터넷 게이트웨이를 통해 인터넷으로 통신할 수 있습니다. 클러스터 또는 노드 이외의 소스에서 모든 인바운드 트래픽을 거부하지만 모든 아웃바운드 트래픽을 허용하는 규칙이 있는 보안 그룹이 배포됩니다. 서브넷에는 Kubernetes가 로드 밸런서를 배포할 수 있도록 태그가 지정됩니다.

- a. [AWS CloudFormation 콘솔](#)을 엽니다.
- b. 탐색 모음에서 Amazon EKS를 지원하는 AWS 리전을 선택합니다.
- c. 스택 생성, 새 리소스 사용(표준)을 선택합니다.
- d. 사전 조건 - 템플릿 준비에서 템플릿 준비 완료가 선택되었는지 확인한 다음 템플릿 지정에서 Amazon S3 URL을 선택합니다.
- e. IPv4만 지원하는 VPC 또는 IPv4 및 IPv6를 지원하는 VPC를 생성할 수 있습니다. Amazon S3 URL 아래의 텍스트 영역에 다음 URL 중 하나를 붙여 넣고 다음을 선택합니다.
 - IPv4

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml
```

- IPv4 및 IPv6

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-ipv6-vpc-public-private-subnets.yaml
```

- a. 스택 세부 정보 지정 페이지에서 파라미터를 입력하고 다음을 선택합니다.
 - 스택 이름: AWS CloudFormation 스택에 대한 스택 이름을 선택합니다. 예를 들어, 이전 단계에서 사용한 템플릿 이름을 사용할 수 있습니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다.

- VpcBlock: VPC에 대한 IPv4 CIDR 범위를 선택합니다. 배포하는 각 노드, 포드 및 로드 밸런서에는 이 블록의 IPv4 주소가 할당됩니다. 기본 IPv4 값은 대부분의 구현에 충분한 IP 주소를 제공하지만 그렇지 않은 경우 변경할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 [VPC 및 서브넷 크기 조정](#)을 참조하세요. 또한 생성된 VPC CIDR 블록을 추가할 수도 있습니다. IPv6 VPC를 생성하는 경우 Amazon의 글로벌 유니캐스트 주소 공간에서 IPv6 CIDR 범위가 자동으로 할당됩니다.
 - PublicSubnet01Block: 퍼블릭 서브넷 1의 IPv4 CIDR 블록을 지정합니다. 기본값은 대부분의 구현에 충분한 IP 주소를 제공하지만 그렇지 않은 경우 변경할 수 있습니다. IPv6 VPC를 생성하는 경우 템플릿 내에 이 블록이 지정됩니다.
 - PublicSubnet02Block: 퍼블릭 서브넷 2의 IPv4 CIDR 블록을 지정합니다. 기본값은 대부분의 구현에 충분한 IP 주소를 제공하지만 그렇지 않은 경우 변경할 수 있습니다. IPv6 VPC를 생성하는 경우 템플릿 내에 이 블록이 지정됩니다.
 - PrivateSubnet01Block: 프라이빗 서브넷 1의 IPv4 CIDR 블록을 지정합니다. 기본값은 대부분의 구현에 충분한 IP 주소를 제공하지만 그렇지 않은 경우 변경할 수 있습니다. IPv6 VPC를 생성하는 경우 템플릿 내에 이 블록이 지정됩니다.
 - PrivateSubnet02Block: 프라이빗 서브넷 2의 IPv4 CIDR 블록을 지정합니다. 기본값은 대부분의 구현에 충분한 IP 주소를 제공하지만 그렇지 않은 경우 변경할 수 있습니다. IPv6 VPC를 생성하는 경우 템플릿 내에 이 블록이 지정됩니다.
- b. (선택사항) 스택 옵션 구성 페이지에서 스택 리소스에 태그를 지정하고 다음을 선택합니다.
 - c. 검토 페이지에서 스택 생성을 선택합니다.
 - d. 스택이 생성된 후 콘솔에서 이를 선택하고 출력을 선택합니다.
 - e. 생성된 VPC의 VpcId를 기록합니다. 이 값은 클러스터 및 노드를 생성할 때 필요합니다.
 - f. 생성된 서브넷의 SubnetIds와 퍼블릭 서브넷 또는 프라이빗 서브넷으로 생성했는지 여부를 기록합니다. 2개 이상의 이 값은 클러스터 및 노드를 생성할 때 필요합니다.
 - g. IPv4 VPC를 생성한 경우 이 단계를 건너뛰십시오. IPv6 VPC를 생성한 경우 템플릿으로 생성한 퍼블릭 서브넷에 대해 IPv6 주소 자동 할당 옵션을 사용 설정해야 합니다. 이 설정은 프라이빗 서브넷에 대해 이미 사용 설정되어 있습니다. 설정을 사용 설정하려면 다음 단계를 완료합니다.
 - i. <https://console.aws.amazon.com/vpc/>에서 Amazon VPC 콘솔을 엽니다.
 - ii. 왼쪽 탐색 창에서 서브넷을 선택합니다.
 - iii. 퍼블릭 서브넷(**stack-name**/SubnetPublic01 또는 **stack-name**/SubnetPublic02에 퍼블릭이라는 단어가 포함됨) 중 하나를 선택하고 작업, 서브넷 설정 편집을 선택합니다.
 - iv. IPv6 주소 자동 할당 사용(Enable auto-assign IPv6 address) 확인란을 선택한 다음 저장(Save)을 선택합니다.

- v. 다른 퍼블릭 서브넷에 대해 이전 단계를 다시 완료합니다.

퍼블릭 서브넷만

이 VPC에는 AWS 리전의 다른 가용 영역에 배포되는 3개의 퍼블릭 서브넷이 있습니다. 모든 노드에는 퍼블릭 IPv4 주소가 자동으로 할당되며 [인터넷 게이트웨이](#)를 통해 인터넷 트래픽을 송수신할 수 있습니다. 모든 인바운드 트래픽을 거부하고 모든 아웃바운드 트래픽을 허용하는 [보안 그룹](#)이 배포됩니다. 서브넷에는 Kubernetes가 로드 밸런서를 배포할 수 있도록 태그가 지정됩니다.

- a. [AWS CloudFormation 콘솔](#)을 엽니다.
- b. 탐색 모음에서 Amazon EKS를 지원하는 AWS 리전을 선택합니다.
- c. 스택 생성, 새 리소스 사용(표준)를 선택합니다.
- d. 템플릿 준비에서 템플릿 준비 완료가 선택되었는지 확인한 다음 템플릿 소스에서 Amazon S3 URL을 선택합니다.
- e. Amazon S3 URL 아래의 텍스트 영역에 다음 URL을 붙여 넣고 다음을 선택합니다.

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-vpc-sample.yaml
```

- a. 세부 정보 지정 페이지에서 파라미터를 입력하고 다음을 선택합니다.
 - 스택 이름: AWS CloudFormation 스택에 대한 스택 이름을 선택합니다. 예를 들어, **amazon-eks-vpc-sample**이라고 부를 수 있습니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다.
 - VpcBlock: VPC에 대한 CIDR 블록을 선택합니다. 배포하는 각 노드, 포드 및 로드 밸런서에는 이 블록의 IPv4 주소가 할당됩니다. 기본 IPv4 값은 대부분의 구현에 충분한 IP 주소를 제공하지만 그렇지 않은 경우 변경할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 [VPC 및 서브넷 크기 조정](#)을 참조하세요. 또한 생성된 VPC CIDR 블록을 추가할 수도 있습니다.
 - Subnet01Block: 서브넷 1에 대한 CIDR 블록을 지정합니다. 기본값은 대부분의 구현에 충분한 IP 주소를 제공하지만 그렇지 않은 경우 변경할 수 있습니다.
 - Subnet02Block: 서브넷 2에 대한 CIDR 블록을 지정합니다. 기본값은 대부분의 구현에 충분한 IP 주소를 제공하지만 그렇지 않은 경우 변경할 수 있습니다.
 - Subnet03Block: 서브넷 3에 대한 CIDR 블록을 지정합니다. 기본값은 대부분의 구현에 충분한 IP 주소를 제공하지만 그렇지 않은 경우 변경할 수 있습니다.

- b. (선택사항) 옵션 페이지에서 스택 리소스에 태그를 지정합니다. 다음을 선택합니다.
- c. 검토 페이지에서 생성을 선택합니다.
- d. 스택이 생성된 후 콘솔에서 이를 선택하고 출력을 선택합니다.
- e. 생성된 VPC의 VpcId를 기록합니다. 이 값은 클러스터 및 노드를 생성할 때 필요합니다.
- f. 생성된 서브넷에 대한 SubnetIds를 기록합니다. 2개 이상의 이 값은 클러스터 및 노드를 생성할 때 필요합니다.
- g. (선택 사항) 이 VPC에 배포되는 모든 클러스터는 포트와 서비스에 프라이빗 IPv4 주소를 할당할 수 있습니다. 이 VPC에 배포된 클러스터가 포트와 서비스에 프라이빗 IPv6 주소를 할당하도록 하려면 VPC, 서브넷, 라우팅 테이블 및 보안 그룹을 업데이트해야 합니다. 자세한 내용은 Amazon VPC 사용 설명서의 [IPv4에서 IPv6로 기존 VPC 마이그레이션](#)을 참조하세요. Amazon EKS를 사용하려면 서브넷에 Auto-assign IPv6 주소 옵션이 사용 설정되어 있어야 합니다. 기본적으로 사용 중지되어 있습니다.

프라이빗 서브넷만

이 VPC에는 AWS 리전의 다른 가용 영역에 배포되는 3개의 프라이빗 서브넷이 있습니다. 서브넷에 배포된 리소스는 인터넷에 액세스할 수 없으며, 인터넷도 서브넷의 리소스에 액세스할 수 없습니다. 템플릿은 노드가 일반적으로 액세스해야 하는 여러 AWS 서비스에 대해 AWS 프라이빗 링크를 사용하여 [VPC 엔드포인트](#)를 생성합니다. 노드에 아웃바운드 인터넷 액세스가 필요한 경우 VPC가 생성되면 각 서브넷의 가용 영역에서 퍼블릭 [NAT 게이트웨이](#)를 추가할 수 있습니다. 서브넷에 배포된 리소스를 제외한 모든 인바운드 트래픽을 거부하는 [보안 그룹](#)이 생성됩니다. 보안 그룹은 모든 아웃바운드 트래픽을 허용합니다. 서브넷에는 Kubernetes가 내부 로드 밸런서를 배포할 수 있도록 태그가 지정됩니다. 이 구성을 사용하여 VPC 생성하는 경우 추가 요구 사항 및 고려 사항에 대한 [the section called “프라이빗 클러스터”](#)를 참조하세요.

- a. [AWS CloudFormation 콘솔](#)을 엽니다.
- b. 탐색 모음에서 Amazon EKS를 지원하는 AWS 리전을 선택합니다.
- c. 스택 생성, 새 리소스 사용(표준)을 선택합니다.
- d. 템플릿 준비에서 템플릿 준비 완료가 선택되었는지 확인한 다음 템플릿 소스에서 Amazon S3 URL을 선택합니다.
- e. Amazon S3 URL 아래의 텍스트 영역에 다음 URL을 붙여 넣고 다음을 선택합니다.

```
https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/amazon-eks-fully-private-vpc.yaml
```


a. 세부 정보 지정 페이지에서 파라미터를 입력하고 다음을 선택합니다.

- 스택 이름: AWS CloudFormation 스택에 대한 스택 이름을 선택합니다. 예를 들어, **amazon-eks-ful-private-vpc**라고 부를 수 있습니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다.
- VpcBlock: VPC에 대한 CIDR 블록을 선택합니다. 배포하는 각 노드, 포드 및 로드 밸런서에는 이 블록의 IPv4 주소가 할당됩니다. 기본 IPv4 값은 대부분의 구현에 충분한 IP 주소를 제공하지만 그렇지 않은 경우 변경할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 [VPC 및 서브넷 크기 조정](#)을 참조하세요. 또한 생성된 VPC CIDR 블록을 추가할 수도 있습니다.
- PrivateSubnet01Block: 서브넷 1에 대한 CIDR 블록을 지정합니다. 기본값은 대부분의 구현에 충분한 IP 주소를 제공하지만 그렇지 않은 경우 변경할 수 있습니다.
- PrivateSubnet02Block: 서브넷 2에 대한 CIDR 블록을 지정합니다. 기본값은 대부분의 구현에 충분한 IP 주소를 제공하지만 그렇지 않은 경우 변경할 수 있습니다.
- PrivateSubnet03Block: 서브넷 3에 대한 CIDR 블록을 지정합니다. 기본값은 대부분의 구현에 충분한 IP 주소를 제공하지만 그렇지 않은 경우 변경할 수 있습니다.

b. (선택사항) 옵션 페이지에서 스택 리소스에 태그를 지정합니다. 다음을 선택합니다.

c. 검토 페이지에서 생성을 선택합니다.

d. 스택이 생성된 후 콘솔에서 이를 선택하고 출력을 선택합니다.

e. 생성된 VPC의 VpcId를 기록합니다. 이 값은 클러스터 및 노드를 생성할 때 필요합니다.

f. 생성된 서브넷에 대한 SubnetIds를 기록합니다. 2개 이상의 이 값은 클러스터 및 노드를 생성할 때 필요합니다.

g. (선택 사항) 이 VPC에 배포되는 모든 클러스터는 포드와 서비스에 프라이빗 IPv4 주소를 할당할 수 있습니다. 이 VPC에 배포된 클러스터가 포드와 서비스에 프라이빗 IPv6 주소를 할당하도록 하려면 VPC, 서브넷, 라우팅 테이블 및 보안 그룹을 업데이트해야 합니다. 자세한 내용은 Amazon VPC 사용 설명서의 [IPv4에서 IPv6로 기존 VPC 마이그레이션](#)을 참조하세요. Amazon EKS를 사용하려면 서브넷에 Auto-assign IPv6 주소 옵션이 활성화되어 있어야 합니다(기본적으로 비활성화됨).

클러스터에 대한 Amazon EKS 보안 그룹 요구 사항 보기

이 주제에서는 Amazon EKS 클러스터의 보안 그룹 요구 사항을 설명합니다.

기본 클러스터 보안 그룹

클러스터를 생성할 때 Amazon EKS에서 `eks-cluster-sg-my-cluster-uniqueID` 라는 보안 그룹을 만듭니다. 이 보안 그룹에는 다음의 기본 규칙이 있습니다.

규칙 타입	프로토콜	포트	소스	대상
인바운드	모두	모두	본인	
아웃바운드	모두	모두		0.0.0.0/0IPv4 () 또는 ::/0 (IPv6)
아웃바운드	모두	모두		자체(EFA 트래픽의 경우)

기본 보안 그룹에는 동일한 보안 그룹의 대상으로 Elastic Fabric Adapter(EFA) 트래픽을 허용하는 아웃바운드 규칙이 포함되어 있습니다. 이렇게 하면 클러스터 내의 EFA 트래픽이 활성화되어 AI/ML 및 HPC(고성능 컴퓨팅) 워크로드에 유용합니다. 자세한 내용은 Amazon EC2 사용 설명서의 [Amazon EC2의 AI/ML 및 HPC 워크로드를 위한 Elastic Fabric Adapter](#)를 참조하세요.

Important

클러스터에 아웃바운드 규칙이 필요하지 않은 경우 해당 규칙을 제거할 수 있습니다. 규칙을 제거하더라도 [클러스터 트래픽 제한](#)에 나열된 최소 규칙은 여전히 있어야 합니다. 인바운드 규칙을 제거하면 클러스터가 업데이트될 때마다 Amazon EKS에서 해당 규칙을 다시 생성합니다.

Amazon EKS에서는 다음과 같은 태그를 보안 그룹에 추가합니다. 태그를 제거하면 클러스터가 업데이트될 때마다 Amazon EKS에서 태그를 보안 그룹에 다시 추가합니다.

키	값
kubernetes.io/cluster/ <i>my-cluster</i>	owned
aws:eks:cluster-name	<i>my-cluster</i>

키	값
Name	eks-cluster-sg- <i>my-cluster-uniqueid</i>

Amazon EKS는 이 보안 그룹을 생성하는 다음 리소스에 자동으로 연결합니다.

- 클러스터를 생성할 때 생성되는 2~4개의 탄력적 네트워크 인터페이스(이 문서의 나머지 부분에서는 네트워크 인터페이스로 언급)입니다.
- 생성한 모든 관리형 노드 그룹에 있는 노드의 네트워크 인터페이스입니다.

기본 규칙을 사용하면 클러스터와 노드 간에 모든 트래픽이 자유롭게 흐를 수 있으며 모든 아웃바운드 트래픽은 모든 대상에 대해 허용됩니다. 클러스터를 생성할 때 사용자 고유의 보안 그룹을 지정할 수 있습니다(선택사항). 이렇게 하면 Amazon EKS는 사용자가 지정한 보안 그룹을 클러스터에 대해 생성하는 네트워크 인터페이스에 연결합니다. 그러나 사용자가 생성하는 노드 그룹에는 연결하지 않습니다.

클러스터의 네트워킹 섹션의 AWS Management Console에서 해당 클러스터 보안 그룹의 ID를 확인할 수 있습니다. 또는 다음 AWS CLI 명령을 실행하여 이 작업을 수행할 수 있습니다.

```
aws eks describe-cluster --name my-cluster --query
cluster.resourcesVpcConfig.clusterSecurityGroupId
```

클러스터 트래픽 제한

클러스터와 노드 사이에 열린 포트를 제한해야 하는 경우 [기본 아웃바운드 규칙](#)을 제거하고 클러스터에 필요한 다음과 같은 최소 규칙을 추가할 수 있습니다. [기본 인바운드 규칙](#)을 제거하면 클러스터가 업데이트될 때마다 Amazon EKS에서 해당 규칙을 다시 생성합니다.

규칙 타입	프로토콜	Port	대상
아웃바운드	TCP	443	클러스터 보안 그룹
아웃바운드	TCP	10250	클러스터 보안 그룹
아웃바운드(DNS)	TCP 및 UDP	53	클러스터 보안 그룹

또한 다음 트래픽에 대한 규칙을 추가해야 합니다.

- 노드에서 노드 간 통신에 사용할 것으로 예상하는 모든 프로토콜 및 포트입니다.
- 노드가 실행 시 클러스터 내부 검사 및 노드 등록을 위해 Amazon EKS API에 액세스할 수 있는 아웃바운드 인터넷 액세스. 노드에 인터넷 액세스가 없는 경우 [인터넷 액세스가 제한된 프라이빗 클러스터 배포](#)를 검토하여 추가 고려 사항을 확인하세요.
- DockerHub와 같이 이미지를 가져오는 데 필요한 Amazon ECR 또는 다른 컨테이너 레지스트리 API에서 컨테이너 이미지를 가져오기 위한 노드 액세스. 자세한 내용은 AWS 일반 참조의 [AWS IP 주소 범위](#)를 참조하세요.
- Amazon S3에 대한 노드 액세스.
- IPv4 및 IPv6 주소에 대해 별도의 규칙이 필요합니다.
- 하이브리드 노드를 사용하는 경우 클러스터에 추가 보안 그룹을 추가하여 온프레미스 노드 및 포드와의 통신을 허용해야 합니다. 자세한 내용은 [the section called “네트워킹 준비”](#) 섹션을 참조하세요.

규칙을 제한하려는 경우 변경된 규칙을 프로덕션 클러스터에 적용하기 전에 모든 포드를 철저히 테스트하는 것이 좋습니다.

Kubernetes 1.14 및 플랫폼 버전 eks.3 이하를 사용하여 클러스터를 처음 배포한 경우 다음을 고려합니다.

- 사용자에게 컨트롤 플레인과 노드 보안 그룹이 있을 수도 있습니다. 생성된 이러한 그룹에는 이전 표에 나열된 제한 규칙이 포함되어 있습니다. 이러한 보안 그룹은 더 이상 필요하지 않으며 제거할 수 있습니다. 그러나 클러스터 보안 그룹에 해당 그룹에 포함된 규칙이 들어 있는지 확인해야 합니다.
- 직접 API를 사용하여 클러스터를 배포했거나, AWS CLI 또는 AWS CloudFormation과 같은 도구를 사용하여 클러스터를 생성하고 클러스터 생성 시 보안 그룹을 지정하지 않은 경우, VPC의 기본 보안 그룹은 Amazon EKS가 생성한 클러스터 네트워크 인터페이스에 적용됩니다.

공유 보안 그룹

Amazon EKS는 공유 보안 그룹을 지원합니다.

- Security Group VPC Associations는 보안 그룹을 동일한 계정 및 리전의 여러 VPC와 연결합니다.
 - Amazon VPC 사용 설명서에서 [여러 VPC와 보안 그룹을 연결](#)하는 방법을 알아보세요.
- 공유 보안 그룹을 사용하면 다른 AWS 계정과 보안 그룹을 공유할 수 있습니다. 계정은 동일한 AWS 조직에 속해야 합니다.
 - Amazon VPC 사용 설명서에서 [조직과 보안 그룹을 공유](#)하는 방법을 알아보세요.

- 보안 그룹은 항상 단일 AWS 리전으로 제한됩니다.

Amazon EKS 고려 사항

- EKS는 공유 또는 다중 VPC 보안 그룹의 요구 사항이 표준 보안 그룹과 동일합니다.

Amazon EKS 클러스터에 대한 네트워킹 추가 기능 관리

Amazon EKS 클러스터에 여러 네트워킹 추가 기능을 사용할 수 있습니다.

내장 추가 기능

Note

콘솔을 사용하는 경우를 제외하고 어떤 방식으로든 클러스터를 생성하는 경우 각 클러스터에는 자체 관리형 버전의 내장 추가 기능이 함께 제공됩니다. 자체 관리형 버전은 AWS Management Console, AWS 명령줄 인터페이스 또는 SDK.에서 관리할 수 없습니다. 자체 관리형 추가 기능의 구성 및 업그레이드를 관리합니다. 자체 관리형 추가 기능 유형을 사용하는 대신 클러스터에 Amazon EKS 유형의 추가 기능을 추가하는 것이 좋습니다. 콘솔에서 클러스터를 생성한 경우 이러한 추가 기능의 Amazon EKS 유형이 설치됩니다.

Kubernetes용 Amazon VPC CNI 플러그인

이 CNI 추가 기능은 탄력적 네트워크 인터페이스를 생성하여 Amazon EC2 노드에 연결합니다. 또한 이 추가 기능은 프라이빗 IPv4 또는 IPv6 주소를 VPC에서 각 포드와 서비스에 할당합니다. 이 추가 기능은 기본적으로 클러스터에 설치됩니다. 자세한 내용은 [the section called “Amazon VPC CNI”](#) 단원을 참조하십시오. 하이브리드 노드를 사용하는 경우 VPC CNI는 여전히 기본적으로 설치되지만 반선호도 규칙을 사용하여 하이브리드 노드에서 실행되지 않습니다. 하이브리드 노드의 CNI 옵션에 대한 자세한 내용은 [the section called “CNI 구성”](#) 섹션을 참조하세요.

CoreDNS

CoreDNS는 Kubernetes 클러스터 DNS로 사용할 수 있는 유연하고 확장 가능한 DNS 서버입니다. CoreDNS는 클러스터의 모든 포드에 대한 이름 확인을 제공합니다. 이 추가 기능은 기본적으로 클러스터에 설치됩니다. 자세한 내용은 [the section called “CoreDNS”](#) 단원을 참조하십시오.

kube-proxy

이 추가 기능은 Amazon EC2 노드의 네트워크 규칙을 유지하고 포드와의 네트워크 통신을 활성화합니다. 이 추가 기능은 기본적으로 클러스터에 설치됩니다. 자세한 내용은 [the section called “kube-proxy”](#) 섹션을 참조하세요.

선택적 AWS 네트워킹 추가 기능

AWS Load Balancer Controller

loadbalancer 유형의 Kubernetes 서비스 객체를 배포하면 컨트롤러가 AWS Network Load Balancer를 생성합니다. Kubernetes 수신 객체를 생성하면 컨트롤러가 AWS Application Load Balancer를 생성합니다. Kubernetes에 내장된 [레거시 클라우드 제공업체](#) 컨트롤러를 사용하는 대신 이 컨트롤러를 사용하여 Network Load Balancer를 프로비저닝하는 것이 좋습니다. 자세한 내용은 [AWS 로드 밸런서 컨트롤러](#) 설명서를 참조하세요.

AWS 게이트웨이 API 컨트롤러

이 컨트롤러를 통해 [Kubernetes 게이트웨이 API](#)를 사용하여 여러 Kubernetes 클러스터 간에 서비스를 연결할 수 있습니다. 컨트롤러는 [Amazon VPC Lattice](#) 서비스를 사용하여 Amazon EC2 인스턴스, 컨테이너 및 서버리스 함수에서 실행되는 Kubernetes 서비스를 연결합니다. 자세한 내용은 [AWS Gateway API 컨트롤러](#) 설명서를 참조하세요.

추가 기능에 대한 자세한 내용은 [the section called “Amazon EKS 추가 기능”](#) 섹션을 참조하세요.

Amazon VPC CNI를 통해 포드에 IP 할당

Tip

Amazon EKS Auto Mode를 사용하면 네트워킹 추가 기능을 설치하거나 업그레이드할 필요가 없습니다. Auto Mode에는 포드 네트워킹 및 로드 밸런싱 기능이 포함되어 있습니다. 자세한 내용은 [EKS Auto Mode](#) 섹션을 참조하세요.

Kubernetes용 Amazon VPC CNI 플러그인 추가 기능은 Amazon EKS 클러스터의 각 Amazon EC2 노드에 배포됩니다. 이 추가 기능은 [탄력적 네트워크 인터페이스](#)를 생성하여 Amazon EC2 노드에 연결합니다. 또한 이 추가 기능은 프라이빗 IPv4 또는 IPv6 주소를 VPC에서 각 포드에 할당합니다.

추가 기능 버전은 클러스터의 각 Fargate 노드와 함께 배포되지만 Fargate 노드에서는 업데이트되지 않습니다. 호환되는 다른 CNI 플러그인을 Amazon EKS 클러스터에서 사용할 수 있지만, 이는 AWS 인 프라에서 실행되는 노드에 대해 Amazon EKS에서 지원하는 유일한 CNI 플러그인입니다. 다른 호환되는 CNI 플러그인에 대한 자세한 내용은 [the section called “대체 CNI 플러그인”](#)을 참조하세요. VPC CNI는 하이브리드 노드에서 사용할 수 없습니다. 하이브리드 노드의 CNI 옵션에 대한 자세한 내용은 [the section called “CNI 구성”](#) 섹션을 참조하세요.

다음 표에는 각 Kubernetes 버전에 사용할 수 있는 Amazon EKS 추가 기능 유형의 최신 버전이 나열되어 있습니다.

Amazon VPC CNI 버전

Kubernetes 버전	Amazon EKS 유형의 VPC CNI 버전
1.33	v1.19.5-eksbuild.3
1.32	v1.19.5-eksbuild.3
1.31	v1.19.5-eksbuild.3
1.30	v1.19.5-eksbuild.3
1.29	v1.19.5-eksbuild.3
1.28	v1.19.5-eksbuild.3
1.27	v1.19.5-eksbuild.3
1.26	v1.19.5-eksbuild.3

Important

이 추가 기능을 자체 관리하는 경우 표의 버전이 사용 가능한 자체 관리 버전과 다를 수 있습니다. 이 추가 기능의 자체 관리형 유형 업데이트에 대한 자세한 내용은 [the section called “업데이트\(자체 관리형\)”](#) 부분을 참조하세요.

⚠ Important

VPC CNI v1.12.0 이상으로 업그레이드하려면 먼저 VPC CNI v1.7.0으로 업그레이드해야 합니다. 한 번에 하나의 마이너 버전을 업데이트하는 것이 좋습니다.

고려 사항

다음은 기능 사용 시 고려 사항입니다.

- 버전은 `major-version.minor-version.patch-version-eksbuild.build-number`(으)로 지정됩니다.
- 각 기능의 버전 호환성을 확인합니다. Kubernetes용 Amazon VPC CNI 플러그인의 각 릴리스의 일부 기능에는 특정 Kubernetes 버전이 필요합니다. 다른 Amazon EKS 기능을 사용할 때 특정 버전의 추가 기능이 필요한 경우 기능 설명서에 설명되어 있습니다. 이전 버전을 실행할 특별한 이유가 없는 한 최신 버전을 실행하는 것이 좋습니다.

Amazon VPC CNI 생성(Amazon EKS 추가 기능)

다음 단계에 따라 Kubernetes용 Amazon VPC CNI 플러그인 Amazon EKS 추가 기능을 생성합니다.

시작하기 전에 고려 사항을 검토하세요. 자세한 내용은 [the section called “고려 사항”](#) 섹션을 참조하세요.

사전 조건

다음은 Kubernetes용 Amazon VPC CNI 플러그인 Amazon EKS 추가 기능의 사전 조건입니다.

- 기존 Amazon EKS 클러스터. 배포하려면 [시작](#) 섹션을 참조하세요.
- 클러스터에 대한 기존 AWS Identity and Access Management(IAM) OpenID Connect(OIDC) 제공자입니다. 이미 있는지 아니면 생성해야 하는지 확인하려면 [the section called “IAM OIDC 제공자”](#) 부분을 참조하세요.
- [AmazonEKS_CNI_Policy](#) IAM 정책(클러스터에서 IPv4 패밀리를 사용하는 경우) 또는 IPv6 정책(클러스터에서 IPv6 패밀리를 사용하는 경우)이 연결된 IAM 역할. VPC CNI 역할에 대한 자세한 내용은 [the section called “IRSA에 대한 구성”](#)을 참조하세요. IPv6 정책에 대한 자세한 내용은 [the section called “IPv6 패밀리를 사용하는 클러스터에 대한 IAM 정책 생성”](#)을 참조하세요.

⚠ Important

Kubernetes용 Amazon VPC CNI 플러그인 버전 v1.16.0~v1.16.1은 CNI 사양 버전 v1.0.0을 구현합니다. CNI 사양 v1.0.0에 대한 자세한 내용은 GitHub의 [Container Network Interface \(CNI\) Specification](#)을 참조하세요.

절차

사전 조건을 완료한 후에는 다음 단계를 사용하여 추가 기능을 생성합니다.

1. 클러스터에 설치된 추가 기능의 버전을 확인하세요.

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
cut -d : -f 3
```

예제 출력은 다음과 같습니다.

```
v1.16.4-eksbuild.2
```

2. 클러스터에 설치된 추가 기능의 유형을 확인하세요. 클러스터를 생성하는 데 사용한 도구에 따라 현재 클러스터에 Amazon EKS 추가 기능이 유형이 설치되어 있지 않을 수 있습니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
addon.addonVersion --output text
```

버전 번호가 반환되는 경우 Amazon EKS 유형의 추가 기능이 클러스터에 설치되고 본 절차의 나머지 단계를 완료할 필요가 없습니다. 오류가 번호가 반환되는 경우 Amazon EKS 유형의 추가 기능이 클러스터에 설치되지 않습니다. 이 절차의 나머지 단계를 완료하여 설치하세요.

3. 현재 설치된 추가 기능의 구성을 저장합니다.

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

4. AWS CLI를 사용하여 추가 기능을 생성합니다. AWS Management Console 또는 `eksctl`를 사용하여 추가 기능을 만들려면 [the section called “추가 기능 생성”](#)을 참조하여 추가 기능 이름을 `vpc-cni`로 지정하세요. 다음 명령을 디바이스에 복사합니다. 필요에 따라 명령을 다음과 같이 수정한 다음에 수정한 명령을 실행합니다.

- *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.
- *v1.19.2-eksbuild.1*을 클러스터 버전의 최신 버전 테이블에 나와 있는 최신 버전으로 바꿉니다. 최신 버전 테이블은 [the section called “Amazon VPC CNI 버전”](#)을 참조하세요.
- *111122223333*을 계정 ID로, *AmazonEKSVPCCNIRole*을 생성한 [기존 IAM 역할](#) 이름으로 바꿉니다. 역할을 지정하려면 클러스터의 IAM OpenID Connect(OIDC) 제공자가 있어야 합니다. 클러스터의 해당 공급자가 이미 있는지 아니면 생성해야 하는지 결정하려면 [the section called “IAM OIDC 제공자”](#) 부분을 참조하세요.

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
v1.19.2-eksbuild.1 \
--service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole
```

Amazon EKS 추가 기능의 기본 설정과 충돌하는 사용자 지정 설정을 현재 추가 기능에 적용한 경우 생성이 실패할 수 있습니다. 생성에 실패하면 문제 해결에 도움이 될 수 있는 오류를 받게 됩니다. 또는 이전 명령에 `--resolve-conflicts OVERWRITE`을(를) 추가할 수 있습니다. 이렇게 하면 추가 기능이 기존 사용자 지정 설정을 덮어쓸 수 있습니다. 추가 기능을 만든 후에는 사용자 지정 설정으로 업데이트할 수 있습니다.

5. 클러스터의 Kubernetes 버전에 맞는 추가 기능의 최신 버전이 클러스터에 추가되었는지 확인합니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
addon.addonVersion --output text
```

추가 기능 생성이 완료되는 데 몇 초 정도 걸릴 수 있습니다.

예제 출력은 다음과 같습니다.

```
v1.19.2-eksbuild.1
```

6. 원래 추가 기능을 사용자 지정 설정했다면 Amazon EKS 추가 기능을 생성하기 전에 이전 단계에서 저장한 구성을 사용하여 EKS 추가 기능을 사용자 지정 설정으로 업데이트하세요. [the section called “업데이트\(EKS 추가 기능\)”](#) 섹션의 단계를 따르세요.
7. (선택사항)cni-metrics-helper 클러스터에 설치합니다. 지표 헬퍼는 네트워크 인터페이스 및 IP 주소 정보를 수집하고 클러스터 수준에서 지표를 집계하고 이 지표를 Amazon CloudWatch에 게시하는 데 사용할 수 있는 도구입니다. 자세한 내용은 GitHub에서 [metrics.proto](#)를 참조하세요.

Amazon VPC CNI 업데이트(Amazon EKS 추가 기능)

Kubernetes용 Amazon VPC CNI 플러그인 추가 기능의 Amazon EKS 유형을 업데이트합니다. 클러스터에 Amazon EKS 유형의 애드온을 추가하지 않은 경우 [the section called “생성”](#)에 따라 설치할 수 있습니다. 또는 [the section called “업데이트\(자체 관리형\)”](#)에 따라 다른 유형의 VPC CNI 설치를 업데이트합니다.

1. 클러스터에 설치된 추가 기능의 버전을 확인하세요. `<cluster-name>`을 클러스터 이름으로 바꿉니다.

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
"addon.addonVersion" --output text
```

예제 출력은 다음과 같습니다.

```
v1.16.4-eksbuild.2
```

버전을 [the section called “Amazon VPC CNI 버전”](#)의 최신 버전 표와 비교합니다. 반환된 버전이 최신 버전 표에 있는 클러스터의 Kubernetes 버전과 동일한 경우 클러스터에 이미 최신 버전이 설치되어 있으므로 이 절차의 나머지 부분을 완료하지 않아도 됩니다. 출력에 버전 번호 대신 오류가 표시되면 클러스터에 설치된 추가 기능의 Amazon EKS 유형이 없는 것입니다. 이 절차로 업데이트하려면 먼저 추가 기능을 생성해야 합니다. VPC CNI 추가 기능의 Amazon EKS 유형을 생성하려면 [the section called “생성”](#)를 따르세요.

2. 현재 설치된 추가 기능의 구성을 저장합니다.

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

3. AWS CLI를 사용하여 추가 기능 업데이트. AWS Management Console 또는 `eksctl`를 사용하여 추가 기능을 업데이트하려면 [the section called “추가 기능 업데이트”](#) 부분을 참조하세요. 다음 명령을 디바이스에 복사합니다. 필요에 따라 명령을 다음과 같이 수정한 다음에 수정한 명령을 실행합니다.

- `my-cluster`를 해당 클러스터의 이름으로 바꿉니다.
- `v1.19.2-eksbuild.1`을 클러스터 버전의 최신 버전 테이블에 나와 있는 최신 버전으로 바꿉니다.
- `111122223333`을 계정 ID로, `AmazonEKSVPCCNIRole`을 생성한 기존 IAM 역할 이름으로 바꿉니다. VPC CNI에 대한 IAM 역할을 생성하려면 [the section called “1단계: Kubernetes용 Amazon VPC CNI 플러그인 IAM 역할 생성”](#)을 참조하세요. 역할을 지정하려면 클러스터의 IAM OpenID

Connect(OIDC) 제공자가 있어야 합니다. 클러스터의 해당 공급자가 이미 있는지 아니면 생성해야 하는지 결정하려면 [the section called "IAM OIDC 제공자"](#) 부분을 참조하세요.

- `--resolve-conflicts PRESERVE` 옵션에서는 추가 기능의 기존 구성 값을 보존합니다. 추가 기능 설정에 사용자 지정 값을 설정하고 이 옵션을 사용하지 않는 경우 Amazon EKS에서는 기본 값으로 해당 값을 덮어씁니다. 이 옵션을 사용하는 경우에는 추가 기능을 업데이트하기 전에 프로덕션 클러스터에서 비프로덕션 클러스터의 필드 및 값 변경 사항을 테스트하는 것이 좋습니다. 이 값을 `OVERWRITE`로 변경하면 모든 설정이 Amazon EKS 기본값으로 변경됩니다. 설정에 사용자 지정 값을 설정한 경우 Amazon EKS 기본값으로 해당 값을 덮어쓸 수도 있습니다. 이 값을 `none`으로 변경하면 Amazon EKS에서는 설정의 값을 변경하지 않지만 업데이트에 실패할 수도 있습니다. 업데이트에 실패하면 충돌 해결에 도움이 되는 오류 메시지가 표시됩니다.
- 구성 설정을 업데이트하지 않는 경우, `--configuration-values '{"env": {"AWS_VPC_K8S_CNI_EXTERNALSNAT": "true"}}'`을(를) 명령에서 제거하세요. 구성 설정을 업데이트하는 경우 `"env": {"AWS_VPC_K8S_CNI_EXTERNALSNAT": "true"}`를 설정하려는 설정으로 바꾸세요. 이 예에서는 `AWS_VPC_K8S_CNI_EXTERNALSNAT` 환경 변수가 `true`로 설정되어 있습니다. 지정한 값은 구성 스키마에 유효해야 합니다. 구성 스키마를 모르는 경우 `aws eks describe-addon-configuration --addon-name vpc-cni --addon-version v1.19.2-eksbuild.1`을 실행하여 `v1.19.2-eksbuild.1`을 추가 기능의 버전 번호(구성을 확인하려는 추가 기능)로 변경하세요. 스키마가 출력에 반환됩니다. 기존 사용자 지정 구성이 있는데 이를 모두 제거하고 모든 설정의 값을 Amazon EKS 기본값으로 다시 설정하려면 명령에서 `"env": {"AWS_VPC_K8S_CNI_EXTERNALSNAT": "true"}`를 제거하여 `{}`을(를) 비워 두십시오. 각 설정에 대한 설명은 GitHub의 [CNI 구성 변수](#)를 참조하세요.

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version v1.19.2-eksbuild.1 \
  --service-account-role-arn arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole \
  --resolve-conflicts PRESERVE --configuration-values '{"env": {"AWS_VPC_K8S_CNI_EXTERNALSNAT": "true"}}'
```

업데이트가 완료되는 데 몇 초 정도 걸릴 수 있습니다.

4. 추가 기능 버전이 업데이트되었는지 확인합니다. `my-cluster`를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni
```

업데이트가 완료되는 데 몇 초 정도 걸릴 수 있습니다.

예제 출력은 다음과 같습니다.

```
{
  "addon": {
    "addonName": "vpc-cni",
    "clusterName": "my-cluster",
    "status": "ACTIVE",
    "addonVersion": "v1.19.2-eksbuild.1",
    "health": {
      "issues": []
    },
    "addonArn": "arn:aws:eks:region:111122223333:addon/my-cluster/vpc-cni/74c33d2f-b4dc-8718-56e7-9fdfa65d14a9",
    "createdAt": "2023-04-12T18:25:19.319000+00:00",
    "modifiedAt": "2023-04-12T18:40:28.683000+00:00",
    "serviceAccountRoleArn": "arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole",
    "tags": {},
    "configurationValues": "{\"env\":{\"AWS_VPC_K8S_CNI_EXTERNALSNAT\": \"true\"}}\"
  }
}
```

Amazon VPC CNI(자체 관리형 추가 기능) 업데이트

Important

자체 관리형 추가 기능 유형을 사용하는 대신 클러스터에 Amazon EKS 유형의 추가 기능을 추가하는 것이 좋습니다. 유형 간의 차이를 잘 모르는 경우 [the section called “Amazon EKS 추가 기능”](#) 부분을 참조하세요. Amazon EKS 추가 기능을 클러스터에 추가하는 방법에 대한 자세한 내용은 [the section called “추가 기능 생성”](#) 섹션을 참조하세요. Amazon EKS 추가 기능을 사용할 수 없는 경우, 사용할 수 없는 이유에 대한 문제를 [컨테이너 로드맵 GitHub 리포지토리에](#) 제출하는 것이 좋습니다.

1. 클러스터에 Amazon EKS 추가 기능 유형이 설치되어 있는지 확인하세요. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query
addon.addonVersion --output text
```

오류 메시자가 번호가 반환되는 경우 Amazon EKS 유형의 추가 기능이 클러스터에 설치되지 않습니다. 추가 기능을 자체 관리하려면 이 절차의 나머지 단계를 완료하여 추가 기능을 업데이트하세요. 버전 번호가 반환되는 경우 Amazon EKS 유형의 추가 기능이 클러스터에 설치됩니다. 업데이트하려면 이 절차를 수행하는 대신 [the section called “추가 기능 업데이트”](#)의 절차를 수행하세요. 추가 기능 유형 간의 차이를 잘 모르는 경우 [the section called “Amazon EKS 추가 기능”](#) 부분을 참조하세요.

2. 클러스터에 현재 설치된 컨테이너 이미지의 버전을 확인하세요.

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
cut -d : -f 3
```

예제 출력은 다음과 같습니다.

```
v1.19.2-eksbuild.1
```

출력에 빌드 번호가 포함되지 않을 수 있습니다.

3. 현재 설정을 백업하면 버전을 업데이트한 후에도 동일한 설정을 구성할 수 있습니다.

```
kubectl get daemonset aws-node -n kube-system -o yaml > aws-k8s-cni-old.yaml
```

사용 가능한 버전을 검토하고 업데이트하려는 버전의 변경 사항을 파악하려면 GitHub의 [releases](#) 내용을 참조하세요. GitHub에서 최신 버전을 사용할 수 있더라도 사용 가능한 최신 버전 표에 나열된 것과 동일한 major.minor.patch 버전으로 업데이트하는 것이 좋습니다. 사용 가능한 최신 버전 테이블은 [the section called “Amazon VPC CNI 버전”](#)을 참조하세요. 표에 나열된 빌드 버전은 GitHub에 나열된 자체 관리형 버전에 지정되어 있지 않습니다. 다음 옵션 중 하나로 작업을 완료하여 버전을 업데이트합니다.

- 추가 기능에 대한 사용자 지정 설정이 없는 경우 GitHub에서 업데이트하려는 [릴리스](#)에 해당하는 To apply this release: 제목 아래에 있는 명령을 실행하세요.
- 사용자 지정 설정이 있는 경우 다음 명령을 사용하여 매니페스트 파일을 다운로드하세요. <https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.19.2/config/master/aws-k8s-cni.yaml> 을 업데이트하려는 GitHub 릴리스의 URL로 변경하세요.

```
curl -O https://raw.githubusercontent.com/aws/amazon-vpc-cni-k8s/v1.19.2/config/
master/aws-k8s-cni.yaml
```

필요한 경우 만든 백업의 사용자 지정 설정으로 매니페스트를 수정한 다음 수정된 매니페스트를 클러스터에 적용합니다. 노드에서 이미지를 가져오는 프라이빗 Amazon EKS Amazon ECR 리포지토리에 액세스할 수 없는 경우, (매니페스트에서 `image:(으)`로 시작하는 줄 참조) 이미지를 다운로드하여 자체 리포지토리에 복사한 다음 리포지토리에서 이미지를 가져오도록 매니페스트를 수정해야 합니다. 자세한 내용은 [the section called “리포지토리에 이미지 복사”](#) 섹션을 참조하세요.

```
kubectl apply -f aws-k8s-cni.yaml
```

4. 이제 클러스터에 새 버전이 설치되어 있는지 확인합니다.

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |  
cut -d : -f 3
```

예제 출력은 다음과 같습니다.

```
v1.19.2
```

5. (선택사항) `cni-metrics-helper` 클러스터에 설치합니다. 지표 헬퍼는 네트워크 인터페이스 및 IP 주소 정보를 수집하고 클러스터 수준에서 지표를 집계하고 이 지표를 Amazon CloudWatch에 게시하는 데 사용할 수 있는 도구입니다. 자세한 내용은 GitHub에서 [metrics.proto](#)를 참조하세요.

IRSA를 사용하도록 Amazon VPC CNI 플러그인 구성

[Kubernetes용 Amazon VPC CNI 플러그인](#)은 Amazon EKS 클러스터에 있는 포트 네트워킹용 네트워킹 플러그인입니다. 플러그인은 Kubernetes 노드에 VPC IP 주소를 할당하고 각 노드의 포트에 대한 필수 네트워킹을 구성하는 역할을 합니다. 이 플러그인의 기능:

- AWS ID 및 액세스 관리(IAM) 권한이 필요합니다. 클러스터에서 IPv4 패밀리를 사용하는 경우 권한은 [AmazonEKS_CNI_Policy](#) AWS 관리형 정책에 지정됩니다. 클러스터에서 IPv6 패밀리를 사용하는 경우 권한이 생성하는 IAM 정책에 추가되어야 합니다. 지침은 [the section called “IPv6 패밀리를 사용하는 클러스터에 대한 IAM 정책 생성”](#) 섹션을 참조하세요. 정책을 Amazon EKS 노드 IAM 역할 또는 별도의 IAM 역할에 연결할 수 있습니다. 정책을 Amazon EKS 노드 IAM 역할에 연결하는 지침은 [the section called “노드 IAM 역할”](#)을 참조하세요. 이 주제에 설명된 대로 별도의 역할에 할당하는 것이 좋습니다.

- 배포되면 `aws-node`라는 Kubernetes 서비스 계정을 생성하고 해당 계정을 사용하도록 구성됩니다. 이 서비스 계정은 `aws-node`라는 Kubernetes `clusterrole`에 바인딩되어 있습니다. 이 역할에는 필요한 Kubernetes 권한이 할당되어 있습니다.

Note

IMDS에 대한 액세스 권한을 차단하지 않는 한 Kubernetes용 Amazon VPC CNI 플러그인의 포드는 [Amazon EKS 노드 IAM 역할](#)에 할당된 권한에 대한 액세스 권한이 있습니다. 자세한 내용은 [작업자 노드에 할당된 인스턴스 프로필에 대한 액세스 제한](#) 부분을 참조하세요.

- 기존 Amazon EKS 클러스터가 필요합니다. 배포하려면 [시작](#) 부분을 참조하세요.
- 클러스터에 대한 기존 AWS Identity and Access Management(IAM) OpenID Connect(OIDC) 공급자가 필요합니다. 이미 있는지 아니면 생성해야 하는지 확인하려면 [the section called “IAM OIDC 제공자”](#) 부분을 참조하세요.

1단계: Kubernetes용 Amazon VPC CNI 플러그인 IAM 역할 생성

1. 클러스터의 IP 패밀리를 확인합니다.

```
aws eks describe-cluster --name my-cluster | grep ipFamily
```

예제 출력은 다음과 같습니다.

```
"ipFamily": "ipv4"
```

출력에서 대신 `ipv6`가 반환될 수 있습니다.

2. IAM 역할을 생성합니다. `eksctl` 또는 `kubect1` 및 AWS CLI를 사용하여 IAM 역할을 생성할 수 있습니다.

`eksctl`

- IAM 역할을 생성하고 클러스터의 IP 패밀리와 일치하는 명령을 사용하여 역할에 IAM 정책을 연결합니다. 명령은 IAM 역할을 생성하는 AWS CloudFormation 스택을 생성 및 배포하고, 여기에 지정한 정책을 연결하고, 생성된 IAM 역할의 ARN으로 기존 `aws-node` Kubernetes 서비스 계정에 주석을 추가합니다.

- IPv4

*my-cluster*를 원하는 값으로 바꿉니다.

```
eksctl create iamserviceaccount \
  --name aws-node \
  --namespace kube-system \
  --cluster my-cluster \
  --role-name AmazonEKSVPCCNIRole \
  --attach-policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
  --override-existing-serviceaccounts \
  --approve
```

- IPv6

*my-cluster*를 원하는 값으로 바꿉니다. **111122223333**을 계정 ID로 바꾸고 ***AmazonEKS_CNI_IPv6_Policy***를 IPv6 정책 이름으로 바꿉니다. IPv6 정책이 없는 경우 [the section called “IPv6 패밀리를 사용하는 클러스터에 대한 IAM 정책 생성”](#) 섹션을 참조하여 생성합니다. 클러스터에서 IPv6를 사용하려면 여러 요구 사항을 충족해야 합니다. 자세한 내용은 [the section called “IPv6”](#) 섹션을 참조하세요.

```
eksctl create iamserviceaccount \
  --name aws-node \
  --namespace kube-system \
  --cluster my-cluster \
  --role-name AmazonEKSVPCCNIRole \
  --attach-policy-arn arn:aws:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy \
  --override-existing-serviceaccounts \
  --approve
```

kubectl 및 AWS CLI

- 클러스터의 OIDC 공급자 URL을 확인합니다.

```
aws eks describe-cluster --name my-cluster --query
  "cluster.identity.oidc.issuer" --output text
```

예제 출력은 다음과 같습니다.

```
https://oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE
```

출력이 반환되지 않은 경우 [해당 클러스터에 대한 IAM OIDC 공급자를 생성해야](#) 합니다.

- ii. 다음 콘텐츠를 *vpc-cni-trust-policy.json*이라는 파일에 복사합니다. **111122223333**을 이전 단계에서 반환된 출력의 계정 ID 및 **EXAMPLED539D4633E53DE1B71EXAMPLE**로 바꿉니다. *region-code*를 클러스터가 있는 AWS 리전으로 바꿉니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
          "oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:aws-node"
        }
      }
    }
  ]
}
```

- iii. 역할을 생성합니다. *AmazonEKSVPCCNIRole*을 선택한 이름으로 바꿀 수 있습니다.

```
aws iam create-role \
  --role-name AmazonEKSVPCCNIRole \
  --assume-role-policy-document file://vpc-cni-trust-policy.json
```

- iv. 필요한 IAM 정책을 역할에 연결합니다. 클러스터의 IP 패밀리와 일치하는 명령을 실행합니다.

- IPv4

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
  --role-name AmazonEKSVPCCNIRole
```

- IPv6

111122223333을 계정 ID로 바꾸고 **AmazonEKS_CNI_IPv6_Policy**를 IPv6 정책 이름으로 바꿉니다. IPv6 정책이 없는 경우 [the section called “IPv6 패밀리를 사용하는 클러스터에 대한 IAM 정책 생성”](#) 섹션을 참조하여 생성합니다. 클러스터에서 IPv6를 사용하려면 여러 요구 사항을 충족해야 합니다. 자세한 내용은 [the section called “IPv6”](#) 섹션을 참조하세요.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy \
  --role-name AmazonEKSVPCCNIRole
```

- v. 다음 명령을 실행하여 이전에 생성한 IAM 역할의 ARN을 사용하여 aws-node 서비스 계정에 주석을 추가합니다. **##** **#**을 사용자의 값으로 바꿉니다.

```
kubectl annotate serviceaccount \
  -n kube-system aws-node \
  eks.amazonaws.com/role-arn=arn:aws:iam::111122223333:role/
AmazonEKSVPCCNIRole
```

3. (선택 사항) Kubernetes 서비스 계정에서 사용하는 AWS 보안 토큰 Kubernetes 서비스 엔드포인트 유형을 구성합니다. 자세한 내용은 [the section called “STS 엔드포인트”](#) 섹션을 참조하세요.

2단계: Kubernetes용 Amazon VPC CNI 플러그인 포드 재배포

1. 서비스 계정에 연결된 기존 포드를 삭제하고 다시 생성하여 자격 증명 환경 변수를 적용합니다. 주석은 현재 주석 없이 실행 중인 포드에는 적용되지 않습니다. 다음 명령은 기존의 aws-node DaemonSet 포드를 삭제하고 서비스 계정 주석을 사용하여 배포합니다.

```
kubectl delete Pods -n kube-system -l k8s-app=aws-node
```

2. 포드가 모두 다시 시작되었는지 확인합니다.

```
kubectl get pods -n kube-system -l k8s-app=aws-node
```

3. 포드 중 하나를 설명하고 AWS_WEB_IDENTITY_TOKEN_FILE 및 AWS_ROLE_ARN 환경 변수가 있는지 확인합니다. **cpjw7**을 이전 단계의 출력에서 반환된 포드 중 하나의 이름으로 바꿉니다.

```
kubectl describe pod -n kube-system aws-node-cpjw7 | grep 'AWS_ROLE_ARN:\|
AWS_WEB_IDENTITY_TOKEN_FILE:'
```

예제 출력은 다음과 같습니다.

```
AWS_ROLE_ARN:                arn:aws:iam::111122223333:role/AmazonEKSVPCCNIRole
  AWS_WEB_IDENTITY_TOKEN_FILE: /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
  AWS_ROLE_ARN:                arn:aws:iam::111122223333:role/
AmazonEKSVPCCNIRole
  AWS_WEB_IDENTITY_TOKEN_FILE: /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
```

포드에 두 개의 컨테이너가 포함되어 있으므로 두 세트의 중복 결과가 반환됩니다. 두 컨테이너의 값이 같은 경우입니다.

포드에서 AWS 리전 엔드포인트를 사용하고 있는 경우 이전 출력에서 다음 줄도 반환됩니다.

```
AWS_STS_REGIONAL_ENDPOINTS=regional
```

3단계: 노드 IAM 역할에서 CNI 정책 제거

현재 [Amazon EKS 노드 IAM 역할](#)에 AmazonEKS_CNI_Policy IAM(IPv4) 정책 또는 [IPv6 정책](#)이 연결되어 있고 별도의 IAM 역할을 생성하여 대신 정책을 연결한 후 aws-node Kubernetes 서비스 계정에 할당했다면 클러스터의 IP 패밀리와 일치하는 AWS CLI 명령으로 노드 역할에서 정책을 제거하는 것이 좋습니다. *AmazonEKSNodeRole*을 노드 역할의 이름으로 바꿉니다.

- IPv4

```
aws iam detach-role-policy --role-name AmazonEKSNodeRole --policy-arn
arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
```

- IPv6

*111122223333*을 계정 ID로 바꾸고 *AmazonEKS_CNI_IPv6_Policy*를 IPv6 정책 이름으로 바꿉니다.

```
aws iam detach-role-policy --role-name AmazonEKSNodeRole --policy-arn
arn:aws:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy
```

IPv6 패밀리를 사용하는 클러스터에 대한 IAM 정책 생성

IPv6 패밀리를 사용하는 클러스터를 생성했고 클러스터에 버전 1.10.1 이상의 Kubernetes용 Amazon VPC CNI 플러그인 추가 기능이 구성되어 있으면 IAM 역할에 할당할 수 있는 IAM 정책을 생성해야 합니다. 생성할 때 IPv6 패밀리로 구성하지 않은 기존 클러스터가 있는 경우 IPv6을 사용하려면 새 클러스터를 생성해야 합니다. 클러스터에서 IPv6 사용에 대한 자세한 내용을 알아보려면 [the section called “IPv6”](#) 섹션을 참조하세요.

1. 다음 텍스트를 복사해 vpc-cni-ipv6-policy.json 파일에 저장합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AssignIpv6Addresses",
        "ec2:DescribeInstances",
        "ec2:DescribeTags",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeInstanceTypes"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateTags"
      ],
      "Resource": [
        "arn:aws:ec2:*:*:network-interface/*"
      ]
    }
  ]
}
```

2. IAM 정책을 생성합니다.

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-document
file://vpc-cni-ipv6-policy.json
```

VPC CNI 모드 및 구성에 대해 알아보기

Kubernetes용 Amazon VPC CNI 플러그인은 포드에 대한 네트워킹을 제공합니다. 다음 테이블을 사용하여 사용 가능한 네트워킹 기능을 자세히 알아봅니다.

네트워킹 기능	자세히 알아보기
클러스터, 포드 및 서비스에 IPv6 주소를 할당하도록 클러스터를 구성합니다.	the section called “IPv6”
포드에 IPv4 소스 네트워크 주소 변환 사용	the section called “아웃바운드 트래픽”
포드에서 송수신하는 네트워크 트래픽 제한	the section called “트래픽 제한”
노드의 보조 네트워크 인터페이스 사용자 지정	the section called “사용자 지정 네트워킹”
노드의 IP 주소 증가	the section called “IP 주소 증가”
포드 네트워크 트래픽에 대해 보안 그룹 사용	the section called “포드의 보안 그룹”
포드에 여러 네트워크 인터페이스 사용	the section called “다중 인터페이스”

클러스터, 포드 및 서비스에 대한 IPv6 주소에 대해 알아보기

적용 대상: Amazon EC2 인스턴스가 있는 포드와 Fargate 포드

기본적으로 Kubernetes는 포드와 서비스에 IPv4 주소를 할당합니다. 포드와 서비스에 IPv4 주소를 할당하는 대신 IPv6 주소를 할당하도록 클러스터를 구성할 수 있습니다. Kubernetes는 듀얼 스택 포드 또는 서비스를 지원하지만, Amazon EKS는 듀얼 스택 포드 또는 서비스를 지원하지 않습니다. 결과적으로 포드와 서비스에 IPv4 및 IPv6 주소를 모두 할당할 수 없습니다.

클러스터를 생성할 때 클러스터에 사용할 IP 패밀리를 선택합니다. 클러스터를 생성한 후에는 패밀리를 변경할 수 없습니다.

Amazon EKS IPv6 클러스터를 배포하기 위한 자습서는 [the section called “배포”](#)을 참조하세요.

다음은 기능 사용 시 고려 사항입니다.

IPv6 기능 지원

- Windows 지원 없음: Windows 포드와 서비스는 지원되지 않습니다.

- Nitro 기반 EC2 노드 필요: IPv6는 AWS Nitro 기반 Amazon EC2 또는 Fargate 노드에서만 사용할 수 있습니다.
- 지원되는 EC2 및 Fargate 노드: Amazon EC2 노드 및 Fargate 노드에서 [the section called “포드의 보안 그룹”](#)와 함께 IPv6를 사용할 수 있습니다.
- Outposts가 지원되지 않음: [AWSOutposts의 Amazon EKS](#)와 IPv6는 함께 사용할 수 없습니다.
- Lustre용 FSx는 지원되지 않음: [the section called “Amazon FSx for Lustre”](#)는 지원되지 않습니다.
- 인스턴스 메타데이터 서비스가 지원되지 않음: Amazon EC2 [인스턴스 메타데이터 서비스](#) IPv6 엔드포인트의 사용은 Amazon EKS에서 지원되지 않습니다.
- 사용자 지정 네트워킹이 지원되지 않음: 이전에 [the section called “사용자 지정 네트워킹”](#)를 사용하여 IP 주소 고갈을 완화한 경우, 대신 IPv6를 사용할 수 있습니다. IPv6에서는 사용자 지정 네트워킹을 사용할 수 없습니다. 네트워크 격리를 위해 사용자 정의 네트워킹을 사용하는 경우 클러스터에 사용자 정의 네트워킹 및 IPv4 패밀리를 계속 사용해야 할 수 있습니다.

IP 주소 할당

- Kubernetes 서비스: Kubernetes 서비스에는 IPv6 주소만 할당됩니다. IPv4 주소는 할당되지 않습니다.
- 포트: 포트에는 IPv6 주소와 호스트-로컬 IPv4 주소가 할당됩니다. 호스트-로컬 IPv4 주소는 VPC CNI와 연결된 호스트-로컬 CNI 플러그인을 사용하여 할당되며 주소는 Kubernetes 컨트롤 플레인에 보고되지 않습니다. 포트가 다른 Amazon VPC 또는 인터넷의 외부 IPv4 리소스와 통신해야 하는 경우에만 사용됩니다. 호스트-로컬 IPv4 주소는 VPC CNI를 통해 워커 노드 기본 ENI의 기본 IPv4 주소에 SNAT 처리됩니다.
- 포트 및 서비스: 포트와 서비스에는 IPv6 주소만 할당됩니다. IPv4 주소는 할당되지 않습니다. 포트는 인스턴스 자체의 NAT를 통해 IPv4 엔드포인트와 통신할 수 있으므로 [DNS64 및 NAT64](#)가 필요하지 않습니다. 트래픽에 퍼블릭 IP 주소가 필요한 경우 트래픽은 퍼블릭 IP로 변환된 소스 네트워크 주소입니다.
- 라우팅 주소: 포트의 소스 IPv6 주소는 VPC 외부에서 통신할 때 노드의 IPv6 주소로 변환된 소스 네트워크 주소가 아닙니다. 인터넷 게이트웨이 또는 외부 전용 인터넷 게이트웨이를 사용하여 라우팅됩니다.
- 노드: 모든 노드에 IPv4 및 IPv6 주소가 할당됩니다.
- Fargate 포트: 각 Fargate 포트는 배포된 서브넷에 대해 지정된 CIDR에서 IPv6 주소를 수신합니다. Fargate 포트를 실행하는 기본 하드웨어 장치는 하드웨어 장치가 배포된 서브넷에 할당된 CIDR에서 고유한 IPv4 및 IPv6 주소를 가져옵니다.

EKS에서 IPv6를 사용하는 방법

- **새 클러스터 생성:** 새 클러스터를 생성하고 해당 클러스터에 IPv6 패밀리를 사용하도록 지정해야 합니다. 이전 버전에서 업데이트한 클러스터에 IPv6 패밀리를 사용 설정할 수 없습니다. 새 클러스터를 생성하는 방법에 대한 지침은 [Considerations](#)을 참조하세요.
- **최신 VPC CNI 사용:** Amazon VPC CNI 버전 1.10.1 이상을 배포합니다. 이 버전 이상은 기본적으로 배포됩니다. 추가 기능을 배포하면 클러스터의 모든 노드 그룹에 있는 모든 노드를 제거하지 않으면 1.10.1 이전 버전에 Amazon VPC CNI 추가 기능을 다운로드할 수 없습니다.
- **IPv6용 VPC CNI 구성:** Amazon EC2 노드를 사용하는 경우 IP 접두사 위임 및 IPv6를 사용하여 Amazon VPC CNI 추가 기능을 구성해야 합니다. 클러스터를 생성할 때 IPv6 패밀리를 선택하면 1.10.1 버전의 추가 기능이 이 구성으로 기본 설정됩니다. 이는 자체 관리형 또는 Amazon EKS 추가 기능 모두에 해당됩니다. IP 접두사 위임에 대한 자세한 내용은 [the section called “IP 주소 증가”](#) 섹션을 참조하세요.
- **IPv4 및 IPv6 주소를 구성:** 클러스터를 생성할 때 지정한 VPC 및 서브넷에는 지정한 VPC 및 서브넷에 할당된 IPv6 CIDR 블록이 있어야 합니다. 또한 할당된 IPv4 CIDR 블록도 있어야 합니다. IPv6만 사용하려는 경우에도 VPC가 작동하려면 IPv4 CIDR 블록이 필요하기 때문입니다. 자세한 내용은 Amazon VPC 사용 설명서의 [IPv6 CIDR 블록을 VPC와 연결](#)을 참조하세요.
- **노드에 IPv6 주소 자동 할당:** 노드를 생성할 때 IPv6 주소를 자동 할당하도록 구성된 서브넷을 지정해야 합니다. 그렇지 않으면 노드를 배포할 수 없습니다. 기본적으로 이 구성은 사용 중지되어 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 [서브넷의 IPv6 주소 지정 속성 수정](#)을 참조하세요.
- **IPv6를 사용할 라우트 테이블 설정:** 서브넷에 할당된 라우팅 테이블에는 IPv6 주소에 대한 경로가 있어야 합니다. 자세한 내용은 Amazon VPC 사용 설명서의 [IPv6로 마이그레이션](#)을 참조하세요.
- **IPv6에 대한 보안 그룹 설정:** 보안 그룹은 IPv6 주소를 허용해야 합니다. 자세한 내용은 Amazon VPC 사용 설명서의 [IPv6로 마이그레이션](#)을 참조하세요.
- **로드 밸런서 설정:** AWS Load Balancer Controller 버전 2.3.1 이상에서 [the section called “애플리케이션 로드 밸런싱”](#)을 사용하여 HTTP 애플리케이션의 로드 밸런싱을 수행하거나 [the section called “네트워크 로드 밸런싱”](#)을 사용하여 네트워크 트래픽을 인스턴스 모드가 아닌 IP 모드의 로드 밸런서가 있는 IPv6 포드로 로드 밸런싱합니다. 자세한 내용은 [the section called “AWS 로드 밸런서 컨트롤러”](#) 섹션을 참조하세요.
- **IPv6 IAM 정책을 추가:** IPv6 IAM 정책을 노드 IAM 또는 CNI IAM 역할에 연결해야 합니다. 두 가지 중 CNI IAM 역할에 연결하는 것이 좋습니다. 자세한 내용은 [the section called “IPv6 패밀리를 사용하는 클러스터에 대한 IAM 정책 생성”](#) 및 [the section called “1단계: Kubernetes용 Amazon VPC CNI 플러그인 IAM 역할 생성”](#) 섹션을 참조하세요.

- 모든 구성 요소 평가: AWS 클러스터를 배포하기 전에 통합하는 애플리케이션, Amazon EKS 애드온 및 IPv6 서비스에 대한 철저한 평가를 수행하세요. 이는 IPv6에서 모든 것이 예상대로 작동하는지 확인하기 위한 것입니다.
- **BootstrapArguments** 자체 관리 노드 그룹 추가: IPv6 패밀리를 사용하는 클러스터에서 자체 관리형 노드 그룹을 생성하는 경우 사용자 데이터에는 노드 시작 시 실행되는 [bootstrap.sh](#) 파일에 대한 다음 BootstrapArguments가 포함되어야 합니다. *your-cidr*를 클러스터 VPC의 IPv6 CIDR 범위로 교체합니다.

```
--ip-family ipv6 --service-ipv6-cidr your-cidr
```

클러스터의 IPv6 CIDR 범위를 모르는 경우, 다음 명령을 사용하여 확인할 수 있습니다(AWS CLI 버전 2.4.9 이상 필요).

```
aws eks describe-cluster --name my-cluster --query
cluster.kubernetesNetworkConfig.serviceIpv6Cidr --output text
```

Amazon EKS **IPv6** 클러스터 및 관리형 Amazon Linux 노드 배포

이 튜토리얼에서는 IPv6 Amazon VPC, IPv6 패밀리가 있는 Amazon EKS 클러스터 및 Amazon EC2 Amazon Linux 노드가 있는 관리형 노드 그룹을 배포합니다. IPv6 클러스터에는 Amazon EC2 Windows 노드를 배포할 수 없습니다. Fargate 노드를 클러스터에 배포할 수도 있지만 이 항목에서는 단순화를 위해 해당 지침을 제공하지 않습니다.

사전 조건

자습서를 시작하기 전에 다음을 완료합니다.

Amazon EKS 클러스터를 생성하고 관리할 때 필요한 다음 도구 및 리소스를 설치하고 구성합니다.

- 모든 설정을 숙지하고 요구 사항을 충족하는 설정으로 클러스터를 배포하는 것이 좋습니다. 자세한 내용은 [the section called “클러스터 생성”](#), [the section called “관리형 노드 그룹”](#) 그리고 이 주제에 대한 [고려 사항](#)을 참조하세요. 클러스터를 생성할 때 일부 설정만 사용 설정할 수 있습니다.
- kubectl 명령줄 도구는 장치 또는 AWS CloudShell에 설치됩니다. 버전은 클러스터의 Kubernetes 버전과 동일하거나 최대 하나 이전 또는 이후의 마이너 버전일 수 있습니다. 예를 들어, 클러스터 버전이 1.29인 경우 kubectl 버전 1.28, 1.29 또는 1.30를 함께 사용할 수 있습니다. kubectl을 설치하거나 업그레이드하려면 [the section called “kubectl 및 eksctl 설정”](#) 부분을 참조하세요.

- 사용하는 IAM 보안 주체에 Amazon EKS IAM 역할, 서비스 연결 역할, AWS CloudFormation, VPC 및 관련 리소스를 사용할 수 있는 권한이 있어야 합니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 사용](#)을 참조하세요.
- eksctl을 사용하는 경우 컴퓨터에 버전 0.210.0 이상을 설치합니다. 설치하거나 업데이트하려면 eksctl 설명서에서 [설치](#)를 참조하세요.
- 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요. AWS CloudShell을 사용하는 경우 AWS CloudShell에 설치된 기본 AWS CLI 버전이 이전 버전일 수 있으므로 [AWS CLI 버전 2.12.3 이상 또는 1.27.160 이상을 설치](#)해야 할 수 있습니다.

eksctl 또는 CLI를 사용하여 IPv6 클러스터를 배포할 수 있습니다.

eksctl을 사용하여 IPv6 클러스터 배포

- ipv6-cluster.yaml 파일을 생성합니다. 다음 명령을 디바이스에 복사합니다. 필요에 따라 명령을 다음과 같이 수정한 다음에 수정한 명령을 실행합니다.
 - **my-cluster**를 클러스터의 이름으로 바꿉니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다.
 - **region-code**를 Amazon EKS에서 지원하는 AWS 리전으로 바꿉니다. AWS 리전 목록은 AWS General Reference 가이드의 [Amazon EKS endpoints and quotas](#)를 참조하세요.
 - 클러스터의 버전이 포함된 version의 값입니다. 자세한 내용은 [the section called “Kubernetes 버전”](#) 섹션을 참조하세요.
 - **my-nodegroup**을 노드 그룹의 이름으로 바꿉니다. 노드 그룹 이름은 63자를 초과할 수 없습니다. 문자나 숫자로 시작하되, 나머지 문자의 경우 하이픈과 밑줄을 포함할 수 있습니다.
 - **t3.medium**을 [AWS Nitro 시스템 인스턴스 유형](#)으로 바꿉니다.

```
cat >ipv6-cluster.yaml <<EOF
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
```

```
metadata:
  name: my-cluster
  region: region-code
  version: "X.XX"

kubernetesNetworkConfig:
  ipFamily: IPv6

addons:
- name: vpc-cni
  version: latest
- name: coredns
  version: latest
- name: kube-proxy
  version: latest

iam:
  withOIDC: true

managedNodeGroups:
- name: my-nodegroup
  instanceType: t3.medium
EOF
```

b. 클러스터를 생성합니다.

```
eksctl create cluster -f ipv6-cluster.yaml
```

클러스터 생성에 몇 분 정도 걸립니다. 다음 출력과 유사한 출력의 마지막 줄이 나타날 때까지 진행하지 마세요.

```
[...]
[#] EKS cluster "my-cluster" in "region-code" region is ready
```

c. 기본 포드에 IPv6 주소가 할당되었는지 확인합니다.

```
kubectl get pods -n kube-system -o wide
```

예제 출력은 다음과 같습니다.

NAME	READY	STATUS	RESTARTS	AGE	IP	
	NODE					NOMINATED NODE
READINESS GATES						
aws-node-rslts	1/1	Running	1	5m36s		
2600:1f13:b66:8200:11a5:ade0:c590:6ac8					ip-192-168-34-75.region-	
code.compute.internal	<none>		<none>			
aws-node-t74jh	1/1	Running	0	5m32s		
2600:1f13:b66:8203:4516:2080:8ced:1ca9					ip-192-168-253-70.region-	
code.compute.internal	<none>		<none>			
coredns-85d5b4454c-cw7w2	1/1	Running	0	56m		
2600:1f13:b66:8203:34e5::					ip-192-168-253-70.region-	
code.compute.internal	<none>		<none>			
coredns-85d5b4454c-tx6n8	1/1	Running	0	56m		
2600:1f13:b66:8203:34e5::1					ip-192-168-253-70.region-	
code.compute.internal	<none>		<none>			
kube-proxy-btpbk	1/1	Running	0	5m36s		
2600:1f13:b66:8200:11a5:ade0:c590:6ac8					ip-192-168-34-75.region-	
code.compute.internal	<none>		<none>			
kube-proxy-jjk2g	1/1	Running	0	5m33s		
2600:1f13:b66:8203:4516:2080:8ced:1ca9					ip-192-168-253-70.region-	
code.compute.internal	<none>		<none>			

d. 기본 서비스에 IPv6 주소가 할당되었는지 확인합니다.

```
kubectl get services -n kube-system -o wide
```

예제 출력은 다음과 같습니다.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	
SELECTOR						
kube-dns	ClusterIP	fd30:3087:b6c2::a	<none>	53/UDP,53/TCP	57m	k8s-
app=kube-dns						

- e. (선택 사항) [샘플 애플리케이션을 배포](#)하거나 [AWS Load Balancer Controller](#)와 샘플 애플리케이션을 배포하여 [the section called “애플리케이션 로드 밸런싱”](#)를 사용하는 HTTP 애플리케이션 또는 [the section called “네트워크 로드 밸런싱”](#)을 사용하는 네트워크 트래픽을 IPv6 포드로 로드 밸런싱합니다.
- f. 이 튜토리얼용으로 생성한 클러스터 및 노드 사용을 마친 후 다음 명령을 사용하여 생성한 리소스를 정리해야 합니다.

```
eksctl delete cluster my-cluster
```

AWS CLI를 사용하여 IPv6 클러스터 배포

⚠ Important

- 이 절차의 모든 단계를 동일한 사용자로 완료해야 합니다. 현재 사용자를 확인하려면 다음 명령을 실행합니다.

```
aws sts get-caller-identity
```

- 이 절차의 모든 단계를 동일한 셸로 완료해야 합니다. 여러 단계에서는 이전 단계에서 설정한 변수를 사용합니다. 변수 값이 다른 셸에서 설정되면 변수를 사용하는 단계가 제대로 작동하지 않습니다. [AWS CloudShell](#)을 사용하여 다음 절차를 완료하는 경우 약 20~30분 동안 키보드나 포인터를 사용하여 상호 작용하지 않으면 셸 세션이 종료된다는 점을 기억하세요. 실행 중인 프로세스는 상호 작용으로 계산되지 않습니다.
- 지침은 Bash 셸용으로 작성되었으며 다른 셸에서 조정해야 할 수도 있습니다.

이 절차 단계의 모든 **##** **#**을 고유한 값으로 바꿉니다.

- 다음 명령을 실행하여 이후 단계에서 사용되는 일부 변수를 설정합니다. **region-code**를 리소스를 배포할 AWS 리전으로 바꿉니다. 값은 Amazon EKS에서 지원하는 모든 AWS 리전일 수 있습니다. AWS 리전 목록은 AWS General Reference 가이드의 [Amazon EKS endpoints and quotas](#)를 참조하세요. **my-cluster**를 클러스터의 이름으로 바꿉니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다. **my-nodegroup**을 노드 그룹의 이름으로 바꿉니다. 노드 그룹 이름은 63자를 초과할 수 없습니다. 문자나 숫자로 시작하되, 나머지 문자의 경우 하이픈과 밑줄을 포함할 수 있습니다. **111122223333**을 계정 ID로 바꿉니다.

```
export region_code=region-code
export cluster_name=my-cluster
export nodegroup_name=my-nodegroup
export account_id=111122223333
```

- Amazon EKS 및 IPv6 요구 사항을 충족하는 퍼블릭 및 프라이빗 서브넷이 있는 Amazon VPC를 생성합니다.

- i. 다음 명령을 실행하여 AWS CloudFormation 스택 이름에 대한 변수를 설정합니다. 선택하는 이름으로 *my-eks-ipv6-vpc*를 바꿀 수 있습니다.

```
export vpc_stack_name=my-eks-ipv6-vpc
```

- ii. AWS CloudFormation 템플릿을 사용하여 IPv6 VPC를 만듭니다.

```
aws cloudformation create-stack --region $region_code --stack-name $vpc_stack_name \
  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-ipv6-vpc-public-private-subnets.yaml
```

스택을 만드는 데 몇 분 정도 걸립니다. 다음 명령을 실행합니다. 명령의 출력이 CREATE_COMPLETE가 될 때까지 다음 단계를 계속하지 마세요.

```
aws cloudformation describe-stacks --region $region_code --stack-name
$vpc_stack_name --query Stacks[].StackStatus --output text
```

- iii. 생성된 퍼블릭 서브넷의 ID를 검색합니다.

```
aws cloudformation describe-stacks --region $region_code --stack-name
$vpc_stack_name \
  --query='Stacks[].Outputs[?OutputKey==`SubnetsPublic`].OutputValue' --output
text
```

예제 출력은 다음과 같습니다.

```
subnet-0a1a56c486EXAMPLE,subnet-099e6ca77aEXAMPLE
```

- iv. 생성된 퍼블릭 서브넷에 대해 IPv6 주소 자동 할당 옵션을 사용 설정합니다.

```
aws ec2 modify-subnet-attribute --region $region_code --subnet-id
subnet-0a1a56c486EXAMPLE --assign-ipv6-address-on-creation
aws ec2 modify-subnet-attribute --region $region_code --subnet-id
subnet-099e6ca77aEXAMPLE --assign-ipv6-address-on-creation
```

- v. 배포된 AWS CloudFormation 스택에서 템플릿으로 생성된 서브넷 및 보안 그룹의 이름을 검색하고 이후 단계에서 사용할 수 있도록 변수에 저장합니다.

```

security_groups=$(aws cloudformation describe-stacks --region $region_code --
stack-name $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SecurityGroups`].OutputValue' --output
text)

public_subnets=$(aws cloudformation describe-stacks --region $region_code --stack-
name $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SubnetsPublic`].OutputValue' --output
text)

private_subnets=$(aws cloudformation describe-stacks --region $region_code --
stack-name $vpc_stack_name \
    --query='Stacks[].Outputs[?OutputKey==`SubnetsPrivate`].OutputValue' --output
text)

subnets=${public_subnets},${private_subnets}

```

- c. 클러스터 IAM 역할을 생성하고 필요한 Amazon EKS IAM 관리형 정책을 연결합니다. Amazon EKS에서 관리하는 Kubernetes 클러스터는 사용자 대신 다른 AWS 서비스를 직접 호출하여 서비스에 사용하는 리소스를 관리합니다.
 - i. 다음 명령을 실행해 eks-cluster-role-trust-policy.json 파일을 생성합니다.

```

cat >eks-cluster-role-trust-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF

```

- ii. 다음 명령을 실행하여 역할 이름에 대한 변수를 설정합니다. *myAmazonEKSClusterRole*을 선택한 이름으로 바꿀 수 있습니다.

```

export cluster_role_name=myAmazonEKSClusterRole

```

iii. 역할을 생성합니다.

```
aws iam create-role --role-name $cluster_role_name --assume-role-policy-document
file://"eks-cluster-role-trust-policy.json"
```

iv. IAM 역할의 ARN을 검색하고 이후 단계를 위해 변수에 저장합니다.

```
CLUSTER_IAM_ROLE=$(aws iam get-role --role-name $cluster_role_name --
query="Role.Arn" --output text)
```

v. 필요한 Amazon EKS 관리형 IAM 정책을 역할에 연결합니다.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSClusterPolicy --role-name $cluster_role_name
```

d. 클러스터를 생성합니다.

```
aws eks create-cluster --region $region_code --name $cluster_name --kubernetes-
version 1.XX \
  --role-arn $CLUSTER_IAM_ROLE --resources-vpc-config subnetIds=
$subnets,securityGroupIds=$security_groups \
  --kubernetes-network-config ipFamily=ipv6
```

- i. 참고: 요청의 가용 영역 중 하나에 Amazon EKS 클러스터를 생성하는 데 충분한 용량이 없다는 오류가 표시될 수 있습니다. 이 경우 오류 출력에는 새 클러스터를 지원할 수 있는 가용 영역이 포함됩니다. 사용자 계정의 지원 가용 영역에 있는 2개 이상의 서브넷을 사용하여 클러스터를 다시 생성합니다. 자세한 내용은 [the section called “용량 부족”](#) 섹션을 참조하세요.

클러스터를 생성하는 데 몇 분 정도 걸립니다. 다음 명령을 실행합니다. 명령의 출력이 ACTIVE가 될 때까지 다음 단계를 계속하지 마세요.

```
aws eks describe-cluster --region $region_code --name $cluster_name --query
cluster.status
```

e. 클러스터와 통신할 수 있도록 클러스터에 대한 kubeconfig 파일을 생성하거나 업데이트합니다.

```
aws eks update-kubeconfig --region $region_code --name $cluster_name
```

기본적으로 config 파일이 ~/.kube에 생성되거나 새 클러스터의 구성이 ~/.kube의 기존 config 파일에 추가됩니다.

f. 노드 IAM 역할을 생성합니다.

i. 다음 명령을 실행해 vpc-cni-ipv6-policy.json 파일을 생성합니다.

```
cat >vpc-cni-ipv6-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:AssignIpv6Addresses",
        "ec2:DescribeInstances",
        "ec2:DescribeTags",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeInstanceTypes"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateTags"
      ],
      "Resource": [
        "arn:aws:ec2:*:*:network-interface/*"
      ]
    }
  ]
}
EOF
```

ii. IAM 정책을 생성합니다.

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-document
file://vpc-cni-ipv6-policy.json
```

iii. 다음 명령을 실행해 node-role-trust-relationship.json 파일을 생성합니다.

```
cat >node-role-trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {

```

```

    "Effect": "Allow",
    "Principal": {
      "Service": "ec2.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
  }
]
}
EOF

```

- iv. 다음 명령을 실행하여 역할 이름에 대한 변수를 설정합니다. *AmazonEKSNodeRole*을 선택한 이름으로 바꿀 수 있습니다.

```
export node_role_name=AmazonEKSNodeRole
```

- v. IAM 역할을 생성합니다.

```
aws iam create-role --role-name $node_role_name --assume-role-policy-document
file://"node-role-trust-relationship.json"
```

- vi. IAM 정책을 IAM 역할에 연결합니다.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::$account_id:policy/
AmazonEKS_CNI_IPv6_Policy \
  --role-name $node_role_name
```

Important

이 자습서에서는 단순화를 위해 이 IAM 역할에 정책이 연결되어 있습니다. 그러나 프로덕션 클러스터에서는 정책을 별도의 IAM 역할에 연결하는 것이 좋습니다. 자세한 내용은 [the section called “IRSA에 대한 구성”](#) 섹션을 참조하세요.

- vii. 필요한 2개의 관리형 IAM 정책을 IAM 역할에 연결합니다.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSWorkerNodePolicy \
  --role-name $node_role_name
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEC2ContainerRegistryReadOnly \
  --role-name $node_role_name
```

viii IAM 역할의 ARN을 검색하고 이후 단계를 위해 변수에 저장합니다.

```
node_iam_role=$(aws iam get-role --role-name $node_role_name --query="Role.Arn" --output text)
```

g. 관리형 노드 그룹을 생성합니다.

i. 이전 단계에서 생성한 서브넷의 ID를 확인합니다.

```
echo $subnets
```

예제 출력은 다음과 같습니다.

```
subnet-0a1a56c486EXAMPLE, subnet-099e6ca77aEXAMPLE, subnet-0377963d69EXAMPLE, subnet-0c05f819d5EXAMPLE
```

ii. 노드 그룹을 생성합니다. *0a1a56c486EXAMPLE*, *099e6ca77aEXAMPLE*, *0377963d69EXAMPLE*, 및 *0c05f819d5EXAMPLE*을 이전 단계의 출력에서 반환된 값으로 바꿉니다. 다음 명령의 이전 출력에서 서브넷 ID 사이의 쉼표를 제거해야 합니다. *t3.medium*을 [AWS Nitro 시스템 인스턴스 유형](#)으로 바꿀 수 있습니다.

```
aws eks create-nodegroup --region $region_code --cluster-name $cluster_name --nodegroup-name $nodegroup_name \
    --subnets subnet-0a1a56c486EXAMPLE subnet-099e6ca77aEXAMPLE \
    subnet-0377963d69EXAMPLE subnet-0c05f819d5EXAMPLE \
    --instance-types t3.medium --node-role $node_iam_role
```

노드 그룹을 만드는 데 몇 분 정도 걸립니다. 다음 명령을 실행합니다. 반환되는 출력이 ACTIVE가 되면 다음 단계로 진행하세요.

```
aws eks describe-nodegroup --region $region_code --cluster-name $cluster_name --nodegroup-name $nodegroup_name \
    --query nodegroup.status --output text
```

h. IP 열에서 기본 포트에 IPv6 주소가 할당되었는지 확인합니다.

```
kubectl get pods -n kube-system -o wide
```

예제 출력은 다음과 같습니다.

NAME	READY	STATUS	RESTARTS	AGE	IP	
	NODE					NOMINATED NODE
READINESS GATES						
aws-node-rslts	1/1	Running	1	5m36s		
2600:1f13:b66:8200:11a5:ade0:c590:6ac8					ip-192-168-34-75.region-	
code.compute.internal	<none>		<none>			
aws-node-t74jh	1/1	Running	0	5m32s		
2600:1f13:b66:8203:4516:2080:8ced:1ca9					ip-192-168-253-70.region-	
code.compute.internal	<none>		<none>			
coredns-85d5b4454c-cw7w2	1/1	Running	0	56m		
2600:1f13:b66:8203:34e5::					ip-192-168-253-70.region-	
code.compute.internal	<none>		<none>			
coredns-85d5b4454c-tx6n8	1/1	Running	0	56m		
2600:1f13:b66:8203:34e5::1					ip-192-168-253-70.region-	
code.compute.internal	<none>		<none>			
kube-proxy-btpbk	1/1	Running	0	5m36s		
2600:1f13:b66:8200:11a5:ade0:c590:6ac8					ip-192-168-34-75.region-	
code.compute.internal	<none>		<none>			
kube-proxy-jjk2g	1/1	Running	0	5m33s		
2600:1f13:b66:8203:4516:2080:8ced:1ca9					ip-192-168-253-70.region-	
code.compute.internal	<none>		<none>			

- i. IP 열에서 기본 서비스에 IPv6 주소가 할당되었는지 확인합니다.

```
kubectl get services -n kube-system -o wide
```

예제 출력은 다음과 같습니다.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	
SELECTOR						
kube-dns	ClusterIP	fd30:3087:b6c2::a	<none>	53/UDP,53/TCP	57m	k8s-
app=kube-dns						

- j. (선택 사항) [샘플 애플리케이션을 배포](#)하거나 [AWS Load Balancer Controller](#)와 샘플 애플리케이션을 배포하여 [the section called “애플리케이션 로드 밸런싱”](#)를 사용하는 HTTP 애플리케이션 또는 [the section called “네트워크 로드 밸런싱”](#)을 사용하는 네트워크 트래픽을 IPv6 포드로 로드 밸런싱합니다.
- k. 이 튜토리얼로 생성한 클러스터 및 노드 사용을 마친 후 다음 명령을 사용하여 생성한 리소스를 정리해야 합니다. 리소스를 삭제하기 전에 이 튜토리얼 외부의 리소스를 사용하고 있지 않은지 확인합니다.

- i. 이전 단계를 완료한 것과 다른 셸에서 이 단계를 완료하는 경우 이전 단계에서 사용한 모든 변수의 값을 설정하고 **## #**을 이전 단계를 완료했을 때 지정한 값으로 바꿉니다. 이전 단계를 완료한 동일한 셸에서 이 단계를 완료하는 경우 다음 단계로 건너뛩니다.

```
export region_code=region-code
export vpc_stack_name=my-eks-ipv6-vpc
export cluster_name=my-cluster
export nodegroup_name=my-nodegroup
export account_id=111122223333
export node_role_name=AmazonEKSNodeRole
export cluster_role_name=myAmazonEKSClusterRole
```

- ii. 노드 그룹을 삭제합니다.

```
aws eks delete-nodegroup --region $region_code --cluster-name $cluster_name --
nodegroup-name $nodegroup_name
```

삭제하는 데 몇 분 정도 걸립니다. 다음 명령을 실행합니다. 출력이 반환되는 경우 다음 단계로 진행하지 마세요.

```
aws eks list-nodegroups --region $region_code --cluster-name $cluster_name --query
nodegroups --output text
```

- iii. 클러스터를 삭제합니다.

```
aws eks delete-cluster --region $region_code --name $cluster_name
```

클러스터를 삭제하는 데 몇 분 정도 걸립니다. 계속하기 전에 다음 명령으로 클러스터가 삭제되었는지 확인합니다.

```
aws eks describe-cluster --region $region_code --name $cluster_name
```

출력이 다음 출력과 유사할 때까지 다음 단계로 진행하지 마세요.

```
An error occurred (ResourceNotFoundException) when calling the DescribeCluster
operation: No cluster found for name: my-cluster.
```

- iv. 생성한 IAM 리소스를 삭제합니다. 이전 단계에서 사용한 이름과 다른 이름을 선택한 경우 **AmazonEKS_CNI_IPv6_PoIicy**를 선택한 이름으로 바꿉니다.

```
aws iam detach-role-policy --role-name $cluster_role_name --policy-arn
arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
aws iam detach-role-policy --role-name $node_role_name --policy-arn
arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
aws iam detach-role-policy --role-name $node_role_name --policy-arn
arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
aws iam detach-role-policy --role-name $node_role_name --policy-arn arn:aws:iam::
$account_id:policy/AmazonEKS_CNI_IPv6_Policy
aws iam delete-policy --policy-arn arn:aws:iam::$account_id:policy/
AmazonEKS_CNI_IPv6_Policy
aws iam delete-role --role-name $cluster_role_name
aws iam delete-role --role-name $node_role_name
```

v. 생성한 VPC AWS CloudFormation 스택을 삭제합니다.

```
aws cloudformation delete-stack --region $region_code --stack-name $vpc_stack_name
```

포드에 대한 아웃바운드 인터넷 액세스 활성화

적용 대상: Linux IPv4 Fargate 노드, Amazon EC2 인스턴스가 있는 Linux 노드

IPv6 패밀리를 사용하여 클러스터를 배포한 경우 IPv6 주소는 네트워크로 변환되지 않기 때문에 이 주제의 정보는 클러스터에 적용되지 않습니다. 클러스터에서 IPv6 사용에 대한 자세한 내용을 알아보려면 [the section called “IPv6”](#) 섹션을 참조하세요.

기본적으로 클러스터의 각 포드에는 포드가 배포된 VPC와 연결된 CIDR(Classless Inter-Domain Routing) 블록의 [프라이빗](#) IPv4 주소가 할당됩니다. 동일한 VPC의 포드는 이러한 프라이빗 IP 주소를 엔드포인트로 사용하여 서로 통신합니다. 포드가 VPC에 연결된 CIDR 블록 내에 있지 않은 IPv4 주소와 통신하는 경우 Amazon VPC CNI 플러그인([Linux](#) 또는 [Windows](#)용)은 기본적으로 *를 통해 포드의 IPv4 주소를 포드가 실행 중인 노드의 기본 [탄력적 네트워크 인터페이스](#)의 기본 프라이빗 IPv4 주소로 변환합니다.

Note

Windows 노드의 경우 고려해야 할 추가 세부 정보가 있습니다. 기본적으로 [Windows용 VPC CNI 플러그인](#)은 동일한 VPC 내의 대상에 대한 트래픽이 SNAT에서 제외되는 네트워킹 구성으로 정의됩니다. 이는 내부 VPC 통신에 SNAT가 비활성화되어 있고 포드에 할당된 IP 주소를 VPC 내에서 라우팅할 수 있음을 의미합니다. 그러나 VPC 외부의 대상에 대한 트래픽은 소스 포트 IP SNAT가 인스턴스 ENI의 기본 IP 주소에 연결되어 있습니다. Windows에 대한 이 기본

구성은 포드가 호스트 인스턴스와 동일한 방식으로 VPC 외부의 네트워크에 액세스할 수 있도록 합니다.

이 동작으로 인해 다음이 발생합니다.

- 실행 중인 노드가 [퍼블릭](#) 또는 [탄력적](#) IP 주소를 할당받고 [퍼블릭 서브넷](#)에 있는 경우에만 포드가 인터넷 리소스와 통신할 수 있습니다. 퍼블릭 서브넷은 인터넷 게이트웨이로 이어지는 라우팅이 있는 [라우팅 테이블](#)과 연결된 서브넷입니다. 가능하면 프라이빗 서브넷에 노드를 배포하는 것이 좋습니다.
- 1.8.0 이전 플러그인 버전의 경우 [VPC 피어링](#), [전송 VPC](#) 또는 [AWS Direct Connect](#)를 사용하여 클러스터 VPC에 연결된 네트워크 또는 VPC에 있는 리소스는 보조 탄력적 네트워크 인터페이스 뒤에 있는 포드와의 통신을 시작할 수 없습니다. 그럼에도 포드는 이러한 리소스와 통신을 시작하고 응답을 수신할 수 있습니다.

사용자 환경에서 다음 설명 중 하나에 해당하는 경우 다음 명령을 사용하여 기본 구성을 변경하십시오.

- 네트워크 또는 [VPC 피어링](#), [전송 VPC](#) 또는 [AWS Direct Connect](#)를 사용하여 클러스터 VPC에 연결된 리소스가 IPv4 주소를 사용하여 포드와 통신을 시작해야 하며 플러그인 버전이 1.8.0보다 이전 버전인 네트워크 또는 VPC에 있습니다.
- 포드는 [프라이빗 서브넷](#)에 있으며 인터넷과 아웃바운드로 통신해야 합니다. 서브넷은 [NAT 게이트웨이](#)로 가는 경로를 가지고 있습니다.

```
kubectl set env daemonset -n kube-system aws-node AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

Note

AWS_VPC_K8S_CNI_EXTERNALSNAT 및 AWS_VPC_K8S_CNI_EXCLUDE_SNAT_CIDRS CNI 구성 변수는 Windows 노드에 적용할 수 없습니다. Windows에서는 SNAT를 비활성화할 수 없습니다. SNAT에서 IPv4 CIDR 목록을 제외하는 경우 Windows 부트스트랩 스크립트에서 ExcludedSnatCidrs 파라미터를 지정하여 이를 정의할 수 있습니다. 이 파라미터 사용에 대한 자세한 내용은 [the section called “부트스트랩 스크립트 구성 파라미터”](#) 섹션을 참조하세요.

호스트 네트워킹

* 포드의 사양에 `hostNetwork=true`(기본값은 `false`)가 포함된 경우 해당 IP 주소가 다른 주소로 변환되지 않습니다. 이 경우는 기본적으로 클러스터에서 실행되는 kube-proxy 및 Kubernetes용 Amazon VPC CNI 플러그인 포드의 경우입니다. 이러한 포드의 경우 IP 주소는 노드의 기본 IP 주소와 동일하므로 포드의 IP 주소는 변환되지 않습니다. 포드의 `hostNetwork` 설정에 대한 자세한 내용은 Kubernetes API Reference의 [PodSpec v1 core](#)를 참조하세요.

Kubernetes 네트워크 정책을 통해 pod 트래픽 제한

기본적으로 클러스터의 pod 또는 다른 네트워크의 pod와 리소스 사이에 IP 주소, 포트 또는 연결에 대해 Kubernetes에는 제한이 없습니다. Kubernetes 네트워크 정책을 사용하여 pod에서 송수신되는 네트워크 트래픽을 제한할 수 있습니다. 자세한 내용은 쿠버네티스 문서의 [네트워크 정책](#)을 참조하세요.

클러스터에 Kubernetes용 Amazon VPC CNI 플러그인 버전 1.13 이하가 있는 경우 타사 솔루션을 구현하여 클러스터에 Kubernetes 네트워크 정책을 적용해야 합니다. 버전 1.14 이상의 플러그인으로 네트워크 정책을 구현할 수 있으므로 타사 솔루션을 사용할 필요가 없습니다. 이 주제에서는 타사 추가 기능을 사용하지 않고 클러스터에서 Kubernetes 네트워크 정책을 사용하도록 클러스터를 구성하는 방법을 알아봅니다.

Kubernetes용 Amazon VPC CNI 플러그인의 네트워크 정책은 다음 구성에서 지원됩니다.

- 클러스터의 Kubernetes용 Amazon VPC CNI 플러그인 버전 1.14 이상.
- IPv4 또는 IPv6 주소에 대해 구성된 클러스터.
- [포드에 대한 보안 그룹](#)과 함께 네트워크 정책을 사용할 수 있습니다. 네트워크 정책을 사용하면 클러스터 내 모든 통신을 제어할 수 있습니다. 포드용 보안 그룹을 통해 포드 내의 애플리케이션에서 AWS 서비스에 대한 액세스를 제어할 수 있습니다.
- 사용자 지정 네트워킹 및 접두사 위임에 네트워크 정책을 사용할 수 있습니다.

고려 사항

아키텍처

- Kubernetes용 Amazon VPC CNI 플러그인 사용을 통해 클러스터에 Kubernetes 네트워크 정책을 적용할 때 Amazon EC2 Linux 노드에만 정책을 적용할 수 있습니다. Fargate 또는 Windows 노드에는 정책을 적용할 수 없습니다.
- 네트워크 정책은 IPv4 또는 IPv6 주소만 적용하지만 둘 다 적용되지는 않습니다. IPv4 클러스터에서 VPC CNI는 포드에 IPv4 주소를 할당하고 IPv4 정책을 적용합니다. IPv6 클러스터에서 VPC

CNI는 포드에 IPv6 주소를 할당하고 IPv6 정책을 적용합니다. IPv6 클러스터에 적용된 모든 IPv4 네트워크 정책 규칙은 무시됩니다. IPv4 클러스터에 적용된 모든 IPv6 네트워크 정책 규칙은 무시됩니다.

네트워크 정책

- 네트워크 정책은 배포의 일부인 포드에만 적용됩니다. `metadata.ownerReferences` 세트가 없는 독립 실행형 포드에는 네트워크 정책이 적용될 수 없습니다.
- 여러 네트워크 정책을 동일한 포드에 적용할 수 있습니다. 동일한 포드를 선택한 둘 이상의 정책이 구성된 경우 모든 정책이 포드에 적용됩니다.
- 네트워크 정책의 각 `ingress:` 또는 `egress:` 섹션에서 각 프로토콜의 고유한 포트 조합 최대 수는 24입니다.
- Kubernetes 서비스의 경우 서비스 포트는 컨테이너 포트와 동일해야 합니다. 이름이 지정된 포트를 사용하는 경우 서비스 사양에도 같은 이름을 사용합니다.

마이그레이션

- 클러스터에서 현재 타사 솔루션을 사용하여 Kubernetes 네트워크 정책을 관리하는 경우 Kubernetes 용 Amazon VPC CNI 플러그인에 동일한 정책을 사용할 수 있습니다. 하지만 동일한 정책을 관리하지 않도록 기존 솔루션을 제거해야 합니다.

설치

- 네트워크 정책 기능에서 `policyendpoints.networking.k8s.aws`라는 PolicyEndpoint 사용자 지정 리소스 정의(CRD)가 생성되고 이를 사용해야 합니다. 사용자 지정 참조의 PolicyEndpoint 개체는 Amazon EKS에서 관리합니다. 이러한 리소스를 수정하거나 삭제해서는 안 됩니다.
- 인스턴스 역할 IAM 보안 인증 정보를 사용하는 포드를 실행하거나 EC2 IMDS에 연결하는 경우 EC2 IMDS에 대한 액세스를 차단하는 네트워크 정책을 주의하여 확인하세요. EC2 IMDS에 대한 액세스를 허용하려면 네트워크 정책을 추가해야 할 수도 있습니다. 자세한 내용은 Amazon EC2 사용 설명서의 [인스턴스 메타데이터 및 사용자 데이터](#)를 참조하세요.

서비스 계정에 대한 IAM 역할을 사용하는 포드 또는 EKS Pod Identity를 사용하는 포드는 EC2 IMDS에 액세스하지 않습니다.

- Kubernetes용 Amazon VPC CNI 플러그인은 각 포드의 추가 네트워크 인터페이스에 네트워크 정책을 적용하지 않고 각 포드의 기본 인터페이스(eth0)에만 적용합니다. 이는 다음 아키텍처에 영향을 미칩니다.
- ENABLE_V4_EGRESS 변수가 true로 설정된 IPv6 포드. 이 변수를 사용하면 IPv4 송신 기능을 통해 IPv6 포드를 클러스터 외부의 항목과 같이 IPv4 엔드포인트에 연결할 수 있습니다. IPv4 송신 기능은 로컬 루프백 IPv4 주소를 사용하여 추가 네트워크 인터페이스를 생성하여 작동합니다.
- Multus와 같은 체인 네트워크 플러그인을 사용하는 경우. 이러한 플러그인은 각 포드에 네트워크 인터페이스를 추가하므로 네트워크 정책은 체인 네트워크 플러그인에 적용되지 않습니다.

Kubernetes 네트워크 정책으로 포드 네트워크 트래픽 제한

Kubernetes 네트워크 정책을 사용하여 포드에서 송수신되는 네트워크 트래픽을 제한할 수 있습니다. 자세한 내용은 쿠버네티스 문서의 [네트워크 정책](#)을 참조하세요.

이 기능을 사용하려면 다음을 구성해야 합니다.

1. 포드 시작 시 정책 적용을 설정합니다. VPC CNI DaemonSet의 aws-node 컨테이너에서 이 작업을 수행합니다.
2. 추가 기능의 네트워크 정책 파라미터를 활성화합니다.
3. Kubernetes 네트워크 정책을 사용하도록 클러스터 구성

시작하기 전에 고려 사항을 검토하세요. 자세한 내용은 [the section called “고려 사항”](#) 섹션을 참조하세요.

사전 조건

다음은 기능의 사전 조건입니다.

최소 클러스터 버전

기존 Amazon EKS 클러스터. 배포하려면 [시작](#) 섹션을 참조하세요. 클러스터는 다음 표에 나열된 Kubernetes 버전 및 플랫폼 버전 중 하나를 실행해야 합니다. 나열된 것보다 이후의 모든 Kubernetes 및 플랫폼 버전도 지원됩니다. 다음 명령에서 *my-cluster*를 본인의 클러스터 이름으로 바꾼 다음 수정된 명령을 실행하여 현재 Kubernetes 버전을 확인할 수 있습니다.

```
aws eks describe-cluster
    --name my-cluster --query cluster.version --output
    text
```

Kubernetes 버전	플랫폼 버전
1.27.4	eks.5
1.26.7	eks.6

최소 VPC CNI 버전

클러스터의 Kubernetes용 Amazon VPC CNI 플러그인 버전 1.14 이상. 다음 명령을 사용하여 현재 버전을 확인할 수 있습니다.

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
cut -d : -f 3
```

버전이 1.14 이하인 경우 [the section called “업데이트\(EKS 추가 기능\)”](#)의 내용을 참조하여 버전을 1.14 이상으로 업그레이드합니다.

최소 Linux 커널 버전

노드의 Linux 커널 버전이 5.10 이상이어야 합니다. `uname -r`을 사용하여 커널 버전을 확인할 수 있습니다. Amazon EKS 최적화 Amazon Linux, Amazon EKS 최적화 가속 Amazon Linux AMI, Bottlerocket AMI의 최신 버전을 사용하는 경우 이미 필수 커널 버전이 설치되어 있습니다.

Amazon EKS 최적화 가속 Amazon Linux AMI 버전 v20231116 이상에는 커널 버전 5.10이 있습니다.

1단계: 포트 시작 시 정책 적용 설정

Kubernetes용 Amazon VPC CNI 플러그인은 포트 프로비저닝과 병행하여 포트에 대한 네트워크 정책을 구성합니다. 새 포트에 대해 모든 정책이 구성될 때까지 새 포트의 컨테이너는 기본 허용 정책으로 시작됩니다. 이를 표준 모드라고 합니다. 기본 허용 정책은 새 포드를 오가는 모든 수신 및 송신 트래픽이 허용됨을 의미합니다. 예를 들어, 새 포트가 활성 정책으로 업데이트될 때까지 포트에는 방화벽 규칙이 적용되지 않습니다(모든 트래픽이 허용됨).

NETWORK_POLICY_ENFORCING_MODE 변수가 `strict`로 설정되면 VPC CNI를 사용하는 포트는 기본 거부 정책으로 시작하고, 정책이 구성됩니다. 이를 엄격 모드라고 합니다. 엄격 모드에서는 클러스터에서 포트가 액세스해야 하는 모든 엔드포인트에 대한 네트워크 정책이 있어야 합니다. 이 요구 사항은 CoreDNS 포트에 적용됩니다. 호스트 네트워킹을 사용하는 포트에는 기본 거부 정책이 구성되어 있지 않습니다.

VPC CNI DaemonSet의 `aws-node` 컨테이너에서 환경 변수

`NETWORK_POLICY_ENFORCING_MODE`를 `strict`로 설정하여 기본 네트워크 정책을 변경할 수 있습니다.

```
env:
  - name: NETWORK_POLICY_ENFORCING_MODE
    value: "strict"
```

2단계: 추가 기능의 네트워크 정책 파라미터 활성화

네트워크 정책 기능은 기본적으로 노드의 포트 8162를 지표에 사용합니다. 또한 이 기능은 상태 프로브에 포트 8163을 사용했습니다. 이러한 포트를 사용해야 하는 노드 또는 포드 내부에서 다른 애플리케이션을 실행하면 앱이 실행되지 않습니다. VPC CNI 버전 v1.14.1 이상에서는 이러한 포트를 변경할 수 있습니다.

다음 절차를 사용하여 추가 기능에 대한 네트워크 정책 파라미터를 활성화합니다.

AWS Management Console

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 왼쪽 탐색 창에서 클러스터를 선택한 다음 Amazon VPC CNI 추가 기능을 구성할 클러스터의 이름을 선택합니다.
3. 추가 기능 탭을 선택합니다.
4. 추가 기능 상자의 오른쪽 상단에 있는 상자를 선택한 다음에 편집을 선택합니다.
5. **Amazon VPC CNI** 구성 페이지에서
 - a. 버전 목록에서 v1.14.0-eksbuild.3 이상 버전을 선택합니다.
 - b. 선택적 구성 설정을 확장합니다.
 - c. 구성 값에 JSON 키 `"enableNetworkPolicy"`: 및 값 `"true"`를 입력합니다. 결과 텍스트는 유효한 JSON 객체여야 합니다. 이 키와 값이 텍스트 상자의 유일한 데이터인 경우, 키와 값을 중괄호(`{ }`)로 둘러쌉니다.

다음 예제에서는 네트워크 정책 기능이 활성화되어 있고 지표 및 상태 프로브가 기본 포트 번호로 설정되어 있습니다.

```
{
  "enableNetworkPolicy": "true",
  "nodeAgent": {
```

```

    "healthProbeBindAddr": "8163",
    "metricsBindAddr": "8162"
  }
}
```

Helm

helm을 통해 Kubernetes용 Amazon VPC CNI 플러그인을 설치한 경우 구성을 업데이트하여 포트를 변경할 수 있습니다.

1. 다음 명령을 실행하여 포트를 변경합니다. 키 `nodeAgent.metricsBindAddr` 또는 키 `nodeAgent.healthProbeBindAddr`의 값에 각각 포트 번호를 설정합니다.

```

helm upgrade --set nodeAgent.metricsBindAddr=8162 --set
nodeAgent.healthProbeBindAddr=8163 aws-vpc-cni --namespace kube-system eks/aws-vpc-
cni
```

kubectl

1. 편집기에서 `aws-node DaemonSet`를 엽니다.

```

kubectl edit daemonset -n kube-system aws-node
```

2. VPC CNI `aws-node` 대몬 세트 매니페스트의 `aws-network-policy-agent` 컨테이너에 있는 `args:`의 다음 명령 인수에서 포트 번호를 바꿉니다.

```

- args:
  - --metrics-bind-addr=:8162
  - --health-probe-bind-addr=:8163
```

3단계: 각 노드에 Berkeley Packet Filter(BPF) 파일 시스템 탑재

각 노드에 Berkeley Packet Filter(BPF) 파일 시스템을 탑재해야 합니다.

Note

클러스터가 버전 1.27 이상인 경우 모든 Amazon EKS 최적화 Amazon Linux 및 Bottlerocket AMI 1.27 이상에 이미 이 기능이 있습니다.

다른 모든 클러스터 버전의 경우 Amazon EKS 최적화 Amazon Linux를 v20230703 이상으로 업그레이드하거나 Bottlerocket AMI를 버전 v1.0.2 이상으로 업그레이드한 경우 이 단계를 건너뛸 수 있습니다.

1. 각 노드에 버클리 패킷 필터(BPF) 파일 시스템을 탑재합니다.

```
sudo mount -t bpf bpf fs /sys/fs/bpf
```

2. 그런 다음 Amazon EC2 Auto Scaling 그룹용 시작 템플릿의 사용자 데이터에 동일한 명령을 추가합니다.

4단계: Kubernetes 네트워크 정책을 사용하도록 클러스터 구성

Amazon EKS 추가 기능 또는 자체 관리형 추가 기능에 대해 이를 설정할 수 있습니다.

Amazon EKS 추가 기능

AWS CLI로 다음 명령을 실행하여 Kubernetes 네트워크 정책을 사용하도록 클러스터를 구성할 수 있습니다. my-cluster를 본인의 클러스터 이름으로 바꾸고 IAM 역할 ARN을 사용 중인 역할로 바꿉니다.

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version v1.14.0-eksbuild.3 \
  --service-account-role-arn arn:aws:iam::123456789012:role/AmazonEKSVPCCNIRole \
  --resolve-conflicts PRESERVE --configuration-values '{"enableNetworkPolicy": "true"}'
```

AWS Management Console을 사용하여 이를 구성하려면 다음 단계를 따르세요.

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 왼쪽 탐색 창에서 클러스터를 선택한 다음 Amazon VPC CNI 추가 기능을 구성할 클러스터의 이름을 선택합니다.
3. 추가 기능 탭을 선택합니다.
4. 추가 기능 상자의 오른쪽 상단에 있는 상자를 선택한 다음에 편집을 선택합니다.
5. **Amazon VPC CNI** 구성 페이지에서
 - a. 버전 목록에서 v1.14.0-eksbuild.3 이상 버전을 선택합니다.
 - b. 선택적 구성 설정을 확장합니다.

- c. 구성 값에 JSON 키 "enableNetworkPolicy": 및 값 "true"를 입력합니다. 결과 텍스트는 유효한 JSON 객체여야 합니다. 이 키와 값이 텍스트 상자의 유일한 데이터인 경우, 키와 값을 중괄호({ })로 둘러쌉니다. 다음 예시는 네트워크 정책이 활성화된 것을 보여줍니다.

```
{ "enableNetworkPolicy": "true" }
```

다음 스크린샷은 이 시나리오의 예를 보여줍니다.



Configure Amazon VPC CNI

Amazon VPC CNI [Info](#)

Listed by 	Category networking	Status  Active

Version

Select the version for this add-on.

v1.17.1-eksbuild.1

Select IAM role

Select an IAM role to use with this add-on. To create a new role, follow the instructions in the [Amazon EKS User Guide](#).

Optional configuration settings

Add-on configuration schema

Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.

```
{
  "$ref": "#/definitions/VpcCni",
  "$schema": "http://json-schema.org/draft-06/schema#",
  "definitions": {
    "Affinity": {
      "type": [
        "object",
        "null"
      ]
    },
    "EniConfig": {
      "additionalProperties": false,

```

Configuration values [Info](#)

Specify any additional JSON or YAML configurations that should be applied to the add-on.

1

{ "enableNetworkPolicy": "true" }

자체 관리형 추가 기능

Helm

helm을 통해 Kubernetes용 Amazon VPC CNI 플러그인을 설치한 경우 구성을 업데이트하여 네트워크 정책을 활성화할 수 있습니다.

1. 다음 명령을 실행하여 네트워크 정책을 활성화합니다.

```
helm upgrade --set enableNetworkPolicy=true aws-vpc-cni --namespace kube-system eks/
aws-vpc-cni
```

kubectl

1. 편집기에서 amazon-vpc-cni ConfigMap을 엽니다.

```
kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
```

2. 다음 줄을 ConfigMap의 data에 추가합니다.

```
enable-network-policy-controller: "true"
```

줄을 추가한 후에는 ConfigMap이 다음 예와 같을 것입니다.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: amazon-vpc-cni
  namespace: kube-system
data:
  enable-network-policy-controller: "true"
```

3. 편집기에서 aws-node DaemonSet를 엽니다.

```
kubectl edit daemonset -n kube-system aws-node
```

- a. VPC CNI aws-node 대몬 세트 매니페스트의 aws-network-policy-agent 컨테이너에 있는 args:의 명령 인수 --enable-network-policy=false에서 false를 true로 바꿉니다.

```
- args:
  - --enable-network-policy=true
```

5단계. 다음 단계

구성을 완료한 후 `aws-node` 포드가 클러스터에서 실행 중인지 확인합니다.

```
kubectl get pods -n kube-system | grep 'aws-node\|amazon'
```

예제 출력은 다음과 같습니다.

aws-node-gmqp7 24h	2/2	Running	1 (24h ago)
aws-node-pinsh 24h	2/2	Running	1 (24h ago)

버전 1.14 이상에서는 `aws-node` 포드에 2개의 컨테이너가 있습니다. 이전 버전에서 네트워크 정책이 비활성화된 경우 `aws-node` 포드에 하나의 컨테이너만 있습니다.

이제 클러스터에 Kubernetes 네트워크 정책을 배포할 수 있습니다.

Kubernetes 네트워크 정책을 구현하려면 Kubernetes `NetworkPolicy` 객체를 생성하고 클러스터에 배포합니다. `NetworkPolicy` 객체 범위는 네임스페이스로 지정됩니다. 정책을 구현하여 레이블 선택기, 네임스페이스 및 IP 주소 범위를 기반으로 포드 사이의 트래픽을 허용 또는 거부합니다. `NetworkPolicy` 객체 생성에 대한 자세한 내용은 쿠버네티스 문서의 [네트워크 정책](#)을 참조하세요.

Kubernetes `NetworkPolicy` 객체의 적용은 확장 Berkeley 패킷 필터(eBPF)를 사용하여 구현됩니다. iptables 기반 구현에 따라 지연 시간을 낮추고 CPU 활용률 감소 및 순차 조회 방지를 포함한 성능 특성을 제공합니다. 추가로 eBPF 프로브는 복잡한 커널 수준 문제를 디버깅하고 관찰성을 개선하는 데 도움이 되도록 컨텍스트가 풍부한 데이터에 대한 액세스를 제공합니다. Amazon EKS는 프로브를 활용하여 각 노드에 정책 결과를 기록하고 외부 로그 수집기로 데이터를 내보내 디버깅을 지원하는 eBPF 기반 내보내기를 지원합니다. 자세한 내용은 [eBPF 설명](#)을 참조하세요.

Amazon EKS 포드 네트워크 트래픽에 대한 Kubernetes 네트워크 정책 비활성화

Kubernetes 네트워크 정책을 비활성화하여 Amazon EKS 포드 네트워크 트래픽 제한 중지

1. 모든 Kubernetes 네트워크 정책을 나열합니다.

```
kubectl get netpol -A
```

2. 각 Kubernetes 네트워크 정책을 삭제합니다. 네트워크 정책을 비활성화하기 전에 모든 네트워크 정책을 삭제해야 합니다.

```
kubectl delete netpol <policy-name>
```

3. 편집기에서 aws-node DaemonSet를 엽니다.

```
kubectl edit daemonset -n kube-system aws-node
```

4. VPC CNI aws-node 데몬 세트 매니페스트의 aws-network-policy-agent 컨테이너에 있는 args:의 명령 인수 --enable-network-policy=true에서 true를 false로 바꿉니다.

```
- args:
  - --enable-network-policy=true
```

Amazon EKS에 대한 Kubernetes 네트워크 정책 문제 해결

네트워크 정책을 사용하는 네트워크 연결의 문제를 해결하고 조사하려면 [네트워크 정책 로그](#)를 읽고 [eBPF SDK](#)의 도구를 실행하여 문제를 해결할 수 있습니다.

네트워크 정책 로그

네트워크 정책에 의해 연결이 허용 또는 거부되는지 여부는 흐름 로그에 기록됩니다. 각 노드의 네트워크 정책 로그에는 네트워크 정책이 있는 모든 포드의 흐름 로그가 포함됩니다. 네트워크 정책 로그는 /var/log/aws-routed-eni/network-policy-agent.log에 저장됩니다. 다음은 network-policy-agent.log 파일의 예입니다.

```
{"level":"info","timestamp":"2023-05-30T16:05:32.573Z","logger":"ebpf-client","msg":"Flow Info: ","Src IP":"192.168.87.155","Src Port":38971,"Dest IP":"64.6.160","Dest Port":53,"Proto":"UDP","Verdict":"ACCEPT"}
```

네트워크 정책 로그는 기본적으로 비활성화되어 있습니다. 네트워크 정책 로그를 활성화하려면 다음 단계를 수행합니다.

Note

네트워크 정책 로그에는 VPC CNI aws-node 대몬 세트 매니페스트의 aws-network-policy-agent 컨테이너용 vCPU 1개가 추가로 필요합니다.

Amazon EKS 추가 기능

AWS Management Console

- a. [Amazon EKS 콘솔](#)을 엽니다.
- b. 왼쪽 탐색 창에서 클러스터를 선택한 다음 Amazon VPC CNI 추가 기능을 구성할 클러스터의 이름을 선택합니다.
- c. 추가 기능 탭을 선택합니다.
- d. 추가 기능 상자의 오른쪽 상단에 있는 상자를 선택한 다음에 편집을 선택합니다.
- e. **Amazon VPC CNI** 구성 페이지에서
 - i. 버전 드롭다운 목록에서 v1.14.0-eksbuild.3 이상 버전을 선택합니다.
 - ii. 선택적 구성 설정을 확장합니다.
 - iii. 최상위 JSON 키 "nodeAgent":를 입력하고 값은 키 "enablePolicyEventLogs":와 구성 값의 "true" 값이 포함된 객체입니다. 결과 텍스트는 유효한 JSON 객체여야 합니다. 다음 예시는 네트워크 정책과 네트워크 정책 로그가 활성화되어 있으며 CloudWatch Logs로 전송된 것을 보여줍니다.

```
{
  "enableNetworkPolicy": "true",
  "nodeAgent": {
    "enablePolicyEventLogs": "true"
  }
}
```

다음 스크린샷은 이 시나리오의 예를 보여줍니다.



[EKS](#) > [Clusters](#) > > [Add-on](#) > [vpc-cni](#) > **Edit add-on**

Configure Amazon VPC CNI

Amazon VPC CNI [Info](#)

Listed by



Category

networking

Status

Active

Version

Select the version for this add-on.

v1.17.1-eksbuild.1

Select IAM role

Select an IAM role to use with this add-on. To create a new role, follow the instructions in the [Amazon EKS User Guide](#).

▼ Optional configuration settings

Add-on configuration schema

Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.

```
{
  "$ref": "#/definitions/VpcCni",
  "$schema": "http://json-schema.org/draft-06/schema#",
  "definitions": {
    "Affinity": {
      "type": [
        "object",
        "null"
      ]
    },
  },
  "EniConfig": {
    "additionalProperties": false,
  }
}
```

Configuration values [Info](#)

Specify any additional JSON or YAML configurations that should be applied to the add-on.

```
1 {  
2   "enableNetworkPolicy": "true",  
3   "nodeAgent": {  
4     "enablePolicyEventLogs": "true"  
5   }  
6 }
```

AWS CLI

- a. 다음 AWS CLI 명령을 실행합니다. `my-cluster`를 본인의 클러스터 이름으로 바꾸고 IAM 역할 ARN을 사용 중인 역할로 바꿉니다.

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version v1.14.0-eksbuild.3 \
    --service-account-role-arn arn:aws:iam::123456789012:role/AmazonEKSVPCCNIRole \
    --resolve-conflicts PRESERVE --configuration-values '{"nodeAgent": {"enablePolicyEventLogs": "true"}}'
```

자체 관리형 추가 기능

Helm

helm을 통해 Kubernetes용 Amazon VPC CNI 플러그인을 설치한 경우 구성을 업데이트하여 네트워크 정책을 쏠 수 있습니다.

- a. 다음 명령을 실행하여 네트워크 정책을 활성화합니다.

```
helm upgrade --set nodeAgent.enablePolicyEventLogs=true aws-vpc-cni --namespace kube-system eks/aws-vpc-cni
```

kubectl

kubectl을 통해 Kubernetes용 Amazon VPC CNI 플러그인을 설치한 경우 구성을 업데이트하여 네트워크 정책을 쏠 수 있습니다.

- a. 편집기에서 `aws-node` DaemonSet를 엽니다.

```
kubectl edit daemonset -n kube-system aws-node
```

- b. VPC CNI `aws-node` 대몬 세트 매니페스트의 `aws-network-policy-agent` 컨테이너에 있는 `args:`의 명령 인수 `--enable-policy-event-logs=false`에서 `false`를 `true`로 바꿉니다.

```
- args:
  - --enable-policy-event-logs=true
```

네트워크 정책 로그를 Amazon CloudWatch Logs로 전송

Amazon CloudWatch Logs와 같은 서비스를 사용하여 네트워크 정책 로그를 모니터링할 수 있습니다. 다음 방법을 사용하여 네트워크 정책 로그를 CloudWatch Logs로 보낼 수 있습니다.

EKS 클러스터의 경우 정책 로그는 `/aws/eks/cluster-name/cluster/` 아래에 있고 자체 관리형 K8S 클러스터의 경우 로그는 `/aws/k8s-cluster/cluster/` 아래에 있습니다.

Kubernetes용 Amazon VPC CNI 플러그인을 사용하여 네트워크 정책 로그 전송

네트워크 정책을 활성화하면 두 번째 컨테이너가 노드 에이전트용 `aws-node` 포드에 추가됩니다. 이 노드 에이전트는 네트워크 정책 로그를 CloudWatch Logs로 보낼 수 있습니다.

Note

네트워크 정책 로그는 노드 에이전트에 의해서만 전송됩니다. VPC CNI에서 생성한 다른 로그는 포함되지 않습니다.

사전 조건

- VPC CNI에 사용하는 IAM 역할에 다음 권한을 스탠자 또는 별도의 정책으로 추가합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups",
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "*"
    }
  ]
}
```

Amazon EKS 추가 기능

AWS Management Console

- a. [Amazon EKS 콘솔](#)을 엽니다.
- b. 왼쪽 탐색 창에서 클러스터를 선택한 다음 Amazon VPC CNI 추가 기능을 구성할 클러스터의 이름을 선택합니다.
- c. 추가 기능 탭을 선택합니다.
- d. 추가 기능 상자의 오른쪽 상단에 있는 상자를 선택한 다음에 편집을 선택합니다.
- e. **Amazon VPC CNI** 구성 페이지에서
 - i. 버전 드롭다운 목록에서 v1.14.0-eksbuild.3 이상 버전을 선택합니다.
 - ii. 선택적 구성 설정을 확장합니다.
 - iii. 최상위 JSON 키 "nodeAgent":를 입력하고 값은 키 "enableCloudWatchLogs":와 구성 값의 "true" 값이 포함된 객체입니다. 결과 텍스트는 유효한 JSON 객체여야 합니다. 다음 예시는 네트워크 정책과 네트워크 정책 로그가 활성화되어 있으며 로그가 CloudWatch Logs로 전송된 것을 보여줍니다.

```
{
  "enableNetworkPolicy": "true",
  "nodeAgent": {
    "enablePolicyEventLogs": "true",
    "enableCloudWatchLogs": "true",
  }
}
```

다음 스크린샷은 이 시나리오의 예를 보여줍니다.



[EKS](#) > [Clusters](#) > > [Add-on](#) > [vpc-cni](#) > **Edit add-on**

Configure Amazon VPC CNI

Amazon VPC CNI [Info](#)

Listed by



Category

networking

Status

🟢 Active

Version

Select the version for this add-on.

v1.17.1-eksbuild.1

Select IAM role

Select an IAM role to use with this add-on. To create a new role, follow the instructions in the [Amazon EKS User Guide](#).

▼ Optional configuration settings

Add-on configuration schema

Refer to the JSON schema below. The configuration values entered in the code editor will be validated against this schema.

```
{
  "$ref": "#/definitions/VpcCni",
  "$schema": "http://json-schema.org/draft-06/schema#",
  "definitions": {
    "Affinity": {
      "type": [
        "object",
        "null"
      ]
    }
  },
  "EniConfig": {
    "additionalProperties": false,

```

Configuration values [Info](#)

Specify any additional JSON or YAML configurations that should be applied to the add-on.

```
1 {
2   "enableNetworkPolicy": "true",
3   "nodeAgent": {
4     "enablePolicyEventLogs": "true",
5     "enableCloudWatchLogs": "true"
6   }
7 }
```

AWS CLI

- a. 다음 AWS CLI 명령을 실행합니다. `my-cluster`를 본인의 클러스터 이름으로 바꾸고 IAM 역할 ARN을 사용 중인 역할로 바꿉니다.

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-
version v1.14.0-eksbuild.3 \
    --service-account-role-arn arn:aws:iam::123456789012:role/AmazonEKSVPCCNIRole
\
    --resolve-conflicts PRESERVE --configuration-values '{"nodeAgent":
{"enablePolicyEventLogs": "true", "enableCloudWatchLogs": "true"}}'
```

자체 관리형 추가 기능

Helm

helm을 통해 Kubernetes용 Amazon VPC CNI 플러그인을 설치한 경우 구성을 업데이트하여 네트워크 정책 로그를 CloudWatch Logs로 전송할 수 있습니다.

- a. 다음 명령을 실행하여 네트워크 정책 로그를 활성화하고 CloudWatch Logs로 전송합니다.

```
helm upgrade --set nodeAgent.enablePolicyEventLogs=true --set
nodeAgent.enableCloudWatchLogs=true aws-vpc-cni --namespace kube-system eks/aws-
vpc-cni
```

kubectl

- a. 편집기에서 `aws-node` DaemonSet를 엽니다.

```
kubectl edit daemonset -n kube-system aws-node
```

- b. VPC CNI `aws-node` 대몬 세트 매니페스트의 `aws-network-policy-agent` 컨테이너에 있는 `args:`에서 두 명령 인수 `--enable-policy-event-logs=false` 및 `--enable-cloudwatch-logs=false`에서 `false`를 `true`로 바꿉니다.

```
- args:
  - --enable-policy-event-logs=true
  - --enable-cloudwatch-logs=true
```

Fluent Bit DaemonSet으로 네트워크 정책 로그 전송

DaemonSet에서 Fluent Bit를 사용하여 노드에서 로그를 전송하는 경우 네트워크 정책에서 네트워크 정책 로그를 포함하도록 구성을 추가할 수 있습니다. 다음 예제 구성을 사용할 수 있습니다.

```
[INPUT]
  Name          tail
  Tag           eksnp.*
  Path          /var/log/aws-routed-eni/network-policy-agent*.log
  Parser        json
  DB            /var/log/aws-routed-eni/flb_npagent.db
  Mem_Buf_Limit 5MB
  Skip_Long_Lines On
  Refresh_Interval 10
```

eBPF SDK 포함

Kubernetes용 Amazon VPC CNI 플러그인은 노드에 eBPF SDK 도구 모음을 설치합니다. eBPF SDK 도구를 사용하여 네트워크 정책 관련 문제를 식별할 수 있습니다. 예를 들어, 다음 명령은 노드에서 실행 중인 프로그램을 나열합니다.

```
sudo /opt/cni/bin/aws-eks-na-cli ebpf progs
```

이 명령을 실행하려면 임의의 방법을 사용하여 노드에 연결할 수 있습니다.

Amazon EKS 네트워크 정책에 대한 Stars 데모

이 데모는 Amazon EKS 클러스터에 프런트엔드, 백엔드 및 클라이언트 서비스를 생성합니다. 데모는 또한 각 서비스 간 이용 가능한 수신 및 발신 경로를 보여주는 관리 그래픽 사용자 인터페이스를 생성합니다. 프로덕션 워크로드를 실행하지 않는 클러스터에서 데모를 완료하는 것이 좋습니다.

네트워크 정책을 생성하기 전에 모든 서비스는 양방향으로 통신할 수 있습니다. 네트워크 정책을 적용하면 클라이언트가 프런트엔드 서비스와만 통신할 수 있고 백엔드는 프런트엔드의 트래픽만 수신하는 것을 볼 수 있습니다.

1. 프런트엔드, 백엔드, 클라이언트 및 관리 사용자 인터페이스 서비스를 적용합니다.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/namespace.yaml
```

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/management-ui.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/backend.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/frontend.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/client.yaml
```

2. 클러스터의 모든 포드를 봅니다.

```
kubectl get pods -A
```

예제 출력은 다음과 같습니다.

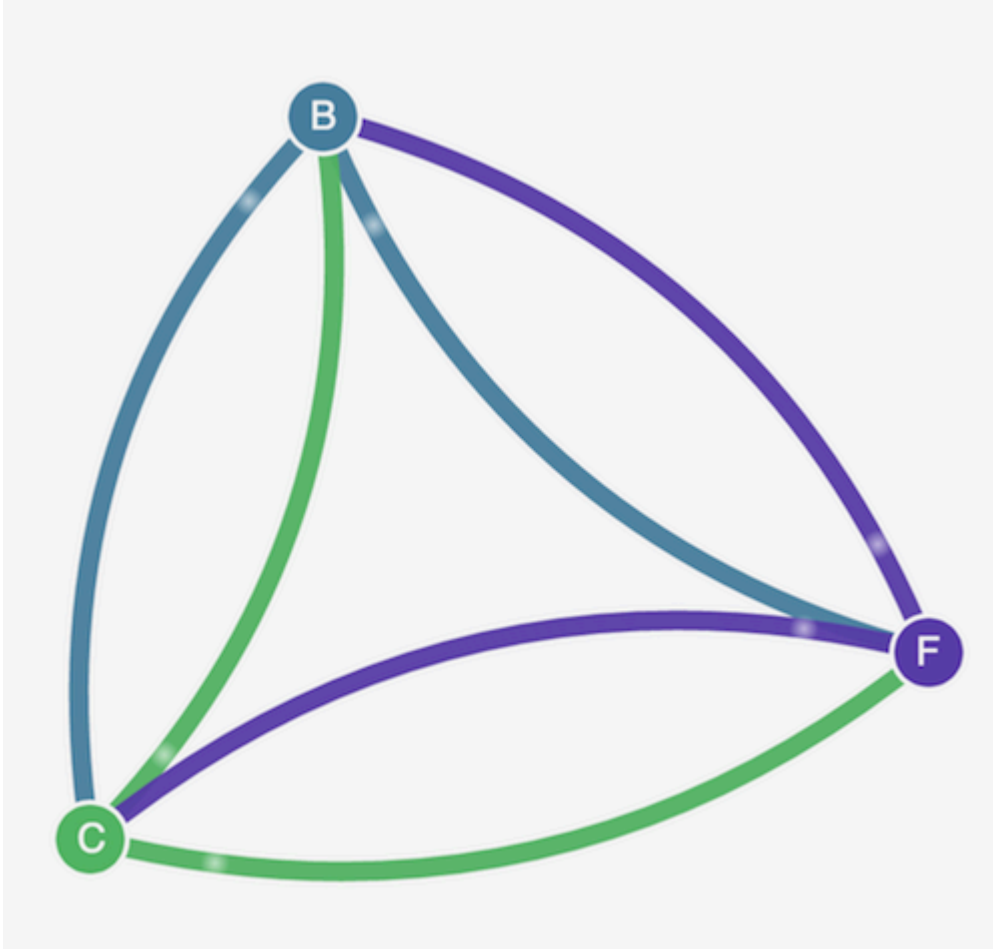
출력에서 다음 출력에 표시된 네임스페이스에 포드가 표시되어야 합니다. 포드의 ##과 READY 열의 포드 수는 다음 출력과 다릅니다. 이름이 비슷하고 STATUS 열에 Running이 있는 포드가 보일 때까지 계속하지 마세요.

NAMESPACE	NAME	READY	STATUS
RESTARTS	AGE		
[...]			
client	client-xlffc	1/1	Running 0
5m19s			
[...]			
management-ui	management-ui-qrb2g	1/1	Running 0
5m24s			
stars	backend-sz87q	1/1	Running 0
5m23s			
stars	frontend-cscnf	1/1	Running 0
5m21s			
[...]			

3. 관리 사용자 인터페이스에 연결하려면 클러스터에서 실행 중인 서비스의 EXTERNAL-IP에 연결합니다.

```
kubectl get service/management-ui -n management-ui
```

4. 브라우저를 열어 이전 단계의 위치로 이동합니다. 관리 사용자 인터페이스가 표시됩니다. C 노드는 클라이언트 서비스이고, F 노드는 프런트엔드 서비스이며, B 노드는 백엔드 서비스입니다. 각 노드는 기타 모든 노드에 대한 전체 통신 액세스를 보유합니다(진한 색상의 선으로 표시).



5. `stars` 및 `client` 네임스페이스 모두에 다음 네트워크 정책을 적용하여 서비스를 각각 격리시킵니다.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny
spec:
  podSelector:
    matchLabels: {}
```

다음 명령을 사용하여 두 네임스페이스 모두에 정책을 적용할 수 있습니다.

```
kubectl apply -n stars -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/apply_network_policies.files/default-deny.yaml
kubectl apply -n client -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/apply_network_policies.files/default-deny.yaml
```

6. 브라우저를 새로 고칩니다. 관리 사용자 인터페이스가 더 이상 노드에 도달하지 않아 사용자 인터페이스에 표시되지 않는 것을 확인할 수 있습니다.
7. 다음의 다른 네트워크 정책을 적용하여 관리 사용자 인터페이스가 서비스에 액세스하도록 허용합니다. 이 정책을 적용하여 UI 허용:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: stars
  name: allow-ui
spec:
  podSelector:
    matchLabels: {}
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
                role: management-ui
```

이 정책을 적용하여 클라이언트를 허용합니다.

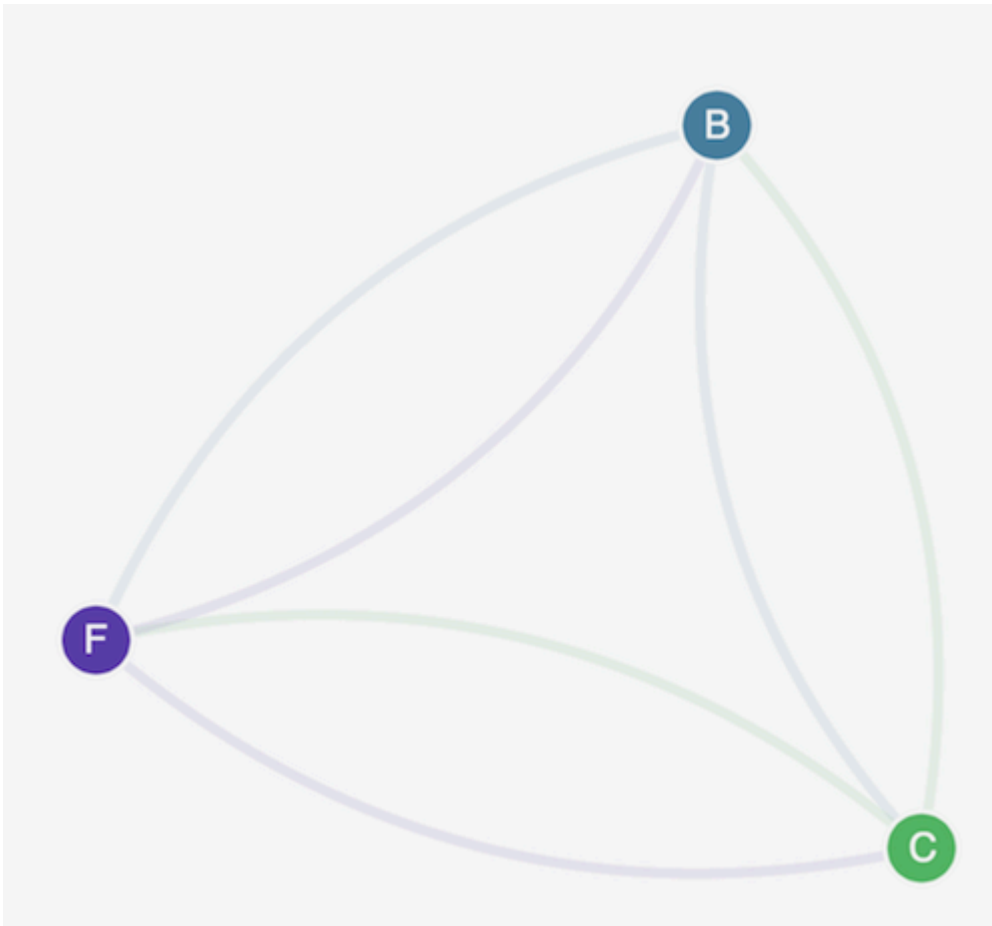
```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: client
  name: allow-ui
spec:
  podSelector:
    matchLabels: {}
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
```

```
role: management-ui
```

다음 명령을 사용하여 두 정책을 모두 적용할 수 있습니다.

```
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/apply_network_policies.files/allow-ui.yaml
kubectl apply -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/apply_network_policies.files/allow-ui-client.yaml
```

8. 브라우저를 새로 고칩니다. 관리 사용자 인터페이스가 노드에 다시 도달할 수 있지만, 노드가 서로 통신할 수 없음을 확인할 수 있습니다.



9. 다음 네트워크 정책을 적용하여 프런트엔드 서비스에서 백엔드 서비스로의 트래픽을 허용합니다.

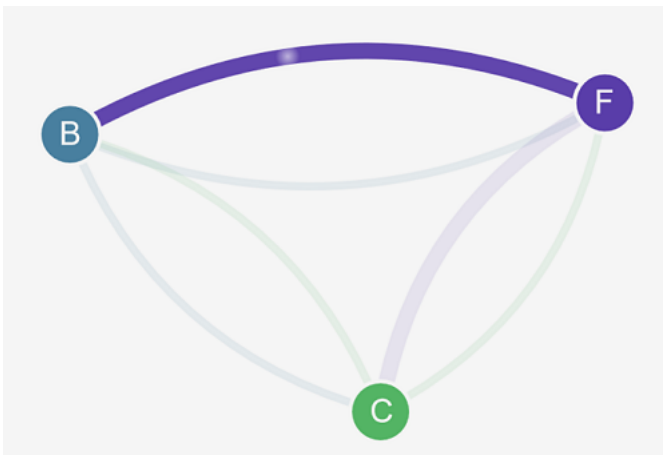
```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
```

```

namespace: stars
name: backend-policy
spec:
  podSelector:
    matchLabels:
      role: backend
  ingress:
    - from:
        - podSelector:
            matchLabels:
              role: frontend
      ports:
        - protocol: TCP
          port: 6379

```

10 브라우저를 새로 고칩니다. 프론트엔드가 백엔드와 통신할 수 있음을 알 수 있습니다.



11 다음 네트워크 정책을 적용하여 클라이언트에서 프론트엔드 서비스로의 트래픽을 허용합니다.

```

kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  namespace: stars
  name: frontend-policy
spec:
  podSelector:
    matchLabels:
      role: frontend
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:

```

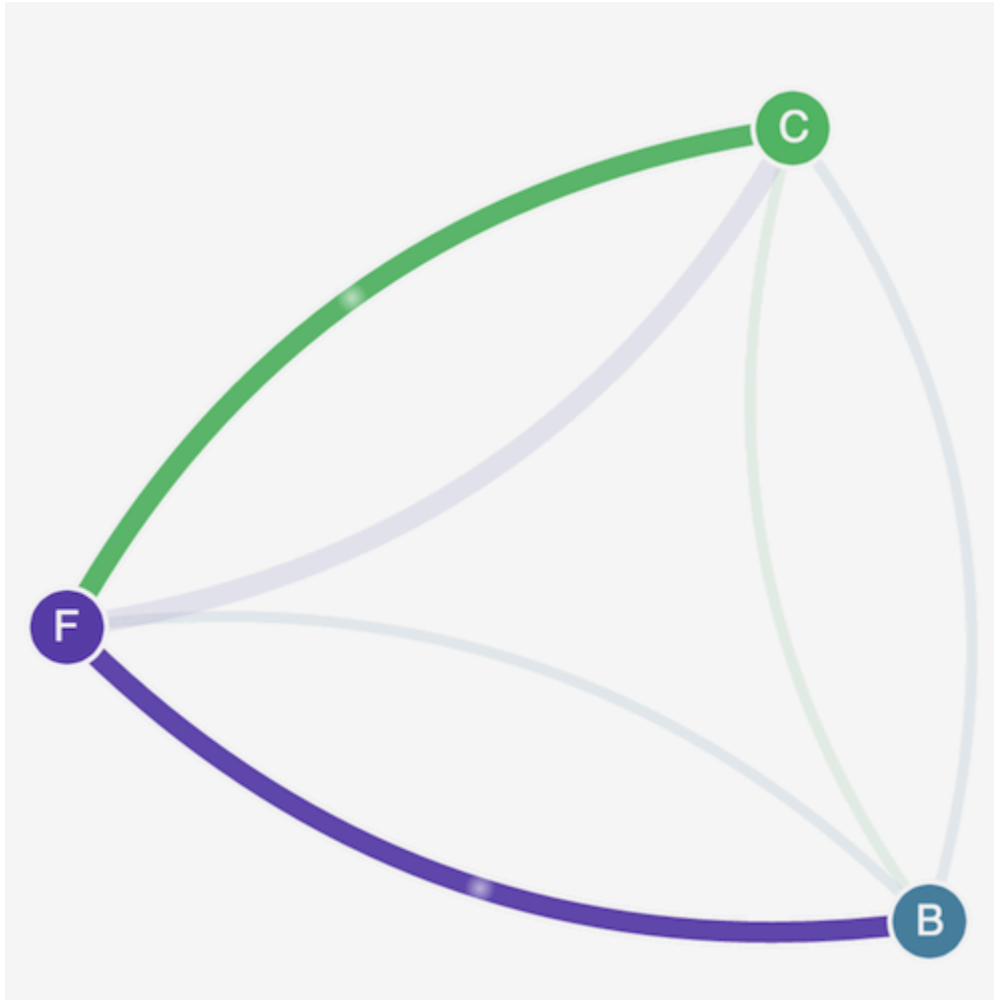


```

    role: client
  ports:
    - protocol: TCP
      port: 80

```

12 브라우저를 새로 고칩니다. 클라이언트가 프런트엔드 서비스와 통신할 수 있음을 알게 됩니다. 프런트엔드 서비스는 여전히 백엔드 서비스와 통신할 수 있습니다.



13(선택사항) 데모를 마쳤으면 리소스를 삭제할 수 있습니다.

```

kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/client.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/frontend.yaml

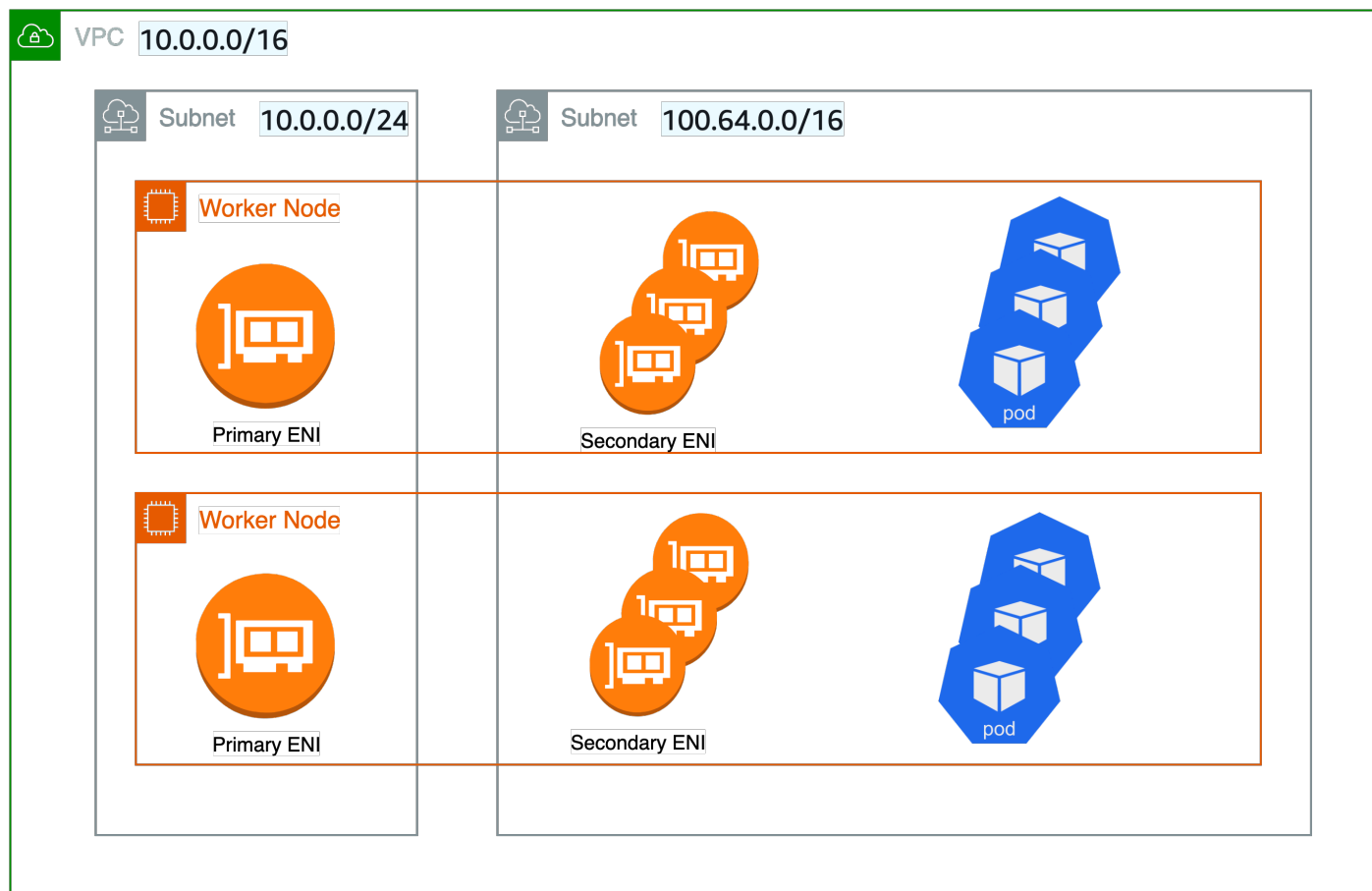
```

```
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/backend.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/management-ui.yaml
kubectl delete -f https://raw.githubusercontent.com/aws-samples/eks-workshop/2f9d29ed3f82ed6b083649e975a0e574fb8a4058/content/beginner/120_network-policies/calico/stars_policy_demo/create_resources.files/namespace.yaml
```

리소스를 삭제한 후에도 클러스터에서 예기치 않은 방식으로 네트워킹을 방해할 수 있는 네트워크 정책 엔드포인트가 노드에 있을 수 있습니다. 이러한 규칙을 제거하는 유일한 방법은 노드를 재부팅하거나 모든 노드를 종료하고 휴지통으로 이동하는 것입니다. 모든 노드를 종료하려면 Auto Scaling 그룹의 원하는 개수를 0으로 설정한 다음, 원하는 개수로 백업하거나 노드를 종료하면 됩니다.

사용자 지정 네트워킹을 통해 대체 서브넷에 포드 배포

적용 대상: Linux IPv4 Fargate 노드, Amazon EC2 인스턴스가 있는 Linux 노드



기본적으로 Kubernetes용 Amazon VPC CNl 플러그인은 Amazon EC2 노드의 [보조 Elastic 네트워크 인터페이스](#)(네트워크 인터페이스)를 생성하는 경우 노드의 기본 네트워크 인터페이스와 동일한 서브넷에서 생성합니다. 또한 기본 네트워크 인터페이스에 연결된 보조 네트워크 인터페이스에 동일한 보안 그룹을 연결합니다. 다음 중 하나 이상의 이유로 플러그인이 다른 서브넷에서 보조 네트워크 인터페이스를 생성하거나 다른 보안 그룹을 보조 네트워크 인터페이스에 연결하거나, 둘 다 하려고 할 수 있습니다.

- 기본 네트워크 인터페이스가 있는 서브넷에서 사용할 수 있는 IPv4 주소의 수는 제한되어 있습니다. 이렇게 하면 서브넷에서 생성할 수 있는 포드 수가 제한될 수 있습니다. 보조 네트워크 인터페이스에 다른 서브넷을 사용하면 포드에 사용 가능한 IPv4 주소 수를 늘릴 수 있습니다.
- 보안상의 이유로 포드는 노드의 기본 네트워크 인터페이스와 다른 서브넷 또는 보안 그룹을 사용해야 할 수 있습니다.
- 노드는 퍼블릭 서브넷에서 구성되며, 포드를 프라이빗 서브넷에 배치할 수 있습니다. 퍼블릭 서브넷과 연결된 라우팅 테이블에는 인터넷 게이트웨이로 가는 경로가 포함됩니다. 프라이빗 서브넷과 연결된 라우팅 테이블에는 인터넷 게이트웨이로 가는 경로가 포함되지 않습니다.

Tip

사용자 지정 네트워킹을 사용하지 않고 Amazon EKS 클러스터에 직접 새 서브넷 또는 기존 서브넷을 추가할 수도 있습니다. 자세한 내용은 [the section called “관리 콘솔에서 Amazon EKS 클러스터에 기존 VPC 서브넷 추가”](#) 단원을 참조하십시오.

고려 사항

다음은 기능 사용 시 고려 사항입니다.

- 사용자 지정 네트워킹을 사용 설정하면 기본 네트워크 인터페이스에 할당된 IP 주소가 포드에 할당되지 않습니다. 보조 네트워크 인터페이스의 IP 주소만 포드에 할당됩니다.
- 클러스터에서 IPv6 패밀리를 사용하는 경우 사용자 지정 네트워킹을 사용할 수 없습니다.
- 사용자 지정 네트워킹을 사용하여 IPv4 주소 소모를 완화하려는 경우 대신 IPv6 패밀리를 사용하여 클러스터를 생성할 수 있습니다. 자세한 내용은 [the section called “IPv6”](#) 단원을 참조하십시오.
- 보조 네트워크 인터페이스에 지정된 서브넷에 배포된 포드는 노드의 기본 네트워크 인터페이스와 다른 서브넷 및 보안 그룹을 사용할 수 있다고 해도 서브넷과 보안 그룹은 노드와 동일한 VPC에 있어야 합니다.

- Fargate의 경우 서브넷은 Fargate 프로필을 통해 제어됩니다. 자세한 내용은 [the section called “프로필 정의”](#) 섹션을 참조하세요.

Amazon EKS 노드에서 보조 네트워크 인터페이스 사용자 지정

자습서를 시작하기 전에 다음을 완료합니다.

- 고려 사항 검토
- Kubernetes용 Amazon VPC CNI 플러그인이 보조 네트워크 인터페이스를 생성하고 포드에 IP 주소를 할당하는 방법 숙지하기. 자세한 내용을 알아보려면 GitHub의 [ENI 할당](#)을 참조하세요.
- 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.
- kubectl 명령줄 도구는 장치 또는 AWS CloudShell에 설치됩니다. kubectl을 설치하거나 업그레이드하려면 [the section called “kubectl 및 eksctl 설정”](#) 부분을 참조하세요.
- Bash 셸에서 이 주제의 단계를 완료하는 것이 좋습니다. Bash 셸을 사용하지 않는 경우 줄 연속 문자 및 변수 설정 및 사용 방식과 같은 일부 스크립트 명령을 통해 셸이 조정되어야 합니다. 또한 셸의 인용 및 이스케이프 규칙이 다를 수 있습니다. 자세한 내용은 AWS 명령줄 인터페이스 사용 설명서의 [AWS CLI에서 문자열에 따옴표 사용하기](#)를 참조하세요.

본 자습서에서는 예제 값을 바꾸라고 언급된 경우를 제외하고는 이를 사용하는 것이 좋습니다. 프로덕션 클러스터의 단계를 완료하는 경우 예제 값을 바꿀 수 있습니다. 동일한 터미널에서 모든 단계를 완료하는 것이 좋습니다. 이는 변수가 여러 단계에서 설정되고 사용되면서 서로 다른 터미널에 존재하지 않기 때문입니다.

이 주제의 명령은 [AWS CLI 예제 사용](#)에 나열된 규칙에 따라 형식이 지정되었습니다. 사용 중인 AWS CLI [프로필](#)에 정의된 기본 AWS 리전과 다른 AWS 리전에 있는 리소스에 대해 명령줄의 명령을 실행하는 경우에는 명령에 `--region us-west-2`를 추가하고 us-west-2를 해당 AWS 지역으로 바꿔야 합니다.

프로덕션 클러스터에 사용자 지정 네트워킹을 배포하려면 [the section called “2단계: VPC 구성”](#)로 이동합니다.

1단계: 테스트 VPC 및 클러스터 생성

다음 절차를 활용하면 테스트 VPC 및 클러스터를 생성하고 해당 클러스터에 대해 사용자 지정 네트워킹을 구성할 수 있습니다. 프로덕션 클러스터에서 사용할 수 있는 관련 없는 기능들은 이 주제에서 다루지 않으므로 프로덕션 워크로드에 테스트 클러스터를 사용하지 않는 것이 좋습니다. 자세한 내용은 [the section called “클러스터 생성”](#) 단원을 참조하십시오.

1. 다음 명령을 실행하여 `account_id` 변수를 정의합니다.

```
account_id=$(aws sts get-caller-identity --query Account --output text)
```

2. VPC를 생성합니다.

a. 테스트 시스템에 배포하는 경우 Amazon EKS AWS CloudFormation 템플릿을 사용하여 VPC를 생성합니다.

```
aws cloudformation create-stack --stack-name my-eks-custom-networking-vpc \
  --template-url https://s3.us-west-2.amazonaws.com/amazon-eks/
cloudformation/2020-10-29/amazon-eks-vpc-private-subnets.yaml \
  --parameters ParameterKey=VpcBlock,ParameterValue=192.168.0.0/24 \
  ParameterKey=PrivateSubnet01Block,ParameterValue=192.168.0.64/27 \
  ParameterKey=PrivateSubnet02Block,ParameterValue=192.168.0.96/27 \
  ParameterKey=PublicSubnet01Block,ParameterValue=192.168.0.0/27 \
  ParameterKey=PublicSubnet02Block,ParameterValue=192.168.0.32/27
```

b. AWS CloudFormation 스택을 만드는 데 몇 분 정도 걸립니다. 스택의 배포 상태를 확인하려면 다음 명령을 실행합니다.

```
aws cloudformation describe-stacks --stack-name my-eks-custom-networking-vpc --
query Stacks\[ \].StackStatus --output text
```

명령의 출력이 `CREATE_COMPLETE`가 될 때까지 다음 단계를 계속하지 마세요.

c. 템플릿에서 생성한 프라이빗 서브넷 ID 값으로 변수를 정의합니다.

```
subnet_id_1=$(aws cloudformation describe-stack-resources --stack-name my-eks-
custom-networking-vpc \
  --query "StackResources[?
LogicalResourceId=='PrivateSubnet01'].PhysicalResourceId" --output text)
subnet_id_2=$(aws cloudformation describe-stack-resources --stack-name my-eks-
custom-networking-vpc \
```

```
--query "StackResources[?
LogicalResourceId=='PrivateSubnet02'].PhysicalResourceId" --output text)
```

- d. 이전 단계에서 검색된 서브넷의 가용 영역으로 변수를 정의합니다.

```
az_1=$(aws ec2 describe-subnets --subnet-ids $subnet_id_1 --query
'Subnets[*].AvailabilityZone' --output text)
az_2=$(aws ec2 describe-subnets --subnet-ids $subnet_id_2 --query
'Subnets[*].AvailabilityZone' --output text)
```

3. 클러스터 IAM 역할을 생성합니다.

- a. 다음 명령을 실행하여 IAM 신뢰 정책 JSON 파일을 생성합니다.

```
cat >eks-cluster-role-trust-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

- b. Amazon EKS 클러스터 IAM 역할을 생성합니다. 필요한 경우 eks-cluster-role-trust-policy.json 앞에 이전 단계에서 파일을 작성한 컴퓨터의 경로를 붙입니다. 명령은 사용자가 이전 단계에서 생성한 신뢰 정책을 역할에 연결합니다. [IAM 역할](#)을 생성하려면 역할을 생성하는 IAM 보안 주체에 iam:CreateRole 작업(권한)을 할당해야 합니다.

```
aws iam create-role --role-name myCustomNetworkingAmazonEKSClusterRole --assume-
role-policy-document file://"eks-cluster-role-trust-policy.json"
```

- c. 역할에 [AmazonEKSClusterPolicy](#)라는 Amazon EKS 관리 정책을 연결합니다. IAM 정책을 [IAM 보안 주체](#)에 연결하려면 정책을 연결하는 보안 주체에 iam:AttachUserPolicy 또는 iam:AttachRolePolicy의 IAM 작업(권한) 중 하나를 할당해야 합니다.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/
AmazonEKSClusterPolicy --role-name myCustomNetworkingAmazonEKSClusterRole
```

4. Amazon EKS 클러스터를 생성하고 이와 통신하도록 디바이스를 구성합니다.

a. 클러스터를 생성합니다.

```
aws eks create-cluster --name my-custom-networking-cluster \
  --role-arn arn:aws:iam::$account_id:role/myCustomNetworkingAmazonEKSClusterRole \
  --resources-vpc-config subnetIds=$subnet_id_1,"$subnet_id_2"
```

Note

요청의 가용 영역 중 하나에 Amazon EKS 클러스터를 생성하는 데 충분한 용량이 없다는 오류가 표시될 수 있습니다. 이 경우 오류 출력에는 새 클러스터를 지원할 수 있는 가용 영역이 포함됩니다. 사용자 계정의 지원 가용 영역에 있는 2개 이상의 서브넷을 사용하여 클러스터를 다시 생성합니다. 자세한 내용은 [the section called “용량 부족”](#) 섹션을 참조하세요.

b. 클러스터를 생성하는 데 몇 분 정도 걸립니다. 클러스터의 배포 상태를 확인하려면 다음 명령을 실행합니다.

```
aws eks describe-cluster --name my-custom-networking-cluster --query
cluster.status
```

명령의 출력이 "ACTIVE"가 될 때까지 다음 단계를 계속하지 마세요.

c. 클러스터와 통신하도록 kubectl을 구성합니다.

```
aws eks update-kubeconfig --name my-custom-networking-cluster
```

2단계: VPC 구성

이 자습서를 시작하려면 [the section called “1단계: 테스트 VPC 및 클러스터 생성”](#)에서 생성된 VPC가 있어야 합니다. 프로덕션 클러스터의 경우 모든 예제 값을 사용자 값으로 바꿔 사용자 VPC 맞게 단계를 조정합니다.

1. 현재 설치된 Kubernetes용 Amazon VPC CNI 플러그인이 최신 버전인지 확인합니다. Amazon EKS 추가 기능 유형의 최신 버전을 확인하고 사용 중인 버전을 이 버전으로 업데이트하려면 [the section called “추가 기능 업데이트”](#) 부분을 참조하세요. 자체 관리형 추가 기능 유형의 최신 버전을 확인하고 사용 중인 버전을 이 버전으로 업데이트하려면 [the section called “Amazon VPC CNI”](#) 부분을 참조하세요.
2. 클러스터 VPC ID를 검색하여 이후 단계에서 사용할 수 있도록 변수에 저장합니다.

```
vpc_id=$(aws eks describe-cluster --name my-custom-networking-cluster --query
"cluster.resourcesVpcConfig.vpcId" --output text)
```

3. 추가 CIDR(Classless Inter-Domain Routing) 블록을 클러스터의 VPC 연결합니다. CIDR 블록은 기존에 연결된 모든 CIDR 블록과 겹칠 수 없습니다.
 - a. VPC에 연결된 현재 CIDR 블록을 봅니다.

```
aws ec2 describe-vpcs --vpc-ids $vpc_id \
  --query 'Vpcs[*].CidrBlockAssociationSet[*].{CIDRBlock: CidrBlock, State:
  CidrBlockState.State}' --out table
```

예제 출력은 다음과 같습니다.

```
-----
|          DescribeVpcs          |
+-----+-----+
|  CIDRBlock  |  State  |
+-----+-----+
| 192.168.0.0/24 | associated |
+-----+-----+
```

- b. 추가 CIDR 블록을 VPC에 연결합니다. 다음 명령에서 CIDR 블록 값을 바꿉니다. 자세한 내용을 알아보려면 Amazon VPC 사용 설명서의 [추가 IPv4 CIDR 블록을 VPC와 연결](#)을 참조하세요.

```
aws ec2 associate-vpc-cidr-block --vpc-id $vpc_id --cidr-block 192.168.1.0/24
```

- c. 새 블록이 연결되어 있는지 확인합니다.

```
aws ec2 describe-vpcs --vpc-ids $vpc_id --query
'Vpcs[*].CidrBlockAssociationSet[*].{CIDRBlock: CidrBlock, State:
  CidrBlockState.State}' --out table
```


예제 출력은 다음과 같습니다.

```
-----
|           DescribeVpcs           |
+-----+-----+
|   CIDRBlock   |   State   |
+-----+-----+
| 192.168.0.0/24 | associated |
| 192.168.1.0/24 | associated |
+-----+-----+
```

새 CIDR 블록의 State가 associated가 될 때까지 다음 단계를 진행하지 마세요.

4. 기존 서브넷이 있는 각 가용 영역에서 사용하려는 만큼 서브넷을 생성합니다. 이전 단계에서 VPC와 연결한 CIDR 블록 내에 있는 CIDR 블록을 지정합니다.
 - a. 새 서브넷을 생성합니다. 다음 명령에서 CIDR 블록 값을 바꿉니다. 서브넷은 기존 서브넷이 있는 것과 다른 VPC CIDR 블록에서 생성되어야 하지만 기존 서브넷과 동일한 가용 영역에서 생성되어야 합니다. 이 예제에서는 하나의 서브넷이 현재 프라이빗 서브넷이 존재하는 각 가용 영역의 새 CIDR 블록에서 생성됩니다. 생성된 서브넷의 ID는 이후 단계에서 사용할 수 있도록 변수에 저장됩니다. Name 값은 이전 단계에서 Amazon EKS VPC 템플릿을 사용하여 생성된 서브넷에 할당된 값과 일치합니다. 이름은 필요하지 않습니다. 다른 이름을 사용할 수 있습니다.

```
new_subnet_id_1=$(aws ec2 create-subnet --vpc-id $vpc_id --availability-zone $az_1
--cidr-block 192.168.1.0/27 \
--tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=my-eks-custom-
networking-vpc-PrivateSubnet01},{Key=kubernetes.io/role/internal-elb,Value=1}]' \
--query Subnet.SubnetId --output text)
new_subnet_id_2=$(aws ec2 create-subnet --vpc-id $vpc_id --availability-zone $az_2
--cidr-block 192.168.1.32/27 \
--tag-specifications 'ResourceType=subnet,Tags=[{Key=Name,Value=my-eks-custom-
networking-vpc-PrivateSubnet02},{Key=kubernetes.io/role/internal-elb,Value=1}]' \
--query Subnet.SubnetId --output text)
```

Important

기본적으로 새로운 서브넷은 VPC의 [기본 라우팅 테이블](#)과 암시적으로 연결됩니다. 이 라우팅 테이블을 사용하면 VPC 배포된 모든 리소스 간에 통신할 수 있습니다. 그러나 VPC와 연결된 CIDR 블록 외부에 있는 IP 주소를 보유한 리소스와는 통신할 수 없습니다.

자체 라우팅 테이블을 서브넷에 연결하여 이 동작을 변경할 수 있습니다. 자세한 내용을 알아보려면 Amazon VPC 사용 설명서의 [서브넷 라우팅 테이블](#)을 참조하세요.

b. VPC에서 현재 서브넷을 확인합니다.

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$vpc_id" \
  --query 'Subnets[*].{SubnetId: SubnetId,AvailabilityZone:
  AvailabilityZone,CidrBlock: CidrBlock}' \
  --output table
```

예제 출력은 다음과 같습니다.

DescribeSubnets			
AvailabilityZone	CidrBlock	SubnetId	
us-west-2d	192.168.0.0/27	subnet-example1	
us-west-2a	192.168.0.32/27	subnet-example2	
us-west-2a	192.168.0.64/27	subnet-example3	
us-west-2d	192.168.0.96/27	subnet-example4	
us-west-2a	192.168.1.0/27	subnet-example5	
us-west-2d	192.168.1.32/27	subnet-example6	

생성한 192.168.1.0 CIDR 블록의 서브넷이 192.168.0.0 CIDR 블록의 서브넷과 동일한 가용 영역에 있는지 확인할 수 있습니다.

3단계: Kubernetes 리소스 구성

1. AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG DaemonSet에서 true 환경 변수를 aws-node로 설정합니다.

```
kubectl set env daemonset aws-node -n kube-system
AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
```

2. [클러스터 보안 그룹](#)의 ID를 검색하여 이후 단계에서 사용할 수 있도록 변수에 저장합니다. Amazon EKS는 클러스터를 생성하는 경우 이 보안 그룹을 자동으로 생성합니다.

```
cluster_security_group_id=$(aws eks describe-cluster --name my-custom-networking-cluster --query cluster.resourcesVpcConfig.clusterSecurityGroupId --output text)
```

3. 포드를 배포하려는 각 서브넷에 대해 ENIConfig 사용자 지정 리소스를 만듭니다.

a. 각 네트워크 인터페이스 구성에 대해 고유한 파일을 생성합니다.

다음 명령은 이전 단계에서 생성된 두 서브넷용 ENIConfig 파일을 별도로 생성합니다. name의 값은 고유해야 합니다. 이름은 서브넷이 있는 가용 영역과 동일합니다. 클러스터 보안 그룹은 ENIConfig에 할당됩니다.

```
cat >$az_1.yaml <<EOF
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
  name: $az_1
spec:
  securityGroups:
    - $cluster_security_group_id
  subnet: $new_subnet_id_1
EOF
```

```
cat >$az_2.yaml <<EOF
apiVersion: crd.k8s.amazonaws.com/v1alpha1
kind: ENIConfig
metadata:
  name: $az_2
spec:
  securityGroups:
    - $cluster_security_group_id
  subnet: $new_subnet_id_2
EOF
```

프로덕션 클러스터의 경우 이전 명령을 다음과 같이 변경할 수 있습니다.

- \$cluster_security_group_id를 각 ENIConfig에 사용할 기존 [보안 그룹](#)의 ID로 바꿉니다.
- 가능하면 ENIConfig를 사용할 가용 영역과 동일하게 ENIConfigs의 이름을 지정하는 것이 좋습니다. 여러 이유로 가용 영역의 이름과 다른 ENIConfigs의 이름을 사용해야 할 수 있습니다. 예를 들어, 동일한 가용 영역에 두 개 이상의 서브넷이 있고 사용자 지정 네트워킹에서 둘 다 사용하려는 경우 동일한 가용 영역에 대한 여러 ENIConfigs가 필요합니다. 각

ENIConfig는 고유한 이름이 필요하므로 가용 영역 이름을 사용하여 ENIConfigs에 두 개 이상의 이름을 지정할 수 없습니다.

ENIConfig 이름이 모두 가용 영역 이름과 동일하지 않은 경우 \$az_1 및 \$az_2를 이전 명령의 사용자 자신의 이름으로 바꾸고 이 자습서의 뒷부분에서 [ENIConfig를 사용하여 노드에 주석을 달니다](#).

Note

프로덕션 클러스터에서 사용할 유효한 보안 그룹을 지정하지 않고 다음을 사용하는 경우

- Kubernetes용 Amazon VPC CNI 플러그인의 버전 1.8.0 이상을 사용하는 경우 노드의 기본 Elastic 네트워크 인터페이스와 연결된 보안 그룹이 사용됩니다.
- Kubernetes용 Amazon VPC CNI 플러그인을 1.8.0보다 이전 버전으로 설정하면 VPC의 기본 보안 그룹이 보조 네트워크 인터페이스에 할당됩니다.

Important

- `AWS_VPC_K8S_CNI_EXTERNALSNAT=false`는 Kubernetes용 Amazon VPC CNI 플러그인 구성의 기본 설정입니다. 기본 설정을 사용하는 경우 VPC와 연결된 CIDR 블록 중 하나에 있지 않은 IP 주소로 향하는 트래픽은 노드의 기본 네트워크 인터페이스의 보안 그룹 및 서브넷을 사용합니다. 보조 네트워크 인터페이스를 생성하는 데 사용된 ENIConfigs에서 정의된 서브넷 및 보안 그룹은 이 트래픽에는 사용되지 않습니다. 이 설정에 대한 자세한 내용을 알아보려면 [the section called “아웃바운드 트래픽”](#) 섹션을 참조하세요.
- 포드에 보안 그룹도 사용하는 경우 SecurityGroupPolicy에 지정된 보안 그룹이 ENIConfigs에 지정된 보안 그룹 대신 사용됩니다. 자세한 내용은 [the section called “포드의 보안 그룹”](#) 단원을 참조하십시오.

b. 다음 명령을 사용하여 생성된 각 사용자 지정 리소스 파일을 클러스터에 적용합니다.

```
kubectl apply -f $az_1.yaml
kubectl apply -f $az_2.yaml
```

4. ENIConfigs가 생성되었는지 확인합니다.

```
kubectl get ENIConfigs
```

예제 출력은 다음과 같습니다.

NAME	AGE
us-west-2a	117s
us-west-2d	105s

5. 프로덕션 클러스터에서 사용자 지정 네트워킹을 활성화하고 사용 중인 가용 영역 이외의 것으로 ENIConfigs의 이름을 지정한 경우 [다음 단계](#)로 이동하여 Amazon EC2 노드를 배포합니다.

Kubernetes가 클러스터에 생성된 모든 새로운 Amazon EC2 노드에 가용 영역의 ENIConfig를 자동으로 적용할 수 있습니다.

- a. 이 자습서의 테스트 클러스터의 경우 [다음 단계](#)로 이동합니다.

프로덕션 클러스터의 경우 `k8s.amazonaws.com/eniConfig` 환경 변수에 대한 키 [ENI_CONFIG_ANNOTATION_DEF](#)가 포함된 주석이 `aws-node DaemonSet`의 컨테이너 사양에 있는지 확인합니다.

```
kubectl describe daemonset aws-node -n kube-system | grep
ENI_CONFIG_ANNOTATION_DEF
```

출력이 반환되는 경우 주석이 존재합니다. 출력이 반환되지 않으면 변수가 설정되지 않은 것입니다. 프로덕션 클러스터의 경우 이 설정이나 다음 단계의 설정 중 하나를 사용할 수 있습니다. 이 설정을 사용하는 경우 다음 단계의 설정을 재정의합니다. 이 튜토리얼에서는 다음 단계의 설정이 사용됩니다.

- b. `aws-node DaemonSet`을 업데이트하여 클러스터에 생성된 모든 새로운 Amazon EC2 노드에 가용 영역의 ENIConfig를 자동으로 적용합니다.

```
kubectl set env daemonset aws-node -n kube-system
ENI_CONFIG_LABEL_DEF=topology.kubernetes.io/zone
```

4단계: Amazon EC2 노드 배포

1. 노드 IAM 역할을 생성합니다.

- a. 다음 명령을 실행하여 IAM 신뢰 정책 JSON 파일을 생성합니다.

```
cat >node-role-trust-relationship.json <<EOF
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Effect": "Allow",
    "Principal": {
      "Service": "ec2.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
  }
]
}
EOF
```

- b. IAM 역할을 생성하고 이후 단계에서 사용할 수 있도록 반환된 Amazon 리소스 이름(ARN)을 변수에 저장합니다.

```
node_role_arn=$(aws iam create-role --role-name myCustomNetworkingNodeRole --
assume-role-policy-document file://"node-role-trust-relationship.json" \
--query Role.Arn --output text)
```

- c. 필요한 3개의 IAM 관리형 정책을 IAM 역할에 연결합니다.

```
aws iam attach-role-policy \
--policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy \
--role-name myCustomNetworkingNodeRole
aws iam attach-role-policy \
--policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly \
--role-name myCustomNetworkingNodeRole
aws iam attach-role-policy \
--policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \
--role-name myCustomNetworkingNodeRole
```

Important

이 튜토리얼에서는 단순화를 위해 노드 IAM 역할에 [AmazonEKS_CNI_Policy](#) 정책이 연결되어 있습니다. 그러나 프로덕션 클러스터에서는 정책을 Kubernetes용 Amazon VPC CNI 플러그인에서만 사용되는 별도의 IAM 역할에 연결하는 것이 좋습니다. 자세한 내용은 [the section called “IRSA에 대한 구성”](#) 단원을 참조하십시오.

2. 다음 노드 그룹 유형 중 하나를 생성합니다. 배포할 인스턴스 유형을 확인하려면 [the section called “Amazon EC2 인스턴스 유형”](#) 섹션을 참조하세요. 이 자습서에서는 관리형, 시작 템플릿을 사용하

지 않거나 AMI ID가 지정되지 않은 시작 템플릿을 사용하는 경우, (Without a launch template or with a launch template without an AMI ID specified)의 옵션을 완료합니다. 프로덕션 워크로드에 노드 그룹을 사용하려는 경우 노드 그룹을 배포하기 전에 [관리형 노드 그룹](#) 및 [자체 관리형 노드 그룹](#) 옵션을 모두 숙지하는 것이 좋습니다.

- 관리형 - 다음 옵션 중 하나를 사용하여 노드 그룹을 배포합니다.
 - 시작 템플릿을 사용하지 않거나 AMI ID가 지정되지 않은 시작 템플릿을 사용하는 경우, (Without a launch template or with a launch template without an AMI ID specified) - 다음 명령을 실행합니다. 이 자습서에서는 예제 값을 사용합니다. 프로덕션 노드 그룹의 경우 모든 예제 값을 사용자 자신의 것으로 바꿉니다. 노드 그룹 이름은 63자를 초과할 수 없습니다. 문자나 숫자로 시작하되, 나머지 문자의 경우 하이픈과 밑줄을 포함할 수 있습니다.

```
aws eks create-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name my-nodegroup \
    --subnets $subnet_id_1 $subnet_id_2 --instance-types t3.medium --node-role $node_role_arn
```

- AMI ID가 지정된 시작 템플릿을 사용하는 경우 (With a launch template with a specified AMI ID)
 - A. 노드에 대한 Amazon EKS 권장 최대 포드 수를 결정합니다. [각 Amazon EC2 인스턴스 유형에 대한 Amazon EKS 권장 최대 포드](#)의 지침에 따라 해당 주제의 3단계에 `--cni-custom-networking-enabled`를 추가합니다. 다음 단계에서 사용할 수 있도록 출력을 적어 둡니다.
 - B. 시작 템플릿에서 Amazon EKS 최적화 AMI ID 또는 Amazon EKS 최적화 AMI에서 빌드한 사용자 지정 AMI를 지정한 다음 [시작 템플릿을 사용하여 노드 그룹 배포](#)를 지정하고 시작 템플릿에 다음 사용자 데이터를 입력합니다. 이 사용자 데이터는 인수를 `bootstrap.sh` 파일에 전달합니다. 부트스트랩 파일에 대한 자세한 내용은 GitHub에서 [bootstrap.sh](#) 섹션을 참조하세요. 20을 이전 단계의 값(권장) 또는 고유한 값으로 바꿉니다.

```
/etc/eks/bootstrap.sh my-custom-networking-cluster --use-max-pods false --
kubelet-extra-args '--max-pods=20'
```

Amazon EKS 최적화 AMI에서 빌드되지 않은 사용자 지정 AMI를 생성한 경우 직접 구성을 사용자 지정하여 생성해야 합니다.

- 자체 관리형
 - i. 노드에 대한 Amazon EKS 권장 최대 포드 수를 결정합니다. [the section called “각 Amazon EC2 인스턴스 유형의 Amazon EKS 권장 최대 포드 수”](#)의 지침을 따라 해당 주제의 3단계에 --

cni-custom-networking-enabled를 추가하세요. 다음 단계에서 사용할 수 있도록 출력을 적어 둡니다.

- ii. [the section called “Amazon Linux”](#)의 지침에 따라 노드 그룹을 배포합니다.

BootstrapArguments 파라미터에 다음 텍스트를 지정합니다. 20을 이전 단계의 값(권장) 또는 고유한 값으로 바꿉니다.

```
--use-max-pods false --kubelet-extra-args '--max-pods=20'
```

Note

프로덕션 클러스터의 노드가 훨씬 더 많은 수의 포드를 지원하도록 하려는 경우 다시 [the section called “각 Amazon EC2 인스턴스 유형의 Amazon EKS 권장 최대 포드 수”](#)에서 스크립트를 실행합니다. 또한 --cni-prefix-delegation-enabled 옵션을 명령에 추가합니다. 예를 들어 m5.large 인스턴스 유형에 대해 110이 반환됩니다. 이 기능을 사용 설정하는 방법에 대한 지침은 [the section called “IP 주소 증가”](#) 부분을 참조하세요. 사용자 지정 네트워킹에서 이 기능을 사용할 수 있습니다.

3. 노드 그룹을 생성하는 데 몇 분 정도 걸립니다. 다음 명령을 사용하여 관리형 노드 그룹의 생성 상태를 확인할 수 있습니다.

```
aws eks describe-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name my-nodegroup --query nodegroup.status --output text
```

반환되는 출력이 ACTIVE가 될 때까지 다음 단계를 진행하지 마세요.

4. 튜토리얼에서는 이 단계를 건너뛸 수 있습니다.

프로덕션 클러스터의 경우 사용 중인 가용 영역과 동일하게 ENIConfigs의 이름을 지정하지 않은 경우 노드에서 사용해야 하는 ENIConfig 이름으로 해당 노드에 주석을 달아야 합니다. 각 가용 영역에 서브넷이 하나만 있고 가용 영역과 동일한 이름으로 ENIConfigs의 이름을 지정한 경우에는 이 단계가 필요하지 않습니다. 이는 [이전 단계](#)에서 활성화한 경우 Kubernetes용 Amazon VPC CNI 플러그인이 자동으로 올바른 ENIConfig를 노드에 연결하기 때문입니다.

- a. 클러스터의 노드 목록을 가져옵니다.

```
kubectl get nodes
```

예제 출력은 다음과 같습니다.

NAME	STATUS	ROLES	AGE	VERSION
ip-192-168-0-126.us-west-2.compute.internal eks-810597c	Ready	<none>	8m49s	v1.22.9-
ip-192-168-0-92.us-west-2.compute.internal eks-810597c	Ready	<none>	8m34s	v1.22.9-

- b. 각 노드가 있는 가용 영역을 확인합니다. 이전 단계에서 반환된 각 노드에 대해 다음 명령을 실행합니다. 이때 이전 출력을 기반으로 IP 주소를 바꿉니다.

```
aws ec2 describe-instances --filters Name=network-interface.private-dns-name,Values=ip-192-168-0-126.us-west-2.compute.internal \
--query 'Reservations[].Instances[].{AvailabilityZone: Placement.AvailabilityZone, SubnetId: SubnetId}'
```

예제 출력은 다음과 같습니다.

```
[
  {
    "AvailabilityZone": "us-west-2d",
    "SubnetId": "subnet-Example5"
  }
]
```

- c. 각 노드에 서브넷 ID 및 가용 영역에 대해 생성한 ENIConfig로 주석을 겁니다. 여러 노드에 동일한 ENIConfig로 주석을 달 수 있지만, 노드에는 하나의 ENIConfig로만 주석을 달 수 있습니다. 예제 값을 사용자의 값으로 바꿉니다.

```
kubectl annotate node ip-192-168-0-126.us-west-2.compute.internal
k8s.amazonaws.com/eniConfig=EniConfigName1
kubectl annotate node ip-192-168-0-92.us-west-2.compute.internal
k8s.amazonaws.com/eniConfig=EniConfigName2
```

5. 사용자 지정 네트워킹 기능 사용으로 전환하기 전에 프로덕션 클러스터에 실행 중인 포드가 있는 노드가 있는 경우 다음 태스크를 완료해야 합니다.

- 사용자 지정 네트워킹 기능을 사용하는 사용 가능한 노드가 있는지 확인합니다.
- 노드를 차단하고 드레이닝하여 포드를 정상적으로 종료합니다. 자세한 내용을 알아보려면 Kubernetes 문서의 [안전한 노드 드레이닝](#)을 참조하세요.
- 노드를 종료합니다. 노드가 기존 관리형 노드 그룹에 있는 경우 노드 그룹을 삭제할 수 있습니다. 다음 명령을 실행합니다.

```
aws eks delete-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name my-nodegroup
```

k8s.amazonaws.com/eniConfig 레이블에 등록된 새 노드만 사용자 지정 네트워킹 기능을 사용합니다.

6. 포드가 이전 단계에서 생성한 서브넷 중 하나에 연결된 CIDR 블록의 IP 주소를 할당하는지 확인합니다.

```
kubectl get pods -A -o wide
```

예제 출력은 다음과 같습니다.

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP
	NODE			NOMINATED	NODE	READINESS
GATES						
kube-system	aws-node-2rk4	1/1	Running	0	7m19s	
	192.168.0.92		ip-192-168-0-92.us-west-2.compute.internal	<none>		
	<none>					
kube-system	aws-node-k96wp	1/1	Running	0	7m15s	
	192.168.0.126		ip-192-168-0-126.us-west-2.compute.internal	<none>		
	<none>					
kube-system	coredns-657694c6f4-smcgr	1/1	Running	0	56m	
	192.168.1.23		ip-192-168-0-92.us-west-2.compute.internal	<none>		
	<none>					
kube-system	coredns-657694c6f4-stwv9	1/1	Running	0	56m	
	192.168.1.28		ip-192-168-0-92.us-west-2.compute.internal	<none>		
	<none>					
kube-system	kube-proxy-jgshq	1/1	Running	0	7m19s	
	192.168.0.92		ip-192-168-0-92.us-west-2.compute.internal	<none>		
	<none>					
kube-system	kube-proxy-wx9vk	1/1	Running	0	7m15s	
	192.168.0.126		ip-192-168-0-126.us-west-2.compute.internal	<none>		
	<none>					

Coredns 포드에 VPC 추가한 192.168.1.0 CIDR 블록의 IP 주소가 할당되는지 볼 수 있습니다. 사용자 지정 네트워킹을 사용하지 않는 경우 원래 CIDR 블록만 VPC와 연결되기 때문에 192.168.0.0 CIDR 블록의 주소가 할당되었을 것입니다.

포드의 spec이 hostNetwork=true를 포함하는 경우 노드의 기본 IP 주소가 할당됩니다. 추가한 서브넷의 주소는 할당되지 않습니다. 기본적으로 이 값은 false로 설정됩니다. 이 값은 클러스터에서 실행되는 Kubernetes용 Amazon VPC CNI 플러그인(aws-node) 포드와 kube-proxy에 대해 true로 설정됩니다. 따라서 kube-proxy 및 플러그인의 aws-node 포드에 이전 출력의 192.168.1.x 주소가 할당되지 않습니다. 포드의 hostNetwork 설정에 대한 자세한 내용은 Kubernetes API Reference의 [PodSpec v1 core](#)를 참조하세요.

5단계: 튜토리얼 리소스 삭제

튜토리얼을 완료하면 생성한 리소스를 삭제하는 것이 좋습니다. 그런 다음 단계를 조정하여 프로덕션 클러스터에 대해 사용자 지정 네트워킹을 사용 설정할 수 있습니다.

1. 생성한 노드 그룹이 테스트용일 경우 삭제하세요.

```
aws eks delete-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name my-nodegroup
```

2. AWS CLI 출력에 클러스터가 삭제되었다고 표시된 이후에도 삭제 프로세스는 실제로 완료되지 않을 수 있습니다. 삭제 프로세스는 몇 분 정도 걸릴 수 있습니다. 다음 명령을 실행하여 완료되었는지 확인합니다.

```
aws eks describe-nodegroup --cluster-name my-custom-networking-cluster --nodegroup-name my-nodegroup --query nodegroup.status --output text
```

반환된 출력이 다음 출력과 비슷하게 될 때까지 진행하지 마세요.

```
An error occurred (ResourceNotFoundException) when calling the DescribeNodegroup operation: No node group found for name: my-nodegroup.
```

3. 생성한 노드 그룹이 테스트용일 경우 노드 IAM 역할을 삭제하세요.

a. 역할에서 정책을 분리합니다.

```
aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
aws iam detach-role-policy --role-name myCustomNetworkingNodeRole --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
```

b. 역할을 삭제합니다.

```
aws iam delete-role --role-name myCustomNetworkingNodeRole
```

4. 클러스터를 삭제하세요.

```
aws eks delete-cluster --name my-custom-networking-cluster
```

다음 명령을 사용하여 클러스터가 삭제되었는지 확인합니다.

```
aws eks describe-cluster --name my-custom-networking-cluster --query cluster.status  
--output text
```

다음과 비슷한 출력이 반환되면 클러스터가 성공적으로 삭제됩니다.

```
An error occurred (ResourceNotFoundException) when calling the DescribeCluster  
operation: No cluster found for name: my-custom-networking-cluster.
```

5. 클러스터 IAM 역할을 삭제합니다.

a. 역할에서 정책을 분리합니다.

```
aws iam detach-role-policy --role-name myCustomNetworkingAmazonEKSClusterRole --  
policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
```

b. 역할을 삭제합니다.

```
aws iam delete-role --role-name myCustomNetworkingAmazonEKSClusterRole
```

6. 이전 단계에서 생성한 서브넷을 삭제합니다.

```
aws ec2 delete-subnet --subnet-id $new_subnet_id_1  
aws ec2 delete-subnet --subnet-id $new_subnet_id_2
```

7. 생성한 VPC를 삭제합니다.

```
aws cloudformation delete-stack --stack-name my-eks-custom-networking-vpc
```

접두사를 사용하여 Amazon EKS 노드에 추가 IP 주소 할당

적용 대상: Amazon EC2 인스턴스가 있는 Linux 및 Windows 노드

적용 대상: 퍼블릭 및 프라이빗 서브넷

각 Amazon EC2 인스턴스는 최대 탄력적 네트워크 인터페이스 수와 각 네트워크 인터페이스에 할당할 수 있는 최대 IP 주소 수를 지원합니다. 각 노드에는 네트워크 인터페이스마다 IP 주소가 하나씩 필요합니다. 사용 가능한 다른 모든 IP 주소를 Pods에 할당할 수 있습니다. 각 Pod에는 자체 IP 주소가 필요합니다. 따라서 사용 가능한 컴퓨팅 및 메모리 리소스가 있지만 Pods에 할당할 IP 주소가 부족하여 추가 Pods를 수용할 수 없는 노드가 있을 수 있습니다.

노드에 개별 보조 IP 주소를 할당하는 대신 IP 접두사를 할당하여 노드가 Pods에 할당할 수 있는 IP 주소 수를 크게 늘릴 수 있습니다. 각 접두사에는 여러 IP 주소가 포함됩니다. IP 접두사 할당을 위해 클러스터를 구성하지 않는 경우 클러스터는 포드 연결에 필요한 네트워크 인터페이스와 IP 주소를 구성하기 위해 더 많은 Amazon EC2 애플리케이션 프로그래밍 인터페이스(API)를 직접적으로 호출해야 합니다. 클러스터가 더 큰 규모로 성장함에 따라 이러한 API 직접 호출의 빈도가 증가하여 포드 및 인스턴스 시작 시간이 더 길어질 수 있습니다. 이로 인해 규모가 크고 까다로운 워크로드의 수요를 충족하기 위해 확장이 지연되고 확장 요구 사항을 충족하기 위해 추가 클러스터와 VPC를 프로비저닝해야 하므로 비용 및 관리 오버헤드가 추가됩니다. 자세한 내용은 GitHub의 [Kubernetes Scalability thresholds](#)를 참조하세요.

Kubernetes용 Amazon VPC CNI 플러그인 기능과의 호환성

IP 접두사를 다음 기능과 함께 사용할 수 있습니다.

- IPv4 소스 네트워크 주소 변환-자세한 내용은 [the section called “아웃바운드 트래픽”](#) 섹션을 참조하세요.
- 클러스터, 포드 및 서비스에 대한 IPv6 주소-자세한 내용은 [the section called “IPv6”](#) 섹션을 참조하세요.
- Kubernetes 네트워크 정책을 사용하여 트래픽 제한-자세한 내용은 [the section called “Kubernetes 정책”](#) 섹션을 참조하세요.

다음 목록에서는 적용되는 Amazon VPC CNI 플러그인 설정에 대한 정보를 제공합니다. 각 설정에 대한 자세한 내용은 GitHub의 [amazon-vpc-cni-k8s](#)를 참조하세요.

- WARM_IP_TARGET
- MINIMUM_IP_TARGET

- WARM_PREFIX_TARGET

고려 사항

이 기능을 사용할 때 다음 사항을 고려하세요.

- 각 Amazon EC2 인스턴스 유형은 최대 포드 수를 지원합니다. 관리형 노드 그룹이 여러 인스턴스 유형으로 구성된 경우 클러스터의 인스턴스에 대한 최대 포드 수 가운데 최소 수가 클러스터의 모든 노드에 적용됩니다.
- 기본적으로 노드에서 실행할 수 있는 최대 Pods 수는 110개이지만 이 수를 변경할 수 있습니다. 숫자를 변경하고 기존 관리형 노드 그룹이 있는 경우 노드 그룹의 다음 AMI 또는 시작 템플릿 업데이트로 인해 새 노드에 변경된 값이 나타납니다.
- IP 주소 할당에서 IP 접두사 할당으로 전환할 때 기존 노드를 순차적으로 바꾸는 대신 새 노드 그룹을 생성하여 사용 가능한 IP 주소 수를 늘리는 것이 좋습니다. IP 주소와 접두사가 모두 할당된 노드에서 포드를 실행하면 광고된 IP 주소 용량에 불일치가 발생하여 노드의 향후 워크로드에 영향을 줄 수 있습니다. 전환을 수행하는 권장 방법은 Amazon EKS 모범 사례 안내서의 [Replace all nodes when migrating from Secondary IP mode to Prefix Delegation mode or vice versa](#)를 참조하세요.
- 보안 그룹 범위는 노드 수준입니다. 자세한 내용은 [보안 그룹](#)을 참조하세요.
- 네트워크 인터페이스에 할당된 IP 접두사는 노드당 높은 포드 밀도를 지원하며, 시작 시간이 가장 빠릅니다.
- IP 접두사 및 IP 주소는 표준 Amazon EC2 탄력적 네트워크 인터페이스와 연결됩니다. 특정 보안 그룹이 필요한 포드에는 브랜치 네트워크 인터페이스의 기본 IP 주소가 할당됩니다. IP 주소 또는 IP 접두사의 IP 주소를 얻는 포드와 동일한 노드에서 브랜치 네트워크 인터페이스를 얻는 포드와 혼합할 수 있습니다.
- Linux 노드만 있는 클러스터의 경우
 - 네트워크 인터페이스에 접두사를 할당하도록 추가 기능을 구성한 후에는 클러스터의 모든 노드 그룹에 있는 모든 노드를 제거하지 않으면 1.9.0(또는 1.10.1) 이전 버전으로 Kubernetes용 Amazon VPC CNI 플러그인 추가 기능을 다운그레이드할 수 없습니다.
 - `POD_SECURITY_GROUP_ENFORCING_MODE=standard` 및 `AWS_VPC_K8S_CNI_EXTERNALSNAT=false`로 설정하고 포드에 보안 그룹도 사용하는 경우 포드가 VPC 외부의 엔드포인트와 통신할 때 포드에 할당한 보안 그룹이 아닌 노드의 보안 그룹이 사용됩니다.

또한 [포드에 보안 그룹](#)을 사용하고 `POD_SECURITY_GROUP_ENFORCING_MODE=strict`인 경우 Pods가 VPC 외부의 엔드포인트와 통신할 때 Pod's 보안 그룹이 사용됩니다.

Amazon EKS 노드에 대해 사용 가능한 IP 주소 증가

노드에 개별 보조 IP 주소를 할당하는 대신 IP 접두사를 할당하여 노드가 포트에 할당할 수 있는 IP 주소 수를 크게 늘릴 수 있습니다.

사전 조건

- 기존 클러스터가 있어야 합니다. 배포하려면 [the section called “클러스터 생성”](#) 부분을 참조하세요.
- Amazon EKS 노드가 있는 서브넷에는 충분한 연속 /28(IPv4 클러스터용) 또는 /80(IPv6 클러스터용) CIDR(Classless Inter-Domain Routing) 블록이 있어야 합니다. IPv6 클러스터에는 Linux 노드만 있을 수 있습니다. IP 주소가 서브넷 CIDR 전체에 분산되어 있는 경우 IP 접두사를 사용하지 못할 수 있습니다. 다음과 같이 하는 것이 좋습니다:
 - 서브넷 CIDR 예약을 사용하여 예약된 범위 내의 IP 주소가 아직 사용 중이더라도 릴리스 시 IP 주소가 다시 할당되지 않도록 합니다. 이렇게 하면 분할 없이 접두사를 할당할 수 있습니다.
 - IP 접두사가 할당된 워크로드 실행에 특별히 사용되는 새 서브넷을 사용합니다. IP 접두사를 할당할 때 Windows 및 Linux 워크로드를 모두 동일한 서브넷에서 실행할 수 있습니다.
- 노드에 IP 접두사를 할당하려면 노드가 AWS Nitro를 기반으로 해야 합니다. NITO 기반이 아닌 인스턴스는 계속해서 개별 보조 IP 주소를 할당하지만 NITO 기반 인스턴스보다 포트에 할당할 IP 주소 수가 상당히 적습니다.
- Linux 노드만 있는 클러스터의 경우 – 클러스터가 IPv4 패밀리용으로 구성된 경우 버전 1.9.0 이상의 Kubernetes용 Amazon VPC CNI 플러그인 추가 기능이 설치되어 있어야 합니다. 현재 설치된 버전은 다음과 같은 명령으로 확인할 수 있습니다.

```
kubectl describe daemonset aws-node --namespace kube-system | grep Image | cut -d "/" -f 2
```

클러스터가 IPv6 패밀리용으로 구성된 경우 버전 1.10.1의 추가 기능이 설치되어 있어야 합니다. 플러그인 버전이 필요한 버전보다 이전인 경우 업데이트해야 합니다. 자세한 내용은 [Amazon VPC CNI를 사용하여 포트에 IP 할당](#)의 업데이트 섹션을 참조하세요.

- Windows 노드만 있는 클러스터의 경우
 - 클러스터와 해당 플랫폼 버전은 다음 표에 나와 있는 버전 이상이어야 합니다. 클러스터 버전을 업그레이드하려면 [the section called “Kubernetes 버전 업데이트”](#) 섹션을 참조하세요. 클러스터가 최소 플랫폼 버전이 아닌 경우 Amazon EKS가 플랫폼 버전을 업데이트할 때까지 노드에 IP 접두사를 할당할 수 없습니다.

Kubernetes 버전	플랫폼 버전
1.27	eks.3
1.26	eks.4

다음 명령에서 *my-cluster*를 클러스터 이름으로 바꾸고 수정된 명령(`aws eks describe-cluster --name my-cluster --query 'cluster.{\"Kubernetes Version\": version, \"Platform Version\": platformVersion}'`)을 실행하여 현재 Kubernetes 및 플랫폼 버전을 확인할 수 있습니다.

- 클러스터에 대해 Windows 지원이 활성화되어 있어야 합니다. 자세한 내용은 [the section called “Windows 지원 사용”](#) 섹션을 참조하세요.

노드에 IP 주소 접두사 할당

노드에 IP 주소 접두사를 할당하도록 클러스터를 구성합니다. 해당 노드의 운영 체제와 일치하는 절차를 완료합니다.

Linux

- 파라미터를 활성화하여 Amazon VPC CNI DaemonSet의 네트워크 인터페이스에 접두사를 할당합니다. 클러스터를 배포하는 경우 Kubernetes용 Amazon VPC CNI 플러그인 버전 1.10.1 이상의 추가 기능이 함께 배포됩니다. IPv6 패밀리를 사용하여 클러스터를 생성한 경우 이 설정은 기본적으로 true로 설정되었습니다. IPv4 패밀리를 사용하여 클러스터를 생성한 경우 이 설정은 기본적으로 false로 설정되었습니다.

```
kubectl set env daemonset aws-node -n kube-system ENABLE_PREFIX_DELEGATION=true
```

Important

서브넷에 사용 가능한 IP 주소가 있더라도 서브넷이 인접한 /28 블록을 사용할 수 없는 경우 Kubernetes용 Amazon VPC CNI 플러그인 로그에 다음 오류가 표시됩니다.

```
InsufficientCidrBlocks: The specified subnet does not have enough free cidr blocks to satisfy the request
```


이는 서브넷에 분산된 기존 보조 IP 주소의 조각화로 인해 발생할 수 있습니다. 이 오류를 해결하려면 새 서브넷을 생성하고 거기서 포드를 시작하거나 Amazon EC2 서브넷 CIDR 예약을 사용하여 접두사 할당과 함께 사용할 서브넷 내에 공간을 예약합니다. 자세한 내용은 Amazon VPC 사용 설명서의 [서브넷 CIDR 예약](#)을 참조하세요.

2. 시작 템플릿이 없는 관리형 노드 그룹을 배포하거나 AMI ID를 지정하지 않은 시작 템플릿을 사용하여 배포할 계획이고, 필수 구성 요소에 나열된 버전보다 늦거나 같은 버전의 Kubernetes용 Amazon VPC CNI 플러그인을 사용하는 경우 다음 단계로 건너뛰세요. 관리형 노드 그룹은 자동으로 최대 포드 수를 계산합니다.

AMI ID를 지정한 시작 템플릿을 사용하여 자체 관리형 노드 그룹 또는 관리형 노드 그룹을 배포하는 경우 Amazon EKS 권장 노드의 최대 포드 수를 결정해야 합니다. `--cni-prefix-delegation-enabled`를 3단계에 추가하여 [각 Amazon EC2 인스턴스 유형에 대한 Amazon EKS 권장 최대 포드](#)의 지침을 따릅니다. 이후 단계에서 사용하기 위해 출력을 적어 둡니다.

Important

관리형 노드 그룹은 maxPods의 값에 최대 수를 적용합니다. vCPU가 30개 미만인 인스턴스의 경우 최대 수는 110이고 다른 모든 인스턴스의 경우 최대 수는 250입니다. 이 최대 수는 접두사 위임의 활성화 여부에 관계없이 적용됩니다.

3. IPv6용으로 구성된 클러스터를 사용하는 경우 다음 단계로 건너뛵니다.

다음 옵션 중 하나에서 파라미터를 지정합니다. 자신에게 적합한 옵션과 제공할 값을 확인하려면 GitHub의 [WARM_PREFIX_TARGET, WARM_IP_TARGET, and MINIMUM_IP_TARGET](#)을 참조하세요.

#을 0보다 큰 값으로 바꿀 수 있습니다.

- WARM_PREFIX_TARGET

```
kubectl set env ds aws-node -n kube-system WARM_PREFIX_TARGET=1
```

- WARM_IP_TARGET 또는 MINIMUM_IP_TARGET - 둘 중 하나의 값이 설정되면 WARM_PREFIX_TARGET에 설정된 모든 값을 재정의합니다.

```
kubectl set env ds aws-node -n kube-system WARM_IP_TARGET=5
```

```
kubectl set env ds aws-node -n kube-system MINIMUM_IP_TARGET=2
```

4. 하나 이상의 Amazon EC2 Nitro Amazon Linux 2 인스턴스 유형으로 다음 유형의 노드 그룹 중 하나를 생성합니다. Nitro 인스턴스 유형의 목록은 Amazon EC2 사용 설명서에서 [Nitro 시스템에 구축된 인스턴스](#) 섹션을 참조하세요. 이 기능은 Windows에서 지원되지 않습니다. **110**을 포함하는 옵션의 경우, 3단계의 값(권장) 또는 고유한 값으로 바꿉니다.

- 자체 관리형-[자체 관리형 Amazon Linux 노드 생성](#)의 지침에 따라 노드 그룹을 배포합니다. BootstrapArguments 파라미터에 다음 텍스트를 지정합니다.

```
--use-max-pods false --kubelet-extra-args '--max-pods=110'
```

eksctl을 사용하여 노드 그룹을 생성하려면 다음 명령을 사용합니다.

```
eksctl create nodegroup --cluster my-cluster --managed=false --max-pods-per-node 110
```

- 관리형 - 다음 옵션 중 하나를 사용하여 노드 그룹을 배포합니다.
 - 시작 템플릿이 없거나 AMI ID가 지정되지 않은 시작 템플릿-[클러스터에 대한 관리형 노드 그룹 생성](#)의 절차를 완료합니다. 관리형 노드 그룹은 자동으로 Amazon EKS 권장 최대 max-pods 값을 계산합니다.
 - AMI ID가 지정된 시작 템플릿을 사용하는 경우(With a launch template with a specified AMI ID) - 시작 템플릿에서 Amazon EKS 최적화 AMI ID 또는 Amazon EKS 최적화 AMI에서 빌드한 사용자 지정 AMI를 지정한 다음 [시작 템플릿을 사용하여 노드 그룹 배포](#)를 지정하고 시작 템플릿에 다음 사용자 데이터를 입력합니다. 이 사용자 데이터는 인수를 bootstrap.sh 파일에 전달합니다. 부트스트랩 파일에 대한 자세한 내용은 GitHub에서 [bootstrap.sh](#) 섹션을 참조하세요.

```
/etc/eks/bootstrap.sh my-cluster \
--use-max-pods false \
--kubelet-extra-args '--max-pods=110'
```

eksctl을 사용하여 노드 그룹을 생성하려면 다음 명령을 사용합니다.

```
eksctl create nodegroup --cluster my-cluster --max-pods-per-node 110
```

Amazon EKS 최적화 AMI에서 빌드되지 않은 사용자 지정 AMI를 생성한 경우 직접 구성을 사용자 지정하여 생성해야 합니다.

Note

또한 인스턴스의 서브넷과 다른 서브넷의 포드에 IP 주소를 할당하려면 이 단계에서 기능을 활성화해야 합니다. 자세한 내용은 [the section called “사용자 지정 네트워킹”](#) 섹션을 참조하세요.

Windows**1. IP 접두사의 할당을 활성화합니다.**

- a. 편집을 위해 amazon-vpc-cni ConfigMap을 엽니다.

```
kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
```

- b. 다음 줄을 data 섹션에 추가합니다.

```
enable-windows-prefix-delegation: "true"
```

- c. 파일을 저장하고 편집기를 닫습니다.
- d. 이 줄이 ConfigMap에 추가되었는지 확인합니다.

```
kubectl get configmap -n kube-system amazon-vpc-cni -o "jsonpath={.data.enable-windows-prefix-delegation}"
```

반환된 출력이 true가 아닌 경우 오류가 있을 수 있습니다. 단계를 다시 완료해 봅니다.

⚠ Important

서브넷에 사용 가능한 IP 주소가 있더라도 서브넷이 인접한 /28 블록을 사용할 수 없는 경우 Kubernetes용 Amazon VPC CNI 플러그인 로그에 다음 오류가 표시됩니다.

```
InsufficientCidrBlocks: The specified subnet does not have enough free cidr blocks to satisfy the request
```

이는 서브넷에 분산된 기존 보조 IP 주소의 조각화로 인해 발생할 수 있습니다. 이 오류를 해결하려면 새 서브넷을 생성하고 거기서 포드를 시작하거나 Amazon EC2 서브넷 CIDR

예약을 사용하여 접두사 할당과 함께 사용할 서브넷 내에 공간을 예약합니다. 자세한 내용은 Amazon VPC 사용 설명서의 [서브넷 CIDR 예약](#)을 참조하세요.

2. (선택 사항) 클러스터의 사전 규모 조정 및 동적 규모 조정 동작을 제어하기 위한 추가 구성을 지정합니다. 자세한 내용은 GitHub에서 [Windows의 접두사 위임 모드를 사용한 구성 옵션](#)을 참조하세요.

a. 편집을 위해 amazon-vpc-cni ConfigMap을 엽니다.

```
kubectl edit configmap -n kube-system amazon-vpc-cni -o yaml
```

b. **##** #을 0보다 큰 값으로 바꾸고 필요한 항목을 ConfigMap의 data 섹션에 추가합니다. warm-ip-target 또한 minimum-ip-target에 대한 값을 설정하면 해당 값이 warm-prefix-target에 대해 설정된 모든 값을 재정의합니다.

```
warm-prefix-target: "1"
warm-ip-target: "5"
minimum-ip-target: "2"
```

c. 파일을 저장하고 편집기를 닫습니다.

3. 하나 이상의 Amazon EC2 Nitro 인스턴스 유형으로 Windows 노드 그룹을 생성합니다. Nitro 인스턴스 유형의 목록은 Amazon EC2 사용 설명서에서 [Nitro 시스템에 구축된 인스턴스](#) 섹션을 참조하세요. 기본적으로 노드에 배포할 수 있는 최대 포드 수는 110개입니다. 이 수를 늘리거나 줄이려면 부트스트랩 구성의 사용자 데이터에 다음을 지정합니다. **max-pods-quantity**를 최대 포드 값으로 바꿉니다.

```
-KubeletExtraArgs '--max-pods=max-pods-quantity'
```

관리형 노드 그룹을 배포하는 경우 시작 템플릿에 이 구성을 추가해야 합니다. 자세한 내용은 [the section called “시작 템플릿”](#) 섹션을 참조하세요. Windows 부트스트랩 스크립트의 구성 파라미터에 대한 자세한 내용은 [the section called “부트스트랩 스크립트 구성 파라미터”](#) 섹션을 참조하세요.

최대 포드 및 사용 가능한 IP 주소 확인

1. 노드가 배포되면 클러스터의 노드를 확인합니다.

```
kubectl get nodes
```

예제 출력은 다음과 같습니다.

NAME	STATUS	ROLES	AGE	VERSION
ip-192-168-22-103.region-code.compute.internal eks-6b7464	Ready	<none>	19m	v1.XX.X-
ip-192-168-97-94.region-code.compute.internal eks-6b7464	Ready	<none>	19m	v1.XX.X-

2. 노드 중 하나를 설명하여 노드에 대한 max-pods 값과 사용 가능한 IP 주소 수를 결정합니다.
192.168.30.193을 이전 출력에서 반환된 노드 중 하나의 이름에 있는 IPv4 주소로 바꿉니다.

```
kubectl describe node ip-192-168-30-193.region-code.compute.internal | grep 'pods\|PrivateIPv4Address'
```

예제 출력은 다음과 같습니다.

```
pods: 110
vpc.amazonaws.com/PrivateIPv4Address: 144
```

이전 출력에서 110은 **144**개의 IP 주소를 사용할 수 있더라도 Kubernetes가 노드에 배포할 최대 포드 수입니다.

개별 포드에 보안 그룹 할당

적용 대상: Amazon EC2 인스턴스가 있는 Linux 노드

적용 대상: 프라이빗 서브넷

포드용 보안 그룹은 Amazon EC2 보안 그룹을 Kubernetes 포드와 통합합니다. Amazon EC2 보안 그룹을 사용하여 여러 Amazon EC2 인스턴스 유형 및 Fargate 에서 실행 중인 노드에 배포하는 포드와의 인바운드 및 아웃바운드 네트워크 트래픽을 허용하는 규칙을 정의할 수 있습니다. 이 기능에 대한 자세한 설명은 [포드에 대한 보안 그룹 소개](#) 블로그 게시물을 참조하세요.

Kubernetes용 Amazon VPC CNI 플러그인 기능과의 호환성

다음 기능과 함께 포드에 대한 보안 그룹을 사용할 수 있습니다.

- IPv4 소스 네트워크 주소 변환-자세한 내용은 [the section called “아웃바운드 트래픽”](#) 섹션을 참조하세요.
- 클러스터, 포드 및 서비스에 대한 IPv6 주소-자세한 내용은 [the section called “IPv6”](#) 섹션을 참조하세요.

- Kubernetes 네트워크 정책을 사용하여 트래픽 제한 - 자세한 내용은 [the section called “Kubernetes 정책”](#) 섹션을 참조하세요.

고려 사항

포드의 보안 그룹을 배포하기 전에 다음 제한 사항 및 조건을 고려하세요.

- 포드에 대한 보안 그룹은 Windows 노드와 함께 사용할 수 없습니다.
- 포드의 보안 그룹은 Amazon VPC CNI 플러그인 버전 1.16.0 이상을 사용하여 Amazon EC2 노드가 포함된 IPv6 패밀리에 대해 구성된 클러스터에서 사용할 수 있습니다. Amazon VPC CNI 플러그인 버전 1.7.7 이상을 사용하여 Fargate 노드만 포함하는 IPv6 패밀리에 대해 구성된 클러스터에서 포드의 보안 그룹을 사용할 수 있습니다. 자세한 내용은 [the section called “IPv6”](#) 섹션을 참조하세요.
- 포드에 대한 보안 그룹은 대다수 [Nitro 기반](#) Amazon EC2 인스턴스 패밀리에서 지원하지만, 패밀리의 모든 세대에서 지원하지는 않습니다. 예를 들면 m5, c5, r5, m6g, c6g 및 r6g 인스턴스 패밀리와 세대는 지원됩니다. t 패밀리의 인스턴스 유형은 지원되지 않습니다. 지원되는 인스턴스 유형의 전체 목록은 GitHub의 [limits.go](#) 파일을 참조하세요. 노드는 해당 파일에 IsTrunkingCompatible: true가 있는 나열된 인스턴스 유형 중 하나여야 합니다.
- 사용자 지정 네트워킹 및 포드의 보안 그룹을 함께 사용하는 경우 ENIConfig에 지정된 보안 그룹 대신 포드의 보안 그룹에 지정된 보안 그룹이 사용됩니다.
- 버전 1.10.2 이하의 Amazon VPC CNI 플러그인을 사용하고 포드 사양에 terminationGracePeriodSeconds 설정이 포함된 경우 설정 값은 0이 될 수 없습니다.
- 버전 1.10 이하의 Amazon VPC CNI 플러그인이나 기본 설정인 `POD_SECURITY_GROUP_ENFORCING_MODE=strict`로 버전 1.11를 사용하는 경우 externalTrafficPolicy를 Local로 설정한 인스턴스 대상을 사용하는 유형 NodePort 및 LoadBalancer의 Kubernetes 서비스는 보안 그룹을 할당하는 포드에서 지원하지 않습니다. 인스턴스 대상에 로드 밸런서를 사용하는 데 대한 자세한 내용은 [the section called “네트워크 로드 밸런싱”](#) 섹션을 참조하세요.
- 버전 1.10 이하의 Amazon VPC CNI 플러그인이나 기본 설정인 `POD_SECURITY_GROUP_ENFORCING_MODE=strict`로 버전 1.11를 사용하는 경우 소스 NAT는 보안 그룹이 할당된 에서 아웃바운드 트래픽에 대해 사용 중지되므로 아웃바운드 보안 그룹 규칙이 적용됩니다. 인터넷에 액세스하려면 보안 그룹이 할당된 포드를 NAT 게이트웨이 또는 인스턴스로 구성된 프라이빗 서브넷에 배포된 노드에서 실행해야 합니다. 퍼블릭 서브넷에 할당된 보안 그룹이 있는 포드는 인터넷에 액세스할 수 없습니다.

`POD_SECURITY_GROUP_ENFORCING_MODE=standard`로 설정된 버전 1.11 이상의 플러그인을 사용하는 경우 VPC 외부로 향하는 포드 트래픽은 인스턴스의 기본 네트워크 인터페이스의 IP 주소

로 변환됩니다. 이 트래픽의 경우 포드의 보안 그룹의 규칙이 아닌 기본 네트워크 인터페이스에 대한 보안 그룹의 규칙이 사용됩니다.

- 연결된 보안 그룹이 있는 포드를 통해 Calico 네트워크 정책을 사용하려면 버전 1.11.0 이상의 Amazon VPC CNI 플러그인을 사용하고 `POD_SECURITY_GROUP_ENFORCING_MODE=standard`로 설정해야 합니다. 그렇지 않으면 연결된 보안 그룹이 있는 포드에서 주고받는 트래픽 흐름에는 Calico 네트워크 정책이 적용되지 않으며, Amazon EC2 보안 그룹으로만 적용이 제한됩니다. Amazon VPC CNI 버전을 업데이트하려면 [the section called “Amazon VPC CNI”](#) 섹션을 참조하세요.
- [NodeLocal DNSCache](#)를 사용하는 클러스터에서 보안 그룹을 사용하는 Amazon EC2 노드에서 실행되는 포드는 `POD_SECURITY_GROUP_ENFORCING_MODE=standard`로 설정된 버전 1.11.0 이상의 Amazon VPC CNI 플러그인에서만 지원됩니다. Amazon VPC CNI 플러그인 버전을 업데이트하려면 [the section called “Amazon VPC CNI”](#) 섹션을 참조하세요.
- 포드의 보안 그룹은 변동이 많은 포드의 경우 포드 시작 지연 시간 증가로 이어질 수 있습니다. 이는 리소스 컨트롤러의 속도 제한 때문입니다.
- EC2 보안 그룹 범위는 포드 수준입니다. 자세한 내용은 [보안 그룹](#)을 참조하세요.

`POD_SECURITY_GROUP_ENFORCING_MODE=standard` 및 `AWS_VPC_K8S_CNI_EXTERNALSNAT=false`를 설정하는 경우 VPC 외부의 엔드포인트로 향하는 트래픽은 포드의 보안 그룹이 아닌 노드의 보안 그룹을 사용합니다.

Amazon EKS 포드의 보안 그룹에 대해 Kubernetes용 Amazon VPC CNI 플러그인 구성

Amazon EC2 인스턴스에서 포드를 사용하는 경우 보안 그룹에 대한 Kubernetes용 Amazon VPC CNI 플러그인을 구성해야 합니다.

Fargate 포드만 사용하고 클러스터에 Amazon EC2 노드가 없는 경우 [the section called “SecurityGroupPolicy”](#) 섹션을 참조하세요.

1. 현재 Kubernetes용 Amazon VPC CNI 플러그인 버전은 다음 명령을 통해 확인할 수 있습니다.

```
kubectl describe daemonset aws-node --namespace kube-system | grep amazon-k8s-cni: |
cut -d : -f 3
```

예제 출력은 다음과 같습니다.

```
v1.7.6
```

Kubernetes용 Amazon VPC CNI 플러그인 버전이 1.7.7 이하인 경우 플러그인을 버전 1.7.7 이상으로 업데이트합니다. 자세한 내용은 [the section called “Amazon VPC CNI”](#) 단원을 참조하세요.

2. Amazon EKS 클러스터와 연결된 [클러스터 역할](#)에 [AmazonEKSVPCResourceController](#) 관리형 IAM 정책을 추가합니다. 정책을 사용하면 역할이 네트워크 인터페이스, 프라이빗 IP 주소, 네트워크 인스턴스의 연결 및 분리를 관리할 수 있습니다.

- a. 클러스터 IAM 역할의 이름을 검색하고 변수에 저장합니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
cluster_role=$(aws eks describe-cluster --name my-cluster --query cluster.roleArn --output text | cut -d / -f 2)
```

- b. 책을 역할에 연결합니다.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonEKSVPCResourceController --role-name $cluster_role
```

3. aws-node DaemonSet에서 ENABLE_POD_ENI 변수를 true로 설정하여 Amazon VPC CNI 추가 기능이 의 네트워크 인터페이스를 관리하도록 합니다. 이 설정이 true로 설정되면 클러스터의 각 노드에서 추가 기능이 cninode 사용자 지정 리소스를 생성합니다. VPC 리소스 컨트롤러는 설명 aws-k8s-trunk-eni를 사용하여 트렁크 네트워크 인터페이스라는 하나의 특수 네트워크 인터페이스를 생성하고 연결합니다.

```
kubectl set env daemonset aws-node -n kube-system ENABLE_POD_ENI=true
```

Note

트렁크 네트워크 인터페이스는 인스턴스 유형에서 지원하는 최대 네트워크 인터페이스 수에 포함됩니다. 인스턴스 유형별로 지원되는 최대 네트워크 인터페이스 수 목록은 Amazon EC2 사용 설명서에서 [인스턴스 유형별 네트워크 인터페이스당 IP 주소](#) 섹션을 참조하세요. 노드에 이미 최대 수의 표준 네트워크 인터페이스가 연결되어 있는 경우 VPC 리소스 컨트롤러가 공간을 예약합니다. 컨트롤러가 표준 네트워크 인터페이스를 분리 및 삭제하고 트렁크 네트워크 인터페이스를 만든 다음 인스턴스에 연결할 수 있도록 실행 중인 포드를 스케일 다운해야 합니다.

4. 다음 명령을 사용하여 CNINode 사용자 지정 리소스를 포함하는 노드를 확인할 수 있습니다. No resources found가 반환된 경우 몇 초 정도 기다렸다가 다시 시도하세요. 이전 단계에서는 몇 초 정도 걸리는 Kubernetes용 Amazon VPC CNI 플러그인 포드를 다시 시작해야 합니다.


```
kubectl get cninode -A
NAME FEATURES
ip-192-168-64-141.us-west-2.compute.internal [{"name":"SecurityGroupsForPods"}]
ip-192-168-7-203.us-west-2.compute.internal [{"name":"SecurityGroupsForPods"}]
```

1.15 이전의 VPC CNI 버전을 사용하는 경우 CNINode 사용자 지정 리소스 대신 노드 레이블이 사용됩니다. 다음 명령을 사용하여 노드 레이블 `aws-k8s-trunk-eni` 레이블이 `true`로 설정된 노드를 확인할 수 있습니다. No resources found가 반환된 경우 몇 초 정도 기다렸다가 다시 시도하세요. 이전 단계에서는 몇 초 정도 걸리는 Kubernetes용 Amazon VPC CNI 플러그인 포드를 다시 시작해야 합니다.

```
kubectl get nodes -o wide -l vpc.amazonaws.com/has-trunk-attached=true
-
```

트렁크 네트워크 인터페이스가 만들어지면 트렁크 또는 표준 네트워크 인터페이스에서 포드에 보조 IP 주소를 할당합니다. 노드가 삭제되면 트렁크 인터페이스가 자동으로 삭제됩니다.

이후 단계에서 포드에 대한 보안 그룹을 배포하면 VPC 리소스 컨트롤러가 `aws-k8s-branch-eni`의 설명과 함께 브랜치 네트워크 인터페이스라는 특수한 네트워크 인터페이스를 생성하고 거기에 보안 그룹을 연결합니다. 브랜치 네트워크 인터페이스는 노드에 연결된 표준 및 트렁크 네트워크 인터페이스와 함께 생성됩니다.

라이브니스 또는 준비 상태 프로브를 사용하는 경우 kubelet가 TCP를 사용하여 브랜치 네트워크 인터페이스의 포드에 연결할 수 있도록 TCP 초기 demux도 비활성화해야 합니다. TCP 초기 Demux를 비활성화하려면 다음 명령을 실행합니다.

```
kubectl patch daemonset aws-node -n kube-system \
  -p '{"spec": {"template": {"spec": {"initContainers": [{"env": [{"name": "DISABLE_TCP_EARLY_DEMUX", "value": "true"}], "name": "aws-vpc-cni-init"}]}}}'
```

Note

다음 단계에 설명된 대로 Kubernetes용 Amazon VPC CNI 플러그인 1.11.0 이상을 사용하고 `POD_SECURITY_GROUP_ENFORCING_MODE=standard`를 설정한 경우에는 이전 명령을 실행할 필요가 없습니다.

- 클러스터에서 NodeLocal DNSCache를 사용하거나, 자체 보안 그룹이 있는 포드를 통해 Calico 네트워크 정책을 사용하거나, 보안 그룹을 할당하려는 포드에 대해 externalTrafficPolicy를 Local로 설정한 인스턴스 대상을 사용하는 NodePort 및 LoadBalancer 유형의 Kubernetes 서비스가 있는 경우 버전 1.11.0 이상의 Kubernetes용 Amazon VPC CNI 플러그인 추가 기능을 사용하고 있어야 하며 다음 설정을 활성화해야 합니다.

```
kubectl set env daemonset aws-node -n kube-system
  POD_SECURITY_GROUP_ENFORCING_MODE=standard
```

중요: 포드 보안 그룹 규칙은 동일한 노드에 있는 포드 간 또는 포드와 서비스(예: **kubelet** 또는 **nodeLocalDNS**) 간의 트래픽에는 적용되지 않습니다. 동일한 노드에서 서로 다른 보안 그룹을 사용하는 포드는 서로 다른 서브넷에 구성되어 있기 때문에 통신할 수 없으며, 이러한 서브넷 사이에 라우팅이 비활성화되어 있습니다. 포드에서 VPC 외부의 주소로 향하는 아웃바운드 트래픽은 인스턴스의 기본 네트워크 인터페이스의 IP 주소로 변환된 네트워크 주소입니다 (AWS_VPC_K8S_CNI_EXTERNALSNAT=true로 설정하지 않은 경우 제외). 이 트래픽의 경우 포드의 보안 그룹의 규칙이 아닌 기본 네트워크 인터페이스에 대한 보안 그룹의 규칙이 사용됩니다. ** 이 설정을 기존 포드에 적용하려면 포드 또는 포드가 실행되고 있는 노드를 다시 시작해야 합니다.

- 포드에 대한 보안 그룹 정책을 사용하는 방법을 보려면 [the section called “SecurityGroupPolicy”](#) 섹션을 참조하세요.

Amazon EKS 포드에 대한 보안 그룹 정책 사용

포드에 대한 보안 그룹을 사용하려면 기존 보안 그룹이 있어야 합니다. 다음 단계에서는 포드의 보안 그룹 정책을 사용하는 방법을 보여 줍니다. 달리 명시되지 않는 한, 변수가 터미널에 걸쳐 지속되지 않는 다음 단계에서 사용되므로 동일한 터미널에서 모든 단계를 완료합니다.

Amazon EC2 인스턴스에 포드가 있는 경우 이 절차를 사용하기 전에 플러그인을 구성해야 합니다. 자세한 내용은 [the section called “구성”](#) 단원을 참조하십시오.

- 리소스를 배포할 Kubernetes 네임스페이스를 만듭니다. 사용하려는 네임스페이스의 이름으로 **my-namespace**를 바꿀 수 있습니다.

```
kubectl create namespace my-namespace
```

- Amazon EKS SecurityGroupPolicy를 클러스터에 배포합니다.

- 다음 콘텐츠를 디바이스에 복사합니다. 서비스 계정 레이블에 따라 포드를 선택하려는 경우 **podSelector**를 serviceAccountSelector로 바꿀 수 있습니다. 두 선택기 중 하나만 지정해야 합니다. 비어 있는 podSelector(예: podSelector: {})는 네임스페이스

의 모든 포드를 선택합니다. *my-role*을 자신의 역할 이름으로 변경할 수 있습니다. 비어 있는 serviceAccountSelector는 네임스페이스의 모든 서비스 계정을 선택합니다. *my-security-group-policy*를 자신의 SecurityGroupPolicy에 대한 이름으로 바꾸고 SecurityGroupPolicy를 생성하려는 네임스페이스로 *my-namespace*를 바꿀 수 있습니다.

*my_pod_security_group_id*를 기존 보안 그룹의 ID로 바꿔야 합니다. 기존 보안 그룹이 없으면 하나를 생성해야 합니다. 자세한 내용은 [Amazon EC2 사용 설명서](#)의 [Linux 인스턴스용 Amazon EC2 보안 그룹](#)을 참조하세요. 보안 그룹 ID는 1~5개를 지정할 수 있습니다. 둘 이상의 ID를 지정하면 모든 보안 그룹의 모든 규칙 조합이 선택한 포드에 적용됩니다.

```
cat >my-security-group-policy.yaml <<EOF
apiVersion: vpcresources.k8s.aws/v1beta1
kind: SecurityGroupPolicy
metadata:
  name: my-security-group-policy
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      role: my-role
  securityGroups:
    groupIds:
      - my_pod_security_group_id
EOF
```

Important

포드에 대해 지정한 보안 그룹 또는 그룹은 다음 기준을 충족해야 합니다.

- 존재해야 합니다. 존재하지 않을 경우 선택기와 일치하는 포드를 배포하면 포드가 생성 프로세스에 계속 남아 있습니다. 포드에 대해 설명하면 다음과 유사한 오류 메시지가 표시됩니다. An error occurred (InvalidSecurityGroupID.NotFound) when calling the CreateNetworkInterface operation: The securityGroup ID '*sg-05b1d815d1EXAMPLE*' does not exist.
- 보안 그룹에서는 프로브를 구성한 모든 포트를 통해 노드에 적용된 보안 그룹의 인바운드 통신(kubelet의 경우)을 허용해야 합니다.
- CoreDNS를 실행하는 (또는 포드가 실행되는 노드)에 할당된 보안 그룹에 대한 TCP 및 UDP 포트 53을 통한 아웃바운드 통신을 허용해야 합니다. CoreDNS 포드의 보안 그룹

에서는 지정하는 보안 그룹의 인바운드 TCP 및 UDP 포트 53 트래픽을 허용해야 합니다.

- 통신하려면 필요한 다른 포트와 통신하는 데 필요한 인바운드 및 아웃바운드 규칙이 있어야 합니다.
 - Fargate에서 보안 그룹을 사용하는 경우 포트가 Kubernetes 컨트롤 플레인과 통신할 수 있는 규칙이 있어야 합니다. 이 작업을 수행할 수 있는 가장 쉬운 방법은 클러스터 보안 그룹을 보안 그룹 중 하나로 지정하는 것입니다.
- 보안 그룹 정책은 새로 예약된 포트에만 적용되며, 실행 중인 포트에는 영향을 주지 않습니다.

b. 정책을 배포합니다.

```
kubectl apply -f my-security-group-policy.yaml
```

3. 이전 단계에서 지정한 *podSelector*의 *my-role* 값과 일치하는 레이블을 사용하여 샘플 애플리케이션을 배포하세요.

a. 다음 콘텐츠를 디바이스에 복사합니다. **## #**을 사용자 값으로 바꾼 다음 수정된 명령을 실행합니다. *my-role*을 바꾸는 경우 이전 단계에서 선택기에 지정한 값과 동일한지 확인합니다.

```
cat >sample-application.yaml <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
  namespace: my-namespace
  labels:
    app: my-app
spec:
  replicas: 4
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
        role: my-role
    spec:
      terminationGracePeriodSeconds: 120
      containers:
```

```

      - name: nginx
        image: public.ecr.aws/nginx/nginx:1.23
        ports:
          - containerPort: 80
    ---
apiVersion: v1
kind: Service
metadata:
  name: my-app
  namespace: my-namespace
  labels:
    app: my-app
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
EOF

```

- b. 다음 명령을 사용하여 애플리케이션을 배포합니다. 애플리케이션을 배포할 때 `role` 레이블과 일치하는 Kubernetes용 Amazon VPC CNI 플러그인과 이전 단계에서 지정한 보안 그룹이 포트에 적용됩니다.

```
kubectl apply -f sample-application.yaml
```

4. 샘플 애플리케이션을 통해 배포된 포드를 봅니다. 이 주제의 나머지 부분에서는 이 터미널을 TerminalA로 언급합니다.

```
kubectl get pods -n my-namespace -o wide
```

예제 출력은 다음과 같습니다.

NAME	READY	STATUS	RESTARTS	AGE	IP
NODE			NOMINATED	NODE	READINESS GATES
my-deployment-5df6f7687b-4fbjm	1/1	Running	0	7m51s	192.168.53.48
ip-192-168-33-28.region-code.compute.internal			<none>		<none>
my-deployment-5df6f7687b-j9fl4	1/1	Running	0	7m51s	192.168.70.145
ip-192-168-92-33.region-code.compute.internal			<none>		<none>
my-deployment-5df6f7687b-rjxcz	1/1	Running	0	7m51s	192.168.73.207
ip-192-168-92-33.region-code.compute.internal			<none>		<none>

my-deployment-5df6f7687b-zmb42	1/1	Running	0	7m51s	192.168.63.27
ip-192-168-33-28.region-code.compute.internal			<none>		<none>

Note

포드가 멈춘 경우 다음 팁을 시도해 보세요.

- 포드가 Waiting 상태에서 멈춘 경우 `kubectl describe pod my-deployment-xxxxxxxxxx-xxxxx -n my-namespace` 를 실행합니다. `Insufficient permissions: Unable to create Elastic Network Interface.`(권한 부족: 탄력적 네트워크 인터페이스를 생성할 수 없습니다.)가 표시되면 이전 단계에서 IAM 클러스터 역할에 IAM 정책을 추가했는지 확인합니다.
- 포드가 Pending 상태로 멈춰 있는 경우 노드 인스턴스 유형이 [limits.go](https://aws.amazon.com/ec2/instance-types/)에 나열되어 있고 인스턴스 유형에서 지원하는 최대 브랜치 네트워크 인터페이스 수와 노드 그룹의 노드 수를 곱한 값이 아직 충족되지 않았는지 확인합니다. 예를 들어, m5.large 인스턴스는 9개의 브랜치 네트워크 인터페이스를 지원합니다. 노드 그룹에 5개의 노드가 있는 경우 노드 그룹에 대해 최대 45개의 브랜치 네트워크 인터페이스를 생성할 수 있습니다. 배포하려는 46 번째 포드는 연결된 보안 그룹이 있는 다른 포드가 삭제될 때까지 Pending 상태로 남아 있습니다.

`kubectl describe pod my-deployment-xxxxxxxxxx-xxxxx -n my-namespace` 를 실행할 때 다음 메시지와 유사한 메시지가 표시될 경우 무시해도 됩니다. 이 메시지는 Kubernetes용 Amazon VPC CNI 플러그인이 호스트 네트워킹을 설정하려고 할 때 네트워크 인터페이스가 생성되는 동안 실패할 경우 나타날 수 있습니다. 플러그인은 네트워크 인터페이스가 생성될 때까지 이 이벤트를 기록합니다.

```
Failed to create Pod sandbox: rpc error: code = Unknown desc = failed to set up
sandbox container "e24268322e55c8185721f52df6493684f6c2c3bf4fd59c9c121fd4cdc894579f"
network for Pod "my-deployment-5df6f7687b-4fbjm": networkPlugin
cni failed to set up Pod "my-deployment-5df6f7687b-4fbjm-c89wx_my-namespace" network:
add cmd: failed to assign an IP address to container
```

인스턴스 유형에서 실행할 수 있는 최대 포드 수를 초과할 수 없습니다. 각 인스턴스 유형에서 실행할 수 있는 최대 포드 수 목록은 GitHub의 [eni-max-pods.txt](https://github.com/aws/amazon-vpc-cni-k8s/blob/master/docs/eni-max-pods.txt)를 참조하세요. 연결된 보안 그룹이 있는 포드를 삭제하거나 포드가 실행 중인 노드를 삭제하면 VPC 리소스 컨트롤러가 브랜치 네트워크 인터페이스를 삭제합니다. 보안 그룹에 대해 포드를 사용하는 포드가 있는 클러스터를 삭제하면 컨트롤러가 브랜치 네트워크 인터페이스를 삭제하지 않으므로 직접 삭제해야 합니다. 네트워크 인터페

이스를 삭제하는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 [네트워크 인터페이스 삭제](#)를 참조하세요.

5. 별도의 터미널에서 포트 중 하나에 셸을 적용합니다. 이 주제의 나머지 부분에서는 이 터미널을 TerminalB로 언급합니다. `5df6f7687b-4fbjm`을 이전 단계의 결과에서 반환된 포트 중 하나의 ID로 바꾸세요.

```
kubectl exec -it -n my-namespace my-deployment-5df6f7687b-4fbjm -- /bin/bash
```

6. TerminalB의 셸에서 샘플 애플리케이션이 작동하는지 확인합니다.

```
curl my-app
```

예제 출력은 다음과 같습니다.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

애플리케이션이 실행되는 모든 포드는 사용자가 생성한 보안 그룹과 연결되기 때문에 이 결과가 표시됩니다. 해당 그룹에는 보안 그룹이 연결되는 모든 포트 간에 모든 트래픽을 허용하는 규칙이 포함되어 있습니다. DNS 트래픽은 해당 보안 그룹에서 노드와 연결된 클러스터 보안 그룹으로 아웃바운드 트래픽이 허용됩니다. 노드가 포트에서 이름 조회를 수행한 CoreDNS 포드를 실행하고 있습니다.

7. 보안 그룹에서 클러스터 보안 그룹으로의 DNS 통신을 허용하는 보안 그룹 규칙을 TerminalA에서 제거합니다. 이전 단계에서 클러스터 보안 그룹에 DNS 규칙을 추가하지 않은 경우 `$my_cluster_security_group_id`를 사용자가 규칙을 생성한 보안 그룹의 ID로 바꿉니다.

```
aws ec2 revoke-security-group-ingress --group-id $my_cluster_security_group_id --
security-group-rule-ids $my_tcp_rule_id
aws ec2 revoke-security-group-ingress --group-id $my_cluster_security_group_id --
security-group-rule-ids $my_udp_rule_id
```

8. TerminalB에서 다시 애플리케이션에 액세스하려고 시도합니다.

```
curl my-app
```

예제 출력은 다음과 같습니다.

```
curl: (6) Could not resolve host: my-app
```

포드에서 클러스터 보안 그룹을 연결해 주는 CoreDNS 포드에 더 이상 액세스할 수 없기 때문에 이 시도는 실패합니다. 클러스터 보안 그룹에는 포드에 연결된 보안 그룹에서 DNS 통신을 허용하는 보안 그룹 규칙이 더 이상 없습니다.

이전 단계에서 포드 중 하나에 대해 반환된 IP 주소를 사용하여 애플리케이션에 액세스하려고 시도하는 경우 보안 그룹이 연결되어 있는 포드 간에 모든 포트가 허용되고 이름 조회가 필요 없기 때문에 응답이 계속 표시됩니다.

9. 실험이 끝나면 생성한 샘플 보안 그룹 정책, 애플리케이션 및 보안 그룹을 제거할 수 있습니다. TerminalA에서 다음 명령을 실행합니다.

```
kubectl delete namespace my-namespace
aws ec2 revoke-security-group-ingress --group-id $my_pod_security_group_id --
security-group-rule-ids $my_inbound_self_rule_id
wait
sleep 45s
aws ec2 delete-security-group --group-id $my_pod_security_group_id
```

Multus와 함께 포드에 여러 네트워크 인터페이스 연결

Multus CNI는 여러 네트워크 인터페이스를 포드에 연결할 수 있는 Amazon EKS용 컨테이너 네트워크 인터페이스(CNI) 플러그인입니다. 자세한 내용은 GitHub에서 [Multus-CNI](#) 설명서를 참조하세요.

Amazon EKS에서 각 포드에는 Amazon VPC CNI 플러그인에서 할당하는 하나의 네트워크 인터페이스가 있습니다. Multus를 사용하면 여러 인터페이스가 있는 멀티 호밍 포드를 생성할 수 있습니다. 이는 여러 다른 CNI 플러그인을 호출할 수 있는 CNI 플러그인인 '메타 플러그인'으로 작동하는 Multus를 통해 생성됩니다. AWS의 Multus 지원은 Amazon VPC CNI 플러그인이 기본 위임 플러그인으로 구성됩니다.

- Amazon EKS는 단일 루트 I/O 가상화(SR-IOV) 및 데이터 영역 개발 키트(DPDK) CNI 플러그인을 구축하지도 게시하지도 않습니다. 그러나 Multus 관리형 호스트 디바이스 및 ipvlan 플러그인을 통해 Amazon EC2 탄력적 네트워크 어댑터(ENA)에 직접 연결하여 패킷 가속화를 실현할 수 있습니다.

- Amazon EKS는 추가 CNI 플러그 인을 간단하게 연결할 수 있는 일반 프로세스를 제공하는 Multus를 지원합니다. Multus와 연결 프로세스가 지원되지만 AWS에서는 연결될 수 있는 호환 가능한 모든 CNI 플러그 인이나 연결 구성과 관련이 없는 CNI 플러그 인에서 발생할 수 있는 문제에 대한 지원을 제공하지 않습니다.
- Amazon EKS는 Multus 플러그 인에 대한 지원 및 수명 주기 관리를 제공하지만 추가 네트워크 인터페이스와 연결된 IP 주소 또는 추가 관리에 대해서는 책임을 지지 않습니다. Amazon VPC CNI 플러그 인을 사용하는 기본 네트워크 인터페이스의 IP 주소와 관리는 변경되지 않습니다.
- Amazon VPC CNI 플러그 인만 기본 위임 플러그 인으로 공식 지원됩니다. 기본 네트워킹에 대해 Amazon VPC CNI 플러그 인을 사용하지 않도록 선택한 경우 게시된 Multus 설치 매니페스트를 수정하여 기본 위임 플러그 인을 대체 CNI로 재구성해야 합니다.
- Multus는 Amazon VPC CNI를 기본 CNI로 사용하는 경우에만 지원됩니다. 상위 인터페이스 및 보조 또는 기타 인터페이스에 사용되는 경우 Amazon VPC CNI를 지원하지 않습니다.
- Amazon VPC CNI 플러그 인이 포드에 할당된 추가 네트워크 인터페이스를 관리하지 못하도록 하려면 네트워크 인터페이스에 다음 태그를 추가합니다.

키

: node.k8s.amazonaws.com/no_manage

USD 상당

: true

- Multus는 네트워크 정책과 호환되지만, 포드에 연결된 추가 네트워크 인터페이스의 일부일 수 있는 포트 및 IP 주소를 포함하도록 정책을 강화해야 합니다.

구현에 대한 자세한 내용은 GitHub에서 [Multus 설정 안내서](#)를 참조하세요.

Amazon EKS 클러스터에 대한 대체 CNI 플러그인

[Kubernetes용 Amazon VPC CNI 플러그인](#)은 Amazon EKS 및 Amazon EC2 노드에서 지원하는 유일한 CNI 플러그인입니다. Amazon EKS는 Amazon EKS Hybrid Nodes용 Cilium 및 Calico의 핵심 기능을 지원합니다. Amazon EKS는 업스트림 Kubernetes를 실행하므로 클러스터의 Amazon EC2 노드에 대체 호환 CNI 플러그인을 설치할 수 있습니다. 클러스터에 Fargate 노드가 있는 경우 해당 Kubernetes용 Amazon VPC CNI 플러그인은 이미 Fargate 노드에 있습니다. Fargate 노드와 함께 사용할 수 있는 유일한 CNI 플러그인입니다. Fargate 노드에 대체 CNI 플러그인을 설치하려는 시도가 실패합니다.

Amazon EC2 노드에서 대체 CNI 플러그인을 사용하려는 경우 해당 플러그인에 대한 상용 지원을 받거나 CNI 플러그인 프로젝트의 문제를 해결하고 프로젝트 수정에 기여할 수 있는 전문 지식을 내부적으로 보유하는 것이 좋습니다.

Amazon EKS는 호환 가능한 대체 CNI 플러그인을 지원하는 파트너 네트워크와 관계를 유지합니다. 버전, 자격 요건 및 수행된 테스트에 대한 자세한 내용은 다음 파트너 설명서를 참조하세요.

파트너	Product	설명서
Tigera	Calico	설치 지침
Isovalent	Cilium	설치 지침
주니퍼	클라우드 네이티브 콘트레일 네트워킹(CN2)	설치 지침
VMware	Antrea	설치 지침

Amazon EKS는 모든 사용 사례를 포괄하는 다양한 옵션을 제공하는 것을 목표로 합니다.

호환되는 대체 네트워크 정책 플러그인

[Calico](#)는 컨테이너 네트워킹 및 보안을 위해 널리 채택되는 솔루션입니다. EKS에서 Calico를 사용하면 EKS 클러스터에 대해 완벽하게 규정을 준수하는 네트워크 정책 시행이 가능합니다. 또한 기본 VPC의 IP 주소를 보존하는 Calico의 네트워킹을 사용하도록 선택할 수 있습니다. [Calico Cloud](#)는 Calico Open Source의 기능을 강화하여 고급 보안 및 관찰성 기능을 제공합니다.

연결된 보안 그룹이 있는 포드에서 주고받는 트래픽 흐름에는 Calico 네트워크 정책이 적용되지 않으며, Amazon VPC 보안 그룹 적용으로만 제한됩니다.

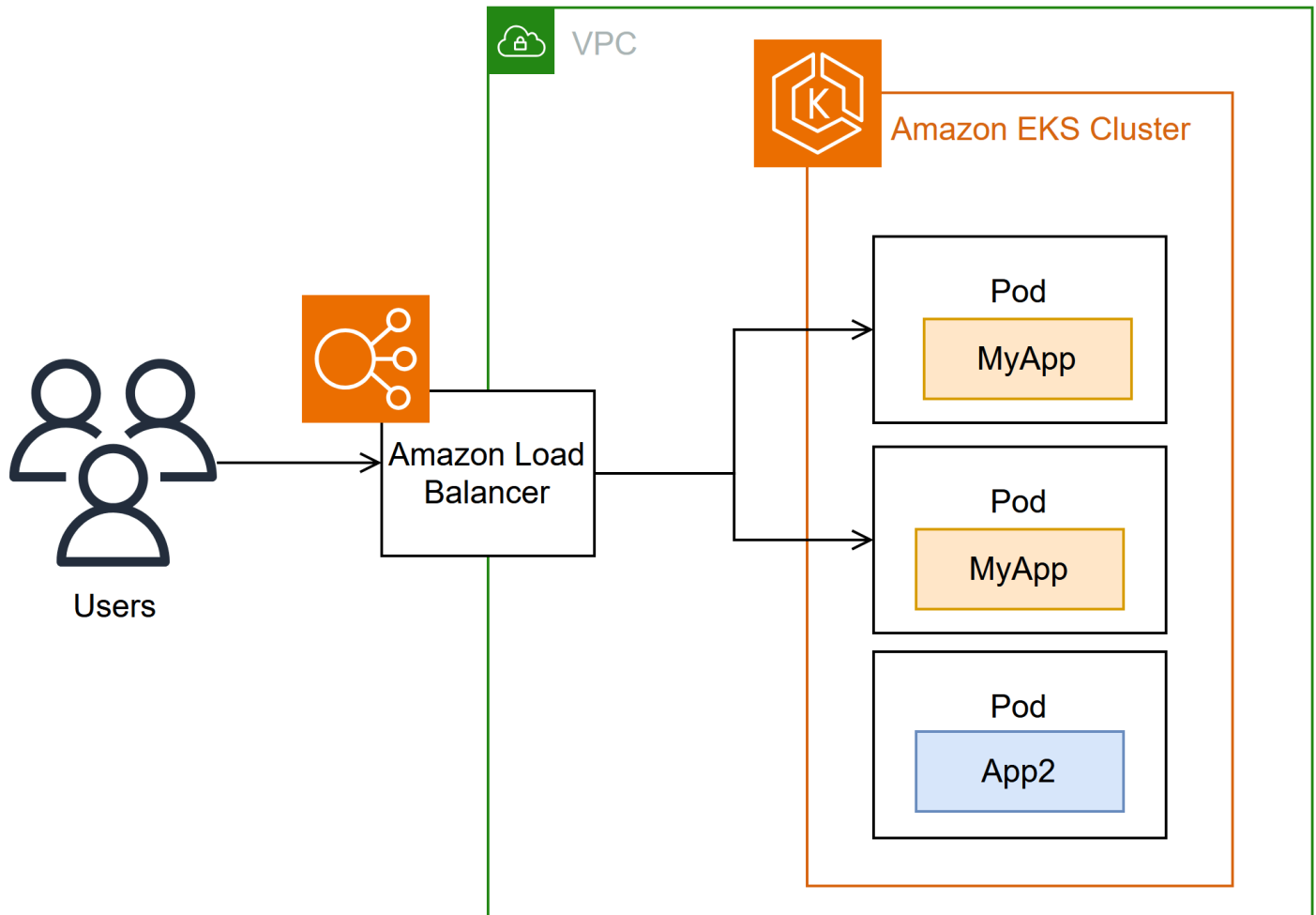
Calico 네트워크 정책 시행을 사용하는 경우 환경 변수 ANNOTATE_POD_IP를 true로 설정하여 Kubernetes 관련 알려진 문제가 발생하지 않도록 하는 것이 좋습니다. 이 기능을 사용하려면 포드에 대한 patch 권한을 aws-node에 추가해야 합니다. aws-node DaemonSet에 패치 권한을 추가하면 플러그인의 보안 범위가 증가합니다. 자세한 내용은 GitHub의 VPC CNI 리포지토리에서 [ANNOTATE_POD_IP](#)를 참조하세요.

Amazon EKS Auto Mode 고려 사항

Amazon EKS Auto Mode는 대체 CNI 플러그인 또는 네트워크 정책 플러그인을 지원하지 않습니다. 자세한 내용은 [EKS Auto Mode](#) 단원을 참조하십시오.

AWS 로드 밸런서 컨트롤러를 통해 인터넷 트래픽 라우팅

AWS 로드 밸런서 컨트롤러는 Kubernetes 클러스터의 AWS Elastic Load Balancer를 관리합니다. 컨트롤러를 사용하여 클러스터 앱을 인터넷에 노출할 수 있습니다. 컨트롤러는 클러스터 Service 또는 Ingress 리소스를 가리키는 AWS 로드 밸런서를 프로비저닝합니다. 다시 말해 컨트롤러는 클러스터의 여러 포드를 가리키는 단일 IP 주소 또는 DNS 이름을 생성합니다.



컨트롤러는 Kubernetes Ingress 또는 Service 리소스를 감시합니다. 이에 대한 응답으로 해당 AWS Elastic Load Balancing 리소스를 생성합니다. Kubernetes 리소스에 주석을 적용하여 로드 밸런서의 특정 동작을 구성할 수 있습니다. 예를 들어, 주석을 사용하여 로드 밸런서에 AWS 보안 그룹을 연결할 수 있습니다.

이 컨트롤러는 다음 리소스를 프로비저닝합니다.

Kubernetes Ingress

LBC는 Kubernetes Ingress 생성 시 [AWS Application Load Balancer\(ALB\)](#)를 생성합니다. [Ingress 리소스에 적용할 수 있는 주석을 검토합니다.](#)

LoadBalancer 유형의 Kubernetes 서비스

LBC는 LoadBalancer 유형의 Kubernetes 서비스 생성 시 [AWS Network Load Balancer\(NLB\)](#)를 생성합니다. [Service 리소스에 적용할 수 있는 주석을 검토합니다.](#)

과거에는 인스턴스 대상에 대해 Kubernetes Network Load Balancer를 사용했지만 IP 대상에 대해서는 LBC를 사용했습니다. AWS Load Balancer Controller 버전 2.3.0 이상에서 대상 유형 중 하나를 사용하여 NLB를 생성할 수 있습니다. NLB 대상 유형에 대한 자세한 내용은 Network Load Balancer 사용 설명서에서 [대상 유형](#)을 참조하세요.

컨트롤러는 GitHub에서 관리되는 [오픈 소스 프로젝트](#)입니다.

컨트롤러를 배포하기 전에 [Application Load Balancer를 사용한 애플리케이션 및 HTTP 트래픽 라우팅](#)과 [the section called “네트워크 로드 밸런싱”](#)의 전제 조건과 고려 사항을 검토하는 것이 좋습니다. 이러한 주제에서는 AWS 로드 밸런서가 포함된 샘플 앱을 배포합니다.

컨트롤러 설치

AWS Load Balancer Controller를 설치하려면 다음 절차 중 하나를 사용합니다.

- Amazon EKS를 처음 사용하는 경우 AWS Load Balancer Controller 설치가 간소화되었으므로 Helm을 사용하여 설치하는 것이 좋습니다. 자세한 내용은 [the section called “Helm을 사용하여 설치”](#) 단원을 참조하십시오.
- 퍼블릭 컨테이너 레지스트리에 대한 네트워크 액세스가 제한된 클러스터와 같은 고급 구성의 경우 Kubernetes 매니페스트를 사용합니다. 자세한 내용은 [the section called “매니페스트를 사용하여 설치”](#) 단원을 참조하십시오.

더 이상 사용되지 않는 컨트롤러 버전에서 마이그레이션

- 사용되지 않는 AWS Load Balancer Controller 버전이 설치된 경우 [the section called “더 이상 사용되지 않는 컨트롤러에서 마이그레이션”](#) 방법을 알아봅니다.
- 사용되지 않는 버전은 업그레이드할 수 없습니다. 이를 제거하고 최신 AWS Load Balancer Controller 버전을 설치해야 합니다.

- 사용되지 않는 버전은 다음과 같습니다.
 - AWS ALB Ingress Controller for Kubernetes("Ingress Controller"), 이전 AWS Load Balancer Controller.
 - 0.1.x 버전의 AWS Load Balancer Controller

레거시 클라우드 공급자

Kubernetes에는 AWS에 대한 레거시 클라우드 제공업체가 포함됩니다. 레거시 클라우드 제공업체는 AWS Load Balancer Controller와 마찬가지로 AWS 로드 밸런서를 프로비저닝할 수 있습니다. 레거시 클라우드 공급자가 Classic Load Balancer를 생성합니다. AWS Load Balancer Controller를 설치하지 않은 경우 Kubernetes에서 기본적으로 레거시 클라우드 제공업체를 사용합니다. AWS Load Balancer Controller를 설치하고 레거시 클라우드 제공업체 사용을 자제해야 합니다.

Important

버전 2.5 이상에서는 AWS Load Balancer Controller가 type: LoadBalancer와 함께 Kubernetes 서비스 리소스의 기본 컨트롤러가 되며 각 서비스에 대한 AWS Network Load Balancer(NLB)를 만듭니다. 이는 서비스에 대한 변형 웹hook을 만들어 이를 수행하며, 이는 type: LoadBalancer의 새 서비스에 대한 spec.loadBalancerClass 필드를 service.k8s.aws/nlb(으)로 설정합니다. 차트 Helm 값을 enableServiceMutatorWebhook에서 false(으)로 설정하여 이 기능을 끄고 [레거시 클라우드 공급자](#)를 기본 컨트롤러로 사용하도록 되돌릴 수 있습니다. 이 기능을 끄지 않는 한 클러스터는 서비스에 새 Classic Load Balancer를 프로비저닝하지 않습니다. 기존 Classic Load Balancer는 계속 작동합니다.

Helm을 사용하여 AWS Load Balancer Controller 설치

Tip

Amazon EKS Auto Mode를 사용하면 네트워킹 추가 기능을 설치하거나 업그레이드할 필요가 없습니다. Auto Mode에는 포트 네트워킹 및 로드 밸런싱 기능이 포함되어 있습니다. 자세한 내용은 [EKS Auto Mode](#) 섹션을 참조하세요.

이 주제에서는 Kubernetes용 패키지 관리자인 Helm과 eksctl을 사용하여 AWS Load Balancer Controller를 설치하는 방법을 설명합니다. 컨트롤러는 기본 옵션으로 설치됩니다. 주석을 사용한 구성

에 관한 세부 정보를 포함한 컨트롤러에 대한 자세한 내용은 GitHub의 [AWS Load Balancer Controller Documentation](#)를 참조하세요.

다음 단계에서 모든 **##** **#**을 고유한 값으로 바꿉니다.

사전 조건

이 자습서를 시작하기 전에 다음 단계를 완료해야 합니다.

- Amazon EKS 클러스터를 생성합니다. 파일을 만들려면 [시작](#) 섹션을 참조하세요.
- 로컬 머신에 [헬름](#)을 설치합니다.
- Kubernetes용 Amazon VPC CNI 플러그인, kube-proxy 및 CoreDNS 추가 기능이 [서비스 계정 토큰](#)에 나열된 최소 버전인지 확인합니다.
- AWS Elastic Load Balancing 개념에 대해 알아보세요. 자세한 내용은 [Elastic Load Balancing User Guide](#)를 참조하세요.
- Kubernetes [서비스](#) 및 [수신](#) 리소스에 대해 알아봅니다.

고려 사항

이 페이지의 구성 단계를 진행하기 전에 다음 사항을 고려하세요.

- IAM 정책과 역할(AmazonEKSLoadBalancerControllerRole)은 동일한 AWS 계정의 여러 EKS 클러스터에서 재사용할 수 있습니다.
- 역할(AmazonEKSLoadBalancerControllerRole)이 원래 생성된 동일한 클러스터에 컨트롤러를 설치하는 경우 역할이 있는지 확인한 후 [2단계: Load Balancer Controller 설치](#)로 이동합니다.
- 서비스 계정에 대한 IAM 역할(IRSA)을 사용하는 경우 각 클러스터에 대해 IRSA를 설정해야 하며 역할의 신뢰 정책에서 OpenID Connect(OIDC) 공급자 ARN은 각 EKS 클러스터에 대해 특정합니다. 또한 기존 AmazonEKSLoadBalancerControllerRole이 있는 새 클러스터에 컨트롤러를 설치하는 경우 새 클러스터의 OIDC 공급자를 포함하도록 역할의 신뢰 정책을 업데이트하고 적절한 역할 주석으로 새 서비스 계정을 생성합니다. OIDC 공급자가 이미 있는지 아니면 생성해야 하는지 확인하려면 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요.

1단계: eksctl을 사용하여 IAM 역할 생성

다음 단계는 AWS 로드 밸런서 컨트롤러 v2.13.0 릴리스 버전을 참조합니다. 모든 릴리스에 대한 자세한 내용은 GitHub의 [AWS Load Balancer Controller Release Page](#)를 참조하세요.

1. 사용자 대신 AWS API를 직접 호출할 수 있는 AWS 로드 밸런서 컨트롤러의 IAM 정책을 다운로드합니다.

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.13.0/docs/install/iam_policy.json
```

- 정부 또는 중국 리전과 같은 비표준 AWS 파티션인 경우 [GitHub의 정책을 검토](#)하고 해당 리전에 적합한 정책을 다운로드합니다.

2. 이전 단계에서 다운로드한 정책을 사용하여 IAM 정책을 만듭니다.

```
aws iam create-policy \
  --policy-name AWSLoadBalancerControllerIAMPolicy \
  --policy-document file://iam_policy.json
```

Note

AWS Management Console에서 정책을 보는 경우 콘솔에 ELB 서비스에 대한 경고는 표시되지만 ELB v2 서비스에 대한 경고는 표시되지 않습니다. 이는 정책의 작업 중 일부가 ELB v2에는 있지만 ELB에는 없기 때문에 발생합니다. ELB에 대한 경고는 무시해도 됩니다.

3. 클러스터 이름, 리전 코드 및 계정 ID의 값을 바꿉니다.

```
eksctl create iamserviceaccount \
  --cluster=<cluster-name> \
  --namespace=kube-system \
  --name=aws-load-balancer-controller \
  --attach-policy-arn=arn:aws:iam::<AWS_ACCOUNT_ID>:policy/AWSLoadBalancerControllerIAMPolicy \
  --override-existing-serviceaccounts \
  --region <aws-region-code> \
  --approve
```

2단계: AWS Load Balancer Controller 설치

1. eks-charts 차트 Helm 리포지토리를 추가합니다. AWS에서는 [이 리포지토리](#)를 GitHub에 유지합니다.

```
helm repo add eks https://aws.github.io/eks-charts
```

2. 최신 차트가 적용되도록 로컬 리포지토리를 업데이트합니다.

```
helm repo update eks
```

3. AWS 로드 밸런서 컨트롤러를 설치합니다.

[Amazon EC2 인스턴스 메타데이터 서비스\(IMDS\)에 대해 제한적인 액세스 권한](#)이 있는 Amazon EC2 노드에 컨트롤러를 배포하거나 Fargate 또는 Amazon EKS Hybrid Nodes에 배포하는 경우, 다음 helm 명령에 다음 플래그를 추가합니다.

- `--set region=region-code`
- `--set vpcId=vpc-xxxxxxxx`

*my-cluster*를 해당 클러스터의 이름으로 바꿉니다. 다음 명령에서 `aws-load-balancer-controller`는 이전 단계에서 생성한 Kubernetes 서비스 계정입니다.

차트 Helm 구성에 관한 자세한 내용은 GitHub에서 [values.yaml](#)을 참조하세요.

```
helm install aws-load-balancer-controller eks/aws-load-balancer-controller \
  -n kube-system \
  --set clusterName=my-cluster \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller \
  --version 1.13.0
```

Important

배포된 차트는 보안 업데이트를 자동으로 수신하지 않습니다. 새 차트가 사용 가능해지면 수동으로 업그레이드해야 합니다. 업그레이드할 때는 이전 명령에서 `upgrade`을 `##`로 변경합니다.

`helm install` 명령은 컨트롤러의 사용자 지정 리소스 정의(CRD)를 자동으로 설치합니다. `helm upgrade` 명령은 설치하지 않습니다. `helm upgrade`, 를 사용하는 경우 CRD를 수동으로 설치해야 합니다. 다음 명령을 실행하여 CRD를 설치합니다.

```
wget https://raw.githubusercontent.com/aws/eks-charts/master/stable/aws-load-balancer-controller/crds/crds.yaml
kubectl apply -f crds.yaml
```


3단계: 컨트롤러가 설치되어 있는지 확인

1. 컨트롤러가 설치되어 있는지 확인합니다.

```
kubectl get deployment -n kube-system aws-load-balancer-controller
```

예제 출력은 다음과 같습니다.

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
aws-load-balancer-controller	2/2	2	2	84s

Helm을 사용하여 배포한 경우 이전 출력이 표시됩니다. Kubernetes 매니페스트를 사용하여 배포한 경우 복제본이 하나만 있습니다.

2. 컨트롤러를 사용하여 AWS 리소스를 프로비저닝하기 전에 클러스터가 특정 요구 사항을 충족해야 합니다. 자세한 내용은 [the section called “애플리케이션 로드 밸런싱”](#) 및 [the section called “네트워크 로드 밸런싱”](#) 섹션을 참조하세요.

매니페스트를 사용하여 AWS Load Balancer Controller 설치

Tip

Amazon EKS Auto Mode를 사용하면 네트워킹 추가 기능을 설치하거나 업그레이드할 필요가 없습니다. Auto Mode에는 포드 네트워킹 및 로드 밸런싱 기능이 포함되어 있습니다. 자세한 내용은 [EKS Auto Mode](#) 섹션을 참조하세요.

이 주제에서는 Kubernetes 매니페스트를 다운로드하고 적용하여 컨트롤러를 설치하는 방법을 설명합니다. GitHub에서 컨트롤러에 대한 전체 [설명서](#)를 볼 수 있습니다.

다음 단계에서 모든 **##** **#**을 고유한 값으로 바꿉니다.

사전 조건

이 자습서를 시작하기 전에 다음 단계를 완료해야 합니다.

- Amazon EKS 클러스터를 생성합니다. 파일을 만들려면 [시작](#) 섹션을 참조하세요.
- 로컬 머신에 [헬름](#)을 설치합니다.

- Kubernetes용 Amazon VPC CNI 플러그인, kube-proxy 및 CoreDNS 추가 기능이 [서비스 계정 토큰](#)에 나열된 최소 버전인지 확인합니다.
- AWS Elastic Load Balancing 개념에 대해 알아보세요. 자세한 내용은 [Elastic Load Balancing User Guide](#)를 참조하세요.
- Kubernetes [서비스](#) 및 [수신](#) 리소스에 대해 알아봅니다.

고려 사항

이 페이지의 구성 단계를 진행하기 전에 다음 사항을 고려하세요.

- IAM 정책과 역할(AmazonEKSLoadBalancerControllerRole)은 동일한 AWS 계정의 여러 EKS 클러스터에서 재사용할 수 있습니다.
- 역할(AmazonEKSLoadBalancerControllerRole)이 원래 생성된 동일한 클러스터에 컨트롤러를 설치하는 경우 역할이 있는지 확인한 후 [2단계: cert-manager 설치](#)로 이동합니다.
- 서비스 계정에 대한 IAM 역할(IRSA)을 사용하는 경우 각 클러스터에 대해 IRSA를 설정해야 하며 역할의 신뢰 정책에서 OpenID Connect(OIDC) 공급자 ARN은 각 EKS 클러스터에 대해 특정합니다. 또한 기존 AmazonEKSLoadBalancerControllerRole이 있는 새 클러스터에 컨트롤러를 설치하는 경우 새 클러스터의 OIDC 공급자를 포함하도록 역할의 신뢰 정책을 업데이트하고 적절한 역할 주석으로 새 서비스 계정을 생성합니다. OIDC 공급자가 이미 있는지 아니면 생성해야 하는지 확인하려면 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요.

1단계: IAM 구성

다음 단계는 AWS Load Balancer Controller v2.11.0 릴리스 버전을 참조하세요. 모든 릴리스에 대한 자세한 내용은 GitHub의 [AWS Load Balancer Controller Release Page](#)를 참조하세요.

1. 사용자 대신 AWS API를 직접 호출할 수 있는 AWS 로드 밸런서 컨트롤러의 IAM 정책을 다운로드합니다.

Example

AWS

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/install/iam_policy.json
```

AWS GovCloud (US)

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/install/iam_policy_us-gov.json
```

```
mv iam_policy_us-gov.json iam_policy.json
```

2. 이전 단계에서 다운로드한 정책을 사용하여 IAM 정책을 만듭니다.

```
aws iam create-policy \
  --policy-name AWSLoadBalancerControllerIAMPolicy \
  --policy-document file://iam_policy.json
```

Note

AWS Management Console에서 정책을 보는 경우 콘솔에 ELB 서비스에 대한 경고는 표시되지만 ELB v2 서비스에 대한 경고는 표시되지 않습니다. 이는 정책의 작업 중 일부가 ELB v2에는 있지만 ELB에는 없기 때문에 발생합니다. ELB에 대한 경고는 무시해도 됩니다.

Example

eksctl

- my-cluster*를 사용자 클러스터 이름으로 바꾸고 **111122223333**을 계정 ID로 바꾼 다음 명령을 실행합니다.

```
eksctl create iamserviceaccount \
  --cluster=my-cluster \
  --namespace=kube-system \
  --name=aws-load-balancer-controller \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --attach-policy-arn=arn:aws:iam::111122223333:policy/AWSLoadBalancerControllerIAMPolicy \
  --approve
```

AWS CLI and kubectl

- a. 클러스터의 OIDC 제공자 ID를 검색하고 변수에 저장합니다.

```
oidc_id=$(aws eks describe-cluster --name my-cluster --query
"cluster.identity.oidc.issuer" --output text | cut -d '/' -f 5)
```

- b. 클러스터의 ID를 가진 IAM OIDC 제공자가 이미 계정에 있는지 확인합니다. 클러스터와 IAM 모두에 OIDC를 구성해야 합니다.

```
aws iam list-open-id-connect-providers | grep $oidc_id | cut -d "/" -f4
```

출력이 반환되면 해당 클러스터에 대한 IAM OIDC 제공자가 이미 있는 것입니다. 출력이 반환되지 않은 경우 해당 클러스터에 대한 IAM OIDC 공급자를 생성해야 합니다. 자세한 내용은 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요.

- c. 다음 콘텐츠를 디바이스에 복사합니다. **111122223333**을 계정 ID로 바꿉니다. **region-code**를 클러스터가 있는 AWS 리전으로 바꿉니다. **EXAMPLED539D4633E53DE1B71EXAMPLE**을 이전 단계에서 반환된 출력으로 바꿉니다.

```
cat >load-balancer-role-trust-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.region-code.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
          "oidc.eks.region-code.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:kube-system:aws-
load-balancer-controller"
        }
      }
    }
  ]
}
```

```
}
EOF
```

- d. IAM 역할을 생성합니다.

```
aws iam create-role \
  --role-name AmazonEKSLoadBalancerControllerRole \
  --assume-role-policy-document file://load-balancer-role-trust-policy.json
```

- e. 필요한 Amazon EKS 관리형 IAM 정책을 IAM 역할에 연결합니다. **111122223333**을 계정 ID로 바꿉니다.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::111122223333:policy/AWSLoadBalancerControllerIAMPolicy \
  --role-name AmazonEKSLoadBalancerControllerRole
```

- f. 다음 콘텐츠를 디바이스에 복사합니다. **111122223333**을 계정 ID로 바꿉니다. 텍스트를 바꾼 후 수정된 명령을 실행하여 `aws-load-balancer-controller-service-account.yaml` 파일을 생성합니다.

```
cat >aws-load-balancer-controller-service-account.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/name: aws-load-balancer-controller
  name: aws-load-balancer-controller
  namespace: kube-system
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/
    AmazonEKSLoadBalancerControllerRole
EOF
```

- g. 클러스터에 Kubernetes 서비스 계정을 생성합니다. `aws-load-balancer-controller`라는 이름의 Kubernetes 서비스 계정은 **AmazonEKSLoadBalancerControllerRole** 이름으로 생성된 IAM 역할을 사용하여 주석을 담니다.

```
kubectl apply -f aws-load-balancer-controller-service-account.yaml
```

2단계: **cert-manager** 설치

다음 방법 중 하나를 사용하여 cert-manager를 설치한 다음 인증서 구성을 Webhook에 주입합니다. 자세한 내용은 cert-manager 문서의 [시작하기](#)를 참조하세요.

quay.io 컨테이너 레지스트리를 사용하여 cert-manager를 설치하는 것이 좋습니다. 노드가 quay.io 컨테이너 레지스트리에 액세스할 수 없는 경우, Amazon ECR을 사용하여 cert-manager를 설치하세요(아래 참조).

Example

Quay.io

- a. 노드에 quay.io 컨테이너 레지스트리에 대한 액세스 권한이 있는 경우, cert-manager를 설치한 다음 Webhook에 인증서 구성을 주입합니다.

```
kubectl apply \
  --validate=false \
  -f https://github.com/jetstack/cert-manager/releases/download/v1.13.5/cert-manager.yaml
```

Amazon ECR

- a. 다음 방법 중 하나를 사용하여 cert-manager를 설치한 다음 인증서 구성을 Webhook에 주입합니다. 자세한 내용은 cert-manager 문서의 [시작하기](#)를 참조하세요.
- b. 매니페스트를 다운로드합니다.

```
curl -Lo cert-manager.yaml https://github.com/jetstack/cert-manager/releases/download/v1.13.5/cert-manager.yaml
```

- c. 다음 이미지를 가져와서 노드가 액세스할 수 있는 리포지토리로 푸시합니다. 이미지를 가져오기, 태그 지정 및 자체 리포지토리로 푸시하는 방법에 대한 자세한 내용은 [the section called “리포지토리에 이미지 복사”](#) 단원을 참조하세요.

```
quay.io/jetstack/cert-manager-cainjector:v1.13.5
quay.io/jetstack/cert-manager-controller:v1.13.5
quay.io/jetstack/cert-manager-webhook:v1.13.5
```

- d. 자체 레지스트리 이름으로 세 개의 이미지에 대한 매니페스트 `quay.io`를 바꿉니다. 다음 명령은 개인 리포지토리의 이름이 원본 리포지토리와 같다고 가정합니다. `111122223333.dkr.ecr.region-code.amazonaws.com`을 개인 레지스트리로 바꿉니다.

```
sed -i.bak -e 's|quay.io|111122223333.dkr.ecr.region-code.amazonaws.com|' ./cert-manager.yaml
```

- e. 매니페스트를 적용합니다.

```
kubectl apply \
  --validate=false \
  -f ./cert-manager.yaml
```

3단계: AWS Load Balancer Controller 설치

1. 컨트롤러 사양을 다운로드합니다. 컨트롤러에 대한 자세한 내용은 GitHub에서 [문서](#)를 참조하세요.

```
curl -Lo v2_11_0_full.yaml https://github.com/kubernetes-sigs/aws-load-balancer-controller/releases/download/v2.11.0/v2_11_0_full.yaml
```

2. 파일에 대해 편집한 항목:

- a. `v2_11_0_full.yaml` 파일을 다운로드 한 경우 다음 명령을 실행하여 매니페스트에서 `ServiceAccount` 섹션을 제거합니다. 이 섹션을 제거하지 않으면 이전 단계에서 서비스 계정에 작성한 필수 주석이 덮어쓰워집니다. 이 섹션을 제거하면 컨트롤러를 삭제할 경우 이전 단계에서 생성한 서비스 계정도 유지됩니다.

```
sed -i.bak -e '690,698d' ./v2_11_0_full.yaml
```

다른 파일 버전을 다운로드한 경우 편집기에서 파일을 열고 다음 줄을 제거합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/component: controller
    app.kubernetes.io/name: aws-load-balancer-controller
  name: aws-load-balancer-controller
  namespace: kube-system
---
```

- b. *my-cluster*를 해당 클러스터 이름으로 바꿔 파일의 Deployment spec 섹션에 있는 *your-cluster-name*을 해당 클러스터의 이름으로 바꿉니다.

```
sed -i.bak -e 's|your-cluster-name|my-cluster|' ./v2_11_0_full.yaml
```

- c. 노드가 Amazon EKS Amazon ECR 이미지 리포지토리에 액세스할 수 없는 경우 다음 이미지를 가져와서 노드가 액세스할 수 있는 리포지토리로 푸시해야 합니다. 이미지를 가져오기, 태깅 및 자체 리포지토리로 푸시하는 방법에 대한 자세한 내용은 [the section called “리포지토리에 이미지 복사”](#) 섹션을 참조하세요.

```
public.ecr.aws/eks/aws-load-balancer-controller:v2.11.0
```

매니페스트에 레지스트리 이름을 추가합니다. 다음 명령은 개인 리포지토리의 이름이 원본 리포지토리와 같다고 가정하고 개인 레지스트리의 이름을 파일에 추가합니다.

*111122223333.dkr.ecr.region-code.amazonaws.com*을 해당 레지스트리로 바꿉니다. 이 라인에서는 개인 리포지토리의 이름을 원본 리포지토리와 동일하게 지정했다고 가정합니다. 그렇지 않은 경우 개인 레지스트리 이름 뒤에 있는 *eks/aws-load-balancer-controller* 텍스트를 해당 리포지토리 이름으로 바꿉니다.

```
sed -i.bak -e 's|public.ecr.aws/eks/aws-load-balancer-controller|111122223333.dkr.ecr.region-code.amazonaws.com/eks/aws-load-balancer-controller|' ./v2_11_0_full.yaml
```

- d. (Fargate 또는 제한된 IMDS에만 필요)

[Amazon EC2 인스턴스 메타데이터 서비스\(IMDS\)에 대해 제한적인 액세스 권한](#)이 있는 Amazon EC2 노드에 컨트롤러를 배포하거나 Fargate 또는 Amazon EKS Hybrid Nodes에 배포하는 경우, `- args:`에서 `following parameters`를 추가합니다.

```
[...]
spec:
  containers:
    - args:
      - --cluster-name=your-cluster-name
      - --ingress-class=alb
      - --aws-vpc-id=vpc-xxxxxxx
      - --aws-region=region-code
```


[...]

3. 파일을 적용합니다.

```
kubectl apply -f v2_11_0_full.yaml
```

4. IngressClass 및 IngressClassParams 매니페스트를 클러스터에 다운로드합니다.

```
curl -Lo v2_11_0_ingclass.yaml https://github.com/kubernetes-sigs/aws-load-balancer-controller/releases/download/v2.11.0/v2_11_0_ingclass.yaml
```

5. 매니페스트를 클러스터에 적용합니다.

```
kubectl apply -f v2_11_0_ingclass.yaml
```

4단계: 컨트롤러가 설치되어 있는지 확인

1. 컨트롤러가 설치되어 있는지 확인합니다.

```
kubectl get deployment -n kube-system aws-load-balancer-controller
```

예제 출력은 다음과 같습니다.

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
aws-load-balancer-controller	2/2	2	2	84s

Helm을 사용하여 배포한 경우 이전 출력이 표시됩니다. Kubernetes 매니페스트를 사용하여 배포한 경우 복제본이 하나만 있습니다.

2. 컨트롤러를 사용하여 AWS 리소스를 프로비저닝하기 전에 클러스터가 특정 요구 사항을 충족해야 합니다. 자세한 내용은 [the section called “애플리케이션 로드 밸런싱”](#) 및 [the section called “네트워크 로드 밸런싱”](#) 섹션을 참조하세요.

더 이상 사용되지 않는 ALB 수신 컨트롤러에서 앱 마이그레이션

이 주제에서는 사용되지 않는 컨트롤러 버전에서 마이그레이션하는 방법을 설명합니다. 보다 구체적으로 AWS Load Balancer Controller의 사용되지 않는 버전을 제거하는 방법을 설명합니다.

- 사용되지 않는 버전은 업그레이드할 수 없습니다. 먼저 제거한 다음, 최신 버전을 설치해야 합니다.

- 사용되지 않는 버전은 다음과 같습니다.
 - AWS ALB Ingress Controller for Kubernetes("Ingress Controller"), 이전 AWS Load Balancer Controller.
 - 0.1.x 버전의 AWS Load Balancer Controller

더 이상 사용되지 않는 컨트롤러 버전 제거

Note

Helm을 사용하거나 Kubernetes 매니페스트를 사용하여 수동으로 사용되지 않는 버전을 설치했을 수 있습니다. 원래 설치한 도구를 사용하여 절차를 완료합니다.

1. incubator/aws-alb-ingress-controller 차트 Helm이 설치된 경우 제거합니다.

```
helm delete aws-alb-ingress-controller -n kube-system
```

2. eks-charts/aws-load-balancer-controller 차트의 버전 0.1.x 이 설치된 경우 제거합니다. 버전 0.1.x 에서 1.0.0으로의 업그레이드는 웹훅 API 버전과 호환되지 않아 작동되지 않습니다.

```
helm delete aws-load-balancer-controller -n kube-system
```

3. 이 컨트롤러가 현재 설치되어 있는지 확인합니다.

```
kubectl get deployment -n kube-system alb-ingress-controller
```

컨트롤러가 설치되어 있지 않은 경우의 출력입니다.

```
Error from server (NotFound): deployments.apps "alb-ingress-controller" not found
```

컨트롤러가 설치된 경우의 출력입니다.

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
alb-ingress-controller	1/1	1	1	122d

4. 다음 명령을 입력하여 컨트롤러를 제거합니다.

```
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-controller/v1.1.8/docs/examples/alb-ingress-controller.yaml
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-alb-ingress-controller/v1.1.8/docs/examples/rbac-role.yaml
```

AWS Load Balancer Controller로 마이그레이션

ALB Ingress Controller for Kubernetes에서 AWS Load Balancer Controller로 마이그레이션하려면 다음을 수행해야 합니다.

1. ALB Ingress Controller를 제거합니다(위 내용 참조).
2. [AWS 로드 밸런서 컨트롤러를 설치합니다.](#)
3. AWS Load Balancer Controller에서 사용하는 IAM 역할에 정책을 더 추가합니다. 이 정책은 LBC가 ALB Ingress Controller for Kubernetes에서 생성한 리소스를 관리할 수 있도록 허용합니다.
4. IAM 정책을 다운로드합니다. 이 정책을 통해 AWS Load Balancer Controller가 ALB Ingress Controller for Kubernetes에서 생성한 리소스를 관리할 수 있습니다. [정책을 볼](#) 수도 있습니다.

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/install/iam_policy_v1_to_v2_additional.json
```

5. 클러스터가 AWS GovCloud(미국 동부) 또는 AWS GovCloud(미국 서부) AWS 리전에 있는 경우, `arn:aws:` 를 `arn:aws-us-gov:`로 대체합니다.

```
sed -i.bak -e 's|arn:aws:|arn:aws-us-gov:|' iam_policy_v1_to_v2_additional.json
```

6. IAM 정책을 생성하고 반환된 ARN 적어둡니다.

```
aws iam create-policy \
  --policy-name AWSLoadBalancerControllerAdditionalIAMPolicy \
  --policy-document file://iam_policy_v1_to_v2_additional.json
```

7. IAM 정책을 AWS Load Balancer Controller에서 사용하는 IAM 역할에 연결합니다. *your-role-name*을 역할의 이름(예: AmazonEKSLoadBalancerControllerRole)으로 바꿉니다.

eksctl을 사용하여 역할을 생성한 경우 생성된 역할 이름을 찾으려면 [AWS CloudFormation 콘솔](#)을 열고 eksctl-*my-cluster*-addon-iamserviceaccount-kube-system-aws-load-balancer-controller 스택을 선택합니다. 리소스 탭을 선택합니다. 물리적 ID 열에 역할 이름이 있습니다.

```
aws iam attach-role-policy \
  --role-name your-role-name \
  --policy-arn arn:aws:iam::111122223333:policy/
AWSLoadBalancerControllerAdditionalIAMPolicy
```

Amazon EKS 클러스터에서 DNS에 대한 CoreDNS 관리

Tip

Amazon EKS Auto Mode를 사용하면 네트워킹 추가 기능을 설치하거나 업그레이드할 필요가 없습니다. Auto Mode에는 포드 네트워킹 및 로드 밸런싱 기능이 포함되어 있습니다. 자세한 내용은 [EKS Auto Mode](#) 섹션을 참조하세요.

CoreDNS는 Kubernetes 클러스터 DNS로 사용할 수 있는 유연하고 확장 가능한 DNS 서버입니다. 하나 이상의 노드가 있는 Amazon EKS 클러스터를 시작하면 클러스터에 배포된 노드 수에 관계없이 CoreDNS 이미지의 복제본 2개가 기본적으로 배포됩니다. CoreDNS 포드는 클러스터의 모든 포드에 대한 이름 확인을 제공합니다. 클러스터에 CoreDNS deployment의 네임스페이스와 일치하는 네임스페이스의 Fargate 프로필이 포함되어 있는 경우 CoreDNS 포드를 Fargate 노드에 배포할 수 있습니다. Fargate 프로파일에 대한 자세한 내용은 [the section called “프로필 정의”](#) 섹션을 참조하세요. CoreDNS에 대한 자세한 내용은 Kubernetes 설명서에서 [서비스 검색에 CoreDNS 사용](#) 섹션을 참조하세요.

CoreDNS 버전

다음 표에는 각 Kubernetes 버전에 사용할 수 있는 Amazon EKS 추가 기능 유형의 최신 버전이 나열되어 있습니다.

Kubernetes 버전	CoreDNS 버전
1.33	v1.12.1-eksbuild.2
1.32	v1.11.4-eksbuild.14
1.31	v1.11.4-eksbuild.14
1.30	v1.11.4-eksbuild.14

Kubernetes 버전	CoreDNS 버전
1.29	v1.11.4-eksbuild.14
1.28	v1.10.1-eksbuild.18
1.27	v1.10.1-eksbuild.18
1.26	v1.9.3-eksbuild.22

Important

이 추가 기능을 자체 관리하는 경우 표의 버전이 사용 가능한 자체 관리 버전과 다를 수 있습니다. 이 추가 기능의 자체 관리형 유형 업데이트에 대한 자세한 내용은 [the section called “업데이트\(자체 관리형\)”](#) 부분을 참조하세요.

중요 CoreDNS 업그레이드 고려 사항

- CoreDNS 배포의 안정성과 가용성을 개선하기 위해 버전 v1.9.3-eksbuild.6 이상과 v1.10.1-eksbuild.30이 PodDisruptionBudget과 함께 배포됩니다. 기존 PodDisruptionBudget 버전을 배포한 경우 이러한 버전으로의 업그레이드가 실패할 수 있습니다. 업그레이드가 실패할 경우 다음 작업 중 하나를 수행하면 문제가 해결됩니다.
- Amazon EKS 추가 기능을 업그레이드할 때 충돌 해결 옵션으로 기존 설정을 재정의하도록 선택합니다. 배포에 다른 사용자 지정 설정을 지정한 경우 업그레이드 후 다른 사용자 지정 설정을 다시 적용할 수 있도록 업그레이드하기 전에 설정을 백업해야 합니다.
- 기존 PodDisruptionBudget을 제거하고 업그레이드를 다시 시도하세요.
- EKS 추가 기능 버전 v1.9.3-eksbuild.3 이상 및 v1.10.1-eksbuild.6 이상에서는 CoreDNS 배포는 /ready 엔드포인트를 사용하도록 readinessProbe를 설정합니다. 이 엔드포인트는 CoreDNS의 Corefile 구성 파일에서 활성화됩니다.

사용자 지정 Corefile을 사용하는 경우 프로브가 사용할 /ready 엔드포인트가 에서 활성화되도록 구성에 ready 플러그인을 추가해야 합니다.

- EKS 추가 기능 버전 v1.9.3-eksbuild.7 이상 및 v1.10.1-eksbuild.4 이상에서는 PodDisruptionBudget을 변경할 수 있습니다. 다음 예제의 필드를 사용하여 선택적

구성 설정에서 추가 기능을 편집하고 이러한 설정을 변경할 수 있습니다. 이 예제는 기본 PodDisruptionBudget을 보여줍니다.

```
{
  "podDisruptionBudget": {
    "enabled": true,
    "maxUnavailable": 1
  }
}
```

maxUnavailable 또는 minAvailable을 설정할 수 있지만 단일 PodDisruptionBudget에서 둘 다 설정할 수는 없습니다. PodDisruptionBudgets에 대한 자세한 내용은 Kubernetes 설명서에서 [PodDisruptionBudget 지정](#)을 참조하세요.

enabled을 false으로 설정한 경우 PodDisruptionBudget은 제거되지 않는다는 점에 유의하세요. 이 필드를 false(으)로 설정한 후에는 PodDisruptionBudget 객체를 삭제해야 합니다. 마찬가지로, PodDisruptionBudget이 있는 버전으로 업그레이드한 후 이전 버전의 추가 기능을 사용하도록 추가 기능을 편집(추가 기능 다운그레이드)해도 PodDisruptionBudget은 제거되지 않습니다. PodDisruptionBudget을 삭제하려면 다음 명령을 실행할 수 있습니다.

```
kubectl delete poddisruptionbudget coredns -n kube-system
```

- EKS 추가 기능 버전 v1.10.1-eksbuild.5 이상에서는 KEP 2067을 준수하도록 기본 허용 범위를 node-role.kubernetes.io/master:NoSchedule에서 node-role.kubernetes.io/control-plane:NoSchedule로 변경합니다. KEP 2067에 대한 자세한 내용은 GitHub의 Kubernetes Enhancement Proposals (KEPs)에서 [KEP-2067: Rename the kubeadm "master" label and taint](#)를 참조하세요.

EKS 추가 기능 버전 v1.8.7-eksbuild.8 이상 및 v1.9.3-eksbuild.9 이상에서 허용 범위는 모든 Kubernetes 버전과 호환되도록 설정됩니다.

- EKS 추가 기능 버전 v1.9.3-eksbuild.11 및 v1.10.1-eksbuild.7 이상에서는 CoreDNS 배포가 topologySpreadConstraints에 대한 기본값을 설정합니다. 기본값은 사용 가능한 여러 가용 영역에 노드가 있는 경우 CoreDNS 포드가 가용 영역에 분산되도록 합니다. 기본값 대신 사용될 사용자 지정 값을 설정할 수 있습니다. 기본값은 다음과 같습니다.

```
topologySpreadConstraints:
- maxSkew: 1
  topologyKey: topology.kubernetes.io/zone
  whenUnsatisfiable: ScheduleAnyway
```

```
labelSelector:
  matchLabels:
    k8s-app: kube-dns
```

CoreDNS v1.11 업그레이드 고려 사항

- EKS 추가 기능 버전 v1.11.1-eksbuild.4 이상에서 컨테이너 이미지는 최소 패키지를 포함하고 셸이 없는 Amazon EKS Distro에서 유지 관리하는 [최소 기본 이미지](#)를 기반으로 합니다. 자세한 내용은 [Amazon EKS Distro](#)를 참조하세요. CoreDNS 이미지의 사용법과 문제 해결은 동일하게 유지됩니다.

CoreDNS Amazon EKS 추가 기능 생성

CoreDNS Amazon EKS 추가 기능을 생성합니다. 추가 기능을 생성하기 전에 클러스터가 있어야 합니다. 자세한 내용은 [the section called “클러스터 생성”](#) 섹션을 참조하세요.

1. 클러스터에 설치된 추가 기능의 버전을 확인하세요.

```
kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -d : -f 3
```

예제 출력은 다음과 같습니다.

```
v1.10.1-eksbuild.13
```

2. 클러스터에 설치된 추가 기능의 유형을 확인하세요. 클러스터를 생성하는 데 사용한 도구에 따라 현재 클러스터에 Amazon EKS 추가 기능이 유형이 설치되어 있지 않을 수 있습니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
addon.addonVersion --output text
```

버전 번호가 반환되는 경우 Amazon EKS 유형의 추가 기능이 클러스터에 설치되고 본 절차의 나머지 단계를 완료할 필요가 없습니다. 오류가 번호가 반환되는 경우 Amazon EKS 유형의 추가 기능이 클러스터에 설치되지 않습니다. 이 절차의 나머지 단계를 완료하여 설치하세요.

3. 현재 설치된 추가 기능의 구성을 저장합니다.

```
kubectl get deployment coredns -n kube-system -o yaml > aws-k8s-coredns-old.yaml
```

4. AWS CLI를 사용하여 추가 기능을 생성합니다. AWS Management Console 또는 `eksctl`를 사용하여 추가 기능을 만들려면 [the section called “추가 기능 생성”](#)을 참조하여 추가 기능 이름을 `coredns`로 지정하세요. 다음 명령을 디바이스에 복사합니다. 필요에 따라 명령을 다음과 같이 수정한 다음에 수정한 명령을 실행합니다.

- `my-cluster`를 해당 클러스터의 이름으로 바꿉니다.
- `v1.11.3-eksbuild.1`을 클러스터 버전의 [최신 버전 테이블](#)에 나와 있는 최신 버전으로 바꿉니다.

```
aws eks create-addon --cluster-name my-cluster --addon-name coredns --addon-version v1.11.3-eksbuild.1
```

Amazon EKS 추가 기능의 기본 설정과 충돌하는 사용자 지정 설정을 현재 추가 기능에 적용한 경우 생성이 실패할 수 있습니다. 생성에 실패하면 문제 해결에 도움이 될 수 있는 오류를 받게 됩니다. 또는 이전 명령에 `--resolve-conflicts OVERWRITE`을(를) 추가할 수 있습니다. 이렇게 하면 추가 기능이 기존 사용자 지정 설정을 덮어쓸 수 있습니다. 추가 기능을 만든 후에는 사용자 지정 설정으로 업데이트할 수 있습니다.

5. 클러스터의 Kubernetes 버전에 맞는 추가 기능의 최신 버전이 클러스터에 추가되었는지 확인합니다. `my-cluster`를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query addon.addonVersion --output text
```

추가 기능 생성이 완료되는 데 몇 초 정도 걸릴 수 있습니다.

예제 출력은 다음과 같습니다.

```
v1.11.3-eksbuild.1
```

6. 원래 추가 기능을 사용자 지정 설정했다면 Amazon EKS 추가 기능을 생성하기 전에 이전 단계에서 저장한 구성을 사용하여 Amazon EKS 추가 기능을 사용자 지정 설정으로 업데이트하세요. 애드온을 업데이트하는 방법은 [the section called “업데이트\(EKS 추가 기능\)”](#)을 참조하세요.

CoreDNS Amazon EKS 추가 기능 업데이트

추가 기능의 Amazon EKS 유형 업데이트. 클러스터에 Amazon EKS 추가 기능을 추가하지 않은 경우 [추가](#)하거나 [the section called “업데이트\(자체 관리형\)”](#) 섹션을 참조하세요.

시작하기 전에 업그레이드 고려 사항을 검토하세요. 자세한 내용은 [the section called “중요 CoreDNS 업그레이드 고려 사항”](#) 섹션을 참조하세요.

1. 클러스터에 설치된 추가 기능의 버전을 확인하세요. `<cluster-name>`을 클러스터 이름으로 바꿉니다.

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
"addon.addonVersion" --output text
```

예제 출력은 다음과 같습니다.

```
v1.10.1-eksbuild.13
```

반환된 버전이 [최신 버전 표](#)에 있는 클러스터의 Kubernetes 버전과 동일한 경우 클러스터에 이미 최신 버전이 설치되어 있으므로 이 절차의 나머지 부분을 완료하지 않아도 됩니다. 출력에 버전 번호 대신 오류가 표시되면 클러스터에 설치된 추가 기능의 Amazon EKS 유형이 없는 것입니다. 이 절차로 업데이트하려면 먼저 [추가 기능을 생성](#)해야 합니다.

2. 현재 설치된 추가 기능의 구성을 저장합니다.

```
kubectl get deployment coredns -n kube-system -o yaml > aws-k8s-coredns-old.yaml
```

3. AWS CLI를 사용하여 추가 기능 업데이트. AWS Management Console 또는 `eksctl`를 사용하여 추가 기능을 업데이트하려면 [the section called “추가 기능 업데이트”](#) 부분을 참조하세요. 다음 명령을 디바이스에 복사합니다. 필요에 따라 명령을 다음과 같이 수정한 다음에 수정한 명령을 실행합니다.

- `my-cluster`를 해당 클러스터의 이름으로 바꿉니다.
- `v1.11.3-eksbuild.1`을 클러스터 버전의 [최신 버전 테이블](#)에 나와 있는 최신 버전으로 바꿉니다.
- `--resolve-conflicts`**PRESERVE** 옵션에서는 추가 기능의 기존 구성 값을 보존합니다. 추가 기능 설정에 사용자 지정 값을 설정하고 이 옵션을 사용하지 않는 경우 Amazon EKS에서는 기본 값으로 해당 값을 덮어씁니다. 이 옵션을 사용하는 경우에는 추가 기능을 업데이트하기 전에 프로덕션 클러스터에서 비프로덕션 클러스터의 필드 및 값 변경 사항을 테스트하는 것이 좋습니다.

이 값을 OVERWRITE로 변경하면 모든 설정이 Amazon EKS 기본값으로 변경됩니다. 설정에 사용자 지정 값을 설정한 경우 Amazon EKS 기본값으로 해당 값을 덮어쓸 수도 있습니다. 이 값을 none으로 변경하면 Amazon EKS에서는 설정의 값을 변경하지 않지만 업데이트에 실패할 수도 있습니다. 업데이트에 실패하면 충돌 해결에 도움이 되는 오류 메시지가 표시됩니다.

- 구성 설정을 업데이트하지 않는 경우, `--configuration-values`
'{"*replicaCount*":3}' 을(를) 명령에서 제거하세요. 구성 설정을 업데이트하는 경우, "*ReplicaCount*":3# 설정하려는 설정으로 바꾸세요. 이 예제에서는 CoreDNS의 복제본 수가 3으로 설정 되어 있습니다. 지정한 값은 구성 스키마에 유효해야 합니다. 구성 스키마를 모르는 경우 `aws eks describe-addon-configuration --addon-name coredns --addon-version v1.11.3-eksbuild.1` 을 실행하여 *v1.11.3-eksbuild.1*을 구성을 확인하려는 추가 기능의 버전 번호로 변경하세요. 스키마가 출력에 반환됩니다. 기존 사용자 지정 구성이 있는데 이를 모두 제거하고 모든 설정의 값을 Amazon EKS 기본값으로 다시 설정하려면 명령에서 "*replicaCount*":3를 제거하여 {}을(를) 비워두세요. CoreDNS 설정에 대한 자세한 내용은 쿠버네티스 문서의 [DNS 서비스 사용자 정의하기](#)를 참조하세요.

```
aws eks update-addon --cluster-name my-cluster --addon-name coredns --addon-version v1.11.3-eksbuild.1 \
  --resolve-conflicts PRESERVE --configuration-values '{"replicaCount":3}'
```

업데이트가 완료되는 데 몇 초 정도 걸릴 수 있습니다.

4. 추가 기능 버전이 업데이트되었는지 확인합니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns
```

업데이트가 완료되는 데 몇 초 정도 걸릴 수 있습니다.

예제 출력은 다음과 같습니다.

```
{
  "addon": {
    "addonName": "coredns",
    "clusterName": "my-cluster",
    "status": "ACTIVE",
    "addonVersion": "v1.11.3-eksbuild.1",
    "health": {
      "issues": []
    },
  },
}
```

```

    "addonArn": "arn:aws:eks:region:111122223333:addon/my-cluster/coredns/
d2c34f06-1111-2222-1eb0-24f64ce37fa4",
    "createdAt": "2023-03-01T16:41:32.442000+00:00",
    "modifiedAt": "2023-03-01T18:16:54.332000+00:00",
    "tags": {},
    "configurationValues": "{\"replicaCount\":3}"
  }
}

```

CoreDNS Amazon EKS 자체 관리형 추가 기능 업데이트

Important

자체 관리형 추가 기능 유형을 사용하는 대신 클러스터에 Amazon EKS 유형의 추가 기능을 추가하는 것이 좋습니다. 유형 간의 차이를 잘 모르는 경우 [the section called “Amazon EKS 추가 기능”](#) 부분을 참조하세요. Amazon EKS 추가 기능을 클러스터에 추가하는 방법에 대한 자세한 내용은 [the section called “추가 기능 생성”](#) 섹션을 참조하세요. Amazon EKS 추가 기능을 사용할 수 없는 경우, 사용할 수 없는 이유에 대한 문제를 [컨테이너 로드맵 GitHub 리포지토리](#)에 제출하는 것이 좋습니다.

시작하기 전에 업그레이드 고려 사항을 검토하세요. 자세한 내용은 [the section called “중요 CoreDNS 업그레이드 고려 사항”](#) 섹션을 참조하세요.

1. 클러스터에 자체 관리형 추가 기능 유형이 설치되어 있는지 확인하세요. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```

aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
addon.addonVersion --output text

```

오류 메시지가 반환되면 자체 관리형 추가 기능 유형이 클러스터에 설치됩니다. 이 절차의 나머지 단계를 완료하여 설치하세요. 버전 번호가 반환되는 경우 Amazon EKS 유형의 추가 기능이 클러스터에 설치됩니다. Amazon EKS 유형의 추가 기능을 업데이트하려면 이 절차를 사용하는 대신 [CoreDNS Amazon EKS 추가 기능 업데이트](#)에 있는 절차를 사용하세요. 추가 기능 유형 간의 차이를 잘 모르는 경우 [the section called “Amazon EKS 추가 기능”](#) 부분을 참조하세요.

2. 클러스터에 현재 설치된 컨테이너 이미지의 버전을 확인하세요.

```
kubectl describe deployment coredns -n kube-system | grep Image | cut -d ":" -f 3
```

예제 출력은 다음과 같습니다.

```
v1.8.7-eksbuild.2
```

3. 현재 CoreDNS 버전이 v1.5.0 이상이지만 [CoreDNS 버전](#) 표에서 나열된 버전보다 낮다면 이 단계를 건너뛵니다. 현재 버전이 1.5.0보다 낮은 경우 프록시 추가 기능 대신 전달 추가 기능을 사용하도록 CoreDNS의 ConfigMap을 수정해야 합니다.

- a. 다음 명령을 통해 ConfigMap을 엽니다.

```
kubectl edit configmap coredns -n kube-system
```

- b. 다음 줄의 proxy를 forward로 바꿉니다. 파일을 저장하고 편집기를 종료합니다.

```
proxy . /etc/resolv.conf
```

4. 원래 1.17 또는 이전 버전에서 Kubernetes에 클러스터를 배포한 경우 CoreDNS 매니페스트에서 더 이상 사용되지 않는 줄을 제거해야 할 수 있습니다.

Important

CoreDNS 버전 1.7.0으로 업데이트하기 전에 이 단계를 완료해야 하지만, 이전 버전으로 업데이트하는 경우에도 이 단계를 완료하는 것이 좋습니다.

- a. CoreDNS 매니페스트에 이 줄이 있는지 확인합니다.

```
kubectl get configmap coredns -n kube-system -o jsonpath='{$.data.Corefile}' | grep upstream
```

결과가 반환되지 않으면 매니페스트에 해당 줄이 없는 것이므로 다음 단계로 건너뛰어 CoreDNS를 업데이트할 수 있습니다. 출력이 반환되면 줄을 제거해야 합니다.

- b. 다음 명령을 사용하여 ConfigMap을 편집하고 파일에서 upstream 단어가 포함된 줄을 제거합니다. 파일에서 다른 것을 변경하지 마세요. 이 줄을 제거한 후 변경 사항을 저장하세요.

```
kubectl edit configmap coredns -n kube-system -o yaml
```

5. 최신 CoreDNS 이미지 버전을 검색합니다.

```
kubectl describe deployment coredns -n kube-system | grep Image
```

예제 출력은 다음과 같습니다.

```
602401143452.dkr.ecr.region-code.amazonaws.com/eks/coredns:v1.8.7-eksbuild.2
```

6. CoreDNS 1.8.3 이상으로 업데이트하는 경우 system:coredns Kubernetes clusterrole에 endpointslices 권한을 추가해야 합니다.

```
kubectl edit clusterrole system:coredns -n kube-system
```

파일의 rules 부분의 기존 권한 줄 아래에 다음 줄을 추가합니다.

```
[...]
- apiGroups:
  - discovery.k8s.io
  resources:
  - endpointslices
  verbs:
  - list
  - watch
[...]
```

7. **602401143452**와 **region-code**를 이전 단계에서 반환된 출력의 값으로 대체하여 CoreDNS 추가 기능을 업데이트합니다. **v1.11.3-eksbuild.1**을 사용 중인 Kubernetes 버전의 [최신 버전 표](#)에 나와 있는 CoreDNS 버전으로 바꿉니다.

```
kubectl set image deployment.apps/coredns -n kube-system
coredns=602401143452.dkr.ecr.region-code.amazonaws.com/eks/coredns:v1.11.3-eksbuild.1
```

예제 출력은 다음과 같습니다.

```
deployment.apps/coredns image updated
```

8. 컨테이너 이미지 버전을 다시 확인하여 이전 단계에서 지정한 버전으로 업데이트되었는지 확인합니다.

```
kubectl describe deployment coredns -n kube-system | grep Image | cut -d ":" -f 3
```

예제 출력은 다음과 같습니다.

```
v1.11.3-eksbuild.1
```

높은 DNS 트래픽을 위한 CoreDNS 포드 확장

하나 이상의 노드가 있는 Amazon EKS 클러스터를 시작하면 클러스터에 배포된 노드 수에 관계없이 기본적으로 CoreDNS 이미지의 복제본 2개 배포가 기본적으로 배포됩니다. CoreDNS 포드는 클러스터의 모든 포드에 대한 이름 확인을 제공합니다. 애플리케이션은 이름 확인을 사용하여 클러스터의 포드와 서비스에 연결하고 클러스터 외부의 서비스에도 연결합니다. 포드의 이름 확인(쿼리) 요청 수가 증가하면 CoreDNS 포드가 압도되어 속도가 느려지고 포드가 처리할 수 없는 요청을 거부할 수 있습니다.

CoreDNS 포드의 증가된 로드를 처리하려면 CoreDNS용 오토 스케일링 시스템을 고려하세요. Amazon EKS는 CoreDNS의 EKS 추가 기능 버전에서 CoreDNS 배포의 Auto Scaling을 관리할 수 있습니다. 이 CoreDNS 오토스케일러는 노드 수와 CPU 코어 수를 포함하여 클러스터 상태를 지속적으로 모니터링합니다. 해당 정보를 기반으로 컨트롤러는 EKS 클러스터에서 CoreDNS 배포의 복제본 수를 동적으로 조정합니다. 이 기능은 CoreDNS v1.9 이상에서 작동합니다. CoreDNS Auto Scaling과 호환되는 버전에 대한 자세한 내용은 다음 섹션을 참조하세요.

전반적인 애플리케이션 가용성과 클러스터 확장성을 향상하려면 이 기능을 다른 [EKS 클러스터 Auto Scaling 모범 사례](#)와 함께 사용하는 것이 좋습니다.

사전 조건

Amazon EKS가 CoreDNS 배포 규모를 조정하기 위한 세 가지 사전 조건이 있습니다.

- CoreDNS의 EKS 추가 기능을 사용해야 합니다.
- 클러스터가 최소 클러스터 버전과 플랫폼 버전을 실행해야 합니다.
- 클러스터가 최소 버전의 CoreDNS EKS 추가 기능을 실행 중이어야 합니다.

최소 클러스터 버전

CoreDNS의 Auto Scaling은 Amazon EKS에서 관리하는 클러스터 컨트롤 플레인의 새로운 구성 요소에 의해 수행됩니다. 따라서 새 구성 요소가 있는 최소 플랫폼 버전을 지원하는 EKS 릴리스로 클러스터를 업그레이드해야 합니다.

새 Amazon EKS 클러스터 배포하려면 [시작](#) 부분을 참조하세요. 클러스터는 다음 표에 나열된 Kubernetes 버전과 플랫폼 버전 중 하나 또는 이후 버전을 실행해야 합니다. 나열된 것보다 이후의 모든 Kubernetes 및 플랫폼 버전도 지원됩니다. 다음 명령에서 *my-cluster*를 본인의 클러스터 이름으로 바꾼 다음 수정된 명령을 실행하여 현재 Kubernetes 버전을 확인할 수 있습니다.

```
aws eks describe-cluster --name my-cluster --query cluster.version --output text
```

Kubernetes 버전	플랫폼 버전
1.29.3	eks.7
1.28.8	eks.13
1.27.12	eks.17
1.26.15	eks.18

Note

이후 Kubernetes 버전의 모든 플랫폼 버전도 지원됩니다. 예를 들어 eks.1의 Kubernetes 버전 1.30 이상이 지원됩니다.

최소 EKS 추가 기능 버전

Kubernetes 버전	1.29	1.28	1.27	1.26
	v1.11.1-eksbuild.9	v1.10.1-eksbuild.11	v1.10.1-eksbuild.11	v1.9.3-eksbuild.15

AWS Management Console에서 CoreDNS Auto Scaling 구성

1. 클러스터가 최소 클러스터 버전 이상인지 확인합니다.

Amazon EKS는 동일한 Kubernetes 버전의 플랫폼 버전 간에 클러스터를 자동으로 업그레이드하며, 이 프로세스는 사용자가 직접 시작할 수 없습니다. 대신 클러스터를 다음 Kubernetes 버전으로 업그레이드하면 클러스터가 해당 K8s 버전과 최신 플랫폼 버전으로 업그레이드됩니다.

새로운 Kubernetes 버전에는 때로는 큰 변화가 도입되기도 합니다. 따라서 프로덕션 클러스터를 업데이트하기 전에 새 Kubernetes 버전의 별도 클러스터를 사용하여 애플리케이션의 동작을 테스트하는 것이 좋습니다.

클러스터를 새 Kubernetes 버전으로 업그레이드하려면 [기존 클러스터를 새 Kubernetes 버전으로 업데이트](#)의 절차를 따릅니다.

2. 자체 관리형 CoreDNS 배포가 아닌 CoreDNS용 EKS 추가 기능이 있는지 확인합니다.

클러스터를 생성하는 데 사용한 도구에 따라 현재 클러스터에 Amazon EKS 추가 기능이 유형이 설치되어 있지 않을 수 있습니다. 클러스터에 설치된 추가 기능의 유형을 확인하려면 다음 명령을 실행합니다. `my-cluster`를 클러스터 이름으로 바꿉니다.

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
addon.addonVersion --output text
```

버전 번호가 반환되면 클러스터에 Amazon EKS 유형의 추가 기능이 설치되어 있으며 다음 단계를 계속할 수 있습니다. 오류가 번호가 반환되는 경우 Amazon EKS 유형의 추가 기능이 클러스터에 설치되지 않습니다. 자체 관리형 버전을 Amazon EKS 추가 기능으로 대체하기 위해 [CoreDNS Amazon EKS 추가 기능 생성](#) 절차의 나머지 단계를 완료합니다.

3. CoreDNS용 EKS 추가 기능이 최소 EKS 추가 기능 버전과 같거나 그보다 높은 버전인지 확인합니다.

클러스터에 설치된 추가 기능의 버전을 확인하세요. AWS Management Console에서 확인하거나 다음 명령을 실행할 수 있습니다.

```
kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -d : -f 3
```

예제 출력은 다음과 같습니다.

v1.10.1-eksbuild.13

이 버전을 이전 섹션의 최소 EKS 추가 기능의 버전과 비교합니다. 필요한 경우 다음 [CoreDNS Amazon EKS 추가 기능 업데이트](#) 절차에 따라 EKS 추가 기능을 상위 버전으로 업그레이드합니다.

4. EKS 추가 기능의 선택적 구성 설정에 Auto Scaling 구성을 추가합니다.

- a. [Amazon EKS 콘솔](#)을 엽니다.
- b. 왼쪽 탐색 창에서 클러스터를 선택한 다음에 추가 기능을 구성할 클러스터의 이름을 선택합니다.
- c. 추가 기능 탭을 선택합니다.
- d. CoreDNS 추가 기능 상자의 오른쪽 상단에 있는 상자를 선택한 다음에 편집을 선택합니다.
- e. CoreDNS 구성 페이지에서
 - i. 사용할 버전을 선택합니다. 이전 단계와 동일한 버전을 유지하고 별도의 작업으로 버전과 구성을 업데이트하는 것이 좋습니다.
 - ii. 선택적 구성 설정을 확장합니다.
 - iii. 구성 값에 중첩 JSON 객체의 JSON 키 "autoscaling": 및 값과 키 "enabled": 및 값 true를 입력합니다. 결과 텍스트는 유효한 JSON 객체여야 합니다. 이 키와 값이 텍스트 상자의 유일한 데이터인 경우, 키와 값을 중괄호({ })로 둘러쌉니다. 다음 예제에서는 Auto Scaling이 활성화되었음을 보여줍니다.

```
{
  "autoScaling": {
    "enabled": true
  }
}
```

- iv. (선택 사항) Auto Scaling을 통해 CoreDNS 포드 수를 조정할 수 있는 최솟값과 최댓값을 제공할 수 있습니다.

다음 예제에서는 Auto Scaling이 활성화되어 있고 모든 선택적 키에 값이 있습니다. 클러스터의 DNS 서비스에 대한 복원력을 제공하려면 최소 CoreDNS 포드 수를 항상 2개 이상으로 설정하는 것이 좋습니다.

```
{
  "autoScaling": {
    "enabled": true,
    "minReplicas": 2,
    "maxReplicas": 10
  }
}
```

```
}
}
```

- f. CoreDNS 포드를 바꿔서 새 구성을 적용하려면 변경 사항 저장을 선택합니다.

Amazon EKS는 CoreDNS용 Kubernetes 배포의 롤아웃을 사용하여 EKS 추가 기능에 변경 사항을 적용합니다. AWS Management Console의 추가 기능 업데이트 기록과 `kubectl rollout status deployment/coredns --namespace kube-system`을 사용하여 롤아웃 상태를 추적할 수 있습니다.

`kubectl rollout`에는 다음 명령이 있습니다.

```
kubectl rollout

history  -- View rollout history
pause    -- Mark the provided resource as paused
restart  -- Restart a resource
resume   -- Resume a paused resource
status   -- Show the status of the rollout
undo     -- Undo a previous rollout
```

롤아웃이 너무 오래 걸리면 Amazon EKS가 롤아웃을 취소하고 유형이 추가 기능 업데이트이고 상태가 실패인 메시지가 추가 기능의 업데이트 기록에 추가됩니다. 문제를 조사하려면 롤아웃 기록부터 시작하고 CoreDNS 포드에서 `kubectl logs`를 실행하여 CoreDNS의 로그를 확인합니다.

5. 업데이트 기록의 새 항목 상태가 성공이면 롤아웃이 완료되었으며 추가 기능이 모든 CoreDNS 포드에서 새 구성을 사용하고 있는 것입니다. 클러스터에 있는 노드 수와 노드의 CPU 코어를 변경하면 Amazon EKS가 CoreDNS 배포의 복제본 수를 확장합니다.

AWS 명령줄 인터페이스에서 CoreDNS Autoscaling 구성

1. 클러스터가 최소 클러스터 버전 이상인지 확인합니다.

Amazon EKS는 동일한 Kubernetes 버전의 플랫폼 버전 간에 클러스터를 자동으로 업그레이드하며, 이 프로세스는 사용자가 직접 시작할 수 없습니다. 대신 클러스터를 다음 Kubernetes 버전으로 업그레이드하면 클러스터가 해당 K8s 버전과 최신 플랫폼 버전으로 업그레이드됩니다.

새로운 Kubernetes 버전에는 때로는 큰 변화가 도입되기도 합니다. 따라서 프로덕션 클러스터를 업데이트하기 전에 새 Kubernetes 버전의 별도 클러스터를 사용하여 애플리케이션의 동작을 테스트 하는 것이 좋습니다.

클러스터를 새 Kubernetes 버전으로 업그레이드하려면 [기존 클러스터를 새 Kubernetes 버전으로 업데이트](#)의 절차를 따릅니다.

2. 자체 관리형 CoreDNS 배포가 아닌 CoreDNS용 EKS 추가 기능이 있는지 확인합니다.

클러스터를 생성하는 데 사용한 도구에 따라 현재 클러스터에 Amazon EKS 추가 기능이 유형이 설치되어 있지 않을 수 있습니다. 클러스터에 설치된 추가 기능의 유형을 확인하려면 다음 명령을 실행합니다. my-cluster를 클러스터 이름으로 바꿉니다.

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns --query
addon.addonVersion --output text
```

버전 번호가 반환되는 경우 Amazon EKS 유형의 추가 기능이 클러스터에 설치됩니다. 오류가 번호가 반환되는 경우 Amazon EKS 유형의 추가 기능이 클러스터에 설치되지 않습니다. 자체 관리형 버전을 Amazon EKS 추가 기능으로 대체하기 위해 [CoreDNS Amazon EKS 추가 기능 생성](#) 절차의 나머지 단계를 완료합니다.

3. CoreDNS용 EKS 추가 기능이 최소 EKS 추가 기능 버전과 같거나 그보다 높은 버전인지 확인합니다.

클러스터에 설치된 추가 기능의 버전을 확인하세요. AWS Management Console에서 확인하거나 다음 명령을 실행할 수 있습니다.

```
kubectl describe deployment coredns --namespace kube-system | grep coredns: | cut -d : -f 3
```

예제 출력은 다음과 같습니다.

```
v1.10.1-eksbuild.13
```

이 버전을 이전 섹션의 최소 EKS 추가 기능의 버전과 비교합니다. 필요한 경우 다음 [CoreDNS Amazon EKS 추가 기능 업데이트](#) 절차에 따라 EKS 추가 기능을 상위 버전으로 업그레이드합니다.

4. EKS 추가 기능의 선택적 구성 설정에 Auto Scaling 구성을 추가합니다.

다음 AWS CLI 명령을 실행합니다. `my-cluster`를 본인의 클러스터 이름으로 바꾸고 IAM 역할 ARN을 사용 중인 역할로 바꿉니다.

```
aws eks update-addon --cluster-name my-cluster --addon-name coredns \
  --resolve-conflicts PRESERVE --configuration-values '{"autoScaling":
{"enabled":true}}'
```

Amazon EKS는 CoreDNS용 Kubernetes 배포의 롤아웃을 사용하여 EKS 추가 기능에 변경 사항을 적용합니다. AWS Management Console의 추가 기능 업데이트 기록과 `kubectl rollout status deployment/coredns --namespace kube-system`을 사용하여 롤아웃 상태를 추적할 수 있습니다.

`kubectl rollout`에는 다음 명령이 있습니다.

```
kubectl rollout

history -- View rollout history
pause   -- Mark the provided resource as paused
restart -- Restart a resource
resume  -- Resume a paused resource
status  -- Show the status of the rollout
undo    -- Undo a previous rollout
```

롤아웃이 너무 오래 걸리면 Amazon EKS가 롤아웃을 취소하고 유형이 추가 기능 업데이트이고 상태가 실패인 메시지가 추가 기능의 업데이트 기록에 추가됩니다. 문제를 조사하려면 롤아웃 기록부터 시작하고 CoreDNS 포드에서 `kubectl logs`를 실행하여 CoreDNS의 로그를 확인합니다.

5. (선택 사항) 오토 스케일링을 통해 CoreDNS 포드 수를 조정할 수 있는 최솟값과 최댓값을 제공할 수 있습니다.

다음 예제에서는 Auto Scaling이 활성화되어 있고 모든 선택적 키에 값이 있습니다. 클러스터의 DNS 서비스에 대한 복원력을 제공하려면 최소 CoreDNS 포드 수를 항상 2개 이상으로 설정하는 것이 좋습니다.

```
aws eks update-addon --cluster-name my-cluster --addon-name coredns \
  --resolve-conflicts PRESERVE --configuration-values '{"autoScaling":
{"enabled":true,"minReplicas":2,"maxReplicas":10}}'
```

6. 다음 명령을 실행하여 추가 기능의 업데이트 상태를 확인합니다.

```
aws eks describe-addon --cluster-name my-cluster --addon-name coredns
```

"status": "ACTIVE" 줄이 표시되면 롤아웃이 완료된 것이며 추가 기능이 모든 CoreDNS 포드에서 새 구성을 사용하고 있는 것입니다. 클러스터에 있는 노드 수와 노드의 CPU 코어를 변경하면 Amazon EKS가 CoreDNS 배포의 복제본 수를 확장합니다.

CoreDNS 지표를 사용하여 Kubernetes DNS 확인 모니터링

EKS 추가 기능 CoreDNS는 kube-dns 서비스의 Prometheus 형식으로 포트 9153의 CoreDNS에서 지표를 노출합니다. Prometheus, Amazon CloudWatch 에이전트 또는 기타 호환 시스템을 사용하여 이러한 지표를 스크레이프(수집)할 수 있습니다.

Prometheus 및 CloudWatch 에이전트 모두와 호환되는 스크레이프 구성의 예시는 Amazon CloudWatch 사용 설명서의 [Prometheus에 대한 CloudWatch 에이전트 구성](#)을 참조하세요.

Amazon EKS 클러스터에서 kube-proxy 관리

Tip

Amazon EKS Auto Mode를 사용하면 네트워킹 추가 기능을 설치하거나 업그레이드할 필요가 없습니다. Auto Mode에는 포트 네트워킹 및 로드 밸런싱 기능이 포함되어 있습니다. 자세한 내용은 [EKS Auto Mode](#) 섹션을 참조하세요.

자체 관리형 추가 기능 유형을 사용하는 대신 클러스터에 Amazon EKS 유형의 추가 기능을 추가하는 것이 좋습니다. 유형 간의 차이를 잘 모르는 경우 [the section called “Amazon EKS 추가 기능”](#) 부분을 참조하세요. Amazon EKS 추가 기능을 클러스터에 추가하는 방법에 대한 자세한 내용은 [the section called “추가 기능 생성”](#) 섹션을 참조하세요. Amazon EKS 추가 기능을 사용할 수 없는 경우, 사용할 수 없는 이유에 대한 문제를 [컨테이너 로드맵 GitHub 리포지토리](#)에 제출하는 것이 좋습니다.

kube-proxy 추가 기능은 Amazon EKS 클러스터의 각 Amazon EC2 노드에 배포됩니다. 노드에 대한 네트워크 규칙을 유지하고 포드와의 네트워크 통신을 가능하게 합니다. 이 추가 기능은 클러스터의 Fargate 노드에 배포되지 않습니다. 자세한 내용은 Kubernetes 문서의 [kube-proxy](#)를 참조하세요.

Amazon EKS 추가 기능으로 설치

kube-proxy 버전

다음 표에는 각 Kubernetes 버전에 사용할 수 있는 Amazon EKS 추가 기능 유형의 최신 버전이 나열되어 있습니다.

Kubernetes 버전	kube-proxy 버전
1.33	v1.33.0-eksbuild.2
1.32	v1.32.3-eksbuild.7
1.31	v1.31.7-eksbuild.7
1.30	v1.30.11-eksbuild.5
1.29	v1.29.15-eksbuild.5
1.28	v1.28.15-eksbuild.20
1.27	v1.27.16-eksbuild.30
1.26	v1.26.15-eksbuild.35

Note

이전 버전의 설명서가 올바르지 않았습니다. kube-proxy 버전 v1.28.5, v1.27.9, v1.26.12는 사용할 수 없습니다.

이 추가 기능을 자체 관리하는 경우 표의 버전이 사용 가능한 자체 관리 버전과 다를 수 있습니다.

kube-proxy 컨테이너 이미지

kube-proxy 컨테이너 이미지는 최소 패키지를 포함하고 셸이 없는 Amazon EKS Distro에서 유지 관리하는 [최소 기본 이미지](#)를 기반으로 합니다. 자세한 내용은 [Amazon EKS Distro](#)를 참조하세요.

다음 테이블에는 각 Amazon EKS 클러스터 버전에 사용할 수 있는 자체 관리형 kube-proxy 컨테이너 이미지의 최신 버전이 나열되어 있습니다.

버전	kube-proxy
1.33	v1.33.0-minimal-eksbuild.2
1.32	v1.32.3-minimal-eksbuild.7
1.31	v1.31.7-minimal-eksbuild.7
1.30	v1.30.11-minimal-eksbuild.5
1.29	v1.29.15-minimal-eksbuild.5
1.28	v1.28.15-minimal-eksbuild.20
1.27	v1.27.16-minimal-eksbuild.30
1.26	v1.26.15-minimal-eksbuild.35

[Amazon EKS 추가 기능 유형을 업데이트](#)할 때 유효한 Amazon EKS 추가 기능 버전을 지정합니다. 이 버전은 이 표에 나와 있는 버전이 아닐 수도 있습니다. [Amazon EKS 추가 기능](#) 버전이 이 추가 기능의 자체 관리형 유형을 업데이트할 때 지정한 컨테이너 이미지 버전과 항상 일치하지는 않기 때문입니다. 이 추가 기능의 자체 관리형 유형을 업데이트할 때는 이 표에 나열된 유효한 컨테이너 이미지 버전을 지정합니다.

Kubernetes **kube-proxy** 자체 관리형 추가 기능 업데이트

Important

자체 관리형 추가 기능 유형을 사용하는 대신 클러스터에 Amazon EKS 유형의 추가 기능을 추가하는 것이 좋습니다. 유형 간의 차이를 잘 모르는 경우 [the section called “Amazon EKS 추가 기능”](#) 부분을 참조하세요. Amazon EKS 추가 기능을 클러스터에 추가하는 방법에 대한 자세한 내용은 [the section called “추가 기능 생성”](#) 섹션을 참조하세요. Amazon EKS 추가 기능을 사용할 수 없는 경우, 사용할 수 없는 이유에 대한 문제를 [컨테이너 로드맵 GitHub 리포지토리](#)에 제출하는 것이 좋습니다.

사전 조건

- 기존 Amazon EKS 클러스터. 배포하려면 [시작](#) 섹션을 참조하세요.

고려 사항

- Amazon EKS 클러스터의 Kube-proxy에는 [Kubernetes와 동일한 호환성 및 skew 정책](#)이 있습니다. [클러스터와 Amazon EKS 추가 기능 버전 호환성을 확인](#)하는 방법을 알아보세요.

- 클러스터에 자체 관리형 추가 기능 유형이 설치되어 있는지 확인하세요. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks describe-addon --cluster-name my-cluster --addon-name kube-proxy --query
addon.addonVersion --output text
```

오류 메시지가 반환되면 자체 관리형 추가 기능 유형이 클러스터에 설치됩니다. 이 주제의 나머지 단계는 추가 기능의 자체 관리형 유형을 업데이트하는 단계입니다. 버전 번호가 반환되는 경우 Amazon EKS 유형의 추가 기능이 클러스터에 설치됩니다. 업데이트하려면 이 항목의 절차 대신 [Amazon EKS 추가 기능 업데이트](#)의 절차를 사용하세요. 추가 기능 유형 간의 차이를 잘 모르는 경우 [the section called “Amazon EKS 추가 기능”](#) 부분을 참조하세요.

- 클러스터에 현재 설치된 컨테이너 이미지의 버전을 확인하세요.

```
kubectl describe daemonset kube-proxy -n kube-system | grep Image
```

예제 출력은 다음과 같습니다.

```
Image:      602401143452.dkr.ecr.region-code.amazonaws.com/eks/kube-proxy:v1.29.1-
eksbuild.2
```

예제 출력에서 *v1.29.1-eksbuild.2*가 클러스터에 설치된 버전입니다.

- 602401143452* 및 *region-code*를 이전 단계의 출력의 값으로 대체하여 kube-proxy 추가 기능을 업데이트합니다. *v1.30.6-eksbuild.3*를 [각 Amazon EKS 클러스터 버전에서 사용 가능한 최신 자체 관리형 kube-proxy 컨테이너 이미지 버전](#) 표에 나열된 kube-proxy 버전으로 바꿉니다.

Important

각 이미지 유형의 매니페스트는 서로 다르며 기본 또는 최소 이미지 유형 사이에서 호환되지 않습니다. 진입점과 인수가 일치하도록 이전 이미지와 동일한 이미지 유형을 사용해야 합니다.


```
kubectl set image daemonset.apps/kube-proxy -n kube-system kube-
proxy=602401143452.dkr.ecr.region-code.amazonaws.com/eks/kube-proxy:v1.30.6-
eksbuild.3
```

예제 출력은 다음과 같습니다.

```
daemonset.apps/kube-proxy image updated
```

- 이제 클러스터에 새 버전이 설치되어 있는지 확인합니다.

```
kubectl describe daemonset kube-proxy -n kube-system | grep Image | cut -d ":" -f 3
```

예제 출력은 다음과 같습니다.

```
v1.30.0-eksbuild.3
```

- 동일한 클러스터에서 x86 및 Arm 노드를 사용하고 있고 클러스터가 2020년 8월 17일 이전에 배포된 경우 다음 명령을 사용하여 여러 하드웨어 아키텍처에 대한 노드 셀렉터를 포함하도록 kube-proxy 매니페스트를 편집합니다. 이는 일회성 작업입니다. 매니페스트에 셀렉터를 추가한 후에는 추가 기능을 업데이트할 때마다 이 옵션을 추가할 필요가 없습니다. 클러스터가 2020년 8월 17일 또는 이후에 배포된 경우, kube-proxy은(는) 이미 멀티 아키텍처를 지원합니다.

```
kubectl edit -n kube-system daemonset/kube-proxy
```

다음 노드 선택기를 편집기에서 파일에 추가한 후 파일을 저장합니다. 편집기에서 이 텍스트를 포함할 위치에 대한 예는 GitHub에서 [CNI 매니페스트](#) 파일을 참조하세요. 이를 통해 Kubernetes는 노드의 하드웨어 아키텍처를 기반으로 올바른 하드웨어 이미지를 가져올 수 있습니다.

```
- key: "kubernetes.io/arch"
  operator: In
  values:
    - amd64
    - arm64
```

- 원래 클러스터가 Kubernetes 버전 1.14 이상으로 생성된 경우 kube-proxy가 이미 Affinity Rule을 포함하므로 이 단계를 건너뛸 수 있습니다. 원래 Kubernetes 버전 1.13 이하로 Amazon EKS 클러스터를 생성했고 클러스터에서 Fargate 노드를 사용하려는 경우 kube-proxy 매니페

스트를 편집하여 NodeAffinity 규칙을 포함시켜 kube-proxy 포드가 Fargate 노드에서 예약하지 못하도록 하세요. 이는 일회성 편집입니다. Affinity Rule을(를) 매니페스트에 추가하면 추가 기능을 업데이트할 때마다 추가할 필요가 없습니다. kube-proxy DaemonSet을 편집합니다.

```
kubectl edit -n kube-system daemonset/kube-proxy
```

편집기에서 파일의 DaemonSet spec 섹션에 다음 Affinity Rule을 추가한 다음 파일을 저장합니다. 편집기에서 이 텍스트를 포함할 위치에 대한 예는 GitHub에서 [CNI 매니페스트](#) 파일을 참조하세요.

```
- key: eks.amazonaws.com/compute-type
  operator: NotIn
  values:
    - fargate
```

Amazon EKS에 워크로드와 추가 기능을 배포하는 방법 알아보기

워크로드는 Kubernetes의 포드에 배포되는 컨테이너에 배포됩니다. 포드에는 하나 이상의 컨테이너가 포함됩니다. 일반적으로 동일한 서비스를 제공하는 하나 이상의 포드가 Kubernetes 서비스에 배포됩니다. 동일한 서비스를 제공하는 여러 포드를 배포한 후에는 다음을 수행할 수 있습니다.

- AWS Management Console을 사용하여 각 클러스터에서 실행 중인 [워크로드에 대한 정보 보기](#).
- Kubernetes [Vertical Pod Autoscaler](#)로 포드를 수직적으로 스케일링할 수 있습니다.
- Kubernetes [Horizontal Pod Autoscaler](#)를 사용하여 수요에 따라 필요한 수의 포드를 수평적으로 스케일링할 수 있습니다.
- 외부(인터넷에 액세스할 수 있는 포드의 경우) 또는 내부(프라이빗 포드의 경우) [Network Load Balancer](#)를 사용하여 여러 포드에 걸쳐 네트워크 트래픽의 균형을 맞출 수 있습니다. 로드 밸런서는 OSI 모델의 계층 4에서 트래픽을 라우팅합니다.
- [Application Load Balancer](#)를 생성하여 포드 전반에서 애플리케이션 트래픽을 분산할 수 있습니다. Application Load Balancer는 OSI 모델의 계층 7에서 트래픽을 라우팅합니다.
- Kubernetes를 처음 사용하는 경우 이 주제는 [샘플 애플리케이션 배포](#)에 도움이 됩니다.
- externalIPs를 사용하여 [서비스에 할당할 수 있는 IP 주소 제한](#)할 수 있습니다.

Linux에서 샘플 애플리케이션 배포

이 주제에서는 Linux 노드의 클러스터에 샘플 애플리케이션을 배포합니다.

사전 조건

- 노드가 하나 이상 있는 기존 Kubernetes 클러스터. 기존 Amazon EKS 클러스터가 없는 경우 [시작](#) 가이드 중 하나를 사용하여 Amazon EKS 클러스터를 배포할 수 있습니다.
- 컴퓨터에 설치된 Kubectl. 자세한 내용은 [the section called “kubectl 및 eksctl 설정”](#) 섹션을 참조하세요.
- 클러스터와 통신하도록 구성된 Kubectl. 자세한 내용은 [the section called “kubectl을 사용하여 클러스터에 액세스”](#) 섹션을 참조하세요.
- 샘플 워크로드를 Fargate에 배포하려는 경우, 이름을 변경하지 않는 한 이 자습서에서 생성한 것과 동일한 네임스페이스(eks-sample-app)를 포함하는 기존 [Fargate 프로파일](#)이 있어야 합니다. [시작](#)에서 gudes 중 하나를 사용하여 클러스터를 만든 경우, 시작 가이드에서 만든 프로파일에는 이 자습

서에서 사용하는 네임스페이스가 지정되어 있지 않으므로 새 프로필을 만들거나 기존 프로필에 네임스페이스를 추가해야 합니다. 또한 VPC에 프라이빗 서브넷이 하나 이상 있어야 합니다.

다음 단계에서 많은 변수를 변경할 수 있지만 지정된 경우에만 변수 값을 변경하는 것이 좋습니다. Kubernetes 포드, 배포 및 서비스를 더 잘 이해하면 다른 값을 변경해 볼 수 있습니다.

네임스페이스 생성

네임스페이스를 사용하면 Kubernetes에서 리소스를 그룹화할 수 있습니다. 자세한 내용은 쿠버네티스 문서의 [네임스페이스](#)를 참조하세요. [AWS Fargate로 컴퓨팅 관리를 간소화](#)하기 위해 샘플 애플리케이션을 배포하려는 경우, 시작할 때 [AWS Fargate를 사용하는 포드 정의](#)에서 namespace의 값이 eks-sample-app인지 확인해야 합니다.

```
kubectl create namespace eks-sample-app
```

Kubernetes 배포를 생성합니다.

Kubernetes 배포를 생성합니다. 이 샘플 배포는 퍼블릭 리포지토리에서 컨테이너 이미지를 가져와 클러스터에 3개의 복제본(개별 포드)을 배포합니다. 자세한 내용은 쿠버네티스 문서의 [디플로이먼트](#)를 참조하세요.

- 다음 콘텐츠를 eks-sample-deployment.yaml이라는 파일에 저장합니다. 샘플 애플리케이션의 컨테이너는 네트워크 스토리지를 사용하지 않지만 필요한 애플리케이션이 있을 수 있습니다. 자세한 내용은 [앱 데이터 스토리지](#) 섹션을 참조하세요.
 - kubernetes.io/arch 키 아래의 amd64 또는 arm64 values는 애플리케이션을 하드웨어 아키텍처 중 하나에 배포할 수 있음을 의미합니다(클러스터에 둘 다 있는 경우). 이는 이 이미지가 다중 아키텍처 이미지이지만 모두가 그렇지 않기 때문에 가능합니다. 이미지를 가져오는 리포지토리에서 [이미지 세부 정보](#)를 확인하여 이미지가 지원되는 하드웨어 아키텍처를 결정할 수 있습니다. 하드웨어 아키텍처 유형을 지원하지 않거나 이미지를 배포하지 않으려는 이미지를 배포하는 경우 매니페스트에서 해당 유형을 제거합니다. 자세한 내용은 쿠버네티스 문서의 [잘 알려진 레이블, 어노테이션, 테인트\(Taint\)](#)를 참조하세요.
 - kubernetes.io/os: linux nodeSelector는 예를 들어, 클러스터에 Linux 및 Windows 노드가 있는 경우 이미지가 Linux 노드에만 배포됨을 의미합니다. 자세한 내용은 쿠버네티스 문서의 [잘 알려진 레이블, 어노테이션, 테인트\(Taint\)](#)를 참조하세요.

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```
name: eks-sample-linux-deployment
namespace: eks-sample-app
labels:
  app: eks-sample-linux-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: eks-sample-linux-app
  template:
    metadata:
      labels:
        app: eks-sample-linux-app
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: kubernetes.io/arch
                    operator: In
                    values:
                      - amd64
                      - arm64
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx:1.23
          ports:
            - name: http
              containerPort: 80
          imagePullPolicy: IfNotPresent
      nodeSelector:
        kubernetes.io/os: linux
```

2. 배포 매니페스트를 클러스터에 적용합니다.

```
kubectl apply -f eks-sample-deployment.yaml
```

서비스 생성

서비스를 사용하면 단일 IP 주소 또는 이름을 통해 모든 복제본에 액세스할 수 있습니다. 자세한 내용은 쿠버네티스 문서의 [서비스](#)를 참조하세요. 샘플 애플리케이션에서는 구현되지 않았지만 다른 AWS

서비스와 상호 작용해야 하는 애플리케이션이 있는 경우 포드에 대한 Kubernetes 서비스 계정을 생성하고 이를 AWS IAM 계정에 연결하는 것이 좋습니다. 서비스 계정을 지정하면 포드가 다른 서비스와 상호 작용하기 위해 지정한 최소 권한만 갖게 됩니다. 자세한 내용은 [the section called “IRSA를 사용한 자격 증명”](#) 단원을 참조하십시오.

1. 다음 콘텐츠를 `eks-sample-service.yaml`이라는 파일에 저장합니다. Kubernetes는 클러스터 내에서만 액세스할 수 있는 자체 IP 주소를 서비스에 할당합니다. 클러스터 외부에서 서비스에 액세스하려면 [AWS 로드 밸런서 컨트롤러](#)를 배포하여 [애플리케이션](#) 또는 [네트워크](#) 트래픽을 서비스에 로드 밸런싱합니다.

```
apiVersion: v1
kind: Service
metadata:
  name: eks-sample-linux-service
  namespace: eks-sample-app
  labels:
    app: eks-sample-linux-app
spec:
  selector:
    app: eks-sample-linux-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

2. 서비스 매니페스트를 클러스터에 적용합니다.

```
kubectl apply -f eks-sample-service.yaml
```

생성된 리소스 검토

1. `eks-sample-app` 네임스페이스에 있는 모든 리소스를 봅니다.

```
kubectl get all -n eks-sample-app
```

예제 출력은 다음과 같습니다.

NAME	READY	STATUS	RESTARTS	AGE
pod/eks-sample-linux-deployment-65b7669776-m6qxz	1/1	Running	0	27m

pod/eks-sample-linux-deployment-65b7669776-mmxvd	1/1	Running	0	27m
pod/eks-sample-linux-deployment-65b7669776-qzn22	1/1	Running	0	27m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
AGE				
service/eks-sample-linux-service	ClusterIP	10.100.74.8	<none>	80/TCP
32m				

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/eks-sample-linux-deployment	3/3	3	3	27m

NAME	DESIRED	CURRENT	READY
AGE			
replicaset.apps/eks-sample-linux-deployment-776d8f8fd8	3	3	3
27m			

이 출력에는 이전 단계에서 배포된 샘플 매니페스트에 지정된 서비스 및 배포가 표시됩니다. 3개의 포드도 확인할 수 있습니다. 동일한 매니페스트에 3 replicas가 지정되었기 때문입니다. 포드에 대한 자세한 내용은 쿠버네티스 문서의 [포드](#)를 참조하세요. Kubernetes는 샘플 매니페스트에 지정되지 않은 경우에도 자동으로 replicaset 리소스를 생성합니다. ReplicaSets에 대한 자세한 내용은 Kubernetes 설명서에서 [ReplicaSet](#)를 참조하세요.

Note

Kubernetes는 매니페스트에 지정된 복제본의 수를 유지합니다. 프로덕션 배포이고 Kubernetes의 복제본 수 수평적 스케일링 또는 포드의 컴퓨팅 리소스 수직적 스케일링을 원하는 경우 [Horizontal Pod Autoscaler로 포드 배포 확장](#) 및 [Vertical Pod Autoscaler로 포드 리소스 조정](#)을 사용하여 이를 수행할 수 있습니다.

2. 배포된 서비스의 세부 정보를 확인합니다.

```
kubectl -n eks-sample-app describe service eks-sample-linux-service
```

예제 출력은 다음과 같습니다.

```
Name:          eks-sample-linux-service
Namespace:     eks-sample-app
Labels:        app=eks-sample-linux-app
Annotations:   <none>
Selector:      app=eks-sample-linux-app
```

```
Type: ClusterIP
IP Families: <none>
IP: 10.100.74.8
IPs: 10.100.74.8
Port: <unset> 80/TCP
TargetPort: 80/TCP
Endpoints: 192.168.24.212:80,192.168.50.185:80,192.168.63.93:80
Session Affinity: None
Events: <none>
```

이전 출력에서 IP:의 값은 클러스터 내의 모든 노드 또는 포트에서 도달할 수 있는 고유한 IP 주소이지만 클러스터 외부에서는 도달할 수 없습니다. Endpoints의 값은 VPC 내에서 서비스의 일부인 포트에 할당된 IP 주소입니다.

- 이전 단계에서 [네임스페이스를 볼](#) 때 출력에 나열된 포트 중 하나의 세부 정보를 봅니다. **776d8f8fd8-78w66**을 포트 중 하나에 대해 반환된 값으로 바꿉니다.

```
kubectl -n eks-sample-app describe pod eks-sample-linux-deployment-65b7669776-m6qxz
```

단축 예제 출력

```
Name: eks-sample-linux-deployment-65b7669776-m6qxz
Namespace: eks-sample-app
Priority: 0
Node: ip-192-168-45-132.us-west-2.compute.internal/192.168.45.132
[...]
IP: 192.168.63.93
IPs:
  IP: 192.168.63.93
Controlled By: ReplicaSet/eks-sample-linux-deployment-65b7669776
[...]
Conditions:
  Type           Status
  Initialized     True
  Ready           True
  ContainersReady True
  PodScheduled    True
[...]
Events:
  Type    Reason      Age    From
  Message
```



```

-----
Normal    Scheduled    3m20s    default-scheduler
Successfully assigned eks-sample-app/eks-sample-linux-deployment-65b7669776-m6qxz to
ip-192-168-45-132.us-west-2.compute.internal
[...]
```

이전 출력에서 IP:의 값은 노드가 있는 서브넷에 할당된 CIDR 블록에서 포드에 할당되는 고유한 IP입니다. 포드에 다른 CIDR 블록의 IP 주소를 할당하려면 기본 동작을 변경할 수 있습니다. 자세한 내용은 [the section called “사용자 지정 네트워킹”](#) 단원을 참조하십시오. Kubernetes 스케줄러가 IP 주소 **192.168.45.132**를 사용하여 Node의 포드를 예약했는지 확인할 수도 있습니다.

Tip

명령줄을 사용하는 대신 AWS Management Console에서 포드, 서비스, 배포 및 기타 Kubernetes 리소스에 대한 많은 세부 정보를 볼 수 있습니다. 자세한 내용은 [the section called “클러스터 리소스 액세스”](#) 단원을 참조하십시오.

포드에서 셸 실행

1. 이전 단계에서 설명한 포드에서 셸을 실행하여 **65b7669776-m6qxz**를 포드 중 하나의 ID로 바꿉니다.

```
kubectl exec -it eks-sample-linux-deployment-65b7669776-m6qxz -n eks-sample-app -- /bin/bash
```

2. 포드 셸에서 이전 단계에서 배포와 함께 설치된 웹 서버의 출력을 봅니다. 서비스 이름만 지정하면 됩니다. 기본적으로 Amazon EKS 클러스터와 함께 배포되는 CoreDNS에 의해 서비스의 IP 주소로 확인됩니다.

```
curl eks-sample-linux-service
```

예제 출력은 다음과 같습니다.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

```
[...]
```

3. 포트 셸에서 포트의 DNS 서버를 봅니다.

```
cat /etc/resolv.conf
```

예제 출력은 다음과 같습니다.

```
nameserver 10.100.0.10
search eks-sample-app.svc.cluster.local svc.cluster.local cluster.local us-
west-2.compute.internal
options ndots:5
```

이전 출력에서 10.100.0.10은 클러스터에 배포된 모든 포트의 nameserver로 자동 할당됩니다.

4. `exit`를 입력하여 포트에서 연결을 해제합니다.

5. 샘플 애플리케이션 사용이 끝나면 다음 명령을 사용하여 샘플 네임스페이스, 서비스 및 배포를 제거할 수 있습니다.

```
kubectl delete namespace eks-sample-app
```

다음 단계

샘플 애플리케이션을 배포한 후 다음 연습 중 일부를 시도해 볼 수 있습니다.

- [Application Load Balancer를 사용하여 애플리케이션 및 HTTP 트래픽 라우팅](#)
- [Network Load Balancer를 사용하여 TCP 및 UDP 트래픽 라우팅](#)

Windows에서 샘플 애플리케이션을 배포하려면

이 주제에서는 Windows 노드의 클러스터에 샘플 애플리케이션을 배포합니다.

사전 조건

- 노드가 하나 이상 있는 기존 Kubernetes 클러스터. 기존 Amazon EKS 클러스터가 없는 경우 [시작 가이드](#) 중 하나를 사용하여 Amazon EKS 클러스터를 배포할 수 있습니다. 클러스터 및 하나 이상의 Amazon EC2 Windows 노드에 대해 [Windows 지원](#)이 활성화되어 있어야 합니다.

- 컴퓨터에 설치된 Kubectl. 자세한 내용은 [the section called “kubectl 및 eksctl 설정”](#) 섹션을 참조하세요.
- 클러스터와 통신하도록 구성된 Kubectl. 자세한 내용은 [the section called “kubectl을 사용하여 클러스터에 액세스”](#) 섹션을 참조하세요.
- 샘플 워크로드를 Fargate에 배포하려는 경우, 이름을 변경하지 않는 한 이 자습서에서 생성한 것과 동일한 네임스페이스(eks-sample-app)를 포함하는 기존 [Fargate 프로필](#)이 있어야 합니다. [시작](#)에서 gudes 중 하나를 사용하여 클러스터를 만든 경우, 시작 가이드에서 만든 프로필에는 이 자습서에서 사용하는 네임스페이스가 지정되어 있지 않으므로 새 프로필을 만들거나 기존 프로필에 네임스페이스를 추가해야 합니다. 또한 VPC에 프라이빗 서브넷이 하나 이상 있어야 합니다.

다음 단계에서 많은 변수를 변경할 수 있지만 지정된 경우에만 변수 값을 변경하는 것이 좋습니다. Kubernetes 포드, 배포 및 서비스를 더 잘 이해하면 다른 값을 변경해 볼 수 있습니다.

네임스페이스 생성

네임스페이스를 사용하면 Kubernetes에서 리소스를 그룹화할 수 있습니다. 자세한 내용은 쿠버네티스 문서의 [네임스페이스](#)를 참조하세요. [AWS Fargate로 컴퓨팅 관리를 간소화](#)하기 위해 샘플 애플리케이션을 배포하려는 경우, 시작할 때 [AWS Fargate를 사용하는 포드 정의](#)에서 namespace의 값이 eks-sample-app인지 확인해야 합니다.

```
kubectl create namespace eks-sample-app
```

Kubernetes 배포를 생성합니다.

이 샘플 배포는 퍼블릭 리포지토리에서 컨테이너 이미지를 가져와 클러스터에 3개의 복제본(개별 포드)을 배포합니다. 자세한 내용은 쿠버네티스 문서의 [디플로이먼트](#)를 참조하세요.

1. 다음 콘텐츠를 eks-sample-deployment.yaml이라는 파일에 저장합니다. 샘플 애플리케이션의 컨테이너는 네트워크 스토리지를 사용하지 않지만 필요한 애플리케이션이 있을 수 있습니다. 자세한 내용은 [앱 데이터 스토리지](#) 단원을 참조하십시오.
 - kubernetes.io/os: windows nodeSelector는 예를 들어, 클러스터에 Windows 및 Linux 노드가 있는 경우 이미지가 Windows 노드에만 배포됨을 의미합니다. 자세한 내용은 쿠버네티스 문서의 [잘 알려진 레이블, 어노테이션, 테인트\(Taint\)](#)를 참조하세요.

```
apiVersion: apps/v1
kind: Deployment
metadata:
```

```

name: eks-sample-windows-deployment
namespace: eks-sample-app
labels:
  app: eks-sample-windows-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: eks-sample-windows-app
  template:
    metadata:
      labels:
        app: eks-sample-windows-app
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: beta.kubernetes.io/arch
                    operator: In
                    values:
                      - amd64
      containers:
        - name: windows-server-iis
          image: mcr.microsoft.com/windows/servercore:ltsc2019
          ports:
            - name: http
              containerPort: 80
          imagePullPolicy: IfNotPresent
          command:
            - powershell.exe
            - -command
            - "Add-WindowsFeature Web-Server; Invoke-WebRequest -UseBasicParsing
              -Uri 'https://dotnetbinaries.blob.core.windows.net/servicemonitor/2.0.1.6/
              ServiceMonitor.exe' -OutFile 'C:\\ServiceMonitor.exe'; echo '<html><body><br/
              ><br/><marquee><H1>Hello EKS!!!<H1><marquee></body><html>' > C:\\inetpub\\wwwroot\\
              \\default.html; C:\\ServiceMonitor.exe 'w3svc'; "
          nodeSelector:
            kubernetes.io/os: windows

```

2. 배포 매니페스트를 클러스터에 적용합니다.

```
kubectl apply -f eks-sample-deployment.yaml
```

서비스 생성

서비스를 사용하면 단일 IP 주소 또는 이름을 통해 모든 복제본에 액세스할 수 있습니다. 자세한 내용은 쿠버네티스 문서의 [서비스](#)를 참조하세요. 샘플 애플리케이션에서는 구현되지 않았지만 다른 AWS 서비스와 상호 작용해야 하는 애플리케이션이 있는 경우 포드에 대한 Kubernetes 서비스 계정을 생성하고 이를 AWS IAM 계정에 연결하는 것이 좋습니다. 서비스 계정을 지정하면 포드가 다른 서비스와 상호 작용하기 위해 지정한 최소 권한만 갖게 됩니다. 자세한 내용은 [the section called “IRSA를 사용한 자격 증명”](#) 단원을 참조하십시오.

1. 다음 콘텐츠를 eks-sample-service.yaml이라는 파일에 저장합니다. Kubernetes는 클러스터 내에서만 액세스할 수 있는 자체 IP 주소를 서비스에 할당합니다. 클러스터 외부에서 서비스에 액세스하려면 [AWS 로드 밸런서 컨트롤러](#)를 배포하여 [애플리케이션](#) 또는 [네트워크](#) 트래픽을 서비스에 로드 밸런싱합니다.

```
apiVersion: v1
kind: Service
metadata:
  name: eks-sample-windows-service
  namespace: eks-sample-app
  labels:
    app: eks-sample-windows-app
spec:
  selector:
    app: eks-sample-windows-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

2. 서비스 매니페스트를 클러스터에 적용합니다.

```
kubectl apply -f eks-sample-service.yaml
```

생성된 리소스 검토

1. eks-sample-app 네임스페이스에 있는 모든 리소스를 봅니다.

```
kubectl get all -n eks-sample-app
```

예제 출력은 다음과 같습니다.

NAME	READY	STATUS	RESTARTS	AGE
pod/eks-sample-windows-deployment-65b7669776-m6qxz	1/1	Running	0	27m
pod/eks-sample-windows-deployment-65b7669776-mmxd	1/1	Running	0	27m
pod/eks-sample-windows-deployment-65b7669776-qzn22	1/1	Running	0	27m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
service/eks-sample-windows-service	ClusterIP	10.100.74.8	<none>	80/
TCP	32m			

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/eks-sample-windows-deployment	3/3	3	3	27m

NAME	DESIRED	CURRENT	READY
replicaset.apps/eks-sample-windows-deployment-776d8f8fd8	3	3	3
27m			

이 출력에는 이전 단계에서 배포된 샘플 매니페스트에 지정된 서비스 및 배포가 표시됩니다. 3개의 포드도 확인할 수 있습니다. 동일한 매니페스트에 3 replicas가 지정되었기 때문입니다. 포드에 대한 자세한 내용은 쿠버네티스 문서의 [포드](#)를 참조하세요. Kubernetes는 샘플 매니페스트에 지정되지 않은 경우에도 자동으로 replicaset 리소스를 생성합니다. ReplicaSets에 대한 자세한 내용은 Kubernetes 설명서에서 [ReplicaSet](#)를 참조하세요.

Note

Kubernetes는 매니페스트에 지정된 복제본의 수를 유지합니다. 프로덕션 배포이고 Kubernetes의 복제본 수 수평적 스케일링 또는 포드의 컴퓨팅 리소스 수직적 스케일링을 원하는 경우 [Horizontal Pod Autoscaler로 포드 배포 확장](#) 및 [Vertical Pod Autoscaler로 포드 리소스 조정](#)을 사용하여 이를 수행할 수 있습니다.

2. 배포된 서비스의 세부 정보를 확인합니다.

```
kubectl -n eks-sample-app describe service eks-sample-windows-service
```

예제 출력은 다음과 같습니다.

```
Name:                eks-sample-windows-service
Namespace:          eks-sample-app
Labels:             app=eks-sample-windows-app
Annotations:        <none>
Selector:           app=eks-sample-windows-app
Type:               ClusterIP
IP Families:        <none>
IP:                 10.100.74.8
IPs:                10.100.74.8
Port:               <unset> 80/TCP
TargetPort:         80/TCP
Endpoints:          192.168.24.212:80,192.168.50.185:80,192.168.63.93:80
Session Affinity:   None
Events:             <none>
```

이전 출력에서 IP:의 값은 클러스터 내의 모든 노드 또는 포트에서 도달할 수 있는 고유한 IP 주소이지만 클러스터 외부에서는 도달할 수 없습니다. Endpoints의 값은 VPC 내에서 서비스의 일부인 포트에 할당된 IP 주소입니다.

- 이전 단계에서 [네임스페이스를 볼](#) 때 출력에 나열된 포트 중 하나의 세부 정보를 봅니다. **776d8f8fd8-78w66**을 포트 중 하나에 대해 반환된 값으로 바꿉니다.

```
kubectl -n eks-sample-app describe pod eks-sample-windows-deployment-65b7669776-m6qxz
```

단축 예제 출력

```
Name:                eks-sample-windows-deployment-65b7669776-m6qxz
Namespace:          eks-sample-app
Priority:            0
Node:               ip-192-168-45-132.us-west-2.compute.internal/192.168.45.132
[...]
IP:                 192.168.63.93
IPs:
  IP:               192.168.63.93
Controlled By:      ReplicaSet/eks-sample-windows-deployment-65b7669776
[...]
```

Conditions:

Type	Status
Initialized	True
Ready	True
ContainersReady	True
PodScheduled	True

[...]

Events:

Type	Reason	Age	From
Message			

Normal Scheduled 3m20s default-scheduler

Successfully assigned eks-sample-app/eks-sample-windows-deployment-65b7669776-m6qxz to ip-192-168-45-132.us-west-2.compute.internal

[...]

이전 출력에서 IP:의 값은 노드가 있는 서브넷에 할당된 CIDR 블록에서 포드에 할당되는 고유한 IP입니다. 포드에 다른 CIDR 블록의 IP 주소를 할당하려면 기본 동작을 변경할 수 있습니다. 자세한 내용은 [the section called “사용자 지정 네트워킹”](#) 단원을 참조하십시오. Kubernetes 스케줄러가 IP 주소 **192.168.45.132**를 사용하여 Node의 포드를 예약했는지 확인할 수도 있습니다.

 Tip

명령줄을 사용하는 대신 AWS Management Console에서 포드, 서비스, 배포 및 기타 Kubernetes 리소스에 대한 많은 세부 정보를 볼 수 있습니다. 자세한 내용은 [the section called “클러스터 리소스 액세스”](#) 단원을 참조하십시오.

포드에서 셸 실행

1. 이전 단계에서 설명한 포드에서 셸을 실행하여 **65b7669776-m6qxz**를 포드 중 하나의 ID로 바꿉니다.

```
kubectl exec -it eks-sample-windows-deployment-65b7669776-m6qxz -n eks-sample-app -- powershell.exe
```

2. 포드 셸에서 이전 단계에서 배포와 함께 설치된 웹 서버의 출력을 봅니다. 서비스 이름만 지정하면 됩니다. 기본적으로 Amazon EKS 클러스터와 함께 배포되는 CoreDNS에 의해 서비스의 IP 주소로 확인됩니다.


```
Invoke-WebRequest -uri eks-sample-windows-service/default.html -UseBasicParsing
```

예제 출력은 다음과 같습니다.

```

StatusCode      : 200
StatusDescription : OK
Content          : < h t m l > < b o d y > < b r / > < b r / > < m a r q u e e > < H
                  1 > H e l l o
                  E K S ! ! ! < H 1 > < m a r q u e e > < / b o d y > < h t m l >

```

3. 포트 셸에서 포트의 DNS 서버를 봅니다.

```
Get-NetIPConfiguration
```

간략한 출력

```

InterfaceAlias      : vEthernet
[...]
IPv4Address         : 192.168.63.14
[...]
DNSServer           : 10.100.0.10

```

이전 출력에서 10.100.0.10은 클러스터에 배포된 모든 포트의 DNS 서버로 자동 할당됩니다.

4. `exit`를 입력하여 포트에서 연결을 해제합니다.
5. 샘플 애플리케이션 사용이 끝나면 다음 명령을 사용하여 샘플 네임스페이스, 서비스 및 배포를 제거할 수 있습니다.

```
kubectl delete namespace eks-sample-app
```

다음 단계

샘플 애플리케이션을 배포한 후 다음 연습 중 일부를 시도해 볼 수 있습니다.

- [Application Load Balancer를 사용하여 애플리케이션 및 HTTP 트래픽 라우팅](#)
- [Network Load Balancer를 사용하여 TCP 및 UDP 트래픽 라우팅](#)

Vertical Pod Autoscaler를 사용하여 포드 리소스 조정

Kubernetes [Vertical Pod Autoscaler](#)는 포드에 대한 CPU 및 메모리 예약을 자동으로 조정하여 애플리케이션의 '크기를 적절히 조정'할 수 있게 지원합니다. 이러한 조정을 통해 클러스터 리소스 사용을 개선하고 다른 포드를 위한 CPU와 메모리를 확보할 수 있습니다. 이 주제에서는 Vertical Pod Autoscaler를 클러스터에 배포하고 제대로 작동하는지 확인하는 방법을 안내합니다.

- 기존 Amazon EKS 클러스터가 있습니다. 그렇지 않은 경우 [시작](#) 섹션을 참조하세요.
- Kubernetes 지표 서버가 설치되어 있습니다. 자세한 내용은 [the section called “지표 서버”](#) 단원을 참조하십시오.
- [Amazon EKS 클러스터와 통신하도록 구성된](#) kubectl 클라이언트를 사용 중입니다.
- OpenSSL 1.1.1 이상이 디바이스에 설치되어 있습니다.

Vertical Pod Autoscaler 배포

이 섹션에서는 클러스터에 Vertical Pod Autoscaler를 배포합니다.

1. 터미널 창을 열고 최신 Vertical Pod Autoscaler 소스 코드를 다운로드할 디렉터리로 이동합니다.
2. [kubernetes/autoscaler](https://github.com/kubernetes/autoscaler) GitHub 리포지토리를 복제합니다.

```
git clone https://github.com/kubernetes/autoscaler.git
```

3. 디렉터리를 vertical-pod-autoscaler로 변경합니다.

```
cd autoscaler/vertical-pod-autoscaler/
```

4. (선택사항) 다른 버전의 Vertical Pod Autoscaler를 이미 배포했다면 다음 명령을 사용하여 제거합니다.

```
./hack/vpa-down.sh
```

5. 노드에 registry.k8s.io 컨테이너 레지스트리에 대한 노드에 인터넷 액세스 권한이 없는 경우, 다음 이미지를 가져와 자체 개인 리포지토리로 푸시해야 합니다. 이미지 가져오기, 자체 개인 리포지토리로 푸시하는 방법에 대한 자세한 내용은 [the section called “리포지토리에 이미지 복사”](#) 섹션을 참조하세요.

```
registry.k8s.io/autoscaling/vpa-admission-controller:0.10.0
```

```
registry.k8s.io/autoscaling/vpa-recommender:0.10.0
registry.k8s.io/autoscaling/vpa-updater:0.10.0
```

이미지를 개인 Amazon ECR 리포지토리에 푸시하는 경우 매니페스트의 `registry.k8s.io`를 레지스트리로 바꿉니다. **111122223333**을 계정 ID로 바꿉니다. **region-code**를 클러스터가 있는 AWS 리전으로 바꿉니다. 다음 명령은 해당 리포지토리의 이름이 매니페스트의 리포지토리 이름과 같다고 가정합니다. 리포지토리의 이름을 다르게 지정하면 리포지토리도 변경해야 합니다.

```
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/admission-controller-deployment.yaml
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/recommender-deployment.yaml
sed -i.bak -e 's/registry.k8s.io/111122223333.dkr.ecr.region-code.amazonaws.com/' ./
deploy/updater-deployment.yaml
```

6. 다음 명령을 사용하여 클러스터에 Vertical Pod Autoscaler를 배포합니다.

```
./hack/vpa-up.sh
```

7. Vertical Pod Autoscaler 포드가 성공적으로 생성되었는지 확인합니다.

```
kubectl get pods -n kube-system
```

예제 출력은 다음과 같습니다.

NAME	READY	STATUS	RESTARTS	AGE
[...]				
metrics-server-8459fc497-kfj8w	1/1	Running	0	83m
vpa-admission-controller-68c748777d-ppspd	1/1	Running	0	7s
vpa-recommender-6fc8c67d85-gljpl	1/1	Running	0	8s
vpa-updater-786b96955c-bgp9d	1/1	Running	0	8s

Vertical Pod Autoscaler 설치 테스트

이 섹션에서는 샘플 애플리케이션을 배포하여 Vertical Pod Autoscaler가 작동 중인지 확인합니다.

1. 다음 명령을 사용하여 예시 `hamster.yaml` Vertical Pod Autoscaler를 배포합니다.

```
kubectl apply -f examples/hamster.yaml
```

2. hamster 예시 애플리케이션에서 포드를 가져옵니다.

```
kubectl get pods -l app=hamster
```

예제 출력은 다음과 같습니다.

```
hamster-c7d89d6db-rglf5    1/1      Running    0          48s
hamster-c7d89d6db-znvz5    1/1      Running    0          48s
```

3. 여러 포드 중 하나를 설명하여 cpu 및 memory 예약을 확인합니다. *c7d89d6db-rglf5*를 이전 단계의 출력에서 반환된 ID 중 하나로 바꿉니다.

```
kubectl describe pod hamster-c7d89d6db-rglf5
```

예제 출력은 다음과 같습니다.

```
[...]
Containers:
  hamster:
    Container ID:  docker://
e76c2413fc720ac395c33b64588c82094fc8e5d590e373d5f818f3978f577e24
    Image:          registry.k8s.io/ubuntu-slim:0.1
    Image ID:        docker-pullable://registry.k8s.io/ubuntu-
slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
    Port:           <none>
    Host Port:      <none>
    Command:
      /bin/sh
    Args:
      -c
      while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
    State:          Running
      Started:       Fri, 27 Sep 2019 10:35:16 -0700
    Ready:          True
    Restart Count:   0
    Requests:
      cpu:          100m
      memory:       50Mi
[...]
```

원래 포드에서는 100millicpu의 CPU와 50메비바이트의 메모리를 예약한 것을 알 수 있습니다. 이 예시 애플리케이션의 경우 100밀리cpu는 포드가 실행해야 하는 것보다 적으므로 CPU가 제한되어 있습니다. 또한 필요한 메모리보다 훨씬 적은 메모리를 예약합니다. Vertical Pod Autoscaler vpa-recommender 배포에서는 hamster 포드를 분석하여 CPU 및 메모리 요구 사항이 적절한지 확인합니다. 조정이 필요하다면 vpa-updater는 업데이트된 값으로 포드를 다시 시작합니다.

4. vpa-updater가 새로운 hamster 포드를 시작할 때까지 기다립니다. 1~2분 정도 걸립니다. 다음 명령을 사용하여 포드를 모니터링할 수 있습니다.

Note

새 포드가 시작되었는지 알 수 없다면 포드 이름을 이전 목록과 비교하세요. 새 포드가 시작되면 새 포드 이름이 표시됩니다.

```
kubectl get --watch Pods -l app=hamster
```

5. 새 hamster 포드가 시작되면 포드를 설명하여 업데이트된 CPU 및 메모리 예약을 확인합니다.

```
kubectl describe pod hamster-c7d89d6db-jxgfv
```

예제 출력은 다음과 같습니다.

```
[...]
Containers:
  hamster:
    Container ID:
      docker://2c3e7b6fb7ce0d8c86444334df654af6fb3fc88aad4c5d710eac3b1e7c58f7db
    Image:
      registry.k8s.io/ubuntu-slim:0.1
    Image ID:
      docker-pullable://registry.k8s.io/ubuntu-slim@sha256:b6f8c3885f5880a4f1a7cf717c07242eb4858fdd5a84b5ffe35b1cf680ea17b1
    Port:
      <none>
    Host Port:
      <none>
    Command:
      /bin/sh
    Args:
      -c
      while true; do timeout 0.5s yes >/dev/null; sleep 0.5s; done
    State:
      Running
    Started:
      Fri, 27 Sep 2019 10:37:08 -0700
```

```

Ready:      True
Restart Count: 0
Requests:
  cpu:      587m
  memory:   262144k
[...]
```

이전 출력에서, cpu 예약이 원래 값의 5배가 넘는 587밀리cpu로 증가했음을 알 수 있습니다. memory은 원래 값의 5배에 달하는 262,144킬로바이트(약 250메비바이트)로 증가했습니다. 이 포드는 리소스가 부족하므로 Vertical Pod Autoscaler는 추정치를 훨씬 더 적절한 값으로 수정하였습니다.

6. hamster-vpa 리소스를 설명하여 새로운 권장 사항을 확인하세요.

```
kubectl describe vpa/hamster-vpa
```

예제 출력은 다음과 같습니다.

```

Name:      hamster-vpa
Namespace: default
Labels:    <none>
Annotations: kubectl.kubernetes.io/last-applied-configuration:
              {"apiVersion":"autoscaling.k8s.io/v1beta2","kind":"VerticalPodAutoscaler","metadata":{"annotations":{},"name":"hamster-vpa","namespace":"d...
API Version: autoscaling.k8s.io/v1beta2
Kind:      VerticalPodAutoscaler
Metadata:
  Creation Timestamp:  2019-09-27T18:22:51Z
  Generation:         23
  Resource Version:    14411
  Self Link:           /apis/autoscaling.k8s.io/v1beta2/namespaces/default/verticalpodautoscalers/hamster-vpa
  UID:                 d0d85fb9-e153-11e9-ae53-0205785d75b0
Spec:
  Target Ref:
    API Version:  apps/v1
    Kind:         Deployment
    Name:         hamster
Status:
  Conditions:
    Last Transition Time:  2019-09-27T18:23:28Z
```

```

Status:                True
Type:                  RecommendationProvided
Recommendation:
  Container Recommendations:
    Container Name:    hamster
    Lower Bound:
      Cpu:             550m
      Memory:          262144k
    Target:
      Cpu:             587m
      Memory:          262144k
    Uncapped Target:
      Cpu:             587m
      Memory:          262144k
    Upper Bound:
      Cpu:             21147m
      Memory:          387863636
  Events:              <none>

```

7. 샘플 애플리케이션에 대한 실험이 끝나면 다음 명령으로 이를 삭제할 수 있습니다.

```
kubectl delete -f examples/hamster.yaml
```

Horizontal Pod Autoscaler를 사용하여 포드 배포 확장

Kubernetes [Horizontal Pod Autoscaler](#)는 배포, 복제 컨트롤러 또는 복제본 집합에 있는 포드의 수를 해당 리소스의 CPU 사용률에 따라 자동으로 조정합니다. 이를 통해 애플리케이션은 증가된 수요를 충족하기 위해 규모를 확장하거나 리소스가 필요 없을 때 규모를 축소할 수 있어 다른 애플리케이션을 위한 노드를 확보할 수 있습니다. 목표 CPU 사용률(백분율)을 설정하면 Horizontal Pod Autoscaler가 이 목표를 충족하기 위해 애플리케이션을 축소 또는 확장합니다.

Horizontal Pod Autoscaler는 Kubernetes의 표준 API 리소스로서, 지표 소스(예: Kubernetes 지표 서버)를 작업할 Amazon EKS 클러스터에 설치하기만 하면 됩니다. 애플리케이션 규모를 조정하기 위해 Horizontal Pod Autoscaler를 클러스터에 배포하거나 설치할 필요가 없습니다. 자세한 내용은 Kubernetes 설명서의 [Horizontal Pod Autoscaler](#)를 참조하십시오.

이 주제를 통해 Amazon EKS 클러스터용 Horizontal Pod Autoscaler를 준비하고 샘플 애플리케이션에서 작동 중인지 확인하세요.

Note

이 주제는 쿠버네티스 문서의 [HorizontalPodAutoscaler 연습](#)에 기반을 두고 있습니다.

- 기존 Amazon EKS 클러스터가 있습니다. 그렇지 않은 경우 [시작](#) 섹션을 참조하세요.
- Kubernetes 지표 서버가 설치되어 있습니다. 자세한 내용은 [the section called “지표 서버”](#) 단원을 참조하십시오.
- [Amazon EKS 클러스터와 통신하도록 구성된](#) kubectl 클라이언트를 사용 중입니다.

Horizontal Pod Autoscaler 테스트 애플리케이션 실행

이 섹션에서는 샘플 애플리케이션을 실행하여 Horizontal Pod Autoscaler가 작동 중인지 확인합니다.

Note

이 예시는 쿠버네티스 문서의 [HorizontalPodAutoscaler 연습](#)에 기반을 두고 있습니다.

1. 다음 명령을 사용하여 간단한 Apache 웹 서버 애플리케이션을 배포합니다.

```
kubectl apply -f https://k8s.io/examples/application/php-apache.yaml
```

이 Apache 웹 서버 포드에는 500 millicpu CPU 제한이 지정되며 포트 80에서 제공됩니다.

2. php-apache 배포를 위해 Horizontal Pod Autoscaler 리소스를 생성합니다.

```
kubectl autoscale deployment php-apache --cpu-percent=50 --min=1 --max=10
```

이 명령을 통해 최소 1개에서 최대 10개의 포드로 배포에 대해 50퍼센트의 CPU 사용률을 달성하려는 자동 조정기가 생성됩니다. 평균 CPU 로드가 50퍼센트 이하인 경우 오토스케일러는 배포 시 포드 수를 최소 1개로 줄이려고 합니다. 로드가 50%보다 큰 경우 자동 조정기는 배포 시 포드 수를 최대 10개로 늘리려고 합니다. 자세한 내용은 쿠버네티스 문서의 [HorizontalPodAutoscaler는 어떻게 작동하는가?](#)를 참조하세요.

3. 자동 조정기의 세부 정보를 볼 수 있는 다음 명령을 사용하여 자동 조정기를 설명하세요.

```
kubectl get hpa
```


예제 출력은 다음과 같습니다.

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
php-apache	Deployment/php-apache	0%/50%	1	10	1	51s

보시다시피 서버에 아직 로드가 없으므로 현재 CPU 로드가 0%입니다. 포드 수가 이미 가장 낮은 경계(1)에 있으므로 크기를 조정할 수 없습니다.

4. 컨테이너를 실행하여 웹 서버의 로드를 생성합니다.

```
kubectl run -i \
  --tty load-generator \
  --rm --image=busybox \
  --restart=Never \
  -- /bin/sh -c "while sleep 0.01; do wget -q -O- http://php-apache; done"
```

5. 배포 확장을 보려면 이전 단계를 실행한 터미널과는 별도의 터미널에서 다음 명령을 주기적으로 실행합니다.

```
kubectl get hpa php-apache
```

예제 출력은 다음과 같습니다.

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
php-apache	Deployment/php-apache	250%/50%	1	10	5	4m44s

복제본 수가 증가하는 데 1분 이상 걸릴 수 있습니다. 실제 CPU 백분율이 목표 백분율보다 높으면 복제본 수가 최대 10까지 증가합니다. 이 경우 해당 백분율이 250%이므로 REPLICAS 수가 계속 증가합니다.

Note

복제본 수가 최대값에 도달하기까지 몇 분 정도 걸릴 수 있습니다. 예를 들어, CPU 로드가 50% 이하로 유지되는 데 복제본이 6개만 필요한 경우 로드가 6개 복제본을 초과해서 확장되지 않습니다.

6. 로드를 중지합니다. 로드를 생성 중인 터미널 창에서 Ctrl+C 키를 누른 채 로드를 중지합니다. 축소가 관찰되는 터미널에서 다음 명령을 다시 실행하여 복제본이 1로 축소되는 것을 확인할 수 있습니다.

```
kubectl get hpa
```

예제 출력은 다음과 같습니다.

NAME	REFERENCE	TARGETS	MINPODS	MAXPODS	REPLICAS	AGE
php-apache	Deployment/php-apache	0%/50%	1	10	1	25m

Note

다시 축소하기 위한 기본 기간은 5분이므로 현재 CPU 백분율이 0%인 경우에도 복제본 수가 다시 1에 도달하기까지 시간이 걸릴 수 있습니다. 시간 프레임은 수정할 수 있습니다. 자세한 내용은 Kubernetes 설명서의 [Horizontal Pod Autoscaler](#)를 참조하십시오.

7. 샘플 애플리케이션에 대한 실험이 끝났으면 php-apache 리소스를 삭제하세요.

```
kubectl delete deployment.apps/php-apache service/php-apache
horizontalpodautoscaler.autoscaling/php-apache
```

Network Load Balancer를 사용하여 TCP 및 UDP 트래픽 라우팅

Note

신규: Amazon EKS Auto Mode는 로드 밸런싱에 대한 일상적인 작업을 자동화합니다. 자세한 내용은 다음을 참조하세요.

- [the section called “로드 밸런서 배포”](#)
- [the section called “서비스 생성”](#)

네트워크 트래픽은 OSI 모델의 L4에서 로드 밸런싱됩니다. L7에서 애플리케이션 트래픽을 로드 밸런싱하려면 Kubernetes ingress를 배포합니다. 이는 AWS Application Load Balancer를 프로비저닝합니다. 자세한 내용은 [the section called “애플리케이션 로드 밸런싱”](#) 단원을 참조하십시오. 두 가지 로드 밸런싱 간의 차이점을 자세히 알아보려면 AWS 웹 사이트에서 [Elastic 로드 밸런싱 기능](#) 부분을 참조하세요.

LoadBalancer 유형의 Kubernetes Service를 생성하면 AWS 클라우드 공급자 로드 밸런서 컨트롤러가 기본적으로 AWS [Classic Load Balancer](#)를 생성하지만 AWS [Network Load Balancer](#)도 생성할 수 있습니다. 이 컨트롤러는 향후 중요한 버그 수정만 수신하고 있습니다. AWS 클라우드 공급자 로드 밸런서 사용에 대한 자세한 내용은 쿠버네티스 문서의 [AWS 클라우드 공급자 로드 밸런서 컨트롤러](#)를 참조하세요. 이 주제에서는 이에 대해 다루지 않습니다.

AWS 클라우드 공급자 로드밸런서 컨트롤러 대신 [AWS 로드밸런서 컨트롤러](#) 버전 2.7.2 이상을 사용하는 것이 좋습니다. AWS Load Balancer Controller는 AWS Network Load Balancer를 생성하지만 AWS Classic Load Balancer는 생성하지 않습니다. 이 주제의 나머지 부분은 AWS 로드 밸런서 컨트롤러 사용에 대한 것입니다.

AWS Network Load Balancer는 Amazon EC2 IP 및 인스턴스 [대상](#), AWS Fargate IP 대상 또는 IP 대상으로 Amazon EKS Hybrid Nodes에 배포된 로 네트워크 트래픽을 로드 밸런싱할 수 있습니다. 자세한 내용은 GitHub의 [AWS Load Balancer Controller](#)를 참조하세요.

사전 조건

AWS 로드 밸런서 컨트롤러를 사용하여 네트워크 트래픽을 로드 밸런싱하려면 먼저 다음 요구 사항을 충족해야 합니다.

- 기존 클러스터를 보유해야 합니다. 기존 클러스터가 없는 경우 [시작](#) 부분을 참조하세요. 기존 클러스터의 버전을 업데이트해야 하는 경우에는 [the section called “Kubernetes 버전 업데이트”](#) 부분을 참조하세요.
- 클러스터에 배포된 AWS 로드 밸런서 컨트롤러. 자세한 내용은 [the section called “AWS 로드 밸런서 컨트롤러”](#) 단원을 참조하십시오. 버전 2.7.2 이상을 사용하는 것이 좋습니다.
- 하나 이상의 서브넷. 가용 영역에서 태그가 지정된 서브넷이 여러 개 있는 경우 컨트롤러는 서브넷 ID가 사전 순으로 가장 먼저 표시되는 서브넷을 선택합니다. 또한 서브넷에는 최소 8개의 사용 가능한 IP 주소가 있어야 합니다.
- AWS Load Balancer Controller 버전 2.1.1 이하를 사용하는 경우 서브넷에 다음과 같이 태깅되어야 합니다. 버전 2.1.2 이상을 사용하는 경우 이 태그는 선택 사항입니다. 동일한 VPC에서 여러 클러스터를 실행 중이거나 AWS 서비스가 VPC에서 서브넷을 공유하는 경우 로드 밸런서가 클러스터별로 프로비저닝되는 위치를 보다 효과적으로 제어하려면 서브넷에 태깅할 수 있습니다. 서브넷 ID를 서비스 객체에 대한 주석으로 명시적으로 지정하면 Kubernetes 및 AWS 로드 밸런서 컨트롤러는 이러한 서브넷을 직접 사용하여 로드 밸런서를 생성합니다. 로드 밸런서를 프로비저닝하는 데 이 방법을 사용하도록 선택한 경우에는 서브넷 태깅이 필요하지 않으며 다음과 같은 프라이빗 및 퍼블릭 서브넷 태깅 요구 사항을 건너뛸 수 있습니다. `<cluster-name>`을 클러스터 이름으로 바꿉니다.
- `ki-kubernetes.io/cluster/<my-cluster>`

- 값 - shared 또는 owned
- 서비스 또는 수신 객체에 대한 주석으로 서브넷 ID를 명시적으로 지정하지 않는 한 퍼블릭 및 프라이빗 서브넷은 다음 요구 사항을 충족해야 합니다. 서브넷 ID를 서비스 또는 수신 객체에 대한 주석으로 명시적으로 지정하여 로드 밸런서를 프로비저닝하면 Kubernetes 및 AWS Load Balancer Controller는 이러한 서브넷을 직접 사용하여 로드 밸런서를 생성하며 다음 태그가 필요하지 않습니다.
 - 프라이빗 서브넷(Public subnets) - 다음 형식으로 태그를 지정해야 합니다. 이렇게 하면 Kubernetes 및 AWS 로드 밸런서 컨트롤러가 서브넷을 내부 로드 밸런서에 사용할 수 있음을 알 수 있습니다. 2020년 3월 26일 이후에 eksctl 또는 Amazon EKS AWS CloudFormation 템플릿을 사용하여 VPC 생성하는 경우 서브넷은 생성될 때 적절하게 태깅됩니다. Amazon EKS AWS CloudFormation VPC 템플릿에 대한 자세한 내용은 [the section called “VPC 생성”](#)를 참조하세요.
 - 키 - kubernetes.io/role/internal-elb
 - 값-1
- 퍼블릭 서브넷(Public subnets) - 다음 형식으로 태그를 지정해야 합니다. 이렇게 하면 Kubernetes가 각 가용 영역에서 퍼블릭 서브넷을 선택하는 대신 외부 로드 밸런서에 대해 이러한 서브넷만 사용하는 것을 알 수 있습니다(서브넷 ID를 기준으로 사전순으로). 2020년 3월 26일 이후에 eksctl 또는 Amazon EKS AWS CloudFormation 템플릿을 사용하여 VPC 생성하는 경우 서브넷은 생성될 때 적절하게 태깅됩니다. Amazon EKS AWS CloudFormation VPC 템플릿에 대한 자세한 내용은 [the section called “VPC 생성”](#)를 참조하세요.
 - 키 - kubernetes.io/role/elb
 - 값-1

서브넷 역할 태그가 명시적으로 추가되지 않은 경우 Kubernetes 서비스 컨트롤러는 클러스터 VPC 서브넷의 라우팅 테이블을 검사하여 서브넷이 프라이빗 또는 퍼블릭 서브넷인지 확인합니다. 하지만 프라이빗 또는 퍼블릭 역할 태그를 명시적으로 추가하는 것을 권장합니다. AWS Load Balancer Controller는 라우팅 테이블을 검사하지 않으며 성공적인 자동 검색을 위해서는 프라이빗 및 퍼블릭 태그가 있어야 합니다.

고려 사항

- 로드 밸런서의 구성은 서비스의 매니페스트에 추가된 주석에 의해 제어됩니다. AWS 로드 밸런서 컨트롤러를 사용할 때와 AWS 클라우드 공급자 로드 밸런서 컨트롤러를 사용할 때의 서비스 주석이 다릅니다. 서비스를 배포하기 전에 AWS 로드 밸런서 컨트롤러에 대한 [주석](#)을 검토해야 합니다.

- [Kubernetes용 Amazon VPC CNI 플러그인](#)을 사용할 때 AWS Load Balancer Controller는 Amazon EC2 IP 또는 인스턴스 대상과 Fargate IP 대상으로 로드 밸런싱할 수 있습니다. [호환 가능한 대체 CNI 플러그인](#)을 사용할 때 Amazon EKS Hybrid Nodes로 로드 밸런싱을 하지 않는 한 컨트롤러는 인스턴스 대상으로만 로드 밸런싱을 수행할 수 있습니다. 하이브리드 노드의 경우 컨트롤러는 IP 대상을 로드 밸런싱할 수 있습니다. Network Load Balancer 대상 유형에 대한 자세한 내용은 Network Load Balancer 사용 설명서에서 [대상 유형](#)을 참조하세요.
- 로드 밸런서가 생성될 때 또는 그 이후에 로드 밸런서에 태그를 추가하려면 서비스 사양에 다음 주석을 추가합니다. 자세한 내용은 AWS Load Balancer Controller 설명서의 [AWS Resource Tags](#)를 참조하세요.

```
service.beta.kubernetes.io/aws-load-balancer-additional-resource-tags
```

- 다음 주석을 추가하여 Network Load Balancer에 [탄력적 IP 주소](#)를 할당할 수 있습니다. **## #**을 Elastic IP 주소의 Allocation IDs로 바꿉니다. Allocation IDs 수는 로드 밸런서에 사용되는 서브넷 수와 일치해야 합니다. 자세한 내용은 [AWS 로드 밸런서 컨트롤러](#) 설명서를 참조하세요.

```
service.beta.kubernetes.io/aws-load-balancer-eip-allocations: eipalloc-
xxxxxxxxxxxxxxxxxxxxx,eipalloc-yyyyyyyyyyyyyyyyyyy
```

- Amazon EKS는 클라이언트 트래픽에 대한 노드의 보안 그룹에 인바운드 규칙 하나를 추가하고, 생성하는 각 Network Load Balancer의 상태 확인을 위해 VPC에 각 로드 밸런서 서브넷에 대한 규칙 하나를 추가합니다. LoadBalancer 유형의 서비스 배포는 Amazon EKS가 보안 그룹에 허용된 최대 규칙 수에 대한 할당량을 초과하는 규칙을 생성하려고 하면 실패할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 Amazon VPC 할당량에 있는 [보안 그룹](#)을 참조하세요. 보안 그룹에 대한 최대 규칙 수를 초과할 가능성을 최소화하려면 다음 옵션을 고려합니다.
- 보안 그룹 할당량당 규칙 증가를 요청합니다. 자세한 내용은 Service Quotas 사용 설명서에서 [할당량 증가 요청](#)을 참조하세요.
- 인스턴스 대상이 아닌 IP 대상을 사용합니다. IP 대상을 사용하면 동일한 대상 포트에 대해 규칙을 공유할 수 있습니다. 로드 밸런서 서브넷은 주석을 사용하여 수동으로 지정할 수 있습니다. 자세한 내용은 GitHub에서 [주석](#)을 참조하세요.
- LoadBalancer 유형의 서비스 대신 수신을 사용하여 서비스에 트래픽을 전송할 수 있습니다. AWS Application Load Balancer에는 Network Load Balancer보다 적은 수의 규칙이 필요합니다. 여러 수신 간에 ALB를 공유할 수 있습니다. 자세한 내용은 [the section called “애플리케이션 로드 밸런싱”](#) 섹션을 참조하세요. 여러 서비스에서 Network Load Balancer를 공유할 수 없습니다.
- 클러스터를 여러 계정에 배포합니다.

- 포드가 Amazon EKS 클러스터의 Windows에서 실행되는 경우 로드 밸런서가 있는 단일 서비스는 최대 1,024개의 백엔드 포드를 지원할 수 있습니다. 각 포드에는 고유한 IP 주소가 있습니다.
- 새 Network Load Balancer만 AWS 로드 밸런서 컨트롤러를 사용하여 생성하는 것이 좋습니다. AWS 클라우드 공급자 로드 밸런서 컨트롤러로 생성된 기존 Network Load Balancer를 교체하려고 하면 여러 Network Load Balancer가 발생하여 애플리케이션 가동 중지를 유발할 수 있습니다.

Network Load Balancer 생성

IP 또는 인스턴스 대상을 사용하여 Network Load Balancer를 생성할 수 있습니다.

Network Load Balancer 생성-IP 대상

- Amazon EC2 노드, Fargate 또는 Amazon EKS Hybrid Nodes에 배포된 포드와 함께 IP 대상을 사용할 수 있습니다. Kubernetes 서비스는 LoadBalancer 유형으로 생성되어야 합니다. 자세한 내용은 Kubernetes 설명서에서 [LoadBalancer 유형](#)을 참조하세요.

IP 대상을 사용하는 로드 밸런서를 생성하려면 서비스 매니페스트에 다음 주석을 추가하고 서비스를 배포합니다. `aws-load-balancer-type`의 `external` 값은 AWS 클라우드 공급자 로드 밸런서 컨트롤러가 아닌 AWS 로드 밸런서 컨트롤러가 Network Load Balancer를 생성하도록 하는 원인입니다. 주석이 있는 [샘플 서비스 매니페스트](#)를 볼 수 있습니다.

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip"
```

Note

IPv6 포드에 로드 밸런싱하는 경우 다음 주석을 추가합니다. IPv6를 통해서는 인스턴스 대상이 아닌 IP 대상으로만 로드 밸런싱할 수 있습니다. 이 주석이 없으면 로드 밸런싱은 IPv4를 통해 이루어집니다.

```
service.beta.kubernetes.io/aws-load-balancer-ip-address-type: dualstack
```

Network Load Balancer는 기본적으로 `internal aws-load-balancer-scheme`으로 생성됩니다. 클러스터를 생성할 때 지정되지 않은 서브넷을 포함하여 클러스터 VPC의 모든 서브넷에서 Network Load Balancer를 시작할 수 있습니다.

Kubernetes는 서브넷의 라우팅 테이블을 검사하여 퍼블릭인지 또는 프라이빗인지 판단합니다. 퍼블릭 서브넷에는 인터넷 게이트웨이를 사용해 인터넷으로 직접 연결되는 경로가 있지만 프라이빗 서브넷은 그렇지 않습니다.

Amazon EC2 노드로 로드 밸런싱하기 위해 퍼블릭 서브넷에 Network Load Balancer를 생성하려는 경우, (Fargate는 프라이빗만 가능) 다음 주석과 함께 `internet-facing`을 지정합니다.

```
service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
```

Note

이전 버전과의 호환성을 위해 `service.beta.kubernetes.io/aws-load-balancer-type: "nlb-ip"` 주석이 여전히 지원됩니다. 그러나 `service.beta.kubernetes.io/aws-load-balancer-type: "nlb-ip"` 대신 새 로드 밸런서에 이전 주석을 사용하는 것이 좋습니다.

Important

서비스를 생성한 후에는 주석을 편집하지 마세요. 수정해야 하는 경우 서비스 객체를 삭제하고 이 주석에 대해 원하는 값으로 다시 작성하세요.

Network Load Balancer 생성-인스턴스 대상

- AWS 클라우드 공급자 로드 밸런서 컨트롤러는 인스턴스 대상만으로 Network Load Balancer를 생성합니다. AWS 로드 밸런서 컨트롤러의 버전 2.2.0 이상도 인스턴스 대상을 사용하여 Network Load Balancer를 생성합니다. AWS 클라우드 공급자 로드 밸런서 컨트롤러보다 이를 사용하여 새 Network Load Balancer를 생성하는 것이 좋습니다. Amazon EC2 노드에 배포된 포드와 함께 Network Load Balancer 인스턴스 대상을 사용할 수 있지만 Fargate에는 사용할 수 없습니다. Fargate에 배포된 포드에서 네트워크 트래픽을 로드 밸런싱하려면 IP 대상을 사용해야 합니다.

프라이빗 서브넷에 Network Load Balancer를 배포하려면 서비스 사양에 다음과 같은 주석이 있어야 합니다. 주석이 있는 [샘플 서비스 매니페스트](#)를 볼 수 있습니다. `aws-load-balancer-type`의 `external` 값은 AWS 클라우드 공급자 로드 밸런서 컨트롤러가 아닌 AWS 로드 밸런서 컨트롤러가 Network Load Balancer를 생성하도록 하는 원인입니다.

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "instance"
```

Network Load Balancer는 기본적으로 `internal` `aws-load-balancer-scheme`으로 생성됩니다. 내부 Network Load Balancer의 경우 Amazon EKS 클러스터는 VPC에서 하나 이상의 프라이빗 서브넷을 사용하도록 구성되어야 합니다. Kubernetes는 서브넷의 라우팅 테이블을 검사하여 퍼블릭인 지 또는 프라이빗인지 판단합니다. 퍼블릭 서브넷에는 인터넷 게이트웨이를 사용해 인터넷으로 직접 연결되는 경로가 있지만 프라이빗 서브넷은 그렇지 않습니다.

Amazon EC2 노드로 로드 밸런싱하기 위해 퍼블릭 서브넷에 Network Load Balancer를 생성하려는 경우 다음 주석과 함께 `internet-facing`을 지정합니다.

```
service.beta.kubernetes.io/aws-load-balancer-scheme: "internet-facing"
```

Important

서비스를 생성한 후에는 주석을 편집하지 마세요. 수정해야 하는 경우 서비스 객체를 삭제하고 이 주석에 대해 원하는 값으로 다시 작성하세요.

(선택사항) 샘플 애플리케이션을 배포합니다.

- 클러스터 VPC의 퍼블릭 또는 프라이빗 서브넷 하나 이상.
 - 클러스터에 배포된 AWS 로드 밸런서 컨트롤러. 자세한 내용은 [the section called “AWS 로드 밸런서 컨트롤러”](#) 단원을 참조하십시오. 버전 2.7.2 이상을 사용하는 것이 좋습니다.
1. Fargate에 배포하는 경우 VPC에 사용 가능한 프라이빗 서브넷이 있는지 확인하고 Fargate 프로필을 생성합니다. Fargate에 배포하지 않는 경우 이 단계를 건너뜁니다. 다음 명령을 실행하여 프로필을 생성하거나, [AWS Management Console](#)에서 명령의 name 및 namespace에 동일한 값을 사용하여 프로필을 생성할 수 있습니다. `##` `#`을 사용자의 값으로 바꿉니다.

```
eksctl create fargateprofile \
  --cluster my-cluster \
  --region region-code \
  --name nlb-sample-app \
  --namespace nlb-sample-app
```

2. 샘플 애플리케이션을 배포합니다.

- a. 애플리케이션에 대한 네임스페이스를 생성합니다.

```
kubectl create namespace nlb-sample-app
```

- b. 다음 콘텐츠를 컴퓨터에서 sample-deployment.yaml이라는 파일에 저장합니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nlb-sample-app
  namespace: nlb-sample-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx:1.23
          ports:
            - name: tcp
              containerPort: 80
```

- c. 매니페스트를 클러스터에 적용합니다.

```
kubectl apply -f sample-deployment.yaml
```

3. IP 대상에 대한 로드 밸런싱을 수행하는 인터넷이 연결된 Network Load Balancer를 사용하여 서비스를 생성합니다.

- a. 다음 콘텐츠를 컴퓨터에서 sample-service.yaml이라는 파일에 저장합니다. Fargate 노드에 배포하는 경우 service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing 줄을 제거합니다.

```
apiVersion: v1
kind: Service
metadata:
  name: nlb-sample-service
```

```

namespace: nlb-sample-app
annotations:
  service.beta.kubernetes.io/aws-load-balancer-type: external
  service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: ip
  service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
  type: LoadBalancer
  selector:
    app: nginx

```

b. 매니페스트를 클러스터에 적용합니다.

```
kubectl apply -f sample-service.yaml
```

4. 서비스가 배포되었는지 확인합니다.

```
kubectl get svc nlb-sample-service -n nlb-sample-app
```

예제 출력은 다음과 같습니다.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP
			PORT(S) AGE
sample-service	LoadBalancer	10.100.240.137	k8s-nlbsampl-nlbsampl-xxxxxxxxxx-xxxxxxxxxxxxxxxxxx.elb.region-code.amazonaws.com 80:32400/TCP 16h

Note

10.100.240.137 및 **xxxxxxxxxx-xxxxxxxxxxxxxxxxxx**의 값은 예제 출력과 다르며 (로드 밸런서마다 고유함) 클러스터가 있는 AWS 리전에 따라 **us-west-2**이 다를 수 있습니다.

5. [Amazon EC2 AWS Management Console](#)을 엽니다. 왼쪽 탐색 창의 Load Balancing 아래에서 대상 그룹을 선택합니다. 이름 열에서 로드 밸런서 열의 값이 이전 단계에서 출력의 EXTERNAL-IP 열에 있는 이름의 일부와 일치하는 대상 그룹의 이름을 선택합니다. 예를 들어, 출력이 이전 출력과 동일한 경우 k8s-default-samplese-**xxxxxxxxxx** 라는 대상 그룹을 선택합니다. 샘플 서비스 매니페스트에 지정되었기 때문에 대상 유형이 IP입니다.

6. 대상 그룹을 선택한 다음 대상 탭을 선택합니다. 등록된 대상에서 이전 단계에서 배포한 3개 복제본의 IP 주소 3개가 표시되어야 합니다. 계속하기 전에 모든 대상의 상태가 정상일 때까지 기다립니다. 모든 대상이 healthy가 될 때까지 몇 분 정도 걸릴 수 있습니다. 대상은 healthy 상태로 변경하기 전에 unhealthy 상태일 수 있습니다.
7. `xxxxxxxxxx-xxxxxxxxxxxxxxxxxx` 및 `us-west-2`을 EXTERNAL-IP에 대한 [이전 단계](#)의 출력에서 반환된 값으로 교체하여 서비스로 트래픽을 보냅니다. 프라이빗 서브넷에 배포한 경우 Bastion Host와 같은 VPC 내의 디바이스에서 페이지를 확인해야 합니다. 자세한 내용은 [AWS 기반 Linux Bastion 호스트](#)를 참조하세요.

```
curl k8s-default-samplese-xxxxxxxxxx-xxxxxxxxxxxxxxxxxx.elb.region-code.amazonaws.com
```

예제 출력은 다음과 같습니다.

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
[...]
```

8. 샘플 배포, 서비스 및 네임스페이스가 완료되면 제거합니다.

```
kubectl delete namespace nlb-sample-app
```

Application Load Balancer를 사용하여 애플리케이션 및 HTTP 트래픽 라우팅

Note

신규: Amazon EKS Auto Mode는 로드 밸런싱에 대한 일상적인 작업을 자동화합니다. 자세한 내용은 다음을 참조하세요.

- [the section called “로드 밸런서 배포”](#)
- [the section called “수신 클래스 생성”](#)

Kubernetes ingress를 생성할 때 애플리케이션 트래픽을 로드 밸런싱하는 AWS Application Load Balancer(ALB)가 프로비저닝됩니다. 자세한 내용은 Application Load Balancer 사용 설명서의 [Application Load Balancer란 무엇입니까?](#) 및 Kubernetes 설명서의 [Ingress](#)를 참조하세요. ALB는 노드 또는 AWS Fargate에 배포되는 포드와 함께 사용됩니다. ALB를 퍼블릭 또는 프라이빗 서브넷에 배포할 수 있습니다.

애플리케이션 트래픽은 OSI 모델의 L7에서 로드 밸런싱됩니다. L4에서 네트워크 트래픽을 로드 밸런싱하려면 LoadBalancer 유형의 Kubernetes service를 배포합니다. 이 유형은 AWS Network Load Balancer를 프로비저닝 합니다. 자세한 내용은 [the section called “네트워크 로드 밸런싱”](#) 섹션을 참조하세요. 두 가지 로드 밸런싱 간의 차이점을 자세히 알아보려면 AWS 웹 사이트에서 [Elastic 로드 밸런싱 기능](#) 부분을 참조하세요.

사전 조건

애플리케이션 트래픽을 애플리케이션으로 로드 밸런싱 하기 전 다음 요구 사항을 충족해야 합니다.

- 기존 클러스터를 보유해야 합니다. 기존 클러스터가 없는 경우 [시작](#) 부분을 참조하세요. 기존 클러스터의 버전을 업데이트해야 하는 경우에는 [the section called “Kubernetes 버전 업데이트”](#) 부분을 참조하세요.
- 클러스터에 배포된 AWS 로드 밸런서 컨트롤러. 자세한 내용은 [the section called “AWS 로드 밸런서 컨트롤러”](#) 섹션을 참조하세요. 버전 2.7.2 이상을 사용하는 것이 좋습니다.
- 서로 다른 가용 영역에 두 개 이상의 서브넷을 보유해야 합니다. AWS Load Balancer Controller는 각 가용 영역에서 서브넷을 하나씩 선택합니다. 가용 영역에서 태그가 지정된 서브넷이 여러 개 있는 경우 컨트롤러는 서브넷 ID가 사전 순으로 가장 먼저 표시되는 서브넷을 선택합니다. 각 서브넷에는 최소 8개의 사용 가능한 IP 주소가 있어야 합니다.

작업자 노드에 연결된 보안 그룹을 여러 개 사용하는 경우 정확히 하나의 보안 그룹에 다음과 같이 태그를 지정해야 합니다. `<cluster-name>`을 클러스터 이름으로 바꿉니다.

- `키-kubernetes.io/cluster/<my-cluster>`
- 값-shared 또는 owned
- AWS Load Balancer Controller 버전 2.1.1 이하를 사용하는 경우 서브넷에 다음과 같은 형식으로 태그를 지정해야 합니다. 버전 2.1.2 이상을 사용하는 경우 태깅은 선택 사항입니다. 그러나 다음 중 하나라도 해당되는 경우 서브넷에 태그를 지정하는 것이 좋습니다. 동일한 VPC에서 실행 중인 클러스터가 여러 개 있거나 VPC에서 서브넷을 공유하는 AWS 서비스가 여러 개 있습니다. 또는 각 클러스터에 대해 로드 밸런서가 프로비저닝되는 위치에 대한 더 정밀한 제어가 필요합니다. `<cluster-name>`을 클러스터 이름으로 바꿉니다.
- `키-kubernetes.io/cluster/<my-cluster>`

- 값-shared 또는 owned
- 퍼블릭 및 프라이빗 서브넷은 다음 요구 사항을 충족해야 합니다. 서비스 또는 수신 개체에서 서브넷 ID를 주석으로 명시적으로 지정하지 않는 한 마찬가지입니다. 서브넷 ID를 서비스 또는 수신 개체에 주석으로 명시적으로 지정하여 로드 밸런서를 프로비저닝한다고 가정합니다. 이 상황에서 Kubernetes 및 AWS Load Balancer Controller는 이러한 서브넷을 직접 사용하여 로드 밸런서를 생성하며 다음 태그는 필요하지 않습니다.
 - 프라이빗 서브넷(Private subnets) - 다음 형식으로 태그를 지정해야 합니다. 이렇게 하면 Kubernetes 및 AWS 로드 밸런서 컨트롤러가 서브넷을 내부 로드 밸런서에 사용할 수 있음을 알 수 있습니다. 2020년 3월 26일 이후에 eksctl 또는 Amazon EKS AWS CloudFormation 템플릿을 통해 VPC를 생성하는 경우 서브넷은 적절하게 태깅됩니다. Amazon EKS AWS CloudFormation VPC 템플릿에 대한 자세한 내용은 [the section called “VPC 생성”](#)를 참조하세요.
 - 키 - kubernetes.io/role/internal-elb
 - 값-1
- 퍼블릭 서브넷(Public subnets) - 다음 형식으로 태그를 지정해야 합니다. 이렇게 하면 Kubernetes에서 외부 로드 밸런서에 지정된 서브넷만 사용할 수 있음을 알 수 있습니다. 따라서 Kubernetes는 각 가용 영역에서 퍼블릭 서브넷을 선택하지 않습니다(서브넷 ID를 기준으로 사전 순으로). 2020년 3월 26일 이후에 eksctl 또는 Amazon EKS AWS CloudFormation 템플릿을 통해 VPC를 생성하는 경우 서브넷은 적절하게 태깅됩니다. Amazon EKS AWS CloudFormation VPC 템플릿에 대한 자세한 내용은 [the section called “VPC 생성”](#)를 참조하세요.
 - 키 - kubernetes.io/role/elb
 - 값-1

서브넷 역할 태그가 명시적으로 추가되지 않은 경우 Kubernetes 서비스 컨트롤러는 클러스터 VPC 서브넷의 라우팅 테이블을 검사합니다. 이는 서브넷이 프라이빗 또는 퍼블릭 인지를 확인하기 위함입니다. 이 동작에는 의존하지 않는 것이 좋습니다. 대신 프라이빗 또는 퍼블릭 역할 태그를 명시적으로 추가합니다. AWS Load Balancer Controller는 라우팅 테이블을 검사하지 않습니다. 또한 자동 검색에 성공하려면 프라이빗 및 퍼블릭 태그가 있어야 합니다.

- [AWS 로드 밸런서 컨트롤러](#)는 kubernetes.io/ingress.class: alb 주석을 사용하여 클러스터에 Kubernetes 수신 리소스가 생성될 때마다 ALB 및 필요한 지원 AWS 리소스를 생성합니다. 수신 리소스는 ALB를 구성하여 HTTP 또는 HTTPS 트래픽을 클러스터 내 다른 포드로 라우팅합니다. 수신 객체에서 AWS 로드 밸런서 컨트롤러를 사용하는지 확인하려면, Kubernetes 수신 사양에 다음 주석을 추가합니다. 자세한 내용은 GitHub의 [수신 사양](#)을 참조하세요.

```
annotations:
  kubernetes.io/ingress.class: alb
```

Note

IPv6 포드로 로드 밸런싱하는 경우 수신 사양에 다음 주석을 추가합니다. IPv6를 통해서만 인스턴스 대상이 아닌 IP 대상으로만 로드 밸런싱할 수 있습니다. 이 주석이 없으면 로드 밸런싱은 IPv4를 통해 이루어집니다.

```
alb.ingress.kubernetes.io/ip-address-type: dualstack
```

- AWS 로드 밸런서 컨트롤러는 다음 트래픽 모드를 지원합니다.
 - 인스턴스 - 클러스터 내의 노드를 ALB의 대상으로 등록합니다. ALB에 도달하는 트래픽은 서비스를 위해 NodePort로 라우팅된 다음 포드로 프록시됩니다. 이것이 기본 트래픽 모드입니다. `alb.ingress.kubernetes.io/target-type: instance` 주석을 사용하여 명시적으로 지정할 수도 있습니다.

Note

Kubernetes 서비스에서 이 트래픽 모드를 사용할 NodePort 또는 LoadBalancer 유형을 지정해야 합니다.

- IP - 포드를 ALB의 대상으로 등록합니다. ALB에 도달하는 트래픽은 서비스를 위해 포드로 직접 라우팅됩니다. 이 트래픽 모드를 사용하려면 `alb.ingress.kubernetes.io/target-type: ip` 주석을 지정해야 합니다. Fargate 또는 Amazon EKS Hybrid Nodes에서 대상 포드가 실행 중인 경우 IP 대상 유형이 필요합니다.
- 컨트롤러에 의해 생성된 ALB에 태그를 지정하려면 컨트롤러에 다음 주석을 추가합니다. `alb.ingress.kubernetes.io/tags`. AWS 로드 밸런서 컨트롤러가 지원하는 모든 사용 가능한 주석의 목록은 GitHub에서 [Ingress 주석](#)을 참조하세요.
- ALB 컨트롤러 버전을 업그레이드하거나 다운그레이드하면 해당 버전에 의존하는 기능에 새로운 변경 사항이 발생할 수 있습니다. 각 릴리스에서 발생하는 새로운 변경 사항에 대한 자세한 내용은 GitHub의 [ALB 컨트롤러 릴리스 정보](#)를 참조하세요.

수신 그룹에서 ALBs 재사용

IngressGroups를 이용하여 여러 서비스 리소스 간에 Application Load Balancer를 공유합니다.

그룹 수신에 가입하려면 Kubernetes 수신 리소스 사양에 다음 주석을 추가합니다.

```
alb.ingress.kubernetes.io/group.name: my-group
```

그룹 이름은 다음과 같아야 합니다.

- 63자 이하의 길이어야 합니다.
- 소문자, 숫자, - 및 .으로 구성되어야 합니다.
- 문자나 숫자로 시작하여 끝나야 합니다.

컨트롤러는 동일한 그룹에 있는 모든 인그레스에 대한 규칙을 자동으로 병합합니다. 단일 ALB로 지원 합니다. 인그레스에 정의된 대부분의 주석은 해당 인그레스에 의해 정의된 경로에만 적용됩니다. 기본적으로 인그레스 리소스는 인그레스 그룹에 속하지 않습니다.

Warning

잠재적 보안 위험

수신 리소스를 만들거나 수정할 수 있는 RBAC 권한이 있는 모든 Kubernetes 사용자가 동일한 신뢰 경계 내에 있는 경우에만 수신에 대한 수신 그룹을 지정합니다. 그룹 이름을 사용하여 주석을 추가하는 경우 다른 Kubernetes 사용자가 동일한 수신 그룹에 속하도록 해당 수신을 생성하거나 수정할 수 있습니다. 이렇게 하면 기존 규칙을 우선 순위가 높은 규칙으로 덮어쓰는 것과 같이 바람직하지 않은 동작이 발생할 수 있습니다.

수신 리소스의 처리 순서를 추가할 수 있습니다.

```
alb.ingress.kubernetes.io/group.order: '10'
```

숫자는 1~1,000 사이여야 합니다. 동일한 인그레스 그룹의 모든 인그레스 중 가장 낮은 수가 먼저 평가 됩니다. 이 주석이 없는 모든 인그레스는 값이 0인 것으로 평가됩니다. 숫자가 높은 중복 규칙은 낮은 수의 규칙을 덮어쓸 수 있습니다. 기본적으로 동일한 인그레스 그룹 내의 규칙 순서는 네임스페이스 및 이름을 기준으로 사전 순으로 결정됩니다.

Important

동일한 인그레스 그룹의 각 인그레스에 고유한 우선 순위 번호가 있는지 확인합니다. 인그레스에서는 중복 주문 번호를 사용할 수 없습니다.

(선택사항) 샘플 애플리케이션을 배포합니다.

- 클러스터 VPC의 퍼블릭 또는 프라이빗 서브넷 하나 이상.
- 클러스터에 배포된 AWS 로드 밸런서 컨트롤러. 자세한 내용은 [the section called “AWS 로드 밸런서 컨트롤러”](#) 섹션을 참조하세요. 버전 2.7.2 이상을 사용하는 것이 좋습니다.

Amazon EC2 노드가 있거나 Fargate 포드가 있거나 둘 다 있는 클러스터에서 샘플 애플리케이션을 실행할 수 있습니다.

1. Fargate에 배포하지 않는 경우 이 단계를 건너뜁니다. Fargate에 배포하는 경우 Fargate 프로필을 생성합니다. 다음 명령을 실행하여 프로필을 생성하거나, [AWS Management Console](#)에서 명령의 name 및 namespace에 동일한 값을 사용하여 프로필을 생성할 수 있습니다. ## #을 사용자의 값으로 바꿉니다.

```
eksctl create fargateprofile \
  --cluster my-cluster \
  --region region-code \
  --name alb-sample-app \
  --namespace game-2048
```

2. 게임 [2048](#)을 샘플 애플리케이션으로 배포하여 AWS 로드 밸런서 컨트롤러가 수신 객체의 결과로 AWS ALB를 생성하는지 확인합니다. 배포하려는 서브넷 유형에 대한 단계를 완료합니다.
 - a. IPv6 패밀리로 생성한 클러스터의 포드에 배포하는 경우 다음 단계로 건너뜁니다.

- 퍼블릭::

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/examples/2048/2048_full.yaml
```

- 프라이빗::

A. 매니페스트를 다운로드합니다.

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/examples/2048/2048_full.yaml
```

B. 파일을 편집하고 alb.ingress.kubernetes.io/scheme: internet-facing 줄을 검색합니다.

C. ### ##을 internal로 변경하고 파일을 저장합니다.

D. 매니페스트를 클러스터에 적용합니다.

```
kubectl apply -f 2048_full.yaml
```

b. [IPv6 패밀리](#)로 생성한 클러스터의 포드에 배포하는 경우 다음 단계를 수행합니다.

i. 매니페스트를 다운로드합니다.

```
curl -O https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/examples/2048/2048_full.yaml
```

ii. 편집기에서 파일을 열고 인그레스 사양의 주석에 다음 줄을 추가합니다.

```
alb.ingress.kubernetes.io/ip-address-type: dualstack
```

iii. 인터넷 연결 포드가 아니라 내부 포드로 로드 밸런싱하는 경우

```
alb.ingress.kubernetes.io/scheme: internet-facing 줄을
```

```
alb.ingress.kubernetes.io/scheme: internal로 변경합니다.
```

iv. 파일을 저장합니다.

v. 매니페스트를 클러스터에 적용합니다.

```
kubectl apply -f 2048_full.yaml
```

3. 몇 분 후, 인그레스 리소스가 다음 명령으로 생성되었는지 확인합니다.

```
kubectl get ingress/ingress-2048 -n game-2048
```

예제 출력은 다음과 같습니다.

NAME	CLASS	HOSTS	ADDRESS
		PORTS	AGE
ingress-2048	<none>	*	k8s-game2048-ingress2-xxxxxxxxxx-yyyyyyyyyy.region-
code.elb.amazonaws.com		80	2m32s

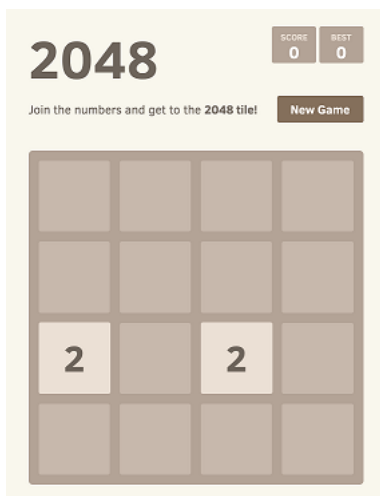
 Note

프라이빗 서브넷에서 로드 밸런서를 생성한 경우 이전 출력에서 ADDRESS 아래의 값은 `internal-`로 시작합니다.

몇 분 후에도 수신이 생성되지 않은 경우 다음 명령을 실행하여 AWS Load Balancer Controller 로그를 확인합니다. 이러한 로그에는 배포 문제를 진단하는 데 사용할 수 있는 오류 메시지가 포함될 수 있습니다.

```
kubectl logs -f -n kube-system -l app.kubernetes.io/instance=aws-load-balancer-controller
```

1. 퍼블릭 서브넷에 배포한 경우 브라우저를 열고 이전 명령 출력에서 ADDRESS URL로 이동하여 샘플 애플리케이션을 봅니다. 아무것도 표시되지 않으면 새로고침 한 후 다시 시도하세요. 프라이빗 서브넷에 배포한 경우 Bastion Host와 같은 VPC 내의 디바이스에서 페이지를 확인해야 합니다. 자세한 내용은 [AWS 기반 Linux Bastion 호스트](#)를 참조하세요.



2. 샘플 애플리케이션에 대한 실험이 끝나면 다음 명령 중 하나를 실행하여 이를 삭제합니다.
 - 매니페스트를 적용했다면 다운로드한 사본을 적용하는 대신 다음 명령을 사용합니다.

```
kubectl delete -f https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.11.0/docs/examples/2048/2048_full.yaml
```

- 매니페스트를 다운로드하여 편집한 경우 다음 명령을 사용합니다.

```
kubectl delete -f 2048_full.yaml
```

서비스에 할당할 수 있는 외부 IP 주소 제한

Kubernetes 서비스는 클러스터 내부에서 다음을 통해 연결할 수 있습니다.

- Kubernetes가 자동으로 할당하는 클러스터 IP 주소

- 서비스 사양에서 externalIPs 속성에 대해 지정하는 모든 IP 주소. 외부 IP 주소는 Kubernetes에서 관리하지 않으며 클러스터 관리자의 책임입니다. externalIPs로 지정된 외부 IP 주소는 클라우드 공급자가 지정하여 LoadBalancer 서비스 유형에 할당되는 외부 IP 주소와 다릅니다.

Kubernetes 서비스에 대한 자세한 내용은 Kubernetes 문서의 [서비스](#)를 참조하세요. 서비스 사양에서 externalIPs에 대해 지정할 수 있는 IP 주소를 제한할 수 있습니다.

1. cert-manager를 배포하여 Webhook 인증서를 관리합니다. 자세한 내용은 [cert-manager](#) 문서를 참조하세요.

```
kubectl apply -f https://github.com/jetstack/cert-manager/releases/download/v1.5.4/cert-manager.yaml
```

2. cert-manager 포드가 실행 중인지 확인하세요.

```
kubectl get pods -n cert-manager
```

예제 출력은 다음과 같습니다.

NAME	READY	STATUS	RESTARTS	AGE
cert-manager-58c8844bb8-nlx7q	1/1	Running	0	15s
cert-manager-cainjector-745768f6ff-696h5	1/1	Running	0	15s
cert-manager-webhook-67cc76975b-4v4nk	1/1	Running	0	14s

3. 기존 서비스를 검토하여 주소를 제한하려는 CIDR 블록 내에 포함되지 않은 외부 IP 주소가 할당되지 않았는지 확인합니다.

```
kubectl get services -A
```

예제 출력은 다음과 같습니다.

NAMESPACE	EXTERNAL-IP	NAME	PORT(S)	AGE	TYPE
cert-manager		cert-manager	9402/TCP	20m	ClusterIP
cert-manager		cert-manager-webhook	443/TCP	20m	ClusterIP
default		kubernetes	443/TCP	2d1h	ClusterIP

externalip-validation-system	externalip-validation-webhook-service	ClusterIP
10.100.234.179 <none>	443/TCP 16s	
kube-system	kube-dns	ClusterIP
10.100.0.10 <none>	53/UDP,53/TCP 2d1h	
my-namespace	my-service	ClusterIP
10.100.128.10 192.168.1.1	80/TCP 149m	

값이 액세스를 제한하려는 블록 내에 있지 않은 IP 주소인 경우 블록 내에 포함되도록 주소를 변경하고 서비스를 다시 배포해야 합니다. 예를 들어, 이전 출력의 my-service 서비스에는 5단계에서 CIDR 블록 예제 내에 없는 외부 IP 주소가 할당되어 있습니다.

4. 외부 IP Webhook 매니페스트를 다운로드합니다. 또한 GitHub에서 [Webhook의 소스 코드](#)를 볼 수도 있습니다.

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/docs/externalip-webhook.yaml
```

5. CIDR 블록을 지정합니다. 편집기에서 다운로드한 파일을 열고 다음 행의 시작 부분에 있는 \#을 제거합니다.

```
#args:
#- --allowed-external-ip-cidrs=10.0.0.0/8
```

10.0.0.0/8을 실제 CIDR 블록으로 바꿉니다. 원하는 만큼 블록을 지정할 수 있습니다. 여러 블록을 지정하는 경우 블록 사이에 쉼표를 추가합니다.

6. 클러스터가 us-west-2 AWS 리전에 없는 경우 파일에서 us-west-2, 602401143452 및 amazonaws.com를 다음 명령으로 바꿉니다. 명령을 실행하기 전에 **region-code** 및 **111122223333**를 [Amazon EKS 추가 기능용 Amazon 컨테이너 이미지 레지스트리 보기](#)의 목록에서 AWS 리전 값으로 바꿉니다.

```
sed -i.bak -e 's|602401143452|111122223333|' externalip-webhook.yaml
sed -i.bak -e 's|us-west-2|region-code|' externalip-webhook.yaml
sed -i.bak -e 's|amazonaws.com||' externalip-webhook.yaml
```

7. 매니페스트를 클러스터에 적용합니다.

```
kubectl apply -f externalip-webhook.yaml
```

CIDR 블록 지정 단계에서 지정한 블록에 포함되지 않은 externalIPs의 지정된 IP 주소를 통해 클러스터에 서비스를 배포하려고 시도할 경우 실패합니다.

한 리포지토리에서 다른 리포지토리로 컨테이너 이미지 복사

이 항목에서는 노드가 액세스할 수 없는 리포지토리에서 컨테이너 이미지를 가져와 노드가 액세스할 수 있는 리포지토리에 그 이미지를 푸시하는 방법에 대해 설명합니다. 이미지를 Amazon ECR 또는 노드가 액세스할 수 있는 대체 리포지토리로 푸시할 수 있습니다.

- 컴퓨터에 설치 및 구성된 도커 엔진. 자세한 내용은 도커 설명서의 [도커 엔진 설치](#) 단원을 참조하세요.
- 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.
- 노드가 Amazon 네트워크를 통해 프라이빗 Amazon ECR 리포지토리에서 컨테이너 이미지를 가져오거나 프라이빗 Amazon ECR 리포지토리에 컨테이너 이미지를 푸시하도록 하려는 경우 사용하는 Amazon ECR용 인터페이스 VPC 엔드포인트입니다. 자세한 내용은 Amazon Elastic Container Registry 사용 설명서의 [Amazon ECR 인터페이스 VPC 엔드포인트 생성](#) 섹션을 참조하세요.

다음 단계를 수행하여 리포지토리에서 컨테이너 이미지를 가져와 자체 리포지토리로 푸시합니다. 이 항목에서 제공하는 다음 예제에서는 [Kubernetes 지표 도우미용 Amazon VPC CNI 플러그인 이미지](#)를 가져옵니다. 다음 단계를 수행할 때 `## #`을 자체 값으로 교체합니다.

1. Amazon ECR 리포지토리나 기타 리포지토리가 아직 없다면 노드가 액세스할 수 있는 리포지토리를 생성합니다. 다음 명령은 Amazon ECR 프라이빗 리포지토리를 생성합니다. Amazon ECR 프라이빗 리포지토리 이름은 문자로 시작해야 합니다. 소문자, 숫자, 하이픈(-), 밑줄(_), 슬래시(/)만 포함할 수 있습니다. 자세한 내용은 Amazon Elastic Container Registry 사용 설명서의 [프라이빗 리포지토리 생성](#) 섹션을 참조하세요.

`cni-metrics-helper`를 선택한 항목으로 바꿀 수 있습니다. 가장 좋은 방법은 각 이미지에 대해 별도의 리포지토리를 생성하는 것입니다. 리포지토리 내에서 이미지 태그가 고유해야 하므로 이 옵션을 사용하는 것이 좋습니다. `region-code`를 [Amazon ECR가 지원하는 AWS 리전](#)으로 바꿉니다.

```
aws ecr create-repository --region region-code --repository-name cni-metrics-helper
```

2. 노드가 가져와야 하는 이미지의 레지스트리, 리포지토리 및 태그(선택사항)를 결정합니다. 이 정보는 `registry/repository[:tag]` 형식을 갖습니다.

이미지 설치에 대한 Amazon EKS 주제 중 다수는 매니페스트 파일을 적용하거나 Helm 차트를 사용하여 이미지를 설치해야 합니다. 그러나 매니페스트 파일을 적용하거나 Helm 차트를 설치하기 전에 먼저 매니페스트 또는 차트의 `values.yaml` 파일 내용을 검토하세요. 그렇게 하면 가져올 레지스트리, 리포지토리 및 태그를 결정할 수 있습니다.

예를 들어, [Kubernetes 지표 도우미용 Amazon VPC CNI 플러그인](#)의 [매니페스트 파일](#)에서 다음 줄을 찾을 수 있습니다. 레지스트리는 `602401143452.dkr.ecr.us-west-2.amazonaws.com`이며, 이는 Amazon ECR 프라이빗 레지스트리입니다. 리포지토리는 `cni-metrics-helper`입니다.

```
image: "602401143452.dkr.ecr.us-west-2.amazonaws.com/cni-metrics-helper:v1.12.6"
```

이미지 위치에 대해 다음과 같은 변형이 나타날 수 있습니다.

- `repository-name:tag`만 해당. 이 경우, `docker.io`는 일반적으로 레지스트리이지만, 레지스트리가 지정되지 않은 경우 Kubernetes가 기본적으로 리포지토리 이름 앞에 추가하기 때문에 지정되지 않는다.
- `repository-name/repository-namespace/repository:tag`. 리포지토리 네임스페이스는 선택 사항이지만 이미지를 범주화하기 위해 리포지토리 소유자가 지정하는 경우도 있습니다. 예를 들어, 모든 [Amazon ECR Public Gallery의 Amazon EC2 이미지](#)는 `aws-ec2` 네임스페이스를 사용합니다.

Helm을 사용하여 이미지를 설치하기 전에 Helm `values.yaml` 파일을 검토하여 이미지 위치를 결정합니다. 예를 들어, [Kubernetes 지표 도우미용 Amazon VPC CNI 플러그인](#)의 [values.yaml](#) 파일에는 다음 줄이 포함됩니다.

```
image:
  region: us-west-2
  tag: v1.12.6
  account: "602401143452"
  domain: "amazonaws.com"
```

3. 매니페스트 파일에 지정된 컨테이너 이미지를 가져옵니다.

- a. [Amazon ECR Public Gallery](#)와 같은 퍼블릭 레지스트리에서 가져오는 경우에는 인증이 필요하지 않으므로 다음 단계로 넘어가도 됩니다. 이 예에서는 CNI 지표 헬퍼 이미지에 대한 리포지토리를 포함하는 Amazon ECR 프라이빗 레지스트리에 대해 인증합니다. Amazon EKS는 [Amazon EKS 추가 기능에 대한 Amazon 컨테이너 이미지 레지스트리 보기](#)에 나열된 각 레지스트리에 이미지를 유지합니다. `602401143452` 및 `region-code`를 다른 레지스트리에 대한 정보로 바꿔 모든 레지스트리에 대해 인증할 수 있습니다. 각각의 [Amazon EKS가 지원되는 AWS 리전](#)에 대해 별도의 레지스트리가 있습니다.

```
aws ecr get-login-password --region region-code | docker login --username AWS --password-stdin 602401143452.dkr.ecr.region-code.amazonaws.com
```

- b. 이미지를 가져옵니다. 이 예제에서는 이전 하위 단계에서 인증한 레지스트리에서 가져옵니다. `602401143452` 및 `region-code`를 이전 하위 단계에서 제공한 정보로 바꿉니다.

```
docker pull 602401143452.dkr.ecr.region-code.amazonaws.com/cni-metrics-helper:v1.12.6
```

4. 레지스트리, 리포지토리 및 태그로 가져온 이미지에 태그를 지정합니다. 다음 예제에서는 매니페스트 파일에서 이미지를 가져와서 첫 번째 단계에서 생성한 Amazon ECR 프라이빗 리포지토리로 푸시한다고 가정합니다. `111122223333`을 계정 ID로 바꿉니다. `region-code`를 Amazon ECR 프라이빗 리포지토리를 생성한 AWS 리전으로 바꿉니다.

```
docker tag cni-metrics-helper:v1.12.6 111122223333.dkr.ecr.region-code.amazonaws.com/cni-metrics-helper:v1.12.6
```

5. 레지스트리를 인증합니다. 이 예제에서는 첫 번째 단계에서 생성한 Amazon ECR 프라이빗 레지스트리에 대해 인증합니다. 자세한 내용은 Amazon Elastic Container Registry 사용 설명서의 [레지스트리 권한](#)을 참조하세요.

```
aws ecr get-login-password --region region-code | docker login --username AWS --password-stdin 111122223333.dkr.ecr.region-code.amazonaws.com
```

6. 리포지토리에 이미지를 푸시합니다. 이 예제에서는 첫 번째 단계에서 생성한 Amazon ECR 프라이빗 레지스트리에 이미지를 푸시합니다. 자세한 내용은 Amazon Elastic 컨테이너 레지스트리 사용 설명서의 [도커 이미지 푸시하기](#) 섹션을 참조하세요.

```
docker push 111122223333.dkr.ecr.region-code.amazonaws.com/cni-metrics-helper:v1.12.6
```

7. 푸시한 이미지에 대해 이전 단계에서 이미지를 결정하는데 `registry/repository:tag`와 함께 사용된 매니페스트 파일을 업데이트합니다. Helm 차트를 사용하여 설치하는 경우 종종 `registry/repository:tag`를 지정하는 옵션이 있습니다. 차트를 설치할 때 리포지토리에 푸시한 이미지에 대한 `registry/repository:tag`를 지정합니다.

Amazon EKS 추가 기능에 대한 Amazon 컨테이너 이미지 레지스트리 보기

[AWS Amazon EKS 추가 기능](#)을 클러스터에 배포하면 노드는 추가 기능의 설치 메커니즘에 지정된 레지스트리에서 설치 매니페스트나 Helm `values.yaml` 파일과 같은 필수 컨테이너 이미지를 가져옵니다. Amazon EKS Amazon ECR 프라이빗 리포지토리에 이미지를 생성합니다. Amazon EKS는 각 Amazon EKS가 지원하는 AWS 리전 리포지토리에 이미지를 복제합니다. 노드는 다음 레지스트리 중에서 컨테이너 이미지를 인터넷을 통해 가져올 수 있습니다. 또는 VPC에서 [Amazon ECR에 대한 인터페이스 VPC 엔드포인트\(AWS 프라이빗 링크\)](#)를 생성한 경우, 노드는 Amazon 네트워크를 통해 이미지를 가져올 수도 있습니다. 레지스트리는 AWS IAM 계정을 통한 인증이 필요합니다. 노드는 관련된 [AmazonEC2ContainerRegistryReadOnly](#) 관리형 IAM 정책의 권한을 갖는 [Amazon EKS 노드 IAM 역할](#)을 이용하여 인증합니다.

AWS 리전	레지스트리
af-south-1	877085696533.dkr.ecr.af-south-1.amazonaws.com
ap-east-1	800184023465.dkr.ecr.ap-east-1.amazonaws.com
ap-east-2	533267051163.dkr.ecr.ap-east-1.amazonaws.com
ap-southeast-3	296578399912.dkr.ecr.ap-southeast-3.amazonaws.com
ap-south-2	900889452093.dkr.ecr.ap-south-2.amazonaws.com
ap-southeast-4	491585149902.dkr.ecr.ap-southeast-4.amazonaws.com

AWS 리전	레지스트리
ap-south-1	602401143452.dkr.ecr.ap-south-1.amazonaws.com
ap-northeast-3	602401143452.dkr.ecr.ap-northeast-3.amazonaws.com
ap-northeast-2	602401143452.dkr.ecr.ap-northeast-2.amazonaws.com
ap-southeast-1	602401143452.dkr.ecr.ap-southeast-1.amazonaws.com/
ap-southeast-2	602401143452.dkr.ecr.ap-southeast-2.amazonaws.com
ap-southeast-7	121268973566.dkr.ecr.ap-southeast-7.amazonaws.com
ap-northeast-1	602401143452.dkr.ecr.ap-northeast-1.amazonaws.com
cn-north-1	918309763551.dkr.ecr.cn-north-1.amazonaws.com.cn
cn-northwest-1	961992271922.dkr.ecr.cn-northwest-1.amazonaws.com.cn
eu-central-1	602401143452.dkr.ecr.eu-central-1.amazonaws.com
eu-west-1	602401143452.dkr.ecr.eu-west-1.amazonaws.com
eu-west-2	602401143452.dkr.ecr.eu-west-2.amazonaws.com
eu-south-1	590381155156.dkr.ecr.eu-south-1.amazonaws.com

AWS 리전	레지스트리
eu-west-3	602401143452.dkr.ecr.eu-west-3.amazonaws.com
eu-south-2	455263428931.dkr.ecr.eu-south-2.amazonaws.com
eu-north-1	602401143452.dkr.ecr.eu-north-1.amazonaws.com
eu-central-2	900612956339.dkr.ecr.eu-central-2.amazonaws.com
il-central-1	066635153087.dkr.ecr.il-central-1.amazonaws.com
mx-central-1	730335286997.dkr.ecr.mx-central-1.amazonaws.com
me-south-1	558608220178.dkr.ecr.me-south-1.amazonaws.com
me-central-1	759879836304.dkr.ecr.me-central-1.amazonaws.com
us-east-1	602401143452.dkr.ecr.us-east-1.amazonaws.com
us-east-2	602401143452.dkr.ecr.us-east-2.amazonaws.com
us-west-1	602401143452.dkr.ecr.us-west-1.amazonaws.com
us-west-2	602401143452.dkr.ecr.us-west-2.amazonaws.com
ca-central-1	602401143452.dkr.ecr.ca-central-1.amazonaws.com

AWS 리전	레지스트리
ca-west-1	761377655185.dkr.ecr.ca-west-1.amazonaws.com
sa-east-1	602401143452.dkr.ecr.sa-east-1.amazonaws.com
us-gov-east-1	151742754352.dkr.ecr.us-gov-east-1.amazonaws.com
us-gov-west-1	013241004608.dkr.ecr.us-gov-west-1.amazonaws.com

Amazon EKS 추가 기능

추가 기능은 Kubernetes 애플리케이션에 대한 지원 운영 기능을 제공하는 소프트웨어이지만 애플리케이션에만 국한되지는 않습니다. 여기에는 기본 AWS 리소스를 사용하여 네트워킹, 컴퓨팅 및 스토리지를 관리할 수 있도록 지원하는 관찰 가능성 에이전트 또는 클러스터가 포함됩니다. 추가 기능 소프트웨어는 일반적으로 Kubernetes 커뮤니티, AWS와 같은 클라우드 공급자 또는 서드 파티 공급업체에 의해 구축 및 유지 관리됩니다. Amazon EKS는 모든 클러스터에 대해 Kubernetes용 Amazon VPC CNI 플러그인, kube-proxy 및 CoreDNS와 같은 자체 관리형 추가 기능을 자동으로 설치합니다. VPC CNI 추가 기능은 Amazon EKS Hybrid Nodes와 호환되지 않으며 하이브리드 노드에 배포되지 않습니다. 원하는 경우 추가 기능의 기본 구성을 변경하고 업데이트할 수 있습니다.

Amazon EKS 추가 기능은 Amazon EKS 클러스터에 대해 큐레이팅된 추가 기능 집합을 설치하고 관리합니다. 모든 Amazon EKS 추가 기능에는 최신 보안 패치, 버그 수정 사항이 포함되어 있으며, Amazon EKS와 작동하도록 AWS에서 검증을 거쳤습니다. Amazon EKS 추가 기능을 사용하면 Amazon EKS 클러스터가 안전하고 안정적일도록 일관되게 보장할 수 있으며, 추가 기능을 설치하고 구성하고 업데이트하기 위해 수행해야 하는 작업량을 줄일 수 있습니다. 자체 관리형 추가 기능(예: kube-proxy)이 클러스터에서 이미 실행 중이고 Amazon EKS 추가 기능으로 사용할 수 있는 경우, kube-proxy Amazon EKS 추가 기능을 설치하여 Amazon EKS 추가 기능의 이점을 활용할 수 있습니다.

Amazon EKS API를 통해 Amazon EKS 추가 기능에 대한 특정 Amazon EKS 관리형 구성 필드를 업데이트할 수 있습니다. 추가 기능이 시작되면 Amazon EKS가 관리하지 않는 구성 필드를 Kubernetes 클러스터 내에서 직접 수정할 수도 있습니다. 해당하는 경우, 여기에는 추가 기능에 대한 특정 구성 필드를 정의하는 작업이 포함됩니다. 이러한 변경 사항은 Amazon EKS에 의해 재정의되지 않습니다. 이는

Kubernetes 서버 측 적용 기능을 사용하여 지원됩니다. 자세한 내용은 [the section called “사용자 지정할 수 있는 필드”](#) 섹션을 참조하세요.

모든 Amazon EKS 노드 유형에서 Amazon EKS 추가 기능을 사용할 수 있습니다. 자세한 내용은 [컴퓨팅 관리](#) 섹션을 참조하세요.

Amazon EKS API, AWS Management Console, AWS CLI 및 `eksctl`을 사용하여 Amazon EKS 추가 기능을 추가, 업데이트 또는 삭제할 수 있습니다. 또한 [AWS CloudFormation](#)을 사용하여 Amazon EKS 추가 기능을 생성할 수도 있습니다.

고려 사항

Amazon EKS 추가 기능을 사용할 때는 다음 사항을 고려해야 합니다.

- 클러스터의 추가 기능을 구성하려면 [IAM 보안 주체](#)에게 추가 기능을 사용할 수 있는 IAM 권한이 있어야 합니다. 자세한 내용은 [Amazon Elastic Kubernetes Service에서 정의한 작업](#)에서 해당 이름의 Addon 작업을 참조하세요.
- Amazon EKS 추가 기능은 클러스터용으로 프로비저닝하거나 구성하는 노드에서 실행됩니다. 노드 유형에는 Amazon EC2 인스턴스, Fargate, 하이브리드 노드가 포함됩니다.
- Amazon EKS가 관리하지 않는 필드를 수정하여 Amazon EKS 추가 기능의 설치를 사용자 지정할 수 있습니다. 자세한 내용은 [the section called “사용자 지정할 수 있는 필드”](#) 섹션을 참조하세요.
- AWS Management Console을 사용하여 클러스터를 생성하는 경우 Amazon EKS kube-proxy, Kubernetes용 Amazon VPC CNI 플러그인 및 CoreDNS Amazon EKS 추가 기능이 클러스터에 자동으로 추가됩니다. `eksctl`을 사용하여 config 파일이 있는 클러스터를 생성하는 경우 `eksctl`은 또한 Amazon EKS 추가 기능을 사용하여 클러스터를 생성할 수 있습니다. config 파일 없이 `eksctl`을 사용하거나 다른 도구를 사용하여 클러스터를 생성할 경우 Amazon EKS 추가 기능이 아니라 자체 관리형 kube-proxy, Kubernetes용 Amazon VPC CNI 플러그인 및 CoreDNS 추가 기능이 설치됩니다. 클러스터 생성 후 Amazon EKS 추가 기능을 수동으로 추가하거나 직접 관리할 수 있습니다. 클러스터를 생성하는 데 사용하는 방법과 관계없이 VPC CNI 추가 기능은 하이브리드 노드에 설치되지 않습니다.
- `eks:addon-cluster-admin ClusterRoleBinding`에서는 `cluster-admin ClusterRole`을 `eks:addon-manager Kubernetes ID`에 바인딩합니다. Kubernetes 네임스페이스를 생성하고 추가 기능을 네임스페이스에 설치하는 `eks:addon-manager ID`에 필요한 권한이 역할에 있습니다. `eks:addon-cluster-admin ClusterRoleBinding`이 제거되는 경우 Amazon EKS 클러스터는 계속 작동하지만, Amazon EKS에서 더 이상 추가 기능을 관리할 수 없습니다. 다음과 같은 플랫폼 버전으로 시작하는 모든 클러스터에서 새 `ClusterRoleBinding`을 사용합니다.

- AWS의 EKS 추가 기능의 하위 집합은 Amazon EKS Hybrid Nodes와의 호환성이 검증되었습니다. 자세한 내용은 [the section called “AWS 추가 기능”](#)의 호환성 테이블을 참조하세요.

Amazon EKS Auto Mode 고려 사항

Amazon EKS Auto Mode에는 다음과 같은 필수 클러스터 기능을 제공하는 기능이 포함되어 있습니다.

- 포트 네트워킹
- 서비스 네트워킹
- 클러스터 DNS
- AutoScaling
- 블록 스토리지
- 로드 밸런서 컨트롤러
- Pod Identity 에이전트
- 노드 모니터링 에이전트

Auto Mode 컴퓨팅을 사용하면 다음과 같이 일반적으로 사용되는 많은 EKS 추가 기능이 중복됩니다.

- Amazon VPC CNI
- kube-proxy
- CoreDNS
- Amazon EBS CSI 드라이버
- EKS Pod Identity 에이전트

그러나 클러스터가 Auto Mode를 자체 관리형 EC2 인스턴스, 관리형 노드 그룹 또는 AWS Fargate와 같은 다른 컴퓨팅 옵션과 결합한 경우에도 이러한 추가 기능은 필요합니다. AWS는 지원되는 컴퓨팅 유형에서만 추가 포드가 자동으로 예약되도록 보장하는 반선택도 규칙을 사용하여 EKS 추가 기능을 개선했습니다. 또한 사용자는 이제 EKS 추가 기능 DescribeAddonVersions API를 활용하여 각 추가 기능 및 해당 특정 버전에 대해 지원되는 computeTypes를 확인할 수 있습니다. 또한 EKS Auto Mode를 사용하면 위에 나열된 컨트롤러가 AWS 소유 인프라에서 실행됩니다. 따라서 다른 유형의 컴퓨팅과 함께 EKS Auto Mode를 사용하지 않는 한 계정에서 이러한 컨트롤러를 볼 수도 없습니다. 이 경우 클러스터에 설치한 컨트롤러가 표시됩니다.

기존 클러스터에서 EKS Auto Mode를 활성화하려는 경우 특정 추가 기능의 버전을 업그레이드해야 할 수 있습니다. 자세한 내용은 EKS Auto Mode에 대한 [the section called “필요한 추가 기능 버전”](#) 섹션을 참조하세요.

지원

AWS는 다양한 수준의 지원을 통해 여러 유형의 추가 기능을 게시합니다.

- AWS 추가 기능: 이 추가 기능은 AWS에서 빌드하며 완벽하게 지원됩니다.
 - AWS 추가 기능을 사용하여 Amazon EFS와 같은 다른 AWS 서비스에서 사용할 수 있습니다.
 - 자세한 내용은 [the section called “AWS 추가 기능”](#) 섹션을 참조하세요.
- AWS Marketplace 추가 기능: 이 추가 기능은 AWS가 스캔하고 독립된 AWS 파트너가 지원합니다.
 - Marketplace 추가 기능을 사용하여 Splunk를 사용한 모니터링과 같이 중요하고 정교한 기능을 클러스터에 추가할 수 있습니다.
 - 자세한 내용은 [the section called “Marketplace 추가 기능”](#) 섹션을 참조하세요.
- 커뮤니티 추가 기능: 이 추가 기능은 AWS가 스캔하지만 오픈 소스 커뮤니티가 지원합니다.
 - 커뮤니티 추가 기능을 사용하여 Kubernetes Metrics Server와 같은 일반적인 오픈 소스 소프트웨어를 설치하는 복잡성을 줄일 수 있습니다.
 - 커뮤니티 추가 기능은 AWS에서 빌드합니다. AWS는 버전 호환성을 위해 커뮤니티 추가 기능만 검증합니다.
 - 자세한 내용은 [the section called “커뮤니티 추가 기능”](#) 섹션을 참조하세요.

다음 테이블에서는 각 추가 기능 유형에 대한 지원 범위를 자세히 보여줍니다.

범주	Feature	AWS 추가 기능	AWS Marketplace 추가 기능	커뮤니티 추가 기능
개발	AWS가 빌드함	예	아니요	예
개발	AWS가 검증함	예	아니요	예*
개발	AWS 파트너가 검증함	아니요	예	아니요
유지 관리	AWS가 스캔함	예	예	예

범주	Feature	AWS 추가 기능	AWS Marketplace 추가 기능	커뮤니티 추가 기능
유지 관리	AWS가 패치를 적용함	예	아니요	예
유지 관리	AWS 파트너가 패치를 적용함	아니요	예	아니요
배포	AWS가 게시함	예	아니요	예
배포	AWS 파트너가 게시함	아니요	예	아니요
지원	AWS가 기본 설치 지원을 제공함	예	예	예
지원	전체 AWS 지원	예	아니요	아니요
지원	전체 AWS 파트너 지원	아니요	예	아니요

*: 커뮤니티 추가 기능에 대한 검증에는 Kubernetes 버전 호환성만 포함됩니다. 예를 들어 클러스터에 커뮤니티 추가 기능을 설치하는 경우 AWS는 클러스터의 Kubernetes 버전과 호환되는지 확인합니다.

AWS Marketplace 추가 기능은 AWS 이외의 외부 소스에서 추가 소프트웨어 종속 항목을 다운로드할 수 있습니다. 이러한 외부 종속 항목은 AWS가 스캔하거나 검증하지 않습니다. 외부 종속 항목을 가져오는 AWS Marketplace 추가 기능을 배포하는 경우 보안 요구 사항을 고려하세요.

AWS 추가 기능

다음과 같은 Amazon EKS 추가 기능을 사용하여 클러스터를 생성할 수 있습니다. eksctl, AWS Management Console 또는 AWS CLI를 사용하여 사용할 수 있는 추가 기능의 최신 목록을 언제든지 볼 수 있습니다. 사용 가능한 모든 추가 기능을 보거나 추가 기능을 설치하려면 [the section called “추가 기능 생성”](#) 섹션을 참조하세요. 추가 기능에 IAM 권한이 필요한 경우 클러스터에 대한 IAM OpenID Connect(OIDC) 제공자가 있어야 합니다. 공급자가 있는지 아니면 생성해야 하는지 확인하려면 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요. 추가 기능을 설치한 후 추가 기능을 업데이트하거나 삭제할 수 있습니다. 자세한 내용은 [the section called “추가 기능 업데이트”](#) 또는 [the section](#)

called “[추가 기능 제거](#)”을 참조하세요. Amazon EKS Hybrid Nodes에서 EKS 추가 기능 실행과 관련된 고려 사항에 대한 자세한 내용은 [the section called “추가 기능 구성](#)” 섹션을 참조하세요.

다음 Amazon EKS 추가 기능을 사용할 수 있습니다.

설명	자세히 알아보기	호환되는 컴퓨팅 유형
클러스터를 위한 네이티브 VPC 네트워킹 제공	the section called “Kubernetes 용 Amazon VPC CNI 플러그인”	EC2
Kubernetes 클러스터 DNS로 사용할 수 있는 유연하고 확장 가능한 DNS 서버입니다.	the section called “CoreDNS”	EC2, Fargate, EKS 자동 모드, EKS Hybrid Nodes
각 Amazon EC2 노드에서 네트워크 규칙 유지 관리	the section called “Kube-proxy ”	EC2, EKS Hybrid Nodes
클러스터에 Amazon EBS 스토리지 제공	the section called “Amazon EBS CSI 드라이버”	EC2
클러스터에 Amazon EFS 스토리지 제공	the section called “Amazon EFS CSI 드라이버”	EC2, EKS Auto Mode
클러스터에 Amazon FSx for Lustre 스토리지 제공	the section called “Amazon FSx CSI 드라이버”	EC2, EKS Auto Mode
클러스터에 Amazon S3 스토리지 제공	the section called “Mountpoint for Amazon S3 CSI 드라이버”	EC2, EKS Auto Mode
추가 노드 상태 문제 감지	the section called “노드 모니터링 에이전트”	EC2, EKS Hybrid Nodes
Amazon EBS CSI 드라이버와 같은 호환되는 CSI 드라이버에서 스냅샷 기능의 사용을 활성화	the section called “CSI 스냅샷 컨트롤러”	EC2, Fargate, EKS 자동 모드, EKS Hybrid Nodes

설명	자세히 알아보기	호환되는 컴퓨팅 유형
SageMaker HyperPod 작업 거버넌스는 Amazon EKS 클러스터에서 여러 팀 간 컴퓨팅 리소스 할당 및 사용을 최적화하여 작업 우선순위 지정 및 리소스 공유의 비효율성을 해소합니다.	the section called “Amazon SageMaker HyperPod 작업 거버넌스”	EC2, EKS 자율 모드
네트워크 흐름 데이터를 수집하여 Amazon CloudWatch에 보고하는 Kubernetes 에이전트로, 클러스터 노드 간 TCP 연결을 포괄적으로 모니터링할 수 있습니다.	the section called “AWS Network Flow Monitor Agent”	EC2, EKS Auto Mode
안전하고 프로덕션 준비가 완료된 AWS 지원 OpenTelemetry 프로젝트 배포	the section called “AWS Distro for OpenTelemetry”	EC2, Fargate, EKS 자동 모드, EKS Hybrid Nodes
AWS CloudTrail 관리 이벤트와 Amazon VPC 흐름 로그를 포함한 기본 데이터 소스를 분석하고 처리하는 보안 모니터링 서비스입니다. Amazon GuardDuty는 또한 Kubernetes 감사 로그 및 런타임 모니터링과 같은 기능을 처리합니다.	the section called “Amazon GuardDuty 에이전트”	EC2, EKS Auto Mode

설명	자세히 알아보기	호환되는 컴퓨팅 유형
AWS에서 제공하는 모니터링 및 관찰성 서비스입니다. 이 추가 기능은 CloudWatch 에이전트를 설치하고 Amazon EKS의 향상된 관찰성을 통해 CloudWatch Application Signals와 CloudWatch Container Insights 모두 활성화합니다.	the section called “Amazon CloudWatch Observability 에이전트”	EC2, EKS 자동 모드, EKS Hybrid Nodes
EC2 인스턴스 프로필이 EC2 인스턴스에 자격 증명을 제공하는 방식과 유사하게 애플리케이션에 대한 자격 증명을 관리하는 기능	the section called “EKS Pod Identity 에이전트”	EC2, EKS Hybrid Nodes
AWS Private CA에서 X.509 인증서를 발급하려면 cert-manager를 활성화합니다. cert-manager가 필요합니다.	the section called “AWS Private CA Connector for Kubernetes”	EC2, Fargate, EKS 자동 모드, EKS Hybrid Nodes

Kubernetes용 Amazon VPC CNI 플러그인

Kubernetes용 Amazon VPC CNI 플러그인 Amazon EKS 추가 기능은 클러스터에 네이티브 VPC 네트워킹을 제공하는 Kubernetes 컨테이너 네트워크 인터페이스(CNI) 플러그인입니다. 이 추가 기능의 자체 관리형 또는 관리형 유형은 기본적으로 각 Amazon EC2 노드에 설치됩니다. 자세한 내용은 [Kubernetes container network interface \(CNI\) 플러그인](#)을 참조하세요.

Note

Amazon EKS Auto Mode 클러스터에 이 추가 기능을 설치할 필요가 없습니다. 자세한 내용은 [the section called “Amazon EKS Auto Mode 고려 사항”](#) 섹션을 참조하세요.

Amazon EKS 추가 기능 이름은 vpc-cni입니다.

필수 IAM 권한

이 추가 기능은 Amazon EKS의 서비스 계정에 대한 IAM 역할 기능을 활용합니다. 자세한 내용은 [the section called “IRSA를 사용한 자격 증명”](#) 섹션을 참조하세요.

클러스터에서 IPv4 패밀리를 사용하는 경우 [AmazonEKS_CNI_Policy](#)의 권한이 필요합니다. 클러스터에서 IPv6 패밀리를 사용하는 경우 [IPv6 모드](#)에서 해당 권한이 있는 [IAM 정책을 만들어야](#) 합니다. 다음 명령을 사용하여 IAM 역할을 생성하고, 정책 중 하나를 이 역할에 연결하고, 추가 기능에서 사용하는 Kubernetes 서비스 계정에 주석을 달 수 있습니다.

*my-cluster*를 클러스터 이름으로 바꾸고 *AmazonEKSVPCCNIRole*을 역할의 이름으로 바꿉니다. 클러스터에서 IPv6 패밀리를 사용하는 경우 *AmazonEKS_CNI_Policy*를 생성한 정책의 이름으로 바꿉니다. 이 명령을 사용하려면 디바이스에 [eksctl](#)이 설치되어 있어야 합니다. 다른 도구를 사용하여 역할을 생성하고 이 역할에 정책을 연결하고, Kubernetes 서비스 계정에 주석을 추가해야 하는 경우 [the section called “IAM 역할 할당”](#) 섹션을 참조하세요.

```
eksctl create iamserviceaccount --name aws-node --namespace kube-system --cluster my-cluster --role-name AmazonEKSVPCCNIRole \
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy --approve
```

정보 업데이트

한 번에 하나의 마이너 버전만 업데이트할 수 있습니다. 예를 들어, 현재 버전이 1.28.x-eksbuild.y 인데 1.30.x-eksbuild.y (으)로 업데이트하려는 경우, 현재 버전을 1.29.x-eksbuild.y (으)로 먼저 업데이트한 다음에 1.30.x-eksbuild.y (으)로 업데이트해야 합니다. 추가 기능을 업데이트하는 방법에 대한 자세한 내용은 [the section called “업데이트\(EKS 추가 기능\)”](#) 부분을 참조하세요.

CoreDNS

CoreDNS Amazon EKS 추가 기능은 Kubernetes 클러스터 DNS 역할을 할 수 있는 유연하고 확장 가능한 DNS 서버입니다. 이 추가 기능의 자체 관리형 또는 관리형 유형은 기본적으로 클러스터를 만들 때 설치되었습니다. 하나 이상의 노드가 있는 Amazon EKS 클러스터를 시작하면 클러스터에 배포된 노드 수에 관계없이 CoreDNS 이미지의 복제본 2개가 기본적으로 배포됩니다. CoreDNS 포드는 클러스터의 모든 포드에 대한 이름 확인을 제공합니다. 클러스터에 CoreDNS 배포용 네임스페이스와 일치하는 네임스페이스가 있는 Fargate 프로필이 포함된 경우 CoreDNS 포드를 Fargate 노드에 배포할 수 있습니다. 자세한 내용은 [the section called “프로필 정의”](#) 섹션을 참조하세요.

Note

Amazon EKS Auto Mode 클러스터에 이 추가 기능을 설치할 필요가 없습니다. 자세한 내용은 [the section called “Amazon EKS Auto Mode 고려 사항”](#) 섹션을 참조하세요.

Amazon EKS 추가 기능 이름은 coredns입니다.

필수 IAM 권한

이 추가 기능에는 권한이 필요하지 않습니다.

추가 정보

CoreDNS에 대해 자세히 알아보려면 쿠버네티스 문서의 [서비스 디스커버리를 위해 CoreDNS 사용하기](#)와 [DNS 서비스 사용자 정의하기](#)를 참조하세요.

Kube-proxy

Kube-proxy Amazon EKS 추가 기능은 각 Amazon EC2 노드에서 네트워크 규칙을 유지합니다. 이를 통해 포드와의 네트워크 통신이 가능합니다. 이 추가 기능의 자체 관리형 또는 관리형 유형은 기본적으로 클러스터의 각 Amazon EC2 노드에 설치됩니다.

Note

Amazon EKS Auto Mode 클러스터에 이 추가 기능을 설치할 필요가 없습니다. 자세한 내용은 [the section called “Amazon EKS Auto Mode 고려 사항”](#) 섹션을 참조하세요.

Amazon EKS 추가 기능 이름은 kube-proxy입니다.

필수 IAM 권한

이 추가 기능에는 권한이 필요하지 않습니다.

정보 업데이트

현재 버전을 업데이트하기 전에 다음 요구 사항을 고려하세요.

- Amazon EKS 클러스터의 Kube-proxy에는 [Kubernetes와 동일한 호환성 및 skew 정책](#)이 있습니다.

추가 정보

kube-proxy에 대한 자세한 내용은 쿠버네티스 문서의 [kube-proxy](#)를 참조하세요.

Amazon EBS CSI 드라이버

Amazon EBS CSI 드라이버 Amazon EKS 추가 기능은 클러스터에 Amazon EBS 스토리지를 제공하는 Kubernetes 컨테이너 스토리지 인터페이스(CSI) 플러그인입니다.

Note

Amazon EKS Auto Mode 클러스터에 이 추가 기능을 설치할 필요가 없습니다. Auto Mode에는 블록 스토리지 기능이 포함되어 있습니다. 자세한 내용은 [the section called “상태 저장 워크로드 배포”](#) 섹션을 참조하세요.

Amazon EKS 추가 기능 이름은 `aws-ebs-csi-driver`입니다.

필수 IAM 권한

이 추가 기능은 Amazon EKS의 서비스 계정에 대한 IAM 역할 기능을 활용합니다. 자세한 내용은 [the section called “IRSA를 사용한 자격 증명”](#) 섹션을 참조하세요. [AmazonEBSCSIDriverPolicy](#) AWS관리형 정책의 권한이 필요합니다. IAM 역할을 생성하고 다음 명령을 사용하여 관리형 정책을 연결할 수 있습니다. `my-cluster`를 클러스터 이름으로 바꾸고 `AmazonEKS_EBS_CSI_DriverRole`을 역할의 이름으로 바꿉니다. 이 명령을 사용하려면 디바이스에 `eksctl`가 설치되어 있어야 합니다. 다른 도구를 사용해야 하거나 암호화에 사용자 지정 [KMS 키](#)를 사용해야 하는 경우 [the section called “1단계-IAM 역할 생성”](#) 섹션을 참조하세요.

```
eksctl create iamserviceaccount \
  --name ebs-csi-controller-sa \
  --namespace kube-system \
  --cluster my-cluster \
  --role-name AmazonEKS_EBS_CSI_DriverRole \
  --role-only \
  --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
  --approve
```

추가 정보

추가 기능에 대한 자세한 내용은 [the section called “Amazon EBS”](#) 섹션을 참조하세요.

Amazon EFS CSI 드라이버

Amazon EFS CSI 드라이버 Amazon EKS 추가 기능은 클러스터에 Amazon EFS 스토리지를 제공하는 Kubernetes 컨테이너 스토리지 인터페이스(CSI) 플러그인입니다.

Amazon EKS 추가 기능 이름은 `aws-efs-csi-driver`입니다.

필수 IAM 권한

필수 IAM 권한 – 이 추가 기능은 Amazon EKS의 서비스 계정에 대한 IAM 역할 기능을 활용합니다. 자세한 내용은 [the section called “IRSA를 사용한 자격 증명”](#) 섹션을 참조하세요.

[AmazonEFSCSIDriverPolicy](#) AWS관리형 정책의 권한이 필요합니다. IAM 역할을 생성하고 다음 명령을 사용하여 관리형 정책을 연결할 수 있습니다. `my-cluster`를 클러스터 이름으로 바꾸고 `AmazonEKS_EFS_CSI_DriverRole`을 역할의 이름으로 바꿉니다. 이러한 명령을 사용하려면 디바이스에 `eksctl`이(가) 설치되어 있어야 합니다. 다른 도구를 사용해야 하는 경우, [the section called “1단계-IAM 역할 생성”](#)을(를) 참조하세요.

```
export cluster_name=my-cluster
export role_name=AmazonEKS_EFS_CSI_DriverRole
eksctl create iamserviceaccount \
  --name efs-csi-controller-sa \
  --namespace kube-system \
  --cluster $cluster_name \
  --role-name $role_name \
  --role-only \
  --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEFSCSIDriverPolicy \
  --approve
TRUST_POLICY=$(aws iam get-role --output json --role-name $role_name --query
  'Role.AssumeRolePolicyDocument' | \
  sed -e 's/efs-csi-controller-sa/efs-csi-*/' -e 's/StringEquals/StringLike/')
aws iam update-assume-role-policy --role-name $role_name --policy-document
"$TRUST_POLICY"
```

추가 정보

추가 기능에 대한 자세한 내용은 [the section called “Amazon EFS”](#) 섹션을 참조하세요.

Amazon FSx CSI 드라이버

Amazon FSx CSI 드라이버 Amazon EKS 추가 기능은 클러스터에 Amazon FSx for Lustre 스토리지를 제공하는 Kubernetes 컨테이너 스토리지 인터페이스(CSI) 플러그인입니다.

Amazon EKS 추가 기능 이름은 `aws-fsx-csi-driver`입니다.

Note

- 클러스터에 기존 Amazon FSx CSI 드라이버를 설치하면 추가 기능 설치에 실패할 수 있습니다. 비 EKS FSx CSI 드라이버가 있는 동안 Amazon EKS 추가 기능 버전을 설치하려고 하면 리소스 충돌로 인해 설치가 실패합니다. 설치 중 `OVERWRITE` 플래그를 사용하여 이 문제를 해결합니다.

```
aws eks create-addon --addon-name aws-fsx-csi-driver --cluster-name my-cluster
--resolve-conflicts OVERWRITE
```

- Amazon FSx CSI 드라이버 EKS 추가 기능을 사용하려면 인증을 위해 EKS Pod Identity 에이전트가 필요합니다. 이 구성 요소가 없으면 Amazon EKS Pod Identity agent is not installed in the cluster 오류와 함께 추가 기능이 실패하여 볼륨 작업을 수행하지 못합니다. FSx CSI 드라이버 추가 기능을 배포하기 전 또는 후에 Pod Identity 에이전트를 설치합니다. 자세한 내용은 [the section called “에이전트 설정”](#) 섹션을 참조하세요.

필수 IAM 권한

이 추가 기능은 Amazon EKS의 서비스 계정에 대한 IAM 역할 기능을 활용합니다. 자세한 내용은 [the section called “IRSA를 사용한 자격 증명”](#) 섹션을 참조하세요. [AmazonFSxFullAccess](#) AWS 관리형 정책의 권한이 필요합니다. IAM 역할을 생성하고 다음 명령을 사용하여 관리형 정책을 연결할 수 있습니다. `my-cluster`를 클러스터 이름으로 바꾸고 `AmazonEKS_FSx_CSI_DriverRole`을 역할 이름으로 바꿉니다. 이 명령을 사용하려면 디바이스에 `eksctl`가 설치되어 있어야 합니다.

```
eksctl create iamserviceaccount \
  --name fsx-csi-controller-sa \
  --namespace kube-system \
  --cluster my-cluster \
  --role-name AmazonEKS_FSx_CSI_DriverRole \
  --role-only \
  --attach-policy-arn arn:aws:iam::aws:policy/AmazonFSxFullAccess \
  --approve
```

추가 정보

추가 기능에 대한 자세한 내용은 [the section called “Amazon FSx for Lustre”](#) 섹션을 참조하세요.

Mountpoint for Amazon S3 CSI 드라이버

Mountpoint for Amazon S3 CSI 드라이버 Amazon EKS 추가 기능은 클러스터에 Amazon S3 스토리지를 제공하는 Kubernetes 컨테이너 스토리지 인터페이스(CSI) 플러그인입니다.

Amazon EKS 추가 기능 이름은 `aws-mountpoint-s3-csi-driver`입니다.

필수 IAM 권한

이 추가 기능은 Amazon EKS의 서비스 계정에 대한 IAM 역할 기능을 활용합니다. 자세한 내용은 [the section called “IRSA를 사용한 자격 증명”](#) 섹션을 참조하세요.

생성된 IAM 역할에는 S3에 대한 액세스 권한을 부여하는 정책이 필요합니다. 정책을 생성할 때 [Mountpoint IAM 권한 권장 사항](#)을 따르세요. 또는 AWS 관리형 정책 [AmazonS3FullAccess](#)를 사용할 수 있지만 이 관리형 정책은 Mountpoint에 필요한 것보다 더 많은 권한을 부여합니다.

IAM 역할을 생성하고 다음 명령을 사용하여 정책을 연결할 수 있습니다. `my-cluster`를 클러스터 이름으로, `region-code`를 올바른 AWS 리전 코드로, `AmazonEKS_S3_CSI_DriverRole`을 역할 이름으로, `AmazonEKS_S3_CSI_DriverRole_ARN`을 역할 ARN으로 바꿉니다. 이러한 명령을 사용하려면 디바이스에 `eksctl`이(가) 설치되어 있어야 합니다. IAM 콘솔 또는 AWS CLI 사용에 대한 지침은 [the section called “2단계 - IAM 역할 생성”](#) 섹션을 참조하세요.

```
CLUSTER_NAME=my-cluster
REGION=region-code
ROLE_NAME=AmazonEKS_S3_CSI_DriverRole
POLICY_ARN=AmazonEKS_S3_CSI_DriverRole_ARN
eksctl create iamserviceaccount \
  --name s3-csi-driver-sa \
  --namespace kube-system \
  --cluster $CLUSTER_NAME \
  --attach-policy-arn $POLICY_ARN \
  --approve \
  --role-name $ROLE_NAME \
  --region $REGION \
  --role-only
```

추가 정보

추가 기능에 대한 자세한 내용은 [the section called “Mountpoint for Amazon S3”](#) 섹션을 참조하세요.

CSI 스냅샷 컨트롤러

컨테이너 스토리지 인터페이스(CSI) 스냅샷 컨트롤러를 사용하면 Amazon EBS CSI 드라이버와 같은 호환 CSI 드라이버에서 스냅샷 기능을 사용할 수 있습니다.

Amazon EKS 추가 기능 이름은 `snapshot-controller`입니다.

필수 IAM 권한

이 추가 기능에는 권한이 필요하지 않습니다.

추가 정보

추가 기능에 대한 자세한 내용은 [the section called “CSI 스냅샷 컨트롤러”](#) 섹션을 참조하세요.

Amazon SageMaker HyperPod 작업 거버넌스

SageMaker HyperPod 작업 거버넌스는 리소스 할당을 간소화하고 Amazon EKS 클러스터에서 팀 및 프로젝트 간 컴퓨팅 리소스를 효율적으로 활용하도록 설계된 강력한 관리 시스템입니다. 이를 통해 관리자는 다음을 설정할 수 있습니다.

- 다양한 작업에 대한 우선순위 수준
- 각 팀에 대한 컴퓨팅 할당
- 각 팀이 유휴 컴퓨팅을 대여 및 차용하는 방법
- 팀이 팀의 작업을 선점하는 경우

또한 HyperPod 작업 거버넌스는 Amazon EKS 클러스터 관찰성을 제공하여 클러스터 용량에 대한 실시간 가시성을 제공합니다. 여기에는 컴퓨팅 가용성 및 사용량, 팀 할당 및 사용률, 작업 실행 및 대기 시간 정보, 정보에 입각한 의사 결정 및 선제적 리소스 관리 설정이 포함됩니다.

Amazon EKS 추가 기능 이름은 `amazon-sagemaker-hyperpod-taskgovernance`입니다.

필수 IAM 권한

이 추가 기능에는 권한이 필요하지 않습니다.

추가 정보

추가 기능에 대한 자세한 내용은 [SageMaker HyperPod 작업 거버넌스](#)를 참조하세요.

AWS Network Flow Monitor Agent

Amazon CloudWatch Network Flow Monitor Agent는 클러스터의 모든 노드에서 TCP 연결 통계를 수집하고 네트워크 흐름 보고서를 Amazon CloudWatch Network Flow Monitor 수집 API에 게시하는 Kubernetes 애플리케이션입니다.

Amazon EKS 추가 기능 이름은 `aws-network-flow-monitoring-agent`입니다.

필수 IAM 권한

이 추가 기능에는 IAM 권한이 필요하지 않습니다.

추가 기능에 `CloudWatchNetworkFlowMonitorAgentPublishPolicy` 관리형 정책을 연결해야 합니다.

필수 IAM 설정에 대한 자세한 내용은 Amazon CloudWatch Network Flow Monitor Agent GitHub 리포지토리의 [IAM Policy](#)를 참조하세요.

관리형 정책에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 [CloudWatchNetworkFlowMonitorAgentPublishPolicy](#)를 참조하세요.

추가 정보

추가 기능에 대한 자세한 내용은 [Amazon CloudWatch Network Flow Monitor Agent GitHub 리포지토리](#)를 참조하세요.

노드 모니터링 에이전트

노드 모니터링 에이전트 Amazon EKS 추가 기능으로 추가 노드 상태 문제를 감지할 수 있습니다. 이러한 추가 상태 신호는 선택적 노드 자동 복구 기능을 통해 필요에 따라 노드를 자동으로 교체할 수도 있습니다.

Note

Amazon EKS Auto Mode 클러스터에 이 추가 기능을 설치할 필요가 없습니다. 자세한 내용은 [the section called “Amazon EKS Auto Mode 고려 사항”](#) 섹션을 참조하세요.

Amazon EKS 추가 기능 이름은 `eks-node-monitoring-agent`입니다.

필수 IAM 권한

이 추가 기능에는 추가 권한이 필요하지 않습니다.

추가 정보

자세한 내용은 [the section called “노드 상태”](#) 섹션을 참조하세요.

AWS Distro for OpenTelemetry

AWS Distro for OpenTelemetry Amazon EKS 추가 기능은 OpenTelemetry 프로젝트의 안전하고 프로덕션 준비가 된 AWS 지원 배포입니다. 자세한 내용은 GitHub의 [AWS Distro for OpenTelemetry](#)를 참조하세요.

Amazon EKS 추가 기능 이름은 adot입니다.

필수 IAM 권한

이 추가 기능에는 고급 구성을 통해 옵트인할 수 있는 사전 구성된 사용자 지정 리소스 중 하나를 사용하는 경우에만 IAM 권한이 필요합니다.

추가 정보

자세한 내용은 AWS Distro for OpenTelemetry 설명서의 [Getting Started with AWS Distro for OpenTelemetry using EKS Add-Ons](#)를 참조하세요.

ADOT는 cert-manager 추가 기능이 클러스터에 사전 조건으로 배포되어야 합니다. 그렇지 않으면 <https://registry.terraform.io/modules/terraform-aws-modules/eks/aws/latest> cluster_addons 속성을 사용하여 직접 배포하는 경우 이 추가 기능이 작동하지 않습니다. 자세한 내용은 AWS Distro for OpenTelemetry 설명서의 [Requirements for Getting Started with AWS Distro for OpenTelemetry using EKS Add-Ons](#)를 참조하세요.

Amazon GuardDuty 에이전트

Amazon GuardDuty 에이전트 Amazon EKS 추가 기능은 GuardDuty 런타임 모니터링에서 분석할 수 있도록 EKS 클러스터 노드에서 [런타임 이벤트](#)(파일 액세스, 프로세스 실행, 네트워크 연결)를 수집합니다. GuardDuty 자체(에이전트 아님)는 AWS CloudTrail 관리 이벤트 및 Amazon VPC 흐름 로그와 같은 [기본 데이터 소스](#)와 Kubernetes 감사 로그 및 런타임 모니터링과 같은 [기능](#)을 분석하고 처리하는 보안 모니터링 서비스입니다.

Amazon EKS 추가 기능 이름은 aws-guardduty-agent입니다.

필수 IAM 권한

이 추가 기능에는 권한이 필요하지 않습니다.

추가 정보

자세한 내용은 [Amazon GuardDuty의 Amazon EKS 클러스터에 대한 런타임 모니터링](#)을 참조하세요.

- Amazon EKS 클러스터에서 잠재적인 보안 위협을 탐지하려면 Amazon GuardDuty 런타임 모니터링을 활성화하고 GuardDuty 보안 에이전트를 Amazon EKS 클러스터에 배포하세요.

Amazon CloudWatch Observability 에이전트

Amazon CloudWatch 관찰성 에이전트 Amazon EKS 추가 기능은 AWS에서 제공하는 모니터링 및 관찰성 서비스입니다. 이 추가 기능은 CloudWatch 에이전트를 설치하고 Amazon EKS의 향상된 관찰성을 통해 CloudWatch Application Signals와 CloudWatch Container Insights를 모두 활성화합니다. 자세한 내용은 [Amazon CloudWatch 에이전트](#)를 참조하세요.

Amazon EKS 추가 기능 이름은 amazon-cloudwatch-observability입니다.

필수 IAM 권한

이 추가 기능은 Amazon EKS의 서비스 계정에 대한 IAM 역할 기능을 활용합니다. 자세한 내용은 [the section called “IRSA를 사용한 자격 증명”](#) 섹션을 참조하세요. [AWSXrayWriteOnlyAccess](#) 및 [CloudWatchAgentServerPolicy](#) AWS 관리형 정책의 권한이 필요합니다. 다음 명령을 사용하여 IAM 역할을 생성하고, 관리형 정책을 이 역할에 연결하고, 추가 기능에서 사용하는 Kubernetes 서비스 계정에 주석을 달 수 있습니다. *my-cluster*를 클러스터 이름으로 바꾸고 *AmazonEKS_Observability_role*을 역할의 이름으로 바꿉니다. 이 명령을 사용하려면 디바이스에 [eksctl](#)가 설치되어 있어야 합니다. 다른 도구를 사용하여 역할을 생성하고 이 역할에 정책을 연결하고, Kubernetes 서비스 계정에 주석을 추가해야 하는 경우 [the section called “IAM 역할 할당”](#) 섹션을 참조하세요.

```
eksctl create iamserviceaccount \
  --name cloudwatch-agent \
  --namespace amazon-cloudwatch \
  --cluster my-cluster \
  --role-name AmazonEKS_Observability_Role \
  --role-only \
  --attach-policy-arn arn:aws:iam::aws:policy/AWSXrayWriteOnlyAccess \
  --attach-policy-arn arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy \
  --approve
```

추가 정보

자세한 내용은 [CloudWatch 에이전트 설치](#)를 참조하세요.

AWS Private CA Connector for Kubernetes

AWS Private CA Connector for Kubernetes는 사용자가 AWS Private Certificate Authority(AWS Private CA)에서 인증서를 얻을 수 있는 cert-manager용 추가 기능입니다.

- Amazon EKS 추가 기능 이름은 `aws-privateca-connector-for-kubernetes`입니다.
- 추가 기능 네임스페이스는 `aws-privateca-issuer`입니다.

이 추가 기능에는 cert-manager가 필요합니다. cert-manager는 Amazon EKS에서 커뮤니티 추가 기능으로 사용할 수 있습니다. 이 추가 기능에 대한 자세한 내용은 [the section called “Cert Manager”](#) 섹션을 참조하세요. 추가 기능 설치에 대한 자세한 내용은 [the section called “추가 기능 생성”](#) 섹션을 참조하세요.

필수 IAM 권한

이 추가 기능에는 IAM 권한이 필요합니다.

EKS Pod Identity를 사용하여 `AWSPriateCAConnectorForKubernetesPolicy` IAM 정책을 `aws-privateca-issuer` Kubernetes 서비스 계정에 연결합니다. 자세한 내용은 [the section called “Pod Identity 사용”](#) 섹션을 참조하세요.

필요한 권한에 대한 자세한 내용은 AWS 관리형 정책 참조의 [AWSPriateCAConnectorForKubernetesPolicy](#)를 참조하세요.

추가 정보

자세한 내용은 [AWS Private CA Issuer for Kubernetes GitHub 리포지토리](#)를 참조하세요.

추가 기능 구성에 대한 자세한 내용은 `aws-privateca-issuer` GitHub 리포지토리의 [values.yaml](#)을 참조하세요. `values.yaml` 버전이 클러스터에 설치된 추가 기능의 버전과 일치하는지 확인합니다.

이 추가 기능은 EKS 자동 모드의 `system NodePool`에서 사용하는 `CriticalAddonsOnly` 테인트를 허용합니다. 자세한 내용은 [the section called “핵심 추가 기능 실행”](#) 섹션을 참조하세요.

EKS Pod Identity 에이전트

Amazon EKS Pod Identity 에이전트 Amazon EKS 추가 기능은 EC2 인스턴스 프로필이 EC2 인스턴스에 자격 증명을 제공하는 방식과 유사하게 애플리케이션에 대한 자격 증명을 관리하는 기능을 제공합니다.

Note

Amazon EKS Auto Mode 클러스터에 이 추가 기능을 설치할 필요가 없습니다. Amazon EKS Auto Mode는 EKS Pod Identity와 통합됩니다. 자세한 내용은 [the section called “Amazon EKS Auto Mode 고려 사항”](#) 섹션을 참조하세요.

Amazon EKS 추가 기능 이름은 eks-pod-identity-agent입니다.

필수 IAM 권한

Pod Identity Agent 추가 기능 자체에는 IAM 역할이 필요하지 않습니다. [Amazon EKS 노드 IAM 역할](#)의 권한을 사용하여 작동하지만 추가 기능에 전용 IAM 역할이 필요하지 않습니다.

정보 업데이트

한 번에 하나의 마이너 버전만 업데이트할 수 있습니다. 예를 들어, 현재 버전이 1.28.x-eksbuild.y인데 1.30.x-eksbuild.y(으)로 업데이트하려는 경우, 현재 버전을 1.29.x-eksbuild.y(으)로 먼저 업데이트한 다음에 1.30.x-eksbuild.y(으)로 업데이트해야 합니다. 추가 기능을 업데이트하는 방법에 대한 자세한 내용은 [the section called “추가 기능 업데이트”](#) 부분을 참조하세요.

커뮤니티 추가 기능

AWS API를 사용하여 Kubernetes Metrics Server와 같은 커뮤니티 추가 기능을 설치할 수 있습니다. 여러 클러스터에서 소프트웨어를 유지 관리하는 복잡성을 줄이기 위해 Amazon EKS 추가 기능으로 커뮤니티 추가 기능을 설치할 수 있습니다.

예를 들어 AWS API, CLI 또는 Management Console을 사용하여 커뮤니티 추가 기능을 설치할 수 있습니다. 클러스터 생성 중에 커뮤니티 추가 기능을 설치할 수 있습니다.

기존 Amazon EKS 추가 기능과 마찬가지로 커뮤니티 추가 기능을 관리합니다. 커뮤니티 추가 기능은 지원 범위가 고유하다는 점에서 기존 추가 기능과 다릅니다.

Note

커뮤니티 추가 기능 사용은 사용자의 재량에 따릅니다. 사용자와 AWS 간 [공동 책임 모델](#)의 일환으로 이러한 서드파티 플러그인에 대해 클러스터에 설치하는 작업을 이해해야 합니다. 또한 커뮤니티 추가 기능에서 클러스터 보안 요구 사항을 충족하는 것도 사용자의 몫입니다. 자세한 내용은 [the section called “EKS에 배포된 소프트웨어 지원”](#) 섹션을 참조하세요.

커뮤니티 추가 기능은 AWS에서 패키징됩니다. AWS는 버전 호환성을 위해 커뮤니티 추가 기능만 검증합니다. 예를 들어 클러스터에 커뮤니티 추가 기능을 설치하는 경우 AWS는 클러스터의 Kubernetes 버전과 호환되는지 확인합니다.

중요한 점은 AWS가 커뮤니티 추가 기능에 대한 완전한 지원을 제공하지 않는다는 것입니다. AWS는 추가 기능 설치 또는 추가 기능 삭제와 같이 AWS API를 사용하여 수행된 수명 주기 작업만 지원합니다.

커뮤니티 추가 기능에 대한 지원이 필요한 경우 기존 프로젝트 리소스를 활용합니다. 예를 들어 프로젝트의 리포지토리에서 GitHub 문제를 생성할 수 있습니다.

추가 기능 유형 결정

AWS CLI를 사용하여 Amazon EKS 추가 기능 유형을 결정할 수 있습니다.

다음 CLI 명령을 사용하여 추가 기능에 대한 정보를 검색합니다. `metrics-server`를 추가 기능 이름으로 바꿀 수 있습니다.

```
aws eks describe-addon-versions --addon-name metrics-server
```

`owner` 필드의 CLI 출력을 검토합니다.

```
{
  "addons": [
    {
      "addonName": "metrics-server",
      "type": "observability",
      "owner": "community",
      "addonVersions": [
```

`owner` 값이 `community`인 경우 추가 기능은 커뮤니티 추가 기능입니다. AWS는 추가 기능의 설치, 업데이트 및 제거에 대한 지원만 제공합니다. 추가 기능 자체의 기능 및 작동에 대한 질문이 있는 경우 GitHub 문제와 같은 커뮤니티 리소스를 사용합니다.

커뮤니티 추가 기능 설치 또는 업데이트

다른 Amazon EKS 추가 기능과 동일한 방식으로 커뮤니티 추가 기능을 설치하거나 업데이트합니다.

- [the section called “추가 기능 생성”](#)
- [the section called “추가 기능 업데이트”](#)
- [the section called “추가 기능 제거”](#)

사용 가능한 커뮤니티 추가 기능

Amazon EKS에서 다음 커뮤니티 추가 기능을 사용할 수 있습니다.

Kubernetes 지표 서버

Kubernetes Metrics Server는 Kubernetes 기본 제공 오토 스케일링 파이프라인에 대한 컨테이너 리소스 지표의 확장 가능하고 효율적인 소스입니다. Kubelets에서 리소스 지표를 수집하고 Horizontal Pod Autoscaler 및 Vertical Pod Autoscaler에서 사용할 수 있도록 지표 API를 통해 Kubernetes apiserver에 공개합니다.

속성	값
추가 기능 이름	metrics-server
네임스페이스	kube-system
설명서	GitHub Readme
서비스 계정 이름	없음
관리형 IAM 정책	없음
사용자 지정 IAM 권한	없음

kube-state-metrics

클러스터 수준 지표를 생성하고 노출하는 추가 기능 에이전트입니다.

Kubernetes API의 Kubernetes 객체 상태는 지표로 노출될 수 있습니다. kube-state-metrics라는 추가 기능 에이전트는 Kubernetes API 서버에 연결하고 클러스터의 개별 객체 상태에서 생성된 지표가 있는 HTTP 엔드포인트를 노출할 수 있습니다. 레이블 및 주석, 시작 및 종료 시간, 상태 또는 객체가 현재 있는 단계와 같은 객체의 상태에 대한 다양한 정보를 노출합니다.

속성	값
추가 기능 이름	kube-state-metrics
네임스페이스	kube-state-metrics

속성	값
설명서	Kubernetes Documentation의 Metrics for Kubernetes Object States
서비스 계정 이름	없음
관리형 IAM 정책	없음
사용자 지정 IAM 권한	없음

Prometheus 익스포터

플러그형 지표 수집기를 사용하여 Go로 작성된 *NIX 커널에서 노출되는 하드웨어 및 OS 지표용 Prometheus 내보내기입니다. Prometheus Node Exporter는 다양한 하드웨어 및 커널 관련 지표를 노출합니다.

속성	값
추가 기능 이름	prometheus-node-exporter
네임스페이스	prometheus-node-exporter
설명서	Prometheus Documentation의 Monitoring Linux host metrics with the Node Exporter
서비스 계정 이름	없음
관리형 IAM 정책	없음
사용자 지정 IAM 권한	없음

Cert Manager

Cert Manager를 사용하여 인증서 생성 및 갱신을 관리할 수 있습니다.

속성	값
추가 기능 이름	cert-manager
네임스페이스	cert-manager
설명서	Cert Manager 문서
서비스 계정 이름	없음
관리형 IAM 정책	없음
사용자 지정 IAM 권한	없음

외부 DNS

외부 DNS EKS 추가 기능을 사용하여 Kubernetes 리소스를 통해 Route53 DNS 레코드를 관리할 수 있습니다.

관리하려는 호스팅 영역에서 외부 DNS 권한을 `route53:ChangeResourceRecordSets`, `route53:ListHostedZones` 및 `route53:ListResourceRecordSets`로 줄일 수 있습니다.

속성	값
추가 기능 이름	external-dns
네임스페이스	external-dns
설명서	GitHub Readme
서비스 계정 이름	external-dns
관리형 IAM 정책	arn:aws:iam::aws:policy/AmazonRoute53FullAccess
사용자 지정 IAM 권한	없음

속성 보기

커뮤니티 추가 기능에 대한 오픈 소스 속성 및 라이선스 정보를 다운로드할 수 있습니다.

1. 속성을 다운로드할 추가 기능의 이름과 버전을 결정합니다.
2. 다음 명령을 이름과 버전으로 업데이트합니다.

```
curl -O https://amazon-eks-docs.s3.amazonaws.com/attributions/<add-on-name>/<add-on-version>/attributions.zip
```

예시:

```
curl -O https://amazon-eks-docs.s3.amazonaws.com/attributions/kube-state-metrics/v2.14.0-eksbuild.1/attributions.zip
```

3. 명령을 사용하여 파일을 다운로드합니다.

이 zip 파일을 사용하여 라이선스 속성에 대한 정보를 봅니다.

AWS Marketplace 추가 기능

이전 Amazon EKS 추가 기능 목록 외에 독립 소프트웨어 공급업체의 작업용 소프트웨어 Amazon EKS 추가 기능을 폭넓게 선택하여 추가할 수도 있습니다. 추가 기능과 설치 요구 사항에 대해 자세히 알아보려면 해당 추가 기능을 선택합니다.

Accuknox

추가 기능 이름은 `accuknox_kubearmor`이고 네임스페이스는 `kubearmor`입니다. Accuknox는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 KubeArmor 설명서의 [Getting Started with KubeArmor](#)를 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Akuity

추가 기능 이름은 `akuity_agent`이고 네임스페이스는 `akuity`입니다. Akuity는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 Akuity Platform 설명서의 [Installing the Akuity 에이전트 on Amazon EKS with the Akuity EKS add-on](#)을 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Calyptia

추가 기능 이름은 `calyptia_fluent-bit`이고 네임스페이스는 `calyptia-fluentbit`입니다. Calyptia는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 Calyptia 설명서 웹사이트의 [Getting Started with Calyptia Core 에이전트](#)를 참조하세요.

서비스 계정 이름

서비스 계정 이름은 `calyptia-fluentbit`입니다.

AWS 관리형 IAM 정책

이 추가 기능은 `AWSMarketplaceMeteringRegisterUsage` 관리형 정책을 사용합니다. 자세한 내용은 AWS 관리형 정책 참조 안내서의 [AWSMarketplaceMeteringRegisterUsage](#)를 참조하세요.

필수 IAM 역할을 생성하는 명령

다음 명령에는 클러스터의 IAM OpenID Connect(OIDC) 제공자가 필요합니다. 공급자가 있는지 아니면 생성해야 하는지 확인하려면 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요. *my-*

*cluster*를 클러스터 이름으로 바꾸고 *my-calyptia-role*을 역할의 이름으로 바꿉니다. 이 명령을 사용하려면 디바이스에 [eksctl](#)가 설치되어 있어야 합니다. 다른 도구를 사용하여 역할을 생성하고 Kubernetes 서비스 계정에 주석을 추가해야 하는 경우 [the section called “IAM 역할 할당”](#) 섹션을 참조하세요.

```
eksctl create iamserviceaccount --name service-account-name --namespace calyptia-fluentbit --cluster my-cluster --role-name my-calyptia-role \
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/AWSMarketplaceMeteringRegisterUsage --approve
```

Cisco Observability Collector

추가 기능 이름은 `cisco_cisco-cloud-observability-collectors`이고 네임스페이스는 `appdynamics`입니다. Cisco는 추가 기능을 게시합니다.

애드온에 대한 자세한 내용은 Cisco AppDynamics 설명서에서 [Cisco Cloud Observability AWS 마켓플레이스 애드온 사용](#)을 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Cisco Observability Operator

추가 기능 이름은 `cisco_cisco-cloud-observability-operators`이고 네임스페이스는 `appdynamics`입니다. Cisco는 추가 기능을 게시합니다.

애드온에 대한 자세한 내용은 Cisco AppDynamics 설명서에서 [Cisco Cloud Observability AWS 마켓플레이스 애드온 사용](#)을 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

CLOUDSOFT

추가 기능 이름은 `cloudsoft_cloudsoft-amp`이고 네임스페이스는 `cloudsoft-amp`입니다. CLOUDSOFT는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 CLOUDSOFT 설명서의 [Amazon EKS ADDON](#)을 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Cribl

추가 기능 이름은 `cribl_cribledge`이고 네임스페이스는 `cribledge`입니다. Cribl은 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 Cribl 설명서의 [Installing the Cribl Amazon EKS Add-on for Edge](#)를 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Dynatrace

추가 기능 이름은 `dynatrace_dynatrace-operator`이고 네임스페이스는 `dynatrace`입니다. Dynatrace는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 dynatrace 설명서의 [Kubernetes monitoring](#)을 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

다트리

추가 기능 이름은 `datree_engine-pro`이고 네임스페이스는 `datree`입니다. Datree는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 Datree 설명서의 [Amazon EKS-intergration](#)을 참조하세요.

서비스 계정 이름

서비스 계정 이름은 `datree-webhook-server-awssmp`입니다.

AWS 관리형 IAM 정책

관리형 정책은 `AWSLicenseManagerConsumptionPolicy`입니다. 자세한 내용은 AWS 관리형 정책 참조 안내서의 [AWSLicenseManagerConsumptionPolicy](#)를 참조하세요.

필수 IAM 역할을 생성하는 명령

다음 명령에는 클러스터의 IAM OpenID Connect(OIDC) 제공자가 필요합니다. 공급자가 있는지 아니면 생성해야 하는지 확인하려면 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요. *my-*

*cluster*를 클러스터 이름으로 바꾸고 *my-datree-role*을 역할의 이름으로 바꿉니다. 이 명령을 사용하려면 디바이스에 [eksctl](#)가 설치되어 있어야 합니다. 다른 도구를 사용하여 역할을 생성하고 Kubernetes 서비스 계정에 주석을 추가해야 하는 경우 [the section called “IAM 역할 할당”](#) 섹션을 참조하세요.

```
eksctl create iamserviceaccount --name datree-webhook-server-awsmp --namespace datree
--cluster my-cluster --role-name my-datree-role \
--role-only --attach-policy-arn arn:aws:iam::aws:policy/service-role/
AWSLicenseManagerConsumptionPolicy --approve
```

사용자 지정 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Datadog

추가 기능 이름은 `datadog_operator`이고 네임스페이스는 `datadog-agent`입니다. Datadog는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 [Installing the Datadog 에이전트 on Amazon EKS with the Datadog Operator Add-on](#)을 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Groundcover

추가 기능 이름은 `groundcover_agent`이고 네임스페이스는 `groundcover`입니다. groundcover가 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 groundcover 설명서의 [Installing the groundcover Amazon EKS Add-on](#)을 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Grafana Labs

추가 기능 이름은 grafana-labs_kubernetes-monitoring이고 네임스페이스는 monitoring입니다. Grafana Labs는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 Grafana Labs 설명서에서 [Configure Kubernetes Monitoring as Add-Amazon EKS](#)를 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Guance

- 게시자 - GUANCE
- 명칭 - guance_datakit
- 네임스페이스 - datakit
- 서비스 계정 이름 - 서비스 계정은 이 추가 기능에서 사용하지 않습니다.
- AWS 관리형 IAM 정책 - 관리형 정책은 이 추가 기능에서 사용하지 않습니다.
- 사용자 지정 IAM 권한 - 사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

- 설정 및 사용 지침 - Guance 설명서의 [Using Amazon EKS add-on](#)을 참조하세요.

HA Proxy

이름은 haproxy-technologies_kubernetes-ingress-ee이고 네임스페이스는 haproxy-controller입니다. HA Proxy는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 Datree 설명서의 [Amazon EKS-intergration](#)을 참조하세요.

서비스 계정 이름

서비스 계정 이름은 customer defined입니다.

AWS 관리형 IAM 정책

관리형 정책은 AWSLicenseManagerConsumptionPolicy입니다. 자세한 내용은 AWS 관리형 정책 참조 안내서의 [AWSLicenseManagerConsumptionPolicy](#)를 참조하세요.

필수 IAM 역할을 생성하는 명령

다음 명령에는 클러스터의 IAM OpenID Connect(OIDC) 제공자가 필요합니다. 공급자가 있는지 아니면 생성해야 하는지 확인하려면 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요. *my-cluster*를 클러스터 이름으로 바꾸고 *my-haproxy-role*을 역할의 이름으로 바꿉니다. 이 명령을 사용하려면 디바이스에 [eksctl](#)가 설치되어 있어야 합니다. 다른 도구를 사용하여 역할을 생성하고 Kubernetes 서비스 계정에 주석을 추가해야 하는 경우 [the section called “IAM 역할 할당”](#) 섹션을 참조하세요.

```
eksctl create iamserviceaccount --name service-account-name --namespace haproxy-controller --cluster my-cluster --role-name my-haproxy-role \
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/service-role/AWSLicenseManagerConsumptionPolicy --approve
```

사용자 지정 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Kpow

추가 기능 이름은 factorhouse_kpow이고 네임스페이스는 factorhouse입니다. Factorhouse는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 Kpow 설명서의 [AWS Marketplace LM](#)을 참조하세요.

서비스 계정 이름

서비스 계정 이름은 kpow입니다.

AWS 관리형 IAM 정책

관리형 정책은 AWSLicenseManagerConsumptionPolicy입니다. 자세한 내용은 AWS 관리형 정책 참조 안내서의 [AWSLicenseManagerConsumptionPolicy](#)를 참조하세요.

필수 IAM 역할을 생성하는 명령

다음 명령에는 클러스터의 IAM OpenID Connect(OIDC) 제공자가 필요합니다. 제공업체가 있는지 아니면 생성해야 하는지 확인하려면 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요. *my-cluster*를 클러스터 이름으로 바꾸고 *my-role-name*을 역할의 이름으로 바꿉니다. 이 명령을 사용하려면 디바이스에 [eksctl](#)이 설치되어 있어야 합니다. 다른 도구를 사용하여 역할을 생성하고 Kubernetes 서비스 계정에 주석을 추가해야 하는 경우 [the section called “IAM 역할 할당”](#) 섹션을 참조하세요.

```
eksctl create iamserviceaccount --name kpow --namespace factorhouse --cluster my-cluster --role-name my-kpow-role \
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/service-role/AWSLicenseManagerConsumptionPolicy --approve
```

사용자 지정 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Kubecost

추가 기능 이름은 kubecost_kubecost이고 네임스페이스는 kubecost입니다. Kubecost는 추가 기능을 게시합니다.

자세한 내용은 Kubecost 설명서의 [AWS Cloud Billing Integration](#)을 참조하세요.

클러스터에 [Amazon EBS가 설치된 Store Kubernetes 볼륨](#)이 있어야 하며, 그렇지 않으면 오류가 발생합니다.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Kasten

추가 기능 이름은 `kasten_k10`이고 네임스페이스는 `kasten-io`입니다. Veeam의 Kasten은 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 Kasten 설명서의 [Installing K10 on AWS using Amazon EKS Add-on](#)을 참조하세요.

클러스터에 기본 StorageClass와 함께 Amazon EBS CSI 드라이버가 설치되어 있어야 합니다.

서비스 계정 이름

서비스 계정 이름은 `k10-k10`입니다.

AWS 관리형 IAM 정책

관리형 정책은 `AWSLicenseManagerConsumptionPolicy`입니다. 자세한 내용은 AWS 관리형 정책 참조 안내서의 [AWSLicenseManagerConsumptionPolicy](#)를 참조하세요.

필수 IAM 역할을 생성하는 명령

다음 명령에는 클러스터의 IAM OpenID Connect(OIDC) 제공자가 필요합니다. 공급자가 있는지 아니면 생성해야 하는지 확인하려면 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요. *my-cluster*를 클러스터 이름으로 바꾸고 *my-kasten-role*을 역할의 이름으로 바꿉니다. 이 명령을 사용하려면 디바이스에 `eksctl`가 설치되어 있어야 합니다. 다른 도구를 사용하여 역할을 생성하고 Kubernetes 서비스 계정에 주석을 추가해야 하는 경우 [the section called “IAM 역할 할당”](#) 섹션을 참조하세요.

```
eksctl create iamserviceaccount --name k10-k10 --namespace kasten-io --cluster my-cluster --role-name my-kasten-role \
    --role-only --attach-policy-arn arn:aws:iam::aws:policy/service-role/AWSLicenseManagerConsumptionPolicy --approve
```

사용자 지정 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Kong

추가 기능 이름은 `kong_konnect-ri`이고 네임스페이스는 `kong`입니다. Kong은 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 Kong 설명서의 [Installing the Kong Gateway EKS Add-on](#)을 참조하세요.

클러스터에 [Amazon EBS가 설치된 Store Kubernetes 볼륨](#)이 있어야 하며, 그렇지 않으면 오류가 발생합니다.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

LeakSignal

추가 기능 이름은 `leaksignal_leakagent`이고 네임스페이스는 `leakagent`입니다. LeakSignal은 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 LeakSignal 설명서의 [https://www.leaksignal.com/docs/Leak에이전트/Deployment/AWS %20EKS%20Addon/\[Leak에이전트 추가 기능 설치\]](https://www.leaksignal.com/docs/Leak에이전트/Deployment/AWS%20EKS%20Addon/[Leak에이전트 추가 기능 설치])를 참조하세요.

클러스터에 [Amazon EBS가 설치된 Store Kubernetes 볼륨](#)이 있어야 하며, 그렇지 않으면 오류가 발생합니다.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

NetApp

추가 기능 이름은 `netapp_trident-operator`이고 네임스페이스는 `trident`입니다. NetApp은 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 NetApp 설명서의 [Configure the Trident EKS add-on](#)을 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

New Relic

추가 기능 이름은 `new-relic_kubernetes-operator`이고 네임스페이스는 `newrelic`입니다. New Relic은 추가 기능을 게시합니다.

애드온에 대한 자세한 내용은 [New Relic 설명서에서 EKS용 New Relic 애드온 설치하기](#)를 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Rafay

추가 기능 이름은 rafay-systems_rafay-operator이고 네임스페이스는 rafay-system입니다. Rafay는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 Rafay 설명서의 [Installing the Rafay Amazon EKS Add-on](#)을 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Rad Security

- 게시자 - RAD SECURITY
- 명칭 - rad-security_rad-security
- 네임스페이스 - ksoc
- 서비스 계정 이름 - 서비스 계정은 이 추가 기능에서 사용하지 않습니다.
- AWS 관리형 IAM 정책 - 관리형 정책은 이 추가 기능에서 사용하지 않습니다.
- 사용자 지정 IAM 권한 - 사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.
- 설정 및 사용 지침 - Rad 보안 문서에서 [AWS 마켓플레이스를 통해 Rad 설치하기](#)를 참조하세요.

SolarWinds

- 게시자 - SOLARWINDS
- 명칭 - solarwinds_swo-k8s-collector-addon
- 네임스페이스 - solarwinds
- 서비스 계정 이름 - 서비스 계정은 이 추가 기능에서 사용하지 않습니다.
- AWS 관리형 IAM 정책 - 관리형 정책은 이 추가 기능에서 사용하지 않습니다.
- 사용자 지정 IAM 권한 - 사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

- 설정 및 사용 지침 - SolarWinds 설명서의 [Monitor an Amazon EKS cluster](#)를 참조하세요.

Solo

추가 기능 이름은 solo-io_istio-distro이고 네임스페이스는 istio-system입니다. Solo는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 Solo.io 설명서의 [Installing Istio](#)를 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Snyk

- 게시자 - SNYK
- 명칭 - snyk_runtime-sensor
- 네임스페이스 - snyk_runtime-sensor
- 서비스 계정 이름 - 서비스 계정은 이 추가 기능에서 사용하지 않습니다.
- AWS 관리형 IAM 정책 - 관리형 정책은 이 추가 기능에서 사용하지 않습니다.
- 사용자 지정 IAM 권한 - 사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.
- 설치 및 사용 지침 - Snyk 사용자 문서의 [Snyk runtime sensor](#)를 참조하세요.

Stormforge

추가 기능 이름은 stormforge_optimize-Live이고 네임스페이스는 stormforge-system입니다. Stormforge는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 StormForge 설명서의 [Installing the StormForge 에이전트](#)를 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Splunk

추가 기능 이름은 `splunk_splunk-otel-collector-chart`이고 네임스페이스는 `splunk-monitoring`입니다. Splunk는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 Splunk 설명서의 [Install the Splunk add-on for Amazon EKS](#)를 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Teleport

추가 기능 이름은 `teleport_teleport`이고 네임스페이스는 `teleport`입니다. Teleport는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 Teleport 설명서의 [How Teleport Works](#)를 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Tetrate

추가 기능 이름은 `tetrate-io_istio-distro`이고 네임스페이스는 `istio-system`입니다. Tetrate Io는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 [Tetrate Istio Distro](#) 웹사이트를 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Upbound Universal Crossplane

추가 기능 이름은 `upbound_universal-crossplane`이고 네임스페이스는 `upbound-system`입니다. Upbound는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 Upbound 설명서의 [Upbound Universal Crossplane\(UXP\)](#)를 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Upwind

추가 기능 이름은 `upwind`이고 네임스페이스는 `upwind`입니다. Upwind는 추가 기능을 게시합니다.

추가 기능에 대한 자세한 내용은 [Upwind 설명서](#)를 참조하세요.

서비스 계정 이름

서비스 계정은 이 추가 기능에서 사용하지 않습니다.

AWS 관리형 IAM 정책

관리형 정책은 이 추가 기능에서 사용하지 않습니다.

사용자 지정 IAM 권한

사용자 지정 권한은 이 추가 기능에서 사용하지 않습니다.

Amazon EKS 추가 기능 생성

Amazon EKS 추가 기능은 Amazon EKS 클러스터용 추가 기능 소프트웨어입니다. 모든 Amazon EKS 추가 기능:

- 최신 보안 패치 및 버그 수정이 포함됩니다.
- Amazon EKS와 작동하도록 AWS에서 검증했습니다.
- 추가 기능 소프트웨어 관리에 필요한 작업량이 감소합니다.

eksctl, AWS Management Console 또는 AWS CLI를 사용하여 Amazon EKS 추가 기능을 생성할 수 있습니다. 추가 기능에 IAM 역할이 필요한 경우 역할 생성에 대한 자세한 내용은 [Amazon EKS 추가 기능](#)에서 특정 추가 기능에 대한 세부 정보를 참조하세요.

사전 조건

추가 기능을 생성하기 전에 다음을 완료합니다.

- 추가 기능을 만들기 전에 클러스터가 존재해야 합니다. 자세한 내용은 [the section called “클러스터 생성”](#) 섹션을 참조하세요.
- 추가 기능에 IAM 역할이 필요한지 확인합니다. 자세한 내용은 [the section called “호환성 확인”](#) 섹션을 참조하세요.
- Amazon EKS 추가 기능 버전이 클러스터와 호환되는지 확인합니다. 자세한 내용은 [the section called “호환성 확인”](#) 섹션을 참조하세요.
- 컴퓨터 또는 AWS CloudShell에 eksctl 명령줄 도구의 버전 0.190.0 이상이 설치되어 있는지 확인합니다. 자세한 내용은 eksctl 웹 사이트에서 [설치](#)를 참조하세요.

절차

eksctl, AWS Management Console 또는 AWS CLI를 사용하여 Amazon EKS 추가 기능을 생성할 수 있습니다. 추가 기능에 IAM 역할이 필요한 경우 역할 생성에 대한 자세한 내용은 [AWS에서 사용 가능한 Amazon EKS 추가 기능](#)에서 특정 추가 기능에 대한 세부 정보를 참조하세요.

추가 기능 생성(eksctl)

1. 클러스터 버전에 사용할 수 있는 추가 기능의 이름을 조회합니다. **1.32**를 클러스터의 버전으로 바꿉니다.

```
eksctl utils describe-addon-versions --kubernetes-version 1.32 | grep AddonName
```

예제 출력은 다음과 같습니다.

```
"AddonName": "aws-ebs-csi-driver",
      "AddonName": "coredns",
      "AddonName": "kube-proxy",
      "AddonName": "vpc-cni",
      "AddonName": "adot",
      "AddonName": "dynatrace_dynatrace-operator",
      "AddonName": "upbound_universal-crossplane",
      "AddonName": "teleport_teleport",
      "AddonName": "factorhouse_kpow",
      [...]
```

2. 생성하려는 추가 기능에 사용할 수 있는 버전을 조회합니다. **1.32**를 클러스터의 버전으로 바꿉니다. 버전을 조회하려는 추가 기능의 이름으로 **name-of-addon**을 바꿉니다. 이름은 이전 단계에서 반환된 이름 중 하나여야 합니다.

```
eksctl utils describe-addon-versions --kubernetes-version 1.32 --name name-of-addon |
grep AddonVersion
```

다음과 같은 출력은 vpc-cni라는 추가 기능에 대해 반환되는 내용의 예입니다. 사용할 수 있는 여러 버전이 추가 기능에 있는지 참조할 수 있습니다.

```
"AddonVersions": [
  "AddonVersion": "v1.12.0-eksbuild.1",
  "AddonVersion": "v1.11.4-eksbuild.1",
  "AddonVersion": "v1.10.4-eksbuild.1",
```

```
"AddonVersion": "v1.9.3-eksbuild.1",
```

- a. 생성하려는 추가 기능이 Amazon EKS 또는 AWS 마켓플레이스 추가 기능인지 결정합니다. AWS 마켓플레이스에는 추가 기능을 생성하려면 추가 단계를 완료해야 하는 타사 추가 기능이 있습니다.

```
eksctl utils describe-addon-versions --kubernetes-version 1.32 --name name-of-addon | grep ProductUrl
```

출력이 반환되지 않는다면 추가 기능이 Amazon EKS입니다. 출력이 반환된다면 추가 기능이 AWS 마켓플레이스 추가 기능입니다. 다음은 `teleport_teleport`라는 추가 기능의 출력입니다.

```
"ProductUrl": "https://aws.amazon.com/marketplace/pp?sku=3bda70bb-566f-4976-806c-f96faef18b26"
```

반환된 URL을 통해 AWS 마켓플레이스에서 추가 기능에 대해 자세히 알아볼 수 있습니다. 추가 기능에 구독이 필요한 경우 AWS 마켓플레이스를 통해 추가 기능을 구독할 수 있습니다. AWS 마켓플레이스에서 추가 기능을 생성하려는 경우 추가 기능 생성에 사용하는 [IAM 보안 주체에 AWSServiceRoleForAWSLicenseManagerRole](#) 서비스 연결 역할을 생성하는 권한이 있어야 합니다. IAM 개체에 권한을 할당하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 자격 증명 권한 추가 및 제거](#)를 참조하세요.

3. Amazon EKS 추가 기능을 생성합니다. 명령을 복사하고 다음과 같이 *user-data*를 교체합니다.

- *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.
- *name-of-addon*을 생성하려는 추가 기능 이름으로 바꿉니다.
- 최신 버전보다 이전 버전의 애드온을 사용하려면 *##*을 사용하려는 이전 단계의 출력에서 반환된 버전 번호로 바꾸세요.
- 추가 기능에서 서비스 계정 역할을 사용하는 경우 *111122223333*을 계정 ID로 바꾸고 *role-name*을 역할 이름으로 바꿉니다. 서비스 계정의 역할 생성에 대한 지침은 생성 중인 추가 기능의 설명서를 참조하세요. 추가 기능 목록은 [the section called “AWS 추가 기능”](#) 부분을 참조하세요. 서비스 계정 역할을 지정하려면 클러스터의 IAM OpenID Connect(OIDC) 제공자가 있어야 합니다. 클러스터의 해당 공급자가 이미 있는지 아니면 생성해야 하는지 결정하려면 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요.

추가 기능에서 서비스 계정 역할을 사용하지 않으면 `--service-account-role-arn:aws:iam::111122223333:role/role-name`을 삭제합니다.

- 이 예시 명령에서는 추가 기능의 기존 자체 관리형 버전(있는 경우)의 구성을 덮어씁니다. 기존 자체 관리형 추가 기능의 구성을 덮어쓰지 않으려면 **--force** 옵션을 제거합니다. 옵션을 제거하고 Amazon EKS 추가 기능에 기존 자체 관리형 추가 기능 구성을 덮어써야 하는 경우 Amazon EKS 추가 기능의 생성에 실패하고 충돌 해결에 도움이 되는 오류 메시지가 표시됩니다. 이 옵션을 사용하여 이러한 설정을 덮어쓰기 때문에 이 옵션을 지정하기 전에 Amazon EKS 추가 기능이 관리해야 하는 설정을 관리하지 않는지 확인합니다.

```
eksctl create addon --cluster my-cluster --name name-of-addon --version latest \
  --service-account-role-arn arn:aws:iam::111122223333:role/role-name --force
```

명령에 사용할 수 있는 모든 옵션의 목록을 참조할 수 있습니다.

```
eksctl create addon --help
```

사용할 수 있는 옵션에 대한 자세한 내용은 eksctl 설명서의 [추가 기능](#)을 참조하세요.

추가 기능 생성(AWS 콘솔)

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 좌측 탐색 창에서 클러스터를 선택합니다.
3. 추가 기능을 생성하려는 클러스터의 이름을 선택합니다.
4. 추가 기능 탭을 선택합니다.
5. 추가 기능 더 가져오기를 선택합니다.
6. 추가 기능 선택 페이지에서 클러스터에 추가할 추가 기능을 선택합니다. 필요한 만큼 Amazon EKS 애드온 및 AWS 마켓플레이스 애드온을 추가할 수 있습니다.

AWS 마켓플레이스 추가 기능의 경우 추가 기능을 생성하는 데 사용 중인 [IAM 보안 주체](#) AWS LicenseManager에서 추가 기능에 대한 자격을 읽는 권한이 있어야 합니다. AWS LicenseManager에는 AWS 리소스에서 자동으로 라이선스를 관리할 수 있는 [AWSServiceRoleForAWSLicenseManagerRole](#) SLR(서비스 연결 역할)이 필요합니다. SLR은 계정 당 한 번만 필요하므로 각 추가 기능 또는 각 클러스터에 대해 별도의 SLR을 생성할 필요가 없습니다. [IAM 보안 주체](#)에 권한을 할당하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 자격 증명 권한 추가 및 삭제](#)를 참조하세요.

설치하려는 AWS 마켓플레이스 추가 기능이 나열되지 않은 경우 페이지 번호를 클릭하여 추가 페이지 결과를 보거나 검색 상자에서 검색할 수 있습니다. 필터링 옵션에서 카테고리, 공급업체 또는

요금 모델별로 필터링한 다음에 검색 결과 중에서 추가 기능을 선택할 수도 있습니다. 설치할 추가 기능을 선택했으면 다음을 선택합니다.

7. 선택한 추가 기능 설정 구성 페이지에서 다음을 수행합니다.

- a. 구독 옵션 보기를 선택하여 구독 옵션 양식을 엽니다. 가격 세부 정보 및 법률 섹션을 검토한 다음 구독 버튼을 선택하여 계속하세요.
- b. 버전의 경우 설치하려는 버전을 선택합니다. 생성 중인 개별 추가 기능에서 다른 버전이 권장되지 않으면 최신으로 표시된 버전이 좋습니다. 추가 기능에 권장 버전이 있는지 확인하려면 생성 중인 추가 기능의 설명서를 참조하세요. 추가 기능 목록은 [the section called “AWS 추가 기능”](#) 부분을 참조하세요.
- c. 추가 기능에 대한 역할을 구성하는 데에는 EKS Pod Identity IAM 역할과 서비스 계정에 대한 IAM 역할(IRSA)이라는 두 가지 옵션이 있습니다. 원하는 옵션에 대해 아래의 적절한 단계를 따릅니다. 선택한 모든 추가 기능의 상태 아래에 구독 필요가 있으면 다음을 선택합니다. 클러스터를 생성한 후 구독하기 전까지 추가로 [해당 추가 기능을 구성](#)할 수 없습니다. 상태 아래에 구독 필요가 없는 추가 기능의 경우 다음을 수행합니다.
 - i. 서비스 계정에 대한 Pod Identity IAM 역할의 경우 기존 EKS Pod Identity IAM 역할을 사용하거나 권장 역할 생성 버튼을 사용하여 역할을 생성할 수 있습니다. 이 필드는 적절한 신뢰 정책이 있는 옵션만 제공합니다. 선택할 역할이 없는 경우에는 일치하는 신뢰 정책이 포함된 기존 역할이 없는 것입니다. 선택한 추가 기능의 서비스 계정에 대한 EKS Pod Identity IAM 역할을 구성하려면 권장 역할 생성을 선택합니다. 역할 생성 마법사가 별도의 창에서 열립니다. 마법사는 다음과 같이 역할 정보를 자동으로 채웁니다. EKS Pod Identity IAM 역할을 생성하려는 각 추가 기능에 대해 다음과 같이 IAM 마법사의 단계를 완료합니다.
 - 신뢰할 수 있는 엔터티 선택 단계에서는 EKS에 대한 AWS 서비스 옵션과 EKS-Pod Identity에 대한 사용 사례가 미리 선택되며 추가 기능에 대한 적절한 신뢰 정책이 자동으로 채워집니다. 예를 들어 역할은 [the section called “EKS Pod Identity의 이점”](#)에 설명된 대로 pods.eks.amazonaws.com IAM 보안 주체가 포함된 적절한 신뢰 정책을 사용하여 생성됩니다. Next(다음)를 선택합니다.
 - 권한 추가 단계에서 역할 정책에 대한 적절한 관리형 정책이 추가 기능에 대해 미리 선택됩니다. 예를 들어 Amazon VPC CNI 추가 기능의 경우 역할은 [the section called “Kubernetes용 Amazon VPC CNI 플러그인”](#)에 설명된 대로 관리형 정책 AmazonEKS_CNI_Policy로 생성됩니다. Next(다음)를 선택합니다.
 - 이름, 검토, 생성 단계의 역할 이름에서 기본 역할 이름은 추가 기능에 대해 자동으로 채워집니다. 예를 들어 Amazon VPC CNI 추가 기능의 경우 역할은 AmazonEKSPodIdentityAmazonVPCCNIRole 이름으로 생성됩니다. 설명에서 기본 설명은 추가 기능에 대한 적절한 설명으로 자동으로 채워집니다. 예를 들어 Amazon VPC CNI 추가 기능의 경우 역할은 Amazon EKS 클러스터에서 실행되는 포드가 AWS 리소스에 액세스하

도록 허용이라는 설명과 함께 생성됩니다. 신뢰 정책에서 추가 기능에 대해 채워진 신뢰 정책을 확인합니다. 역할 생성을 선택합니다.

참고: 기본 역할 이름을 유지하면 EKS가 새 클러스터의 추가 기능 또는 기존 클러스터에 추가 기능을 추가할 때 역할을 미리 선택할 수 있습니다. 여전히 이 이름을 재정의할 수 있으며 클러스터 전체에서 추가 기능에 역할을 사용할 수 있지만 드롭다운에서 역할을 수동으로 선택해야 합니다.

- ii. 상태에서 구독 필요가 표시되지 않고 IRSA를 사용하여 역할을 구성하려는 추가 기능은 IAM 정책을 생성하고 역할에 연결하기 위해 생성하는 추가 기능에 대한 설명서를 참조하세요. 추가 기능 목록은 [the section called “AWS 추가 기능”](#) 부분을 참조하세요. IAM 역할을 선택하려면 클러스터에 대한 IAM OpenID Connect(OIDC) 제공자가 있어야 합니다. 클러스터의 해당 공급자가 이미 있는지 아니면 생성해야 하는지 결정하려면 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요.
 - iii. 선택적 구성 설정을 선택합니다.
 - iv. 추가 기능에 구성이 필요하면 구성 값 상자에 해당 구성을 입력합니다. 추가 기능에 구성 정보가 필요한지 확인하려면 생성 중인 추가 기능의 설명서를 참조하세요. 추가 기능 목록은 [the section called “AWS 추가 기능”](#) 부분을 참조하세요.
 - v. 충돌 해결 방법에 사용할 수 있는 옵션 중 하나를 선택합니다. 충돌 해결 방법에서 재정의의 선택한 경우 기존 추가 기능에 대한 하나 이상의 설정을 Amazon EKS 추가 기능의 설정으로 덮어쓸 수 있습니다. 이 옵션을 사용 설정하지 않고 기존 설정과 충돌이 있는 경우 작업이 실패합니다. 결과 오류 메시지를 사용하여 충돌을 해결할 수 있습니다. 이 옵션을 선택하기 전에 Amazon EKS 추가 기능이 사용자가 자체 관리해야 하는 설정을 관리하지 않는지 확인하세요.
 - vi. 다음을 선택합니다.
8. 검토 및 추가 페이지에서 생성을 선택합니다. 추가 기능 설치가 완료되면 설치한 추가 기능이 표시됩니다.
 9. 설치한 추가 기능 중 구독이 필요한 추가 기능이 있으면 다음과 같은 단계를 완료합니다.
 - a. 추가 기능의 오른쪽 하단에 있는 구독 버튼을 선택합니다. AWS 마켓플레이스의 추가 기능 페이지가 나타납니다. 제품 개요 및 요금 정보와 같은 추가 기능에 대한 정보를 읽어보세요.
 - b. 추가 기능 페이지의 오른쪽 상단에 있는 계속 구독 버튼을 선택합니다.
 - c. 이용 약관을 모두 읽어보세요. 동의하는 경우, 이용 약관 수락을 선택합니다. 구독을 처리하는 데 몇 분 정도 걸릴 수 있습니다. 구독이 처리되는 동안에는 Amazon EKS 콘솔로 돌아가기 버튼이 회색으로 표시됩니다.
 - d. 구독 처리가 완료되면 Amazon EKS 콘솔로 돌아가기 버튼이 더는 회색으로 표시되지 않습니다. 버튼을 선택하여 클러스터의 Amazon EKS 콘솔 추가 기능 탭으로 돌아갑니다.

- e. 구독한 추가 기능의 경우 제거 및 재설치를 선택한 다음에 추가 기능 재설치를 선택합니다. 추가 기능을 설치하는 데 몇 분 정도 걸릴 수 있습니다. 설치가 완료되면 추가 기능을 구성할 수 있습니다.

추가 기능 생성(AWS CLI)

1. 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.
2. 사용할 수 있는 추가 기능을 확인합니다. 사용할 수 있는 모든 추가 기능, 해당 유형 및 게시자를 참조할 수 있습니다. 사용할 수 있는 추가 기능의 URL도 AWS 마켓플레이스를 통해 참조할 수 있습니다. [1.32](#)를 클러스터의 버전으로 바꿉니다.

```
aws eks describe-addon-versions --kubernetes-version 1.32 \
    --query 'addons[].{MarketplaceProductId: marketplaceInformation.productId,
    Name: addonName, Owner: owner Publisher: publisher, Type: type}' --output table
```

예제 출력은 다음과 같습니다.

```
-----
|
| DescribeAddonVersions
|
+-----+
+-----+-----+-----+
+-----+
|                               MarketplaceProductId                               |                               Name
|                               | Owner      | Publisher  |                               |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| None                        | aws      | eks        | storage      | aws-ebs-csi-driver
```

```

| None | aws | eks | networking | coredns
| None | aws | eks | networking | kube-proxy
| None | aws | eks | networking | vpc-cni
| None | aws | eks | observability | adot
| https://aws.amazon.com/marketplace/pp/prodview-brb73nceicv7u |
dynatrace_dynatrace-operator | aws-marketplace | dynatrace | monitoring
|
| https://aws.amazon.com/marketplace/pp/prodview-uhc2iwi5xysoc | upbound_universal-
crossplane | aws-marketplace | upbound | infra-management |
| https://aws.amazon.com/marketplace/pp/prodview-hd2ydsrgqy4li | teleport_teleport
| aws-marketplace | teleport | policy-management |
| https://aws.amazon.com/marketplace/pp/prodview-vgghgqdsplhvc | factorhouse_kpow
| aws-marketplace | factorhouse | monitoring |
| [...] | [...] | [...] | [...] | [...]
+-----+
+-----+-----+-----+
+-----+

```

출력은 다를 수도 있습니다. 이 예시 출력에는 `networking` 유형에 사용할 수 있는 3가지 추가 기능과 `eks` 유형의 게시자가 있는 5가지 추가 기능이 있습니다. Owner 열에 `aws-marketplace`가 있는 추가 기능을 설치하려면 구독이 필요할 수 있습니다. URL을 방문하여 추가 기능에 대해 자세히 알아보고 구독할 수 있습니다.

3. 각 추가 기능에 어떤 버전을 사용할 수 있는지 확인할 수 있습니다. **1.32**를 클러스터의 버전으로 바꾸고 **`vpc-cni`**를 이전 단계에서 반환된 추가 기능 이름으로 바꿉니다.

```
aws eks describe-addon-versions --kubernetes-version 1.32 --addon-name vpc-cni \
  --query 'addons[].addonVersions[0].{Version: addonVersion, Defaultversion:
compatibilities[0].defaultVersion}' --output table
```

예제 출력은 다음과 같습니다.

```

-----
| DescribeAddonVersions |
+-----+-----+
| Defaultversion | Version |
+-----+-----+

```

False	v1.12.0-eksbuild.1	
True	v1.11.4-eksbuild.1	
False	v1.10.4-eksbuild.1	
False	v1.9.3-eksbuild.1	
+-----+-----+		

기본적으로 Defaultversion 열에 True가 있는 버전이 추가 기능을 생성한 버전입니다.

4. (선택사항) 다음과 같은 명령을 실행하여 선택한 추가 기능의 구성 옵션을 찾습니다.

```
aws eks describe-addon-configuration --addon-name vpc-cni --addon-version v1.12.0-eksbuild.1
```

```
{
  "addonName": "vpc-cni",
  "addonVersion": "v1.12.0-eksbuild.1",
  "configurationSchema": "{ \" $ref\": \"#/definitions/VpcCni\", \" $schema\": \"http://
json-schema.org/draft-06/schema#\", \"definitions\": { \"Cri\": { \"additionalProperties
\": false, \"properties\": { \"hostPath\": { \" $ref\": \"#/definitions/HostPath\" } }, \"title
\": \"Cri\", \"type\": \"object\" }, \"Env\": { \"additionalProperties\": false, \"properties
\": { \"ADDITIONAL_ENI_TAGS\": { \"type\": \"string\" }, \"AWS_VPC_CNI_NODE_PORT_SUPPORT
\": { \"format\": \"boolean\", \"type\": \"string\" }, \"AWS_VPC_ENI_MTU\": { \"format\":
\"integer\", \"type\": \"string\" }, \"AWS_VPC_K8S_CNI_CONFIGURE_RPFILTER\": { \"format
\": \"boolean\", \"type\": \"string\" }, \"AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG\": { \"format
\": \"boolean\", \"type\": \"string\" }, \"AWS_VPC_K8S_CNI_EXTERNALSNAT\": { \"format\":
\"boolean\", \"type\": \"string\" }, \"AWS_VPC_K8S_CNI_LOGLEVEL\": { \"type\": \"string\" },
\"AWS_VPC_K8S_CNI_LOG_FILE\": { \"type\": \"string\" }, \"AWS_VPC_K8S_CNI_RANDOMIZESNAT
\": { \"type\": \"string\" }, \"AWS_VPC_K8S_CNI_VETHPREFIX\": { \"type\": \"string\" },
\"AWS_VPC_K8S_PLUGIN_LOG_FILE\": { \"type\": \"string\" }, \"AWS_VPC_K8S_PLUGIN_LOG_LEVEL
\": { \"type\": \"string\" }, \"DISABLE_INTROSPECTION\": { \"format\": \"boolean\", \"type
\": \"string\" }, \"DISABLE_METRICS\": { \"format\": \"boolean\", \"type\": \"string
\" }, \"DISABLE_NETWORK_RESOURCE_PROVISIONING\": { \"format\": \"boolean\", \"type
\": \"string\" }, \"ENABLE_POD_ENI\": { \"format\": \"boolean\", \"type\": \"string
\" }, \"ENABLE_PREFIX_DELEGATION\": { \"format\": \"boolean\", \"type\": \"string\" },
\"WARM_ENI_TARGET\": { \"format\": \"integer\", \"type\": \"string\" }, \"WARM_PREFIX_TARGET
\": { \"format\": \"integer\", \"type\": \"string\" } }, \"title\": \"Env\", \"type\":
\"object\" }, \"HostPath\": { \"additionalProperties\": false, \"properties\": { \"path
\": { \"type\": \"string\" } }, \"title\": \"HostPath\", \"type\": \"object\" }, \"Limits
\": { \"additionalProperties\": false, \"properties\": { \"cpu\": { \"type\": \"string
\" }, \"memory\": { \"type\": \"string\" } }, \"title\": \"Limits\", \"type\": \"object\" },
\"Resources\": { \"additionalProperties\": false, \"properties\": { \"limits\": { \" $ref\":
\"#/definitions/Limits\" }, \"requests\": { \" $ref\": \"#/definitions/Limits\" } }, \"title
\": \"Resources\", \"type\": \"object\" }, \"VpcCni\": { \"additionalProperties\": false,
```

```
\\"properties\\":{\\\"cri\\\":{\\\"$ref\\\":\\\"#/definitions/Cri\\\"},\\\"env\\\":{\\\"$ref\\\":\\\"#/definitions/Env\\\"},\\\"resources\\\":{\\\"$ref\\\":\\\"#/definitions/Resources\\\"}},\\\"title\\\":\\\"VpcCni\\\",\\\"type\\\":\\\"object\\\"}}}
```

출력은 표준 JSON 스키마입니다.

다음은 위의 스키마와 연동하는 유효한 구성 값의 예시입니다.

```
{
  "resources": {
    "limits": {
      "cpu": "100m"
    }
  }
}
```

다음은 위의 스키마와 연동하는 YAML 형식의 유효한 구성 값의 예시입니다.

```
resources:
  limits:
    cpu: 100m
```

5. 추가 기능에 IAM 권한이 필요한지 확인하세요. 필요하다면 (1) EKS Pod Identity를 사용할지 또는 서비스 계정용 IAM 역할(IRSA)을 사용할지 결정하고, (2) 추가 기능과 함께 사용할 IAM 역할의 ARN을 결정하고, (3) 추가 기능이 사용하는 Kubernetes 서비스 계정의 이름을 결정해야 합니다. 자세한 내용은 [the section called “IAM 정보 검색”](#) 섹션을 참조하세요.

- Amazon EKS는 추가 기능에서 지원되는 경우 EKS Pod Identity를 사용할 것을 권장합니다. 이를 위해서는 [Pod Identity 에이전트가 클러스터에 설치되어 있어야](#) 합니다. 추가 기능과 함께 Pod Identity를 사용하는 방법에 대한 자세한 내용은 [the section called “IAM 역할”](#) 섹션을 참조하세요.
- 추가 기능 또는 클러스터가 EKS Pod Identity용으로 설정되지 않은 경우 IRSA를 사용합니다. [클러스터에 IRSA 설정이 되어 있는지 확인합니다.](#)
- [Amazon EKS 추가 기능 설명서를 검토하여 추가 기능에 IAM 권한이 필요한지 여부와 관련 Kubernetes 서비스 계정의 이름을 확인하세요.](#)
 - a. Amazon EKS 추가 기능을 생성합니다. 다음 명령을 디바이스에 복사합니다. 필요에 따라 명령을 다음과 같이 수정한 다음에 수정한 명령을 실행합니다.
- **my-cluster**를 해당 클러스터의 이름으로 바꿉니다.
- 생성하려는 이전 단계의 출력에서 반환된 추가 기능 이름으로 **vpc-cni**를 바꿉니다.

- 사용하려는 이전 단계의 출력에서 반환된 버전으로 *version-number*를 바꿉니다.
- 추가 기능에 IAM 권한이 필요하지 않은 경우 *<service-account-configuration>*을 삭제합니다.
- 다음 중 하나를 수행합니다.
 - 추가 기능에 (1) IAM 권한이 필요하고 (2) 클러스터가 EKS Pod Identity를 사용하는 경우, *<service-account-configuration>*을 다음 Pod Identity 연결로 교체하세요. *<service-account-name>*를 추가 기능에 사용되는 서비스 계정 이름으로 교체합니다. *<role-arn>*를 IAM 역할의 ARN으로 교체합니다. 역할에는 EKS Pod Identity에 필요한 신뢰 정책이 있어야 합니다.

```
--pod-identity-associations 'serviceAccount=<service-account-name>,roleArn=<role-arn>'
```

- 추가 기능에 (1) IAM 권한이 필요하고 (2) 클러스터가 IRSA를 사용하는 경우, *<service-account-configuration>*를 다음 IRSA 구성으로 교체하세요. *111122223333*를 계정 ID로, *role-name*을 생성한 기존 IAM 역할 이름으로 교체합니다. 역할 생성에 대한 지침은 생성 중인 추가 기능의 설명서를 참조하세요. 추가 기능 목록은 [the section called “AWS 추가 기능”](#) 부분을 참조하세요. 서비스 계정 역할을 지정하려면 클러스터의 IAM OpenID Connect(OIDC) 제공자가 있어야 합니다. 클러스터의 해당 공급자가 이미 있는지 아니면 생성해야 하는지 결정하려면 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요.

```
--service-account-role-arn arn:aws::iam::111122223333:role/role-name
```

- 이들 예시 명령에서는 추가 기능의 기존 자체 관리형 버전(있는 경우)의 *--configuration-values* 옵션을 덮어씁니다. 이 값을 문자열이나 파일 입력과 같은 원하는 구성 값으로 바꾸세요. 구성 값을 제공하지 않으려면 *--configuration-values* 옵션을 삭제합니다. AWS CLI로 기존 자체 관리형 추가 기능의 구성을 덮어쓰지 않으려면 *--resolve-conflicts OVERWRITE* 옵션을 제거합니다. 옵션을 제거하고 Amazon EKS 추가 기능에 기존 자체 관리형 추가 기능 구성을 덮어써야 하는 경우 Amazon EKS 추가 기능의 생성에 실패하고 충돌 해결에 도움이 되는 오류 메시지가 표시됩니다. 이 옵션을 사용하여 이러한 설정을 덮어쓰기 때문에 이 옵션을 지정하기 전에 Amazon EKS 추가 기능이 관리해야 하는 설정을 관리하지 않는지 확인합니다.

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version version-number \
    <service-account-configuration> --configuration-values '{"resources": {"limits":{"cpu":"100m"}}}' --resolve-conflicts OVERWRITE
```

```
aws eks create-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
version-number \
    <service-account-configuration> --configuration-values 'file:///example.yaml' --
resolve-conflicts OVERWRITE
```

사용할 수 있는 옵션의 전체 목록은 Amazon EKS 명령줄 참조의 [create-addon](#) 부분을 참조하세요. 생성한 추가 기능에 이전 단계의 Owner 열에 나열된 aws-marketplace가 있으면 생성에 실패하고 다음과 같은 오류와 비슷한 오류 메시지가 표시될 수 있습니다.

```
{
  "addon": {
    "addonName": "addon-name",
    "clusterName": "my-cluster",
    "status": "CREATE_FAILED",
    "addonVersion": "version",
    "health": {
      "issues": [
        {
          "code": "AddonSubscriptionNeeded",
          "message": "You are currently not subscribed to this add-on. To
subscribe, visit the AWS Marketplace console, agree to the seller EULA, select the
pricing type if required, then re-install the add-on"
        }
      ]
    }
  }
}
```

이전 출력의 오류와 비슷한 오류가 표시되는 경우 이전 단계의 출력에 있는 URL을 방문하여 추가 기능을 구독합니다. 구독한 후 create-addon 명령을 다시 실행합니다.

Amazon EKS 추가 기능 업데이트

Amazon EKS에서는 새 버전이 릴리스되거나 새 Kubernetes 마이너 버전으로 클러스터를 업데이트한 후 추가 기능을 자동으로 업데이트하지 않습니다. 기존 클러스터의 추가 기능을 업데이트하려면 업데이트를 시작해야 합니다. 사용자가 업데이트를 시작하면 Amazon EKS에서 추가 기능을 업데이트합니다. 추가 기능을 업데이트하기 전에 추가 기능에 대한 최신 설명서를 검토합니다. 사용 가능한 추가 기능 목록은 [the section called “AWS 추가 기능”](#) 부분을 참조하세요. 추가 기능에 IAM 역할이 필요한 경

우 역할 생성에 대한 자세한 내용은 [AWS에서 사용 가능한 Amazon EKS 추가 기능](#)에서 특정 추가 기능에 대한 세부 정보를 참조하세요.

사전 조건

추가 기능을 생성하기 전에 다음을 완료합니다.

- 추가 기능에 IAM 역할이 필요한지 확인합니다. 자세한 내용은 [the section called “Amazon EKS 추가 기능”](#) 섹션을 참조하세요.
- Amazon EKS 추가 기능 버전이 클러스터와 호환되는지 확인합니다. 자세한 내용은 [the section called “호환성 확인”](#) 섹션을 참조하세요.

절차

eksctl, AWS Management Console 또는 AWS CLI를 사용하여 Amazon EKS 추가 기능을 업데이트할 수 있습니다.

추가 기능 업데이트(eksctl)

1. 클러스터에 설치된 현재 추가 기능과 추가 기능 버전을 확인합니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
eksctl get addon --cluster my-cluster
```

예제 출력은 다음과 같습니다.

NAME	VERSION	STATUS	ISSUES	IAMROLE	UPDATE AVAILABLE
coredns	v1.8.7-eksbuild.2	ACTIVE	0		
kube-proxy	v1.23.7-eksbuild.1	ACTIVE	0		v1.23.8-eksbuild.2
vpc-cni	v1.10.4-eksbuild.1	ACTIVE	0		v1.12.0-eksbuild.1, v1.11.4-eksbuild.1, v1.11.3-eksbuild.1, v1.11.2-eksbuild.1, v1.11.0-eksbuild.1

클러스터에 있는 추가 기능과 버전에 따라 출력이 다르게 보일 수 있습니다. 이전 예시 출력에서는 클러스터의 기존 추가 기능2가지에 사용할 수 있는 더 새로운 버전이 UPDATE AVAILABLE 열에 있는 것을 볼 수 있습니다.

2. 추가 기능을 업데이트합니다.

a. 다음 명령을 디바이스에 복사합니다. 필요에 따라 명령을 다음과 같이 수정합니다.

- *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

- `region-code`를 클러스터가 있는 AWS 리전으로 바꿉니다.
- `vpc-cni`를 업데이트하려는 이전 단계의 출력에서 반환된 애드온의 이름으로 바꿉니다.
- 사용할 수 있는 최신 버전보다 이전의 버전으로 업데이트하려는 경우에는 사용하려는 이전 단계의 출력에서 반환된 버전 번호로 `##`을 바꿉니다. 일부 추가 기능에는 권장 버전이 있습니다. 자세한 내용은 업데이트 중인 추가 기능의 설명서를 참조하세요. 추가 기능 목록은 [the section called “AWS 추가 기능”](#) 섹션을 참조하세요.* 추가 기능이 Kubernetes 서비스 계정과 IAM 역할을 사용하는 경우 `111122223333`을 계정 ID로 바꾸고 `role-name`을 생성한 기존 IAM 역할의 이름으로 바꿉니다. 역할 생성에 대한 지침은 생성 중인 추가 기능의 설명서를 참조하세요. 추가 기능 목록은 [the section called “AWS 추가 기능”](#) 부분을 참조하세요. 서비스 계정 역할을 지정하려면 클러스터의 IAM OpenID Connect(OIDC) 제공자가 있어야 합니다. 클러스터의 해당 공급자가 이미 있는지 아니면 생성해야 하는지 결정하려면 [the section called “IAM OIDC 제공자”](#) 부분을 참조하세요.

추가 기능에서 Kubernetes 서비스 계정과 IAM 역할을 사용하지 않으면

`serviceAccountRoleARN: arn:aws:iam::111122223333:role/role-name` 줄을 삭제합니다.

- `##` 옵션에서는 추가 기능의 기존 값이 보존됩니다. 추가 기능 설정에 사용자 지정 값을 설정하고 이 옵션을 사용하지 않는 경우 Amazon EKS에서는 기본값으로 해당 값을 덮어씁니다. 이 옵션을 사용하는 경우에는 추가 기능을 업데이트하기 전에 프로덕션 클러스터에서 비프로덕션 클러스터의 필드 및 값 변경 사항을 테스트하는 것이 좋습니다. 이 값을 `overwrite`로 변경하면 모든 설정이 Amazon EKS 기본값으로 변경됩니다. 설정에 사용자 지정 값을 설정한 경우 Amazon EKS 기본값으로 해당 값을 덮어쓸 수도 있습니다. 이 값을 `none`으로 변경하면 Amazon EKS에서는 설정의 값을 변경하지 않지만 업데이트에 실패할 수도 있습니다. 업데이트에 실패하면 충돌 해결에 도움이 되는 오류 메시지가 표시됩니다.

```
cat >update-addon.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig
metadata:
  name: my-cluster
  region: region-code

addons:
- name: vpc-cni
  version: latest
  serviceAccountRoleARN: arn:aws:iam::111122223333:role/role-name
  resolveConflicts: preserve
EOF
```


- b. 수정한 명령을 실행하여 `update-addon.yaml` 파일을 생성합니다.
- c. 클러스터에 구성 파일을 적용합니다.

```
eksctl update addon -f update-addon.yaml
```

추가 기능 업데이트에 대한 자세한 내용은 `eksctl` 설명서의 [추가 기능 업데이트](#)를 참조하세요.

추가 기능 업데이트(AWS 콘솔)

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 좌측 탐색 창에서 클러스터를 선택합니다.
3. 추가 기능을 업데이트하려는 클러스터의 이름을 선택합니다.
4. 추가 기능 탭을 선택합니다.
5. 업데이트하려는 추가 기능을 선택합니다.
6. 편집을 선택합니다.
7. **## ## ##** 구성 페이지에서 다음을 수행합니다.
 - a. 사용하려는 버전을 선택합니다. 추가 기능에 권장 버전이 있을 수도 있습니다. 자세한 내용은 업데이트 중인 추가 기능의 설명서를 참조하세요. 추가 기능 목록은 [the section called “AWS 추가 기능”](#) 부분을 참조하세요.
 - b. 추가 기능에 대한 역할을 구성하는 데에는 EKS Pod Identity IAM 역할과 서비스 계정에 대한 IAM 역할(IRSA)이라는 두 가지 옵션이 있습니다. 원하는 옵션에 대해 아래의 적절한 단계를 따릅니다. 선택한 모든 추가 기능의 상태 아래에 구독 필요가 있으면 다음을 선택합니다. 상태 아래에 구독 필요가 없는 추가 기능의 경우 다음을 수행합니다.
 - i. 서비스 계정에 대한 Pod Identity IAM 역할의 경우 기존 EKS Pod Identity IAM 역할을 사용하거나 권장 역할 생성 버튼을 사용하여 역할을 생성할 수 있습니다. 이 필드는 적절한 신뢰 정책이 있는 옵션만 제공합니다. 선택할 역할이 없는 경우에는 일치하는 신뢰 정책이 포함된 기존 역할이 없는 것입니다. 선택한 추가 기능의 서비스 계정에 대한 EKS Pod Identity IAM 역할을 구성하려면 권장 역할 생성을 선택합니다. 역할 생성 마법사가 별도의 창에서 열립니다. 마법사는 다음과 같이 역할 정보를 자동으로 채웁니다. EKS Pod Identity IAM 역할을 생성하려는 각 추가 기능에 대해 다음과 같이 IAM 마법사의 단계를 완료합니다.
 - 신뢰할 수 있는 엔터티 선택 단계에서는 EKS에 대한 AWS 서비스 옵션과 EKS-Pod Identity에 대한 사용 사례가 미리 선택되며 추가 기능에 대한 적절한 신뢰 정책이 자동으로 채워집니다. 예를 들어 역할은 [the section called “EKS Pod Identity의 이점”](#)에 설명된 대로

pods.eks.amazonaws.com IAM 보안 주체가 포함된 적절한 신뢰 정책을 사용하여 생성됩니다. Next(다음)를 선택합니다.

- 권한 추가 단계에서 역할 정책에 대한 적절한 관리형 정책이 추가 기능에 대해 미리 선택됩니다. 예를 들어 Amazon VPC CNI 추가 기능의 경우 역할은 [the section called “Kubernetes용 Amazon VPC CNI 플러그인”](#)에 설명된 대로 관리형 정책 AmazonEKS_CNI_Policy로 생성됩니다. Next(다음)를 선택합니다.
- 이름, 검토, 생성 단계의 역할 이름에서 기본 역할 이름은 추가 기능에 대해 자동으로 채워집니다. 예를 들어 Amazon VPC CNI 추가 기능의 경우 역할은 AmazonEKSPodIdentityAmazonVPCCNIRole 이름으로 생성됩니다. 설명에서 기본 설명은 추가 기능에 대한 적절한 설명으로 자동으로 채워집니다. 예를 들어 Amazon VPC CNI 추가 기능의 경우 역할은 Amazon EKS 클러스터에서 실행되는 포드가 AWS 리소스에 액세스하도록 허용이라는 설명과 함께 생성됩니다. 신뢰 정책에서 추가 기능에 대해 채워진 신뢰 정책을 확인합니다. 역할 생성을 선택합니다.

 Note

기본 역할 이름을 유지하면 EKS가 새 클러스터의 추가 기능 또는 기존 클러스터에 추가 기능을 추가할 때 역할을 미리 선택할 수 있습니다. 여전히 이 이름을 재정의할 수 있으며 클러스터 전체에서 추가 기능에 역할을 사용할 수 있지만 드롭다운에서 역할을 수동으로 선택해야 합니다.

- 상태에서 구독 필요가 표시되지 않고 IRSA를 사용하여 역할을 구성하려는 추가 기능은 IAM 정책을 생성하고 역할에 연결하기 위해 생성하는 추가 기능에 대한 설명서를 참조하세요. 추가 기능 목록은 [the section called “AWS 추가 기능”](#) 부분을 참조하세요. IAM 역할을 선택하려면 클러스터에 대한 IAM OpenID Connect(OIDC) 제공자가 있어야 합니다. 클러스터의 해당 공급자가 이미 있는지 아니면 생성해야 하는지 결정하려면 [the section called “IAM OIDC 제공자”](#) 섹션을 참조하세요.
- 선택적 구성 설정을 확장합니다.
 - 구성 값에 추가 기능 관련 구성 정보를 입력합니다. 자세한 내용은 업데이트 중인 추가 기능의 설명서를 참조하세요. 추가 기능 목록은 [the section called “AWS 추가 기능”](#) ... 충돌 해결 방법의 경우 옵션 중 하나를 선택합니다. 추가 기능 설정에 사용자 지정 값을 설정한 경우, 보존 옵션이 좋습니다. 이 옵션을 선택하지 않으면 Amazon EKS에서 기본값으로 해당 값을 덮어씁니다. 이 옵션을 사용하는 경우에는 추가 기능을 업데이트하기 전에 프로덕션 클러스터에서 비프로덕션 클러스터의 필드 및 값 변경 사항을 테스트하는 것이 좋습니다. 이 값을 overwrite로 변경하면 모든 설정이 Amazon EKS 기본값으로 변경됩니다. 설정에 사용자 지정 값을 설정한 경우 Amazon EKS 기본값으로 해당 값을 덮어쓸 수도 있습니다. 이 값을 none으로 변경하면 Amazon EKS에

서는 설정의 값을 변경하지 않지만 업데이트에 실패할 수도 있습니다. 업데이트에 실패하면 충돌 해결에 도움이 되는 오류 메시지가 표시됩니다.

8. Save changes(변경 사항 저장)를 선택합니다.

추가 기능 업데이트(AWS CLI)

1. 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.
2. 설치한 추가 기능 목록을 참조합니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks list-addons --cluster-name my-cluster
```

예제 출력은 다음과 같습니다.

```
{
  "addons": [
    "coredns",
    "kube-proxy",
    "vpc-cni"
  ]
}
```

3. 업데이트할 추가 기능의 현재 버전을 조회합니다. *my-cluster*를 클러스터 이름으로 바꾸고 업데이트할 추가 기능의 이름으로 *vpc-cni*를 바꿉니다.

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni --query "addon.addonVersion" --output text
```

예제 출력은 다음과 같습니다.

```
v1.10.4-eksbuild.1
```

4. 클러스터 버전에 사용할 수 있는 추가 기능 버전을 확인합니다. **1.32**를 클러스터의 버전으로 바꾸고 업데이트할 추가 기능의 이름으로 **vpc-cni**를 바꿉니다.

```
aws eks describe-addon-versions --kubernetes-version 1.32 --addon-name vpc-cni \
  --query 'addons[].addonVersions[].[Version: addonVersion, Defaultversion:
  compatibilities[0].defaultVersion]' --output table
```

예제 출력은 다음과 같습니다.

```
-----+-----+
| DescribeAddonVersions |
+-----+-----+
| Defaultversion | Version |
+-----+-----+
| False | v1.12.0-eksbuild.1 |
| True | v1.11.4-eksbuild.1 |
| False | v1.10.4-eksbuild.1 |
| False | v1.9.3-eksbuild.1 |
+-----+-----+
```

기본적으로 Defaultversion 열에 True가 있는 버전이 추가 기능을 생성한 버전입니다.

5. 추가 기능을 업데이트합니다. 다음 명령을 디바이스에 복사합니다. 필요에 따라 명령을 다음과 같이 수정한 다음에 수정한 명령을 실행합니다. 이 명령에 대한 자세한 내용은 Amazon EKS 명령줄 참조의 [update-addon](#)을 참조하세요.

- **my-cluster**를 해당 클러스터의 이름으로 바꿉니다.
- **vpc-cni**를 이전 단계의 출력에서 반환된 업데이트하려는 애드온의 이름으로 바꿉니다.
- 업데이트하려는 이전 단계의 출력에서 반환된 버전으로 **version-number**를 바꿉니다. 일부 추가 기능에는 권장 버전이 있습니다. 자세한 내용은 업데이트 중인 추가 기능의 설명서를 참조하세요. 추가 기능 목록은 [the section called “AWS 추가 기능”](#) 섹션을 참조하세요.* 추가 기능이 Kubernetes 서비스 계정과 IAM 역할을 사용하는 경우 **111122223333**을 계정 ID로 바꾸고 **role-name**을 생성한 기존 IAM 역할의 이름으로 바꿉니다 역할 생성에 대한 지침은 생성 중인 추가 기능의 설명서를 참조하세요. 추가 기능 목록은 [the section called “AWS 추가 기능”](#) 부분을 참조하세요. 서비스 계정 역할을 지정하려면 클러스터의 IAM OpenID Connect(OIDC) 제공자가 있어야 합니다. 클러스터의 해당 공급자가 이미 있는지 아니면 생성해야 하는지 결정하려면 [the section called “IAM OIDC 제공자”](#) 부분을 참조하세요.

추가 기능에서 Kubernetes 서비스 계정과 IAM 역할을 사용하지 않으면

`serviceAccountRoleARN: arn:aws:iam::111122223333:role/role-name` 줄을 삭제합니다.

- `--resolve-conflicts PRESERVE` 옵션은 추가 기능의 기존 값을 보존합니다. 추가 기능 설정에 사용자 지정 값을 설정하고 이 옵션을 사용하지 않는 경우 Amazon EKS에서는 기본값으로 해당 값을 덮어씁니다. 이 옵션을 사용하는 경우에는 추가 기능을 업데이트하기 전에 프로덕션 클러스터에서 비프로덕션 클러스터의 필드 및 값 변경 사항을 테스트하는 것이 좋습니다. 이 값을 `OVERWRITE`로 변경하면 모든 설정이 Amazon EKS 기본값으로 변경됩니다. 설정에 사용자 지정 값을 설정한 경우 Amazon EKS 기본값으로 해당 값을 덮어쓸 수도 있습니다. 이 값을 `NONE`으로 변경하면 Amazon EKS에서는 설정의 값을 변경하지 않지만 업데이트에 실패할 수도 있습니다. 업데이트에 실패하면 충돌 해결에 도움이 되는 오류 메시지가 표시됩니다.
- 모든 사용자 지정 구성을 제거하려면 `--configuration-values '{}'` 옵션을 사용하여 업데이트를 수행하세요. 그러면 모든 사용자 지정 구성이 기본값으로 되돌아갑니다. 사용자 지정 구성을 변경하지 않으려는 경우에는 `--configuration-values` 플래그를 제공하지 마세요. 사용자 지정 구성을 조정하려는 경우에는 `{}` 부분을 새 파라미터로 바꿉니다.

```
aws eks update-addon --cluster-name my-cluster --addon-name vpc-cni --addon-version
version-number \
    --service-account-role-arn arn:aws:iam::111122223333:role/role-name --
configuration-values '{}' --resolve-conflicts PRESERVE
```

6. 업데이트 상태를 확인합니다. `my-cluster`를 클러스터 이름으로 바꾸고 업데이트 중인 추가 기능의 이름으로 `vpc-cni`를 바꿉니다.

```
aws eks describe-addon --cluster-name my-cluster --addon-name vpc-cni
```

예제 출력은 다음과 같습니다.

```
{
  "addon": {
    "addonName": "vpc-cni",
    "clusterName": "my-cluster",
    "status": "UPDATING",
  }
}
```

상태가 `ACTIVE`라면 업데이트가 완료된 것입니다.

Amazon EKS 추가 기능 버전의 클러스터와의 호환성 확인

Amazon EKS 추가 기능을 생성하기 전에 Amazon EKS 추가 기능 버전이 클러스터와 호환되는지 확인해야 합니다.

[describe-addon-versions API](#)를 사용하여 EKS 추가 기능의 사용 가능한 버전과 각 추가 기능 버전에서 지원하는 Kubernetes 버전을 나열합니다.

1. `aws sts get-caller-identity` 명령으로 AWS CLI가 설치되어 작동하고 있는지 확인합니다. 이 명령이 작동하지 않는 경우 [AWS CLI 시작](#) 방법을 알아봅니다.
2. 버전 호환성 정보를 검색하려는 추가 기능의 이름(예: `amazon-cloudwatch-observability`)을 확인합니다.
3. 클러스터의 Kubernetes 버전(예: `1.32`)을 확인합니다.
4. AWS CLI를 사용하여 클러스터의 Kubernetes 버전과 호환되는 추가 기능 버전을 검색합니다.

```
aws eks describe-addon-versions --addon-name amazon-cloudwatch-observability --  
kubernetes-version 1.32
```

예제 출력은 다음과 같습니다.

```
{  
  "addons": [  
    {  
      "addonName": "amazon-cloudwatch-observability",  
      "type": "observability",  
      "addonVersions": [  
        {  
          "addonVersion": "vX.X.X-eksbuild.X",  
          "architecture": [  
            "amd64",  
            "arm64"  
          ],  
          "computeTypes": [  
            "ec2",  
            "auto",  
            "hybrid"  
          ],  
          "compatibilities": [  
            {  
              "clusterVersion": "1.32",
```

```

        "platformVersions": [
            "*"
        ],
        "defaultVersion": true
    }
},
]
}
]
}

```

이 출력은 추가 기능 버전 vX.X.X-eksbuild.X이 Kubernetes 클러스터 버전 1.32과 호환됨을 보여줍니다.

컴퓨팅 유형과의 추가 기능 호환성

describe-addon-versions 출력의 computeTypes 필드는 추가 기능이 EKS Auto Mode 관리형 노드 또는 하이브리드 노드와 호환됨을 나타냅니다. auto로 표시된 추가 기능은 EKS Auto Mode의 클라우드 기반 AWS 관리형 인프라에서 작동하는 반면 hybrid로 표시된 추가 기능은 EKS 클라우드 컨트롤 플레인에 연결된 온프레미스 노드에서 실행 가능합니다.

자세한 내용은 [the section called “Amazon EKS Auto Mode 고려 사항”](#) 단원을 참조하십시오.

클러스터에서 Amazon EKS 추가 기능 제거

eksctl, AWS Management Console 또는 AWS CLI를 사용하여 클러스터에서 Amazon EKS 추가 기능을 제거할 수 있습니다.

클러스터에서 Amazon EKS 추가 기능을 제거하는 경우:

- 추가 기능에서 제공되는 기능의 가동 중지 시간이 없습니다.
- 서비스 계정용 IAM 역할(IRSA)을 사용하고 추가 기능에 연결된 IAM 역할이 있는 경우, IAM 역할은 제거되지 않습니다.
- Pod Identity를 사용하는 경우 추가 기능이 소유한 모든 Pod Identity 연결이 제거됩니다. AWS CLI에 `--preserve` 옵션을 지정하면 연결이 보존됩니다.
- Amazon EKS에서는 추가 기능의 설정 관리를 중지합니다.
- 새 버전이 사용할 수 있을 때 콘솔의 알림 표시가 중지됩니다.
- AWS 도구 또는 API를 사용하여 추가 기능을 업데이트할 수 없습니다.

- 자체 관리할 수 있도록 클러스터에 추가 기능 소프트웨어 남겨두기를 선택하거나 클러스터에서 추가 기능 소프트웨어를 제거할 수 있습니다. 추가 기능에서 제공되는 기능에 종속되는 리소스가 클러스터에 없는 경우에만 클러스터에서 추가 기능 소프트웨어를 제거해야 합니다.

사전 조건

추가 기능을 생성하기 전에 다음을 완료합니다.

- 기존 Amazon EKS 클러스터. 배포하려면 [시작](#) 섹션을 참조하세요.
- 추가 기능에 IAM 역할이 필요한지 확인합니다. 자세한 내용은 단원을 참조하세요.
- 장치에 설치된 eksctl 명령줄 도구의 버전 0.210.0 이상 또는 AWS CloudShell이 필요합니다. eksctl을 설치 또는 업그레이드하려면 eksctl 설명서에서 [설치](#)를 참조하세요.

절차

Amazon EKS 추가 기능을 제거하는 경우 다음 두 가지 옵션이 있습니다.

- 클러스터에 추가 기능 소프트웨어 보존 – 이 옵션은 모든 설정에 대한 Amazon EKS 관리를 제거합니다. 또한 업데이트를 시작한 후 Amazon EKS가 업데이트를 알리고 Amazon EKS 추가 기능을 자동으로 업데이트하는 기능을 제거합니다. 그러나 클러스터에 추가 기능 소프트웨어는 유지됩니다. 이 옵션을 사용하면 추가 기능을 Amazon EKS 추가 기능이 아닌 자체 관리형 설치 기능으로 만들 수 있습니다. 이 옵션을 사용하면 추가 기능에 대한 가동 중지 시간이 없습니다.
- 클러스터에서 추가 기능 소프트웨어 완전히 제거(Remove the add-on software entirely from your cluster) - 클러스터에 종속된 리소스가 없는 경우,에만 클러스터에서 Amazon EKS 추가 기능을 제거하는 것이 좋습니다.

eksctl, AWS Management Console 또는 AWS CLI를 사용하여 Amazon EKS 추가 기능을 제거할 수 있습니다.

추가 기능 제거(eksctl)

1. 클러스터에 설치된 현재 추가 기능을 확인합니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
eksctl get addon --cluster my-cluster
```

예제 출력은 다음과 같습니다.

NAME	VERSION	STATUS	ISSUES	IAMROLE	UPDATE AVAILABLE
coredns	v1.8.7-eksbuild.2	ACTIVE	0		
kube-proxy	v1.23.7-eksbuild.1	ACTIVE	0		
vpc-cni	v1.10.4-eksbuild.1	ACTIVE	0		
[...]					

클러스터에 있는 추가 기능과 버전에 따라 출력이 다르게 보일 수 있습니다.

2. 추가 기능을 제거합니다. *my-cluster*를 클러스터의 이름으로 바꾸고 제거하려는 이전 단계의 출력에서 반환된 추가 기능의 이름으로 *name-of-addon*을 바꿉니다. *--##* 옵션을 제거하면 Amazon EKS에서 더는 추가 기능을 관리하지 않을 뿐만 아니라 클러스터에서 추가 기능 소프트웨어가 삭제됩니다.

```
eksctl delete addon --cluster my-cluster --name name-of-addon --preserve
```

추가 기능 업데이트에 대한 자세한 내용은 eksctl 설명서의 [추가 기능 삭제](#)를 참조하세요.

추가 기능 제거(AWS 콘솔)

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 좌측 탐색 창에서 클러스터를 선택합니다.
3. Amazon EKS 추가 기능을 제거하려는 클러스터의 이름을 선택합니다.
4. 추가 기능 탭을 선택합니다.
5. 제거하려는 추가 기능을 선택합니다.
6. 제거를 선택합니다.
7. 제거: **## ## ##** 확인 대화 상자에서 다음을 수행합니다.
 - a. 추가 기능에 대한 Amazon EKS의 설정 관리를 중지하려면 클러스터에 보존을 선택합니다. 클러스터에 추가 소프트웨어를 유지하려면 이렇게 합니다. 이는 추가 기능의 모든 설정을 스스로 관리할 수 있도록 하기 위한 것입니다.
 - b. 추가 기능 이름을 입력합니다.
 - c. 제거를 선택합니다.

추가 기능 제거(AWS CLI)

1. 장치에 설치된 eksctl 명령줄 도구의 버전 0.210.0 이상 또는 AWS CloudShell이 필요합니다. eksctl을 설치 또는 업그레이드하려면 eksctl 설명서에서 [설치](#)를 참조하세요.
2. 설치한 추가 기능 목록을 참조합니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks list-addons --cluster-name my-cluster
```

예제 출력은 다음과 같습니다.

```
{
  "addons": [
    "coredns",
    "kube-proxy",
    "vpc-cni",
    "name-of-addon"
  ]
}
```

3. 설치한 추가 기능을 제거합니다. *my-cluster*를 클러스터 이름으로 바꾸고 제거할 추가 기능의 이름으로 *name-of-addon*을 바꿉니다. *--##*을 제거하면 클러스터에서 추가 기능 소프트웨어가 삭제됩니다.

```
aws eks delete-addon --cluster-name my-cluster --addon-name name-of-addon --preserve
```

축약한 예시 출력은 다음과 같습니다.

```
{
  "addon": {
    "addonName": "name-of-add-on",
    "clusterName": "my-cluster",
    "status": "DELETING",
  }
}
```

4. 제거의 상태를 확인합니다. *my-cluster*를 클러스터 이름으로 바꾸고 제거할 추가 기능의 이름으로 *name-of-addon*을 바꿉니다.

```
aws eks describe-addon --cluster-name my-cluster --addon-name name-of-addon
```

추가 기능이 제거된 후의 예시 출력은 다음과 같습니다.

```
An error occurred (ResourceNotFoundException) when calling the DescribeAddon
operation: No addon: name-of-addon found in cluster: my-cluster
```

Amazon EKS 추가 기능에 대한 IAM 역할

특정 Amazon EKS 추가 기능은 AWS API를 호출하기 위해 IAM 역할과 권한이 필요합니다. 예를 들어, Amazon VPC CNI 추가 기능은 특정 AWS API를 호출하여 계정의 네트워킹 리소스를 구성합니다. 이러한 추가 기능에는 IAM을 사용하여 권한을 부여해야 합니다. 좀 더 구체적으로 설명하자면, 추가 기능을 실행하는 포드의 서비스 계정을 특정 IAM 정책이 있는 IAM 역할과 연결해야 합니다.

클러스터 워크로드에 AWS 권한을 부여하는 방법으로 Amazon EKS 기능인 Pod Identity 사용을 권장합니다. Pod Identity 연결을 사용하여 추가 기능의 서비스 계정을 IAM 역할에 매핑할 수 있습니다. 포드가 연결이 있는 서비스 계정을 사용하는 경우 Amazon EKS는 포드의 컨테이너에 환경 변수를 설정합니다. 환경 변수는 AWS CLI를 포함한 AWS SDK를 구성하여 EKS Pod Identity 보안 인증 정보를 사용합니다. 자세한 내용은 [the section called “Pod Identity”](#) 섹션을 참조하세요.

Amazon EKS 추가 기능은 추가 기능에 해당하는 Pod Identity 연결의 수명 주기를 관리하는 데 도움이 될 수 있습니다. 예를 들어, Amazon EKS 추가 기능과 필요한 Pod Identity 연결을 단일 API 호출로 생성하거나 업데이트할 수 있습니다. Amazon EKS는 제안된 IAM 정책을 검색하기 위한 API도 제공합니다.

1. [Amazon EKS Pod Identity 에이전트](#)가 클러스터에 설치되어 있는지 확인합니다.
2. describe-addon-versions AWS CLI 작업을 사용하여 설치하려는 추가 기능에 IAM 권한이 필요한지 확인하세요. requiresIamPermissions 플래그가 true인 경우 describe-addon-configurations 작업을 사용하여 추가 기능에 필요한 권한을 결정해야 합니다. 응답에는 권장 관리형 IAM 정책 목록이 포함됩니다.
3. describe-addon-configuration CLI 작업을 사용하여 Kubernetes 서비스 계정의 이름과 IAM 정책을 검색합니다. 보안 요구 사항을 기준으로 제안된 정책의 범위를 평가합니다.
4. 제안된 권한 정책과 Pod Identity에 필요한 신뢰 정책을 사용하여 IAM 역할을 생성합니다. 자세한 내용은 [the section called “pod identity 연결 생성\(AWS 콘솔\)”](#) 섹션을 참조하세요.
5. CLI를 사용하여 Amazon EKS 추가 기능을 생성 또는 업데이트합니다. Pod Identity 연결을 하나 이상 지정합니다. Pod Identity 연결은 Kubernetes 서비스 계정의 이름과 IAM 역할의 ARN입니다.
 - 추가 기능 API를 사용하여 생성된 Pod Identity 연결은 해당 추가 기능이 소유합니다. 추가 기능을 삭제하면 Pod Identity 연결도 삭제됩니다. AWS CLI 또는 API를 사용하여 추가 기능을 삭제할 때

preserve 옵션을 사용하면 이러한 연속 삭제를 방지할 수 있습니다. 필요한 경우 Pod Identity 연결을 직접 업데이트하거나 삭제할 수도 있습니다. 추가 기능은 기존 Pod Identity 연결의 소유권을 수임할 수 없습니다. 추가 기능 생성 또는 업데이트 작업을 사용하여 기존 연결을 삭제하고 다시 생성해야 합니다.

- Amazon EKS는 Pod Identity 연결을 사용하여 추가 기능에 대한 IAM 권한을 관리할 것을 권장합니다. 이전 방법인 서비스 계정에 대한 IAM 역할(IRSA)은 여전히 지원됩니다. 추가 기능에 대해 IRSA `serviceAccountRoleArn` 및 Pod Identity 연결을 모두 지정할 수 있습니다. EKS Pod Identity 에이전트가 클러스터에 설치된 경우 `serviceAccountRoleArn`은 무시되고 EKS는 제공된 Pod Identity 연결을 사용합니다. Pod Identity가 활성화되지 않은 경우, `serviceAccountRoleArn`이 사용됩니다.
- 기존 추가 기능의 Pod Identity 연결을 업데이트하면 Amazon EKS가 추가 기능 포드의 롤링 재시작을 시작합니다.

Amazon EKS 추가 기능에 대한 IAM 정보 검색

추가 기능을 생성하기 전에 AWS CLI를 사용하여 다음을 확인하세요.

- 추가 기능에 IAM 권한이 필요한지 여부
- 사용할 제안된 IAM 정책

절차

1. 설치하려는 추가 기능의 이름과 클러스터의 Kubernetes 버전을 결정합니다. 추가 기능에 대한 자세한 내용은 [the section called “Amazon EKS 추가 기능”](#) 섹션을 참조하세요.
2. AWS CLI를 사용하여 추가 기능에 IAM 권한이 필요한지 확인할 수 있습니다.

```
aws eks describe-addon-versions \
--addon-name <addon-name> \
--kubernetes-version <kubernetes-version>
```

예시:

```
aws eks describe-addon-versions \
--addon-name aws-ebs-csi-driver \
--kubernetes-version 1.30
```

다음 샘플 출력을 검토하세요. 또한 `requiresIamPermissions`는 `true`이며 기본 추가 기능 버전입니다. 권장 IAM 정책을 검색할 때 추가 기능 버전을 지정해야 합니다.

```
{
  "addons": [
    {
      "addonName": "aws-ebs-csi-driver",
      "type": "storage",
      "addonVersions": [
        {
          "addonVersion": "v1.31.0-eksbuild.1",
          "architecture": [
            "amd64",
            "arm64"
          ],
          "compatibilities": [
            {
              "clusterVersion": "1.30",
              "platformVersions": [
                "*"
              ],
              "defaultVersion": true
            }
          ],
          "requiresConfiguration": false,
          "requiresIamPermissions": true
        }
      ],
      "requiresConfiguration": false,
      "requiresIamPermissions": true
    },
    [...]
  ]
}
```

3. 추가 기능에 IAM 권한이 필요한 경우 AWS CLI를 사용하여 권장 IAM 정책을 검색하세요.

```
aws eks describe-addon-configuration \
--query podIdentityConfiguration \
--addon-name <addon-name> \
--addon-version <addon-version>
```

예시:

```
aws eks describe-addon-configuration \
--query podIdentityConfiguration \
--addon-name aws-ebs-csi-driver \
```

```
--addon-version v1.31.0-eksbuild.1
```

다음 출력을 검토하세요. `recommendedManagedPolicies`을 기록합니다.

```
[
  {
    "serviceAccount": "ebs-csi-controller-sa",
    "recommendedManagedPolicies": [
      "arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy"
    ]
  }
]
```

4. IAM 역할을 생성하고 관리형 정책을 연결합니다. 또는 관리형 정책을 검토하고 필요에 따라 권한 범위를 좁힐 수도 있습니다. 자세한 내용은 [the section called “pod identity 연결 생성\(AWS 콘솔\)”](#)을 참조하세요.

Pod Identity 지원 참조

다음 표는 특정 Amazon EKS 추가 기능이 EKS Pod Identity를 지원하는지 여부를 나타냅니다.

추가 기능 이름	Pod Identity 지원	필요한 최소 버전
Amazon EBS CSI 드라이버	예	v1.26.0-eksbuild.1
Amazon VPC CNI	예	v1.15.5-eksbuild.1
Amazon EFS CSI 드라이버	예	v2.0.5-eksbuild.1
AWS Distro for OpenTelemetry	예	v0.94.1-eksbuild.1
Mountpoint for Amazon S3 CSI 드라이버	아니요	N/A
Amazon CloudWatch Observability 에이전트	예	v3.1.0-eksbuild.1

이 표는 2024년 10월 28일에 마지막으로 업데이트되었습니다.

Pod Identity를 사용하여 Amazon EKS 추가 기능에 IAM 역할 할당

특정 Amazon EKS 추가 기능에는 IAM 역할과 권한이 필요합니다. Pod Identity 연결을 사용하기 위해 Amazon EKS 추가 기능을 업데이트하기 전에 사용할 역할과 정책을 확인합니다. 자세한 내용은 [the section called “IAM 정보 검색”](#) 섹션을 참조하세요.

1. 결정 필요:

- `cluster-name` - 추가 기능을 설치할 클러스터의 이름입니다.
- `addon-name` - 설치할 추가 기능의 이름입니다.
- `service-account-name` - 추가 기능이 사용하는 Kubernetes 서비스 계정의 이름입니다.
- `iam-role-arn` - 추가 기능에 대한 충분한 권한이 있는 IAM 역할의 ARN입니다. 역할에는 EKS Pod Identity에 필요한 신뢰 정책이 있어야 합니다. 자세한 정보는 [the section called “pod identity 연결 생성\(AWS 콘솔\)”](#) 섹션을 참조하세요.

2. AWS CLI를 사용하여 추가 기능을 업데이트합니다. 추가 기능을 생성할 때 동일한 `--pod-identity-associations` 구문을 사용하여 Pod Identity 연결을 지정할 수도 있습니다. 추가 기능을 업데이트하는 동안 Pod Identity 연결을 지정하면 이전의 모든 Pod Identity 연결이 덮어쓰여진다는 점에 유의하세요.

```
aws eks update-addon --cluster-name <cluster-name> \
--addon-name <addon-name> \
--pod-identity-associations 'serviceAccount=<service-account-name>,roleArn=<role-arn>'
```

예시:

```
aws eks update-addon --cluster-name mycluster \
--addon-name aws-ebs-csi-driver \
--pod-identity-associations 'serviceAccount=ebs-csi-controller-sa,roleArn=arn:aws:iam::123456789012:role/StorageDriver'
```

3. Pod Identity 연결이 생성되었는지 확인합니다.

```
aws eks list-pod-identity-associations --cluster-name <cluster-name>
```

성공한 경우 다음과 유사한 출력 화면이 표시되어야 합니다. EKS 추가 기능의 OwnerARN을 기록해 둡니다.

```
{
```

```

"associations": [
  {
    "clusterName": "mycluster",
    "namespace": "kube-system",
    "serviceAccount": "ebs-csi-controller-sa",
    "associationArn": "arn:aws:eks:us-
west-2:123456789012:podidentityassociation/mycluster/a-4wvljrezsukshq1bv",
    "associationId": "a-4wvljrezsukshq1bv",
    "ownerArn": "arn:aws:eks:us-west-2:123456789012:addon/mycluster/aws-ebs-
csi-driver/9cc7ce8c-2e15-b0a7-f311-426691cd8546"
  }
]
}

```

Amazon EKS 추가 기능에서 Pod Identity 연결 제거

Amazon EKS 추가 기능에서 Pod Identity 연결을 제거합니다.

1. 결정 필요:

- `cluster-name` - 추가 기능을 설치할 EKS 클러스터의 이름입니다.
- `addon-name` - 설치할 Amazon EKS 추가 기능의 이름입니다.

2. 추가 기능을 업데이트하여 Pod Identity 연결의 빈 배열을 지정합니다.

```

aws eks update-addon --cluster-name <cluster-name> \
--addon-name <addon-name> \
--pod-identity-associations "[]"

```

EKS 추가 기능의 Pod Identity 문제 해결

AWS API, SDK 또는 CLI 작업을 시도하는 동안 추가 기능에 오류가 발생하는 경우 다음 사항을 확인하세요.

- Pod Identity 에이전트가 클러스터에 설치되어 있습니다.
 - Pod Identity 에이전트 설치 방법에 대한 자세한 내용은 [the section called “에이전트 설정”](#) 섹션을 참조하세요.
- 추가 기능에 유효한 Pod Identity 연결이 있습니다.
 - AWS CLI를 사용하여 추가 기능이 사용하는 서비스 계정 이름에 대한 연결을 검색할 수 있습니다.


```
aws eks list-pod-identity-associations --cluster-name <cluster-name>
```

- IAM 역할에는 Pod Identity에 필요한 신뢰 정책이 있습니다.
- AWS CLI를 사용하여 추가 기능에 대한 신뢰 정책을 검색할 수 있습니다.

```
aws iam get-role --role-name <role-name> --query Role.AssumeRolePolicyDocument
```

- IAM 역할에는 추가 기능에 필요한 권한이 있습니다.
- AWS CloudTrail을 사용하여 AccessDenied 또는 UnauthorizedOperation 이벤트를 검토합니다.
- Pod Identity 연결의 서비스 계정 이름이 추가 기능이 사용하는 서비스 계정 이름과 일치합니다.
- 사용 가능한 추가 기능에 대한 자세한 내용은 [the section called “AWS 추가 기능”](#) 섹션을 참조하세요.
- 이름이 pod-identity-webhook인 MutatingWebhookConfiguration의 구성 확인
- 웹훅의 admissionReviewVersions는 v1beta1이어야 하고 v1에서는 작동하지 않습니다.

Amazon EKS 추가 기능에 대해 사용자 지정할 수 있는 필드 확인

Amazon EKS 추가 기능은 표준 모범 사례 구성을 사용하여 클러스터에 설치됩니다. Amazon EKS 추가 기능을 클러스터에 추가하는 방법에 대한 자세한 내용은 [the section called “Amazon EKS 추가 기능”](#) 섹션을 참조하세요.

고급 기능을 사용하도록 Amazon EKS 추가 기능의 구성을 사용자 지정하려 할 수 있습니다. Amazon EKS는 Kubernetes 서버 측 적용 기능을 사용하여 Amazon EKS가 관리하지 않는 설정에 대한 구성을 덮어쓰지 않고도 Amazon EKS의 추가 기능을 관리할 수 있습니다. 자세한 내용은 Kubernetes 문서의 [서버 측 적용](#)을 참조하세요. 이를 위해 Amazon EKS는 설치하는 모든 추가 기능에 대해 최소한의 필드 세트를 관리합니다. Amazon EKS가 관리하지 않는 모든 필드 또는 다른 Kubernetes 컨트롤 플레인 프로세스(예: kube-controller-manager)를 문제없이 수정할 수 있습니다.

Important

Amazon EKS가 관리하는 필드를 수정하면 Amazon EKS가 추가 기능을 관리할 수 없으며, 추가 기능이 업데이트될 때 변경 내용이 덮어써질 수 있습니다.

필드 관리 구문

Kubernetes 객체에 대한 세부 정보를 볼 때 관리형 및 비관리형 필드 모두 출력에 반환됩니다. 관리형 필드는 다음 유형 중 하나일 수 있습니다.

- 완전 관리형 - 필드의 모든 키를 Amazon EKS에서 관리합니다. 어떤 값이든 수정하면 충돌이 발생합니다.
- 부분 관리형 - 필드의 일부 키를 Amazon EKS에서 관리합니다. Amazon EKS에서 명시적으로 관리하는 키를 수정하면 충돌이 발생합니다.

두 유형의 필드 모두 `manager: eks`를 사용하여 태그가 지정됩니다.

각 키는 항상 빈 세트에 매핑되는 필드 자체를 나타내는 `.`이거나, 하위 필드 또는 항목을 나타내는 문자열입니다. 필드 관리에 대한 출력은 다음과 같은 유형의 선언으로 구성됩니다.

- `f:name` , 여기서 `##`은 목록의 필드 이름입니다.
- `k:keys` , 여기서 `#`는 목록 항목의 필드 맵입니다.
- `v:value` , 여기서 `#`은 목록 항목의 정확한 JSON 형식 값입니다.
- `i:index` , 여기서 `###`는 목록에서 항목의 위치입니다.

CoreDNS 추가 기능에 대한 다음 출력 부분은 이전 선언을 보여줍니다.

- 완전관리형 필드(Fully managed fields) - 관리형 필드에 `f:(필드)`가 지정되고 `k:(키)`가 지정되지 않은 경우 전체 필드가 관리됩니다. 이 필드의 값을 수정하면 충돌이 발생합니다.

다음 출력에서는 `coredns`라는 컨테이너가 `eks`에 의해 관리되는 것을 확인할 수 있습니다. `args`, `image` 및 `imagePullPolicy` 하위 필드도 `eks`에서 관리합니다. 이러한 필드의 값을 수정하면 충돌이 발생합니다.

```
[...]
f:containers:
  k:{"name":"coredns"}:
    .: {}
    f:args: {}
    f:image: {}
    f:imagePullPolicy: {}
[...]
manager: eks
```

```
[...]
```

- 부분 관리형 필드 - 관리형 키에 값이 지정되면 해당 필드에 대해 선언된 키가 관리됩니다. 지정된 키를 수정하면 충돌이 발생합니다.

다음 출력에서는 eks가 name 키로 설정된 config-volume 및 tmp 볼륨을 관리하는 것을 확인할 수 있습니다.

```
[...]
f:volumes:
  k:{"name":"config-volume"}:
    .: {}
    f:configMap:
      f:items: {}
      f:name: {}
    f:name: {}
  k:{"name":"tmp"}:
    .: {}
    f:name: {}
[...]
manager: eks
[...]
```

- 부분 관리형 필드에 키 추가 - 특정 키 값만 관리되는 경우, 충돌을 일으키지 않고 인수와 같은 추가 키를 필드에 안전하게 추가할 수 있습니다. 다른 키를 추가하는 경우 먼저 필드가 관리되지 않는지 확인합니다. 관리되는 값을 추가하거나 수정하면 충돌이 발생합니다.

다음 출력에서는 name 키와 name 필드가 모두 관리되는 것을 알 수 있습니다. 컨테이너 이름을 추가하거나 수정하면 이 관리 키와 충돌이 발생합니다.

```
[...]
f:containers:
  k:{"name":"coredns"}:
[...]
  f:name: {}
[...]
manager: eks
[...]
```

절차

kubectl을 사용하여 Amazon EKS 추가 기능을 위해 Amazon EKS가 관리하는 필드를 볼 수 있습니다.

Amazon EKS가 관리하지 않는 모든 필드 또는 다른 Kubernetes 컨트롤 플레인 프로세스(예: kube-controller-manager)를 문제없이 수정할 수 있습니다.

1. 조사할 추가 기능을 확인합니다. 클러스터에 배포된 모든 deployments와 DaemonSet을 보려면 [the section called “클러스터 리소스 액세스”](#) 섹션을 참조하세요.
2. 다음 명령을 실행하여 추가 기능에 대한 관리형 필드를 확인합니다.

```
kubectl get type/add-on-name -n add-on-namespace -o yaml
```

예를 들어 다음 명령을 사용하여 CoreDNS 추가 기능에 대한 관리형 필드를 볼 수 있습니다.

```
kubectl get deployment/coredns -n kube-system -o yaml
```

필드 관리는 반환된 출력의 다음 섹션에 나열되어 있습니다.

```
[...]
managedFields:
  - apiVersion: apps/v1
    fieldsType: FieldsV1
    fieldsV1:
[...]
```

Note

출력에 managedFields가 표시되지 않으면 명령에 --show-managed-fields를 추가하고 다시 실행합니다. 사용 중인 kubectl 버전에 따라 관리형 필드가 기본적으로 반환되는지 여부가 결정됩니다.

다음 단계

AWS가 소유하지 않은 필드를 추가 기능에 맞도록 사용자 지정하세요.

배포 중 컨테이너 이미지 서명 검증

[AWS 서명](#)를 사용하고 배포 시 서명된 컨테이너 이미지를 확인하려는 경우 다음 솔루션 중 하나를 사용할 수 있습니다.

- [Gatekeeper](#) 및 [Ratify](#) – Gatekeeper를 승인 컨트롤러로 사용하고 AWS 서명 플러그인으로 구성된 Ratify를 서명 검증을 위한 웹 후크로 사용합니다.
- [Kyverno](#) – 서명 검증을 위해 AWS 서명 플러그인으로 구성된 Kubernetes 정책 엔진입니다.

Note

컨테이너 이미지 서명을 확인하기 전에 선택한 승인 컨트롤러의 필요에 따라 [Notation](#) 트러스트 스토어 및 신뢰 정책을 구성합니다.

클러스터 리소스 구성 및 모니터링

이 장에서는 클러스터를 관리하는 데 도움이 되는 다음 주제를 다룹니다. AWS Management Console 을 사용하여 [Kubernetes 리소스](#)의 정보를 볼 수도 있습니다.

- Kubernetes Dashboard는 Kubernetes 클러스터용 범용 웹 기반 UI입니다. 이를 통해 사용자는 클러스터에서 실행 중인 애플리케이션을 관리하고 문제를 해결할 수 있을 뿐만 아니라 클러스터 자체를 관리할 수 있습니다. 자세한 내용은 [Kubernetes Dashboard](#) GitHub 리포지토리를 참조하세요.
- [the section called “지표 서버”](#) - Kubernetes 지표 서버는 클러스터에서 리소스 사용량 데이터의 집계자입니다. 클러스터에 기본적으로 배포되지 않지만 Kubernetes 대시보드 및 [the section called “Horizontal Pod Autoscaler”](#)와 같은 Kubernetes 추가 기능에 사용됩니다. 이 주제에서는 Metrics Server를 설치하는 방법을 알아봅니다.
- [the section called “Helm을 사용하여 앱 배포”](#) - Kubernetes용 Helm 패키지 관리자는 Kubernetes 클러스터에서 애플리케이션을 설치하고 관리하는 데 도움이 됩니다. 이 주제에서는 Helm 바이너리를 설치하고 실행하는 방법을 설명하므로 로컬 컴퓨터에서 Helm CLI를 사용하여 차트를 설치하고 관리할 수 있습니다.
- [the section called “리소스에 태깅”](#) - Amazon EKS 리소스 관리를 지원하기 위해 태그 형식으로 각 리소스에 고유한 메타데이터를 할당할 수 있습니다. 이 주제에서는 태그를 설명하고 태그를 생성하는 방법을 보여줍니다.
- [the section called “Service quotas”](#) - AWS 계정에는 각 AWS 서비스에 대한 기본 할당량(이전에는 제한이라고 함)이 있습니다. Amazon EKS 할당량과 이러한 할당량을 높이는 방법을 알아봅니다.

Amazon EKS 클러스터 비용 모니터링 및 최적화

비용 모니터링은 Amazon EKS에서 Kubernetes 클러스터를 관리할 때 필수적인 요소입니다. 클러스터 비용에 대한 가시성을 확보하면 리소스 사용률을 최적화하고, 예산을 설정하고, 배포에 대한 데이터 기반 결정을 내릴 수 있습니다. Amazon EKS는 비용을 효과적으로 추적하고 할당할 수 있도록 도와주는 두 가지 비용 모니터링 솔루션을 제공하며, 각 솔루션에는 고유한 장점이 있습니다.

AWS Amazon EKS에 대한 빌링 분할 비용 할당 데이터-이 기본 기능은 AWS 빌링 콘솔과 원활하게 통합되므로 다른 AWS 서비스에서 사용하는 것과 동일한 친숙한 인터페이스와 워크플로를 사용하여 비용을 분석하고 할당할 수 있습니다. 비용 할당 분할을 사용하면 다른 AWS 지출과 함께 Kubernetes 비용을 직접 파악할 수 있으므로 AWS 환경 전반에서 비용을 전체적으로 최적화하기가 더 쉬워집니다. 또한 비용 범주 및 비용 이상 징후 감지와 같은 기존 AWS 빌링 기능을 활용하여 비용 관리 기능을 더욱

향상시킬 수 있습니다. 자세한 내용은 AWS 빌링 사용 설명서의 [분할 비용 할당 데이터에 대한 이해](#)를 참조하세요.

Kubecost - Amazon EKS는 Kubernetes 비용 모니터링 도구인 Kubecost를 지원합니다. Kubecost는 Kubernetes 네이티브 방식으로 기능이 풍부한 비용 모니터링 접근 방식을 제공하며 Kubernetes 리소스별로 세분화된 비용 분석, 비용 최적화 권장 사항, 즉시 사용 가능한 대시보드 및 보고서를 제공합니다. 또한 Kubecost는 AWS 비용 및 사용 보고서를 통합하여 정확한 요금 데이터를 가져오므로 고객이 Amazon EKS 비용을 정확하게 파악할 수 있습니다. [Kubecost 설치](#) 방법을 알아봅니다. 무료 Kubecost 구독에 대한 자세한 내용은 [Kubecost](#) AWS Marketplace 페이지를 참조하세요.

분할 비용 할당으로 AWS 청구에서 포드별 비용 보기

Amazon EKS에 대한 AWS 분할 비용 할당 데이터를 사용한 비용 모니터링

Amazon EKS에 대한 AWS 분할 비용 할당 데이터를 사용하여 Amazon EKS 클러스터의 비용을 세부적으로 파악할 수 있습니다. 이를 통해 Kubernetes 애플리케이션의 비용 및 사용량을 분석하고 최적화하며 차지백할 수 있습니다. Kubernetes 애플리케이션에서 사용하는 Amazon EC2 CPU 및 메모리 리소스를 기반으로 개별 사업부 및 팀에 애플리케이션 비용을 할당합니다. Amazon EKS에 대한 분할 비용 할당 데이터를 사용하면 포드당 비용을 파악할 수 있고 네임스페이스, 클러스터 및 기타 Kubernetes 기본 요소를 사용하여 포드당 비용 데이터를 집계할 수 있습니다. 다음은 Amazon EKS 비용 할당 데이터를 분석하는 데 사용할 수 있는 Kubernetes 기본 요소의 예입니다.

- 클러스터 이름
- 배포
- 네임스페이스
- 노드
- 워크로드 이름
- 워크로드 유형

[사용자 정의 비용 할당 태그](#)도 지원됩니다. 분할 비용 할당 데이터에 대한 자세한 내용은 AWS 빌링 사용 설명서의 [분할 비용 할당 데이터에 대한 이해](#)를 참조하세요.

비용 및 사용 보고서 설정

비용 관리 콘솔, AWS 명령줄 인터페이스 또는 AWS SDK에서 EKS에 대한 분할 비용 할당 데이터를 설정할 수 있습니다.

분할 비용 할당 데이터에 대해 다음을 사용합니다.

1. 분할 비용 할당 데이터를 옵트인합니다. 자세한 내용은 AWS 비용 및 사용 보고서 사용 설명서의 [분할 비용 할당 데이터 활성화](#)를 참조하세요.
2. 새 보고서나 기존 보고서에 데이터를 포함합니다.
3. 보고서를 봅니다. Billing and Cost Management 콘솔을 사용하거나 Amazon Simple Storage Service에서 보고서 파일을 볼 수 있습니다.

Kubecost 설치

Amazon EKS는 포드, 노드, 네임스페이스 및 레이블을 포함한 Kubernetes 리소스별로 분류된 비용을 모니터링하는 데 사용할 수 있는 Kubecost를 지원합니다. 이 주제에서는 Kubecost 설치 및 Kubecost 대시보드 액세스를 다룹니다.

Amazon EKS는 클러스터 비용 가시성을 위해 AWS에 최적화된 Kubecost 번들을 제공합니다. 기존 AWS 지원 계약을 사용하여 지원을 받을 수 있습니다. Kubecost의 사용 가능한 버전에 대한 자세한 내용은 [the section called “Kubecost에 대해 자세히 알아보기”](#) 섹션을 참조하세요.

Note

Kubecost v2에는 몇 가지 주요 새 기능이 도입되었습니다. [Kubecost v2에 대해 자세히 알아보세요.](#)

Kubecost에 대한 자세한 내용은 [Kubecost](#) 설명서와 [FAQ](#)를 참조하세요.

Amazon EKS 최적화 Kubecost 번들 설치

다음 절차 중 하나를 사용하여 Amazon EKS 최적화 Kubecost 번들을 설치할 수 있습니다.

- 시작하기 전에 [Kubecost - 아키텍처 개요](#)를 검토하여 Kubecost가 Amazon EKS에서 작동하는 방식을 이해하는 것이 좋습니다.
- Amazon EKS를 처음 사용하는 경우 Amazon EKS 최적화 Kubecost 번들 설치가 간소화되었으므로 Helm을 사용하여 설치하는 것이 좋습니다. 자세한 내용은 [Amazon EKS 추가 기능을 사용하여 Amazon EKS 클러스터에 Kubecost 배포](#)를 참조하세요.
- 설치를 사용자 지정하려면 Helm을 사용하여 Amazon EKS 최적화 Kubecost 번들을 구성합니다. 자세한 내용은 Kubecost 설명서의 [Helm을 사용하여 Amazon EKS 클러스터에 Kubecost 배포](#)를 참조하세요.

Kubecost 대시보드 액세스

Amazon EKS 최적화 Kubecost 번들 설정이 완료되면 Kubecost 대시보드에 액세스할 수 있어야 합니다. 자세한 내용은 [the section called “Kubecost 대시보드 액세스”](#) 섹션을 참조하세요.

Kubecost 대시보드 액세스

사전 조건

1. kubecost 관련 포드의 상태가 ‘실행 중’인지 확인합니다.

```
kubectl get pods --namespace kubecost
```

Kubecost 대시보드 액세스

1. 디바이스에서 포트 포워딩을 활성화하여 Kubecost 대시보드를 노출합니다.

```
kubectl port-forward deployment/kubecost-cost-analyzer 9090 --namespace kubecost
```

또는 [AWS Load Balancer Controller](#)를 사용하여 Kubecost를 노출하고 인증, 권한 부여, 사용자 관리를 위해 Amazon Cognito를 사용할 수 있습니다. 자세한 내용은 [How to use Application Load Balancer and Amazon Cognito to authenticate users for your Kubernetes web apps](#)을 참조하세요.

2. 이전 단계를 완료한 동일한 디바이스에서 웹 브라우저를 열고 다음 주소를 입력합니다.

```
http://localhost:9090
```

브라우저에 Kubecost 개요 페이지가 표시됩니다. Kubecost가 지표를 수집하는 데 클러스터 크기에 따라 5~10분 이상 걸릴 수 있습니다. 누적 클러스터 비용, 관련 Kubernetes 자산 비용, 월별 집계 비용을 포함한 Amazon EKS 지출을 확인할 수 있습니다.

3. 클러스터 수준에서 비용을 추적하려면 청구할 Amazon EKS 리소스에 태그를 지정합니다. 자세한 내용은 [the section called “리소스에 결제용 태깅”](#) 섹션을 참조하세요.
 - 비용 할당 - 지난 7일 동안 각 네임스페이스와 기타 차원에 대한 월별 Amazon EKS 비용 및 누적 비용을 확인합니다. 이는 애플리케이션의 어느 부분이 Amazon EKS 지출에 기여하는지 파악하는데 유용합니다.
 - 자산 - Amazon EKS 리소스와 연결된 AWS 인프라 자산의 비용을 확인합니다.

Kubecost에 대해 자세히 알아보기

Amazon EKS는 클러스터 비용 가시성을 위해 AWS에 최적화된 Kubecost 번들을 제공합니다. Amazon EKS는 포드, 노드, 네임스페이스 및 레이블을 포함한 Kubernetes 리소스별로 분류된 비용을 모니터링하는 데 사용할 수 있는 Kubecost를 지원합니다.

이 주제에서는 사용 가능한 Kubecost 버전과 사용 가능한 계층 간의 차이점을 다룹니다. EKS는 Kubecost 버전 1 및 버전 2를 지원합니다. 각 버전은 서로 다른 계층에서 사용 가능합니다. Amazon EKS 최적화 Kubecost 번들을 추가 비용 없이 Amazon EKS 클러스터에 사용할 수 있습니다. Prometheus용 Amazon 관리형 서비스 등 관련 AWS 서비스 이용에 대한 요금이 청구될 수 있습니다. 기존 AWS 지원 계약을 사용하여 지원을 받을 수도 있습니다.

Kubernetes 플랫폼 관리자 및 재무 리더는 Kubecost를 사용하여 Amazon EKS 청구 내역을 시각화하고, 비용을 할당하고, 애플리케이션 팀과 같은 조직 단위를 청구할 수 있습니다. 내부 팀과 사업부에 실제 AWS 청구서를 기반으로 투명하고 정확한 비용 데이터를 제공할 수 있습니다. 또한 인프라 환경 및 클러스터 내 사용 패턴을 기반으로 비용 최적화를 위한 맞춤형 권장 사항을 가져올 수도 있습니다. Kubecost에 대한 자세한 내용은 [Kubecost](#) 설명서를 참조하세요.

Kubecost의 사용자 지정 번들과 Kubecost(OpenCost라고도 함) 무료 버전의 차이점은 무엇인가요?

AWS와 Kubecost는 협력하여 사용자 지정 버전의 Kubecost를 제공했습니다. 이 버전에는 추가 요금 없이 상용 기능의 하위 집합이 포함되어 있습니다. Kubecost의 사용자 지정 번들에 포함된 기능은 아래 테이블을 참조하세요.

Kubecost v2

Kubecost v1 및 v2의 차이점은 무엇인가요?

Kubecost 2.0은 이전 버전에서 크게 업그레이드되었으며 완전히 새로운 API 백엔드를 비롯한 주요 새 기능을 포함합니다. 참고로 [할당](#) 및 [자산](#) API는 이전 버전과 완전히 호환됩니다. [원활한 전환을 위해 Kubecost 설명서를 검토하세요](#). 개선 사항의 전체 목록은 [Kubecost v2.0 공지 사항](#)과 [전체 릴리스 정보](#)를 참조하세요.

Important

[업그레이드하기 전에 Kubecost 설명서를 검토하세요](#). 업그레이드는 보고서 가용성에 영향을 미칠 수 있습니다.

핵심 기능 비교:

Feature	Kubecost Free Tier 2.0	Amazon EKS 최적화 Kubecost 번들 2.0	Kubecost Enterprise 2.0
클러스터 비용 가시성	최대 250개 코어의 무제한 클러스터	Prometheus용 Amazon 관리형 서비스와 통합 시 코어 제한이 없는 통합 멀티 클러스터	무제한의 환경(예: 멀티 클라우드)에서 통합 및 무제한 클러스터
배포	사용자 호스팅	사용자 호스팅	사용자 호스팅, Kubecost 호스팅(전용 테넌트), SaaS
지원되는 데이터베이스	로컬 Prometheus	Prometheus 또는 Local Prometheus에 대한 Amazon Managed Service	모든 Prometheus 버전 및 사용자 지정 데이터베이스
데이터베이스 보존 지원(기본 지표)	15일	무제한 이력 데이터	무제한 이력 데이터
Kubecost API 및 UI 보존(ETL)	15일	15일	무제한
하이브리드 클라우드 가시성	-	Amazon EKS 및 Amazon EKS Anywhere 클러스터	멀티 클라우드 및 하이브리드 클라우드
경고 및 반복 보고서	기본 클러스터에서만 지원, 250개의 코어로 제한됨	효율성 알림, 예산 알림, 지출 변경 알림 등 모든 클러스터에서 더 많은 지원	효율성 알림, 예산 알림, 지출 변경 알림 등 모든 클러스터에서 더 많은 지원
저장된 보고서	-	15일 동안의 지표를 사용하는 보고서	무제한 이력 데이터 및 지표를 사용하는 보고서

Feature	Kubecost Free Tier 2.0	Amazon EKS 최적화 Kubecost 번들 2.0	Kubecost Enterprise 2.0
클라우드 결제 통합	기본 클러스터에서만 지원, 250개의 코어로 제한됨	AWS에 대한 사용자 지정 가격 지원(여러 클러스터 및 여러 계정 포함)	모든 클라우드에 대한 맞춤 요금 지원
절감 권장 사항	기본 클러스터에서만 지원, 250개의 코어로 제한됨	기본 클러스터 인사이트(단, 250개의 코어 한도 없음)	멀티 클러스터 인사이트
거버넌스: 감사	-	-	이력 비용 이벤트 감사
Single Sign-On(SO) 지원	-	Amazon Cognito 지원	Okta, Auth0, PingID, KeyCloak 및 이외 사용자 지정
SAML 2.0에서 역할 기반 액세스 제어 (RBAC)	-	-	Okta, Auth0, PingID, KeyCloak 및 이외 사용자 지정
기업 교육 및 온보딩	-	-	풀 서비스 교육 및 FinOps 온보딩
팀	-	-	예

새로운 기능:

다음 기능에는 지표 한도가 적용됩니다.

- Kubecost 집계기
- 네트워크 모니터링
- Kubecost 작업
- 컬렉션
- 이상 탐지
- 컨테이너 요청 적정 크기 조정

- Kubecost 예측
- 필터링 및 집계를 위한 자동 완성

지표 한도:

지표	Kubecost Free Tier 2.0	Amazon EKS 최적화 Kubecost 번들 2.0	Kubecost Enterprise 2.0
클러스터 크기	최대 250개 코어의 무제한 클러스터	무제한	무제한
지표 보존	15일	15일	무제한
다중 클러스터 지원	사용할 수 없음	사용 가능	Available
코어 한도	클러스터당 250개 코어	코어 한도 없음	코어 한도 없음

Kubecost v1

Feature	Kubecost Free Tier	Amazon EKS 최적화 Kubecost 번들	Kubecost Enterprise
배포	사용자 호스팅	사용자 호스팅	사용자 호스팅 또는 Kubecost 호스팅 (SaaS)
지원되는 클러스터의 수	무제한	무제한	무제한
지원되는 데이터베이스	로컬 Prometheus	Local Prometheus 또는 Amazon Managed Service for Prometheus	Prometheus, Amazon Managed Service for Prometheus, Cortex 또는 Thanos
데이터베이스 보존 지원	15일	무제한 이력 데이터	무제한 이력 데이터

Feature	Kubecost Free Tier	Amazon EKS 최적화 Kubecost 번들	Kubecost Enterprise
Kubecost API 보존 (ETL)	15일	15일	무제한 이력 데이터
클러스터 비용 가시성	단일 클러스터	통합 멀티 클러스터	통합 멀티 클러스터
하이브리드 클라우드 가시성	-	Amazon EKS 및 Amazon EKS Anywhere 클러스터	멀티 클라우드 및 하이 브리드 클라우드 지원
경고 및 반복 보고서	-	효율성 알림, 예산 알 림, 지출 변경 알림 등 지원	효율성 알림, 예산 알 림, 지출 변경 알림 등 지원
저장된 보고서	-	15일 데이터를 사용한 보고서	무제한 이력 데이터를 사용하는 보고서
클라우드 결제 통합	각 개별 클러스터에 필 요	AWS에 대한 사용자 지정 가격 지원(여러 클러스터 및 여러 계정 포함)	AWS에 대한 사용자 지정 가격 지원(여러 클러스터 및 여러 계정 포함)
절감 권장 사항	단일 클러스터 인사이 트	단일 클러스터 인사이 트	멀티 클러스터 인사이 트
거버넌스: 감사	-	-	이력 비용 이벤트 감사
Single Sign-On(S SO) 지원	-	Amazon Cognito 지원	Okta, Auth0, PingID, KeyCloak
SAML 2.0이 포함 된 역할 기반 액세스 제어(RBAC)	-	-	Okta, Auth0, PingID, Keycloak
기업 교육 및 온보딩	-	-	풀 서비스 교육 및 FinOps 온보딩

자주 묻는 질문(FAQ)

Amazon EKS에서 Kubecost 사용에 대한 다음과 같은 일반적인 질문과 답변을 참조하세요.

Kubecost API 보존(ETL) 기능은 무엇인가요?

Kubecost ETL 기능은 지표를 집계 및 구성하여 다양한 수준의 세분화에서 비용 가시성을 제공합니다 (예: namespace-level, pod-level 및 deployment-level). Amazon EKS 최적화 Kubecost 번들의 경우 고객은 지난 15일 동안의 지표를 통해 데이터와 인사이트를 얻을 수 있습니다.

알림 및 반복 보고서 기능이란 무엇인가요? 여기에는 어떤 알림 및 보고서가 포함되나요?

Kubecost 알림을 통해 팀은 실시간 Kubernetes 지출 및 클라우드 지출에 대한 업데이트를 받을 수 있습니다. 반복 보고서를 통해 팀은 이력 Kubernetes 및 클라우드 지출에 대한 사용자 지정 보기를 받을 수 있습니다. 둘 다 Kubecost UI 또는 Helm 값을 사용하여 구성할 수 있습니다. 이메일, Slack, Microsoft Teams를 지원합니다.

저장된 보고서에는 어떤 내용이 포함되나요?

Kubecost 저장된 보고서는 비용 및 효율성 지표의 사전 정의된 보기입니다. 여기에는 클러스터, 네임스페이스, 라벨 등에 따른 비용이 포함됩니다.

클라우드 결제 통합이란 무엇인가요?

AWS 청구 API와 통합하면 Kubecost에서 클러스터 외 비용(예: Amazon S3)을 표시할 수 있습니다. 또한, 이를 통해 Kubecost는 스팟 사용량, 절감형 플랜 및 엔터프라이즈 할인을 고려하도록 Kubecost의 클러스터 내 예측을 실제 청구 데이터와 맞게 재조정할 수 있습니다.

절감 권장 사항에는 무엇이 포함되나요?

Kubecost는 사용자가 Kubernetes 인프라 및 지출을 최적화하는 데 도움이 되도록 인사이트와 자동화를 제공합니다.

이 기능은 유료인가요?

아니요. 추가 비용 없이 Amazon EKS 최적화 Kubecost 번들을 사용할 수 있습니다. 포함되지 않은 추가 Kubecost 기능이 필요한 경우 AWS Marketplace를 통해 또는 Kubecost에서 직접 Kubecost의 엔터프라이즈 라이선스를 구입할 수 있습니다.

Amazon EKS 최적화 Kubecost 번들에 대한 지원이 제공되나요?

예, Amazon EKS 최적화 Kubecost 번들을 사용하는 경우에 한합니다.

Amazon EKS 최적화 Kubecost 번들에 대한 지원을 받으려면 어떻게 해야 하나요?

[AWS에 문의하기](#)에서 AWS Support 팀에 지원 사례를 열 수 있습니다.

Amazon EKS 통합에서 제공하는 Kubecost 기능을 사용하려면 라이선스가 필요하나요?

아니요.

보다 정확한 보고를 위해 Kubecost를 AWS 비용 및 사용 보고서와 통합할 수 있나요?

예. AWS 비용 및 사용 보고서에서 데이터를 수집하도록 Kubecost를 구성하여 할인, 스팟 요금, 예약형 인스턴스 요금 등의 정확한 비용 가시성을 얻을 수 있습니다. 자세한 내용은 Kubecost 설명서의 [AWS Cloud Billing Integration](#)을 참조하세요.

이 버전은 Amazon EC2에서 자체 관리형 Kubernetes 클러스터의 비용 관리를 지원하나요?

아니요. Amazon EKS 최적화 Kubecost 번들은 Amazon EKS 클러스터와만 호환됩니다.

Kubecost가 AWS Fargate에서 Amazon EKS에 대한 비용을 추적할 수 있나요?

Kubecost는 Fargate의 Amazon EKS에 대한 클러스터 비용 가시성을 보여주기 위해 최선을 다하지만 Amazon EC2의 Amazon EKS보다 정확도가 낮습니다. 주로 사용량에 대한 요금 청구 방식의 차이 때문입니다. Fargate의 Amazon EKS를 사용하면 사용한 리소스에 대한 요금이 청구됩니다. Amazon EC2 노드의 Amazon EKS에서 프로비저닝된 리소스에 대한 요금이 청구됩니다. Kubecost는 CPU, RAM 및 임시 스토리지를 포함하는 노드 사양을 기반으로 Amazon EC2 노드의 비용을 계산합니다. Fargate를 사용하면 Fargate 포드에 대해 요청된 리소스를 기반으로 비용이 계산됩니다.

Kubecost의 업데이트와 새 버전을 받으려면 어떻게 해야 하나요?

표준 Helm 업그레이드 절차를 사용하여 Kubecost 버전을 업그레이드할 수 있습니다. 최신 버전은 [Amazon ECR Public Gallery](#)에 있습니다.

kubect1-cost CLI가 지원되나요? 설치하려면 어떻게 해야 하나요?

예. Kubect1-cost는 Kubecost 비용 할당 지표에 대한 CLI 액세스를 제공하는 Kubecost(Apache 2.0 라이선스)의 오픈 소스 도구입니다. kubect1-cost를 설치하려면 GitHub의 [설치](#)를 참조하세요.

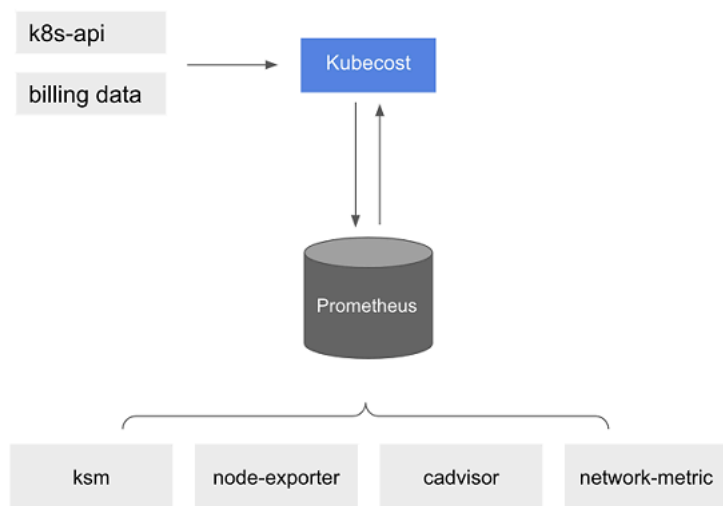
Kubecost 사용자 인터페이스가 지원되나요? 액세스하려면 어떻게 해야 하나요?

Kubecost는 kubect1 포트 전달, 수신 또는 로드 밸런서를 통해 액세스할 수 있는 웹 대시보드를 제공합니다. AWS 로드 밸런서 컨트롤러를 사용하여 Kubecost를 노출하고 인증, 권한 부여 및 사용자 관리

를 위해 Amazon Cognito를 사용할 수 있습니다. 자세한 내용은 AWS 블로그의 [How to use Application Load Balancer and Amazon Cognito to authenticate users for your Kubernetes web apps](#)을 참조하세요.

추가 Kubecost 기능

- Kubecost v1 및 v2 모두에서 다음 기능을 사용할 수 있습니다.
 - 비용 지표 내보내기 - Amazon EKS 최적화 비용 모니터링은 오픈 소스 모니터링 시스템 및 시계열 데이터베이스인 Kubecost 및 Prometheus와 함께 배포됩니다. Kubecost는 Prometheus에서 지표를 읽은 다음 비용 할당 계산을 수행하고 지표를 Prometheus에 다시 씁니다. Kubecost 프론트엔드는 Kubecost에서 지표를 읽고 Kubecost 사용자 인터페이스에 표시합니다. 다음 다이어그램에 아키텍처가 나와 있습니다.



[Prometheus](#)가 사전 설치되어 있으면 추가 분석을 위해 Kubecost 데이터를 현재 비즈니스 인텔리전스 시스템으로 모으는 쿼리를 작성할 수 있습니다. 또한 이를 현재 [Grafana](#) 대시보드의 데이터 소스로 사용하여 내부 팀이 잘 알고 있는 Amazon EKS 클러스터 비용을 표시할 수 있습니다. Prometheus 쿼리를 작성하는 방법에 대한 자세한 내용은 GitHub의 <https://opencost.io/docs/installation/prometheus/> readme 파일을 참조하거나 [Kubecost Github 리포지토리](#)의 예제 Grafana JSON 모델을 참조로 사용합니다.

- AWS 비용 및 사용량 보고서 통합 - Amazon EKS 클러스터에 대한 비용 할당 계산을 수행하기 위해 Kubecost는 AWS 가격 목록 API에서 AWS 서비스 및 AWS 리소스의 공개 가격 정보를 검색합니다. 또한 Kubecost를 AWS 비용 및 사용량 보고서와 통합하여 AWS 계정과 관련된 가격 정보의 정확성을 높일 수도 있습니다. 이 정보에는 엔터프라이즈 할인 프로그램, 예약 인스턴스 사용량, 절감형 플랜 및 스팟 사용량이 포함됩니다. AWS 비용 및 사용량 보고서 통합 작동 방식에 대한 자세한 내용은 Kubecost 설명서의 [AWS 클라우드 결제 통합](#)을 참조하세요.

Kubernetes 지표 서버를 사용한 리소스 사용량 보기

Kubernetes 지표 서버는 클러스터에서 리소스 사용량 데이터의 집계자이며, 기본적으로 Amazon EKS 클러스터에 배포되어 있지 않습니다. 자세한 내용은 GitHub에서 [Kubernetes Metrics Server](#) 섹션을 참조하세요. 지표 서버는 [Horizontal Pod Autoscaler를 사용한 스케일 포드 배포](#) 또는 [Kubernetes 대시보드](#) 등의 다른 Kubernetes 추가 기능에서 일반적으로 사용됩니다. 자세한 내용은 Kubernetes 설명서의 [리소스 지표 파이프라인](#)을 참조하세요. 이 주제에서는 Amazon EKS 클러스터에 Kubernetes 지표 서버를 배포하는 방법을 설명합니다.

Important

지표는 시점 분석을 위한 것이며 기록 분석을 위한 정확한 소스가 아닙니다. 모니터링 솔루션이나 기타 Auto Scaling 이외의 용도로는 사용할 수 없습니다. 모니터링 도구에 대한 자세한 내용은 [클러스터 모니터링](#) 섹션을 참조하세요.

고려 사항

- 매니페스트를 사용하여 Fargate 노드에 Kubernetes Metrics Server를 수동으로 배포하는 경우, 기본값인 10250이 아닌 다른 포트를 사용하도록 metrics-server 배포를 구성합니다. 이 포트는 Fargate용으로 예약되어 있습니다. Metrics Server의 Amazon EKS 추가 기능 버전은 포트 10251을 사용하도록 사전 구성되어 있습니다.
- 보안 그룹과 네트워크 ACL이 metrics-server 포트와 다른 모든 노드 및 포트 간에 포트 10250을 허용하는지 확인하세요. Kubernetes Metrics Server는 여전히 포트 10250을 사용하여 클러스터의 다른 엔드포인트에서 지표를 수집합니다. Fargate 노드에 배포하는 경우 구성된 대체 Metrics Server 포트와 포트 10250을 모두 허용하세요.

Amazon EKS 추가 기능을 사용하여 커뮤니티 추가 기능으로 배포

새로운 기능: 이제 AWS 콘솔 또는 Amazon EKS API를 사용하여 Metrics Server를 커뮤니티 추가 기능으로 배포할 수 있습니다.

AWS 콘솔을 사용하여 배포

1. AWS 콘솔에서 EKS 클러스터 열기
2. '추가 기능' 탭에서 추가 기능 더 가져오기를 선택하세요.

3. '커뮤니티 추가 기능' 섹션에서 Metrics Server를 선택하고 다음을 선택하세요.
4. EKS에서 클러스터에 적합한 추가 기능 버전을 결정합니다. 버전 드롭다운 메뉴를 사용하여 버전을 변경할 수 있습니다.
5. 다음을 선택하고 생성을 선택하여 추가 기능을 설치하세요.

추가 리소스

[the section called “커뮤니티 추가 기능”](#) 단원에 대해 자세히 알아보세요.

다른 Amazon EKS 추가 기능과 동일한 방식으로 커뮤니티 추가 기능을 설치하거나 업데이트합니다.

- [the section called “추가 기능 생성”](#)
- [the section called “추가 기능 업데이트”](#)
- [the section called “추가 기능 제거”](#)

매니페스트를 사용하여 배포

새로운 기능: 이제 AWS 콘솔 또는 Amazon EKS API를 사용하여 Metrics Server를 커뮤니티 추가 기능으로 배포할 수 있습니다. 이 매니페스트 설치 지침은 아카이브됩니다.

1. 다음 명령을 사용하여 지표 서버를 배포합니다.

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/latest/download/components.yaml
```

Fargate를 사용하는 경우 이 파일을 변경해야 합니다. 기본 구성에서 지표 서버는 포트 10250을 사용합니다. 이 포트는 Fargate에 예약되어 있습니다. components.yaml의 포트 10250에 대한 참조를 10251과 같은 다른 포트로 교체합니다.

2. 다음 명령을 사용하여 metrics-server 배포에서 원하는 수의 포드를 실행하고 있는지 확인합니다.

```
kubectl get deployment metrics-server -n kube-system
```

예제 출력은 다음과 같습니다.

NAME	READY	UP-TO-DATE	AVAILABLE	AGE

```
metrics-server 1/1 1 1 6m
```

3. 노드의 리소스(CPU/메모리) 사용량을 표시하여 지표 서버가 작동하는지 테스트합니다.

```
kubectl top nodes
```

4. Error from server (Forbidden) 오류 메시지가 표시되면 Kubernetes RBAC 구성을 업데이트해야 합니다. 클러스터 지표를 읽으려면 Kubernetes RBAC ID에 충분한 권한이 필요합니다. GitHub에서 [지표를 읽는 데 필요한 최소 Kubernetes API 권한](#)을 검토하세요. [역할과 같은 AWS IAM 자격 증명에 Kubernetes API에 대한 액세스 권한 부여](#) 방법을 알아봅니다.

Amazon EKS에서 Helm을 사용하여 애플리케이션 배포

Kubernetes용 Helm 패키지 관리자는 Kubernetes 클러스터에서 애플리케이션을 설치하고 관리하는데 도움이 됩니다. 자세한 내용은 [Helm 설명서](#)를 참조하세요. 이 주제에서는 Helm 바이너리를 설치하고 실행하는 방법을 설명하므로 로컬 시스템에서 Helm CLI를 사용하여 차트를 설치하고 관리할 수 있습니다.

Important

Amazon EKS 클러스터에 Helm 차트를 설치하기 전에 Amazon EKS에 대해 작동하도록 kubectl을 구성해야 합니다. 아직 구성하지 않은 경우 계속하기 전에 [the section called “kubectl을 사용하여 클러스터에 액세스”](#) 섹션을 참조하세요. 클러스터에 대해 다음 명령이 성공한 경우 적절하게 구성한 것입니다.

```
kubectl get svc
```

1. 클라이언트 운영 체제에 맞는 명령을 실행합니다.
 - [Homebrew](#)와 함께 macOS를 사용하는 경우 다음 명령으로 바이너리를 설치합니다.

```
brew install helm
```

- [Chocolatey](#)와 함께 Windows를 사용하는 경우 다음 명령으로 바이너리를 설치합니다.

```
choco install kubernetes-helm
```

- Linux를 사용하는 경우 다음 명령을 사용하여 바이너리를 설치합니다.

```
curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
chmod 700 get_helm.sh
./get_helm.sh
```

Note

openssl을 먼저 설치해야 한다는 메시지가 나타나는 경우 다음 명령으로 설치할 수 있습니다.

```
sudo yum install openssl
```

1. PATH에서 새 바이너리를 선택하려면 현재 터미널 창을 닫고 새 창을 엽니다.
2. 설치한 Helm 버전을 확인합니다.

```
helm version --template='{{ .Version }}\n'
```

예제 출력은 다음과 같습니다.

```
v3.17.2
```

3. 설치된 버전이 클러스터 버전과 호환되는지 확인합니다. 자세한 내용은 [지원되는 버전 스큐](#)를 참조하세요. 예를 들어 3.17.x로 실행하는 경우 지원되는 Kubernetes 버전은 1.29.x~1.32.x 범위를 벗어나면 안 됩니다.
4. 이 시점에서 Helm 명령(예: `helm install chart-name`)을 실행하여 클러스터에서 Helm 차트를 설치, 수정, 삭제 또는 쿼리할 수 있습니다. Helm을 처음 사용하는 경우 설치할 특정 차트가 없으면 다음을 수행할 수 있습니다.
 - 예제 차트를 설치하여 실험합니다. Helm [Quickstart 가이드](#)에서 [예제 차트 설치](#)를 참조하세요.
 - 예제 차트를 생성하여 Amazon ECR로 푸시합니다. 자세한 내용은 Amazon Elastic 컨테이너 레지스트리 사용 설명서의 [Helm 차트 푸시](#)를 참조하세요.
 - [eks-charts](#) GitHub 리포지토리 또는 [ArtifactHub](#)에서 Amazon EKS 차트를 설치합니다.

태그를 사용하여 Amazon EKS 리소스 구성

태그를 사용하여 Amazon EKS 리소스를 관리할 수 있습니다. 이 항목에서는 태그 함수에 대한 개요를 제공하고 태그를 생성하는 방법을 보여 줍니다.

주제

- [태그 기본 사항](#)
- [리소스에 태깅](#)
- [태그 제한](#)
- [리소스에 결제용 태깅](#)
- [콘솔을 사용한 태그 작업](#)
- [CLI, API 또는 eksctl을 사용한 태그 작업](#)

Note

태그는 Kubernetes 레이블 및 주석과 분리된 메타데이터 유형입니다. 이러한 다른 메타데이터 유형에 대한 자세한 내용을 알아보려면 Kubernetes 설명서의 다음 섹션을 참조하세요.

- [레이블 및 선택기](#)
- [Annotations](#)

태그 기본 사항

태그는 AWS 리소스에 할당하는 레이블입니다. 각 태그는 키와 값(선택사항)으로 구성됩니다.

태그를 사용하여 AWS 리소스를 범주화할 수 있습니다. 예를 들어, 용도, 소유자 또는 환경 기준으로 리소스를 범주화할 수 있습니다. 동일한 유형의 리소스가 많은 경우 특정 리소스에 할당한 태그를 사용하여 해당 리소스를 빠르게 식별할 수 있습니다. 예를 들어, Amazon EKS 클러스터에 태그 집합을 정의하면 클러스터의 소유자 및 스택 수준을 추적하는 데 도움이 됩니다. 각 리소스 유형에 대해 일관된 태그 키 집합을 고안하는 것이 좋습니다. 그러면 추가하는 태그에 따라 리소스를 검색하고 필터링할 수 있습니다.

태그를 추가한 후에는 언제든지 태그 키와 값을 편집하거나 리소스에서 태그를 제거할 수 있습니다. 리소스를 삭제하면 리소스 태그도 삭제됩니다.

태그는 Amazon EKS에는 아무런 의미가 없으며 엄격하게 문자열로 해석됩니다. 태그 값을 빈 문자열로 설정할 수 있습니다. 그러나 태그 값을 null로 설정할 수는 없습니다. 해당 리소스에 대해 키가 기존 태그와 동일한 태그를 추가하는 경우 새 값이 이전 값을 덮어씁니다.

AWS ID 및 액세스 관리(IAM)를 사용하는 경우, AWS 계정에서 태그를 관리할 수 있는 권한을 가진 사용자를 제어할 수 있습니다.

리소스에 태깅

다음 Amazon EKS 리소스는 태그를 지원합니다.

- 클러스터
- 관리형 노드 그룹
- Fargate 프로필

다음을 사용하여 이러한 리소스에 태그를 지정할 수 있습니다.

- Amazon EKS 콘솔을 사용하는 경우 새 리소스 또는 기존 리소스에 태그를 언제든지 적용할 수 있습니다. 관련 리소스 페이지의 태그 탭을 사용하여 이 작업을 수행할 수 있습니다. 자세한 내용은 [the section called “콘솔을 사용한 태그 작업”](#) 섹션을 참조하세요.
- eksctl을 사용하는 경우 리소스가 생성될 때 --tags 옵션을 사용하여 리소스에 태그를 적용할 수 있습니다.
- AWS CLI, Amazon EKS API 또는 AWS SDK를 사용 중인 경우 관련 API 작업의 tags 파라미터를 사용하여 새 리소스에 태그를 적용할 수 있습니다. TagResource API 작업을 사용하여 기존 리소스에 태그를 적용할 수도 있습니다. 자세한 내용은 [TagResource](#)를 참조하세요.

일부 리소스 생성 작업을 사용할 때 리소스를 생성함과 동시에 리소스에 대한 태그를 지정할 수도 있습니다. 리소스를 생성하는 동안 태그를 적용할 수 없는 경우 리소스를 생성할 수 없습니다. 이 메커니즘에서는 태그를 지정하려는 리소스가 지정된 태그와 함께 생성되거나 전혀 생성되지 않습니다. 리소스를 생성할 때 태그를 지정하면 리소스를 생성한 후 사용자 지정 태깅 스크립트를 실행할 필요가 없습니다.

태그는 생성한 리소스와 연결된 다른 리소스로 전파되지 않습니다. 예를 들어, Fargate 프로필 태그는 Fargate 프로필과 연결된 다른 리소스(예: 예약된 포트)로 전파되지 않습니다.

태그 제한

태그에 적용되는 제한은 다음과 같습니다.

- 최대 50개의 태그를 리소스와 연결할 수 있습니다.
- 한 리소스에 대해 태그 키를 반복할 수 없습니다. 각 태그 키는 고유해야 하며 하나의 값만 가질 수 있습니다.
- 키는 최대 128자(UTF-8 형식)까지 가능합니다.
- 값은 최대 256자(UTF-8 형식)까지 가능합니다.
- 여러 AWS 서비스 및 리소스에서 태깅 스키마를 사용하는 경우 사용하는 문자 유형을 제한합니다. 일부 서비스는 허용되는 문자에 제한이 있을 수 있습니다. 일반적으로 허용되는 문자는 문자, 숫자, 공백 및 특수 문자 + - = . _ : / @입니다.
- 태그 키와 값은 대소문자를 구분합니다.
- 키 또는 값에 aws:, AWS: 또는 이러한 접두사의 대문자 또는 소문자 조합을 사용하지 않습니다. 이러한 이름은 AWS 전용으로 예약되어 있습니다. 이 접두사가 지정된 태그 키나 값은 편집하거나 삭제할 수 없습니다. 이 접두사가 지정된 태그는 리소스당 태그 수 제한에 포함되지 않습니다.

리소스에 결제용 태깅

Amazon EKS 클러스터에 태그를 적용하면 Cost & Usage Reports에서 비용 할당에 태그를 사용할 수 있습니다. Cost & Usage Reports의 측정 데이터는 모든 Amazon EKS 클러스터에서의 사용량을 보여줍니다. 자세한 내용은 AWS 비용 사용 설명서의 [AWS 비용 및 사용 보고서](#)를 참조하세요.

AWS에서 생성한 비용 할당 태그, 특히 aws:eks:cluster-name을 사용하면 Cost Explorer에서 Amazon EC2 인스턴스 비용을 개별 Amazon EKS 클러스터별로 분류할 수 있습니다. 그러나 이 태그는 컨트롤 플레인 비용을 캡처하지 않습니다. 태그는 Amazon EKS 클러스터에 참여하는 Amazon EC2 인스턴스에 자동으로 추가됩니다. 이 동작은 인스턴스가 Amazon EKS 관리형 노드 그룹인 Karpenter를 사용하여 프로비저닝되었는지 아니면 Amazon EC2를 사용하여 직접 프로비저닝되었는지 여부에 관계없이 발생합니다. 이 특정 태그는 50개 태그 한도에 포함되지 않습니다. 태그를 사용하려면 계정 소유자가 AWS 비용 콘솔에서 또는 API를 사용하여 태그를 활성화해야 합니다. AWS Organizations 관리 계정 소유자가 태그를 활성화하면 태그는 모든 조직 멤버 계정에 대해서도 활성화됩니다.

태그 키 값이 동일한 리소스를 기준으로 결제 정보를 구성할 수도 있습니다. 예를 들어, 특정 애플리케이션 이름으로 여러 리소스에 태그를 지정한 다음 결제 정보를 구성할 수 있습니다. 이렇게 하면 여러 서비스에서 해당 애플리케이션의 총 비용을 볼 수 있습니다. 태그를 사용한 비용 할당 보고서 설정에 대한 자세한 내용은 AWS 비용 사용 설명서에서 [월간 비용 할당 보고서](#)를 참조하세요.

Note

방금 보고서를 활성화한 경우, 24시간 후에 이번 달의 데이터를 볼 수 있습니다.

Cost Explorer는 AWS 프리 티어의 일부로 제공되는 보고 도구입니다. Cost Explorer를 사용하여 지난 13개월 동안의 Amazon EKS 리소스 차트를 볼 수 있습니다. 또한 향후 3개월 동안의 지출을 예상해볼 수 있습니다. 시간 경과에 따른 AWS 리소스의 지출 패턴을 확인할 수 있습니다. 예를 들어, Cost Explorer를 사용하여 추가 조사가 필요한 영역을 알아내고, 비용 이해에 사용할 수 있는 추세를 파악할 수 있습니다. 또한 데이터의 시간 범위를 지정하고 일별 또는 월별 시간 데이터를 볼 수도 있습니다.

콘솔을 사용한 태그 작업

Amazon EKS 콘솔을 사용하면 신규/기존 클러스터 및 관리형 노드 그룹과 연결된 태그를 관리할 수 있습니다.

Amazon EKS 콘솔에서 리소스 관련 페이지를 선택하면 페이지에 이러한 리소스의 목록이 표시됩니다. 예를 들어, 왼쪽 탐색 창에서 클러스터를 선택하는 경우, 콘솔에 Amazon EKS 클러스터 목록이 표시됩니다. 이러한 목록 중 하나(예: 특정 클러스터)에서 리소스를 선택할 때 해당 리소스가 태그를 지원하면 태그 탭에서 해당 태그를 보고 관리할 수 있습니다.

또한 태그를 관리할 수 있는 통합 방법을 제공하는 태그 편집기를 AWS Management Console에서 사용할 수 있습니다. 자세한 내용은 AWS 태그 편집기 사용 설명서의 [태그 편집기로 AWS 리소스 태그](#)를 참조하세요.

생성 시 리소스에 대한 태그 추가

Amazon EKS 클러스터, 관리형 노드 그룹 및 Fargate 프로필을 생성할 때 태그를 추가할 수 있습니다. 자세한 내용은 [the section called “클러스터 생성”](#) 섹션을 참조하세요.

리소스에 대한 태그 추가 및 삭제

리소스 페이지에서 직접 클러스터와 연결된 태그를 추가하거나 삭제할 수 있습니다.

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 탐색 모음에서 사용할 AWS 리전을 선택합니다.
3. 좌측 탐색 창에서 클러스터를 선택합니다.
4. 특정 클러스터를 선택합니다.
5. 태그 탭을 선택한 후 태그 관리를 선택합니다.
6. 태그 관리 페이지에서 필요에 따라 태그를 추가하거나 삭제합니다.
 - 태그를 추가하려면 태그 추가를 선택합니다. 그런 다음 각 태그의 키와 값을 지정합니다.
 - 태그를 삭제하려면 태그 제거를 선택합니다.
7. 추가 또는 삭제하려는 각 태그에 대해 이 프로세스를 반복합니다.

8. 업데이트를 선택하여 완료합니다.

CLI, API 또는 eksctl를 사용한 태그 작업

다음 AWS CLI 명령 또는 Amazon EKS API 작업을 사용하여 리소스에 대한 태그를 추가, 업데이트, 나열 및 삭제합니다. eksctl만 사용하여 하나의 명령으로 새 리소스를 생성하는 동시에 태그를 추가할 수 있습니다.

Task	AWS CLI	AWS Tools for Windows PowerShell	API 작업
하나 이상의 태그를 추가하거나 덮어씁니다.	tag-resource	Add-EKSResourceTag	TagResource
하나 이상의 태그를 삭제합니다.	untag-resource	Remove-EKSResourceTag	UntagResource

다음 예제는 AWS CLI를 사용하여 리소스에 태그를 지정하거나 태그를 제거하는 방법을 보여줍니다.

예제 1: 기존 클러스터에 태깅

다음 명령은 기존 클러스터에 태그를 지정합니다.

```
aws eks tag-resource --resource-arn resource_ARN --tags team=devs
```

예제 2: 기존 클러스터에서 태그 제거

다음 명령은 기존 클러스터에서 태그를 삭제합니다.

```
aws eks untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

예제 3: 리소스의 태그 나열

다음 명령은 기존 리소스와 연결된 태그를 나열합니다.

```
aws eks list-tags-for-resource --resource-arn resource_ARN
```

일부 리소스 생성 작업을 사용할 때 리소스를 생성하는 동시에 태그를 지정할 수 있습니다. 리소스를 생성할 때 태그를 지원하는 작업은 다음과 같습니다.

Task	AWS CLI	AWS Tools for Windows PowerShell	API 작업	eksctl
클러스터 생성	create-cluster	New-EKSCluster	CreateCluster	<code>create cluster</code>
관리형 노드 그룹 생성*	create-nodegroup	New-EKSNodegroup	CreateNodegroup	<code>create nodegroup</code>
Fargate 프로파일 생성	create-fargate-profile	New-EKSFargateProfile	CreateFargateProfile.html	<code>create fargateprofile</code>

- 관리형 노드 그룹을 생성할 때 Amazon EC2 인스턴스에도 태그를 지정하려면 시작 템플릿을 사용하여 관리형 노드 그룹을 생성합니다. 자세한 내용은 [the section called “Amazon EC2 인스턴스 태깅”](#) 섹션을 참조하세요. 인스턴스가 이미 있는 경우 인스턴스에 수동으로 태깅할 수 있습니다. 자세한 내용은 Amazon EC2 사용 설명서의 [리소스에 태깅](#)을 참조하세요.

Amazon EKS 및 Fargate Service Quotas 보기 및 관리

Amazon EKS는 중앙 위치에서 할당량을 보고 관리할 수 있는 AWS 서비스인 Service Quotas와 통합됩니다. 자세한 내용은 Service Quotas 사용 설명서의 [Service Quotas는 무엇인가요?](#)를 참조하세요. Service Quotas 통합과 함께 AWS Management Console 및 AWS CLI를 사용하여 Amazon EKS 및 AWS Fargate 서비스 할당량 값을 빠르게 조회할 수 있습니다.

AWS Management Console를 사용하여 EKS Service Quotas 보기

1. [Service Quotas Console](#)을 엽니다.
2. 왼쪽 탐색 창에서 AWS 서비스를 선택합니다.
3. AWS 서비스 목록에서 Amazon Elastic Kubernetes Service(Amazon EKS) 또는 AWS Fargate를 검색하고 선택합니다.

Service quotas 목록에서 서비스 할당량 이름, 적용된 값(제공된 경우), AWS 기본 할당량 및 할당량 값 조정 가능 여부를 확인할 수 있습니다.

4. 설명 등 서비스 할당량에 대한 추가 정보를 보려면 할당량 이름을 선택합니다.

5. (선택사항) 할당량 증가를 요청하려면 증가시킬 할당량을 선택하고 할당량 증가 요청을 선택한 다음 필요한 정보를 입력하거나 선택한 다음 요청을 선택합니다.

AWS Management Console을 사용하여 서비스 할당량에 대한 추가 작업을 수행하려면 [Service Quotas 사용 설명서](#)를 참조하세요. 할당량 증가를 요청하려면 Service Quotas 사용 설명서의 [할당량 증가 요청](#)을 참조하세요.

AWS CLI를 사용하여 EKS Service Quotas 보기

다음 명령을 실행하여 Amazon EKS 할당량을 확인합니다.

```
aws service-quotas list-aws-default-service-quotas \
  --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
  --service-code eks \
  --output table
```

다음 명령을 실행하여 Fargate 할당량을 확인합니다.

```
aws service-quotas list-aws-default-service-quotas \
  --query 'Quotas[*].
{Adjustable:Adjustable,Name:QuotaName,Value:Value,Code:QuotaCode}' \
  --service-code fargate \
  --output table
```

Note

반환된 할당량은 현재 AWS 리전의 이 계정에서 Fargate에서 동시에 실행할 수 있는 Amazon ECS 태스크 또는 Amazon EKS 포드의 수입입니다.

AWS CLI를 사용하여 서비스 할당량으로 더 많은 작업을 하려면 AWS CLI 명령 참조에서 [서비스 할당량](#)을 참조하세요. 할당량 증가를 요청하려면 AWS CLI 명령 참조의 [request-service-quota-increase](#) 명령을 참조하세요.

Amazon EKS 서비스 할당량

AWS에서는 AWS Management Console을 사용하여 현재 할당량을 볼 것을 권장합니다. 자세한 내용은 [the section called “AWS Management Console를 사용하여 EKS Service Quotas 보기”](#) 단원을 참조하십시오.

기본 EKS Service Quotas을 보려면 AWS 일반 참조의 [Amazon Elastic Kubernetes Service 엔드포인트 및 할당량](#)을 참조하세요.

이러한 서비스 할당량은 Service Quotas 콘솔의 Amazon Elastic Kubernetes Service(Amazon EKS) 아래에 나열됩니다. 조정 가능한 것으로 표시되는 값에 대한 할당량 증가를 요청하려면 Service Quotas 사용 설명서의 [할당량 증가 요청](#)을 참조하세요.

Note

다음 할당량은 Service Quotas에서 사용할 수 없습니다.

- 클러스터당 Pod Identity 연결은 지원되는 각 리전에서 1,000이며 이 할당량은 조정할 수 없습니다.
- 클러스터당 하이브리드 노드에 대해 최대 15개의 CIDR을 원격 노드 네트워크에 사용하고 15개의 CIDR을 원격 포드 네트워크에 사용할 수 있습니다. 이 할당량은 조정할 수 없습니다.

AWS Fargate 서비스 할당량

Service Quotas 콘솔의 AWS Fargate 서비스는 다양한 서비스 할당량을 나열합니다. 사용량이 서비스 할당량에 가까워지면 경고하는 경보를 구성할 수 있습니다. 자세한 내용은 [the section called “Fargate 리소스 사용량 지표 모니터링을 위한 CloudWatch 경보 생성”](#) 섹션을 참조하세요.

새 AWS 계정에서는 초기 할당량이 낮지만 시간이 지남에 따라 증가할 수 있습니다. Fargate는 각 AWS 리전 내의 계정 사용량을 지속적으로 모니터링한 다음 사용량에 따라 할당량을 자동으로 늘립니다. 조정 가능한 것으로 표시된 값에 대한 할당량 증가를 요청할 수도 있습니다. 자세한 내용은 Service Quotas 사용 설명서의 [할당량 증가 요청](#)을 참조하세요.

AWS에서는 AWS Management Console을 사용하여 현재 할당량을 볼 것을 권장합니다. 자세한 내용은 [the section called “AWS Management Console를 사용하여 EKS Service Quotas 보기”](#) 단원을 참조하십시오.

EKS Service Quotas의 기본 AWS Fargate를 보려면 AWS 일반 참조의 [Fargate Service Quotas](#)를 참조하세요.

 Note

Fargate는 Amazon ECS 태스크 및 Amazon EKS 포드 실행 속도 할당량을 추가로 시행합니다. 자세한 내용은 Amazon ECS 가이드의 [AWS Fargate 스로틀링 할당량](#)을 참조하세요.

Amazon EKS의 보안

AWS는 클라우드 보안을 가장 중요하게 생각합니다. AWS 고객으로서 여러분은 가장 높은 보안 요구 사항을 충족하기 위해 설계된 데이터 센터 및 네트워크 아키텍처의 혜택을 받게 됩니다.

보안은 AWS와 사용자의 공동 책임입니다. [공동 책임 모델](#)은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드의 보안 – AWS는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라 보호를 책임집니다. Amazon EKS의 경우, AWS가 제어 영역 노드와 etcd 데이터베이스를 포함한 Kubernetes 제어 영역을 책임집니다. 서드 파티 감사원은 정기적으로 [AWS규정 준수 프로그램](#)의 일환으로 보안 효과를 테스트하고 검증합니다. Amazon EKS에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 [규정 준수 프로그램 제공 범위 내 AWS 서비스](#)를 참조하세요.
- 클라우드 내부의 보안 - 고객의 책임에는 다음 영역이 포함됩니다.
 - Amazon EKS 제어 영역에서 고객 VPC로 트래픽을 전달할 수 있는 보안 그룹의 구성을 포함한 데이터 영역의 보안 구성
 - 노드와 컨테이너 자체의 구성
 - 노드 운영 체제(업데이트 및 보안 패치 포함)
 - 기타 관련 애플리케이션 소프트웨어:
 - 방화벽 규칙과 같은 네트워크 컨트롤 설정 및 관리
 - IAM을 단독 또는 추가로 이용하여 플랫폼 수준의 자격 증명 및 액세스 관리
 - 데이터의 민감도, 회사 요구 사항, 관련 법률 및 규정

Amazon EKS는 규제 대상이며 민감한 애플리케이션에 대한 여러 규정 준수 프로그램의 인증을 받았습니다. Amazon EKS는 [SOC](#), [PCI](#), [ISO](#), [FedRAMP-Moderate](#), [IRAP](#), [C5](#), [K-ISMS](#), [ENS High](#), [OSPAR](#), [HITRUST CSF](#)를 준수하며 [HIPAA](#) 적격 서비스입니다. 자세한 내용은 [액세스 관리](#) 단원을 참조하십시오.

이 문서는 Amazon EKS를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목적에 맞게 Amazon EKS를 구성하는 방법을 보여줍니다. 또한 Amazon EKS 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법을 배우게 됩니다.

Note

Linux 컨테이너는 컨테이너가 액세스할 수 있는 항목을 제한하는 데 도움이 되는 제어 그룹(cgroups)과 네임스페이스로 구성되지만 모든 컨테이너는 호스트 Amazon EC2 인스턴스와 동일한 Linux 커널을 공유합니다. 컨테이너를 루트 사용자(UID 0)로 실행하거나 호스트 네트워크 또는 호스트 PID 네임스페이스와 같은 호스트 리소스 또는 네임스페이스에 대한 액세스 권한을 컨테이너에 부여하는 것은 권장하지 않습니다. 이렇게 하면 컨테이너가 제공하는 격리의 효과가 떨어지기 때문입니다.

주제

- [모범 사례로 Amazon EKS 클러스터 보호](#)
- [Amazon EKS에서 취약성 분석](#)
- [Amazon EMR 클러스터에서 규정 준수 검증](#)
- [Amazon Elastic Kubernetes Service에 대한 보안 고려 사항](#)
- [Kubernetes의 보안 고려 사항](#)
- [Amazon EKS Auto Mode의 보안 고려 사항](#)
- [Amazon EKS용 자격 증명 및 액세스 관리](#)

모범 사례로 Amazon EKS 클러스터 보호

Amazon EKS 보안 모범 사례는 Amazon EKS 모범 사례 가이드의 [보안 모범 사례](#)에 나와 있습니다.

Amazon EKS에서 취약성 분석

보안은 Kubernetes 클러스터 및 애플리케이션을 구성하고 유지 관리하는 데 중요한 고려 사항입니다. 다음에는 EKS 클러스터의 보안 구성을 분석하기 위한 리소스, 취약성을 확인하기 위한 리소스, 해당 분석을 수행할 수 있는 AWS 서비스와의 통합이 나열되어 있습니다.

Amazon EKS에 대한 Center for Internet Security(CIS) 벤치마크

[Center for Internet Security\(CIS\) Kubernetes 벤치마크](#)는 Amazon EKS 보안 구성에 대한 지침을 제공합니다. 이 벤치마크의 기능:

- Kubernetes 구성 요소의 보안 구성을 담당하는 Amazon EC2 노드(관리형 노드 및 자체 관리형 노드 모두)에 적용할 수 있습니다.

- Amazon EKS를 사용할 때 Kubernetes 클러스터와 노드를 안전하게 구성할 수 있도록 커뮤니티에서 승인한 표준 방법을 제공합니다.
- 제어 플레인 로깅 구성, 노드 보안 구성, 정책 및 관리형 서비스의 4개 섹션으로 구성됩니다.
- 현재 Amazon EKS에서 사용할 수 있는 모든 Kubernetes 버전을 지원하며, Kubernetes 클러스터에서 CIS 벤치마크를 사용하여 구성을 확인하기 위한 표준 오픈 소스 도구인 [kube-bench](#)를 사용하여 실행할 수 있습니다.

자세한 내용은 [CIS Amazon EKS 벤치마크 소개](#)를 참조하세요.

CIS 벤치마킹된 AMI로 노드 그룹을 업데이트하는 자동 aws-samples 파이프라인은 [EKS 최적화 AMI 강화 파이프라인](#)을 참조하세요.

Amazon EKS 플랫폼 버전

Amazon EKS 플랫폼 버전은 활성화된 Kubernetes API 서버 플러그와 현재 Kubernetes 패치 버전 등 클러스터 컨트롤 플레인의 기능을 나타냅니다. 새 클러스터는 최신 플랫폼 버전과 함께 배포됩니다. 세부 정보는 [the section called “플랫폼 버전”](#)을 참조하세요.

새로운 Kubernetes 버전으로 [Amazon EKS 클러스터를 업데이트](#)할 수 있습니다. Amazon EKS에서 새로운 Kubernetes 버전을 사용할 수 있게 되면 미리 클러스터를 업데이트하여 최신 버전을 사용하는 것이 좋습니다. EKS의 Kubernetes 버전에 대한 자세한 내용은 [the section called “Kubernetes 버전”](#) 섹션을 참조하세요.

운영 체제 취약성 목록

AL2023 취약성 목록

[Amazon Linux 보안 센터](#)에서 Amazon Linux 2023의 보안 또는 프라이버시 이벤트를 추적하거나 관련 [RSS 피드](#)를 구독하세요. 보안 및 프라이버시 이벤트에는 영향을 받는 문제의 개요, 패키지, 인스턴스 업데이트로 문제를 해결하는 방법이 포함됩니다.

Amazon Linux 2 취약성 목록

[Amazon Linux 보안 센터](#)에서 Amazon Linux 2의 보안 또는 프라이버시 이벤트를 추적하거나 관련 [RSS 피드](#)를 구독하세요. 보안 및 프라이버시 이벤트에는 영향을 받는 문제의 개요, 패키지, 인스턴스 업데이트로 문제를 해결하는 방법이 포함됩니다.

Amazon Inspector로 노드 탐지

[Amazon Inspector](#)를 사용하여 노드의 의도하지 않은 네트워크 액세스 가능성과 이러한 Amazon EC2 인스턴스 취약성을 검사할 수 있습니다.

Amazon GuardDuty로 클러스터 및 노드 탐지

Amazon GuardDuty는 AWS 환경 내 계정, 컨테이너, 워크로드 및 데이터를 보호하는 데 도움이 되는 위협 감지 서비스입니다. 다른 기능 중에서도 GuardDuty는 EKS 클러스터에 대한 잠재적 위협을 탐지하는 두 가지 기능인 EKS Protection과 런타임 모니터링을 제공합니다.

자세한 내용은 [the section called “Amazon GuardDuty”](#) 섹션을 참조하세요.

Amazon EMR 클러스터에서 규정 준수 검증

AWS 서비스가 특정 규정 준수 프로그램의 범위 내에 있는지 확인하려면 [규정 준수 프로그램별 범위 내 AWS 서비스](#)를 참조하고 관심 있는 규정 준수 프로그램을 선택하세요. 일반 정보는 [AWS 규정 준수 프로그램](#)을 참조하세요.

AWS Artifact를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 [AWS Artifact의 보고서 다운로드](#)를 참조하세요.

AWS 서비스를 사용할 때 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표, 관련 법률 및 규정에 따라 결정됩니다. AWS는 규정 준수를 지원하기 위해 다음과 같은 리소스를 제공합니다.

- [보안 규정 준수 및 거버넌스](#) - 이러한 솔루션 구현 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수 기능을 배포하는 단계를 제공합니다.
- [HIPAA 적격 서비스 참조](#) - HIPAA 적격 서비스가 나열되어 있습니다. 모든 AWS 서비스가 HIPAA에 적합한 것은 아닙니다.
- [AWS 규정 준수 리소스](#) - 사용자의 업계와 위치에 해당할 수 있는 워크북 및 안내서 모음입니다.
- [AWS 고객 규정 준수 가이드](#) - 규정 준수의 관점에서 공동 책임 모델을 이해합니다. 이 가이드에서는 AWS 서비스를 보호하기 위한 모범 사례를 요약하고 여러 프레임워크(미국 표준 기술 연구소(NIST), 결제 카드 산업 보안 표준 위원회(PCI), 국제 표준화기구(ISO) 등)에서 보안 제어에 대한 지침을 매핑합니다.
- AWS 구성 개발자 가이드의 [규칙으로 리소스 평가하기](#) - AWS 구성 서비스는 리소스 구성이 내부 관행, 업계 가이드라인 및 규정을 얼마나 잘 준수하는지 평가합니다.

- [AWS 보안 허브](#)- 이 AWS 서비스는 AWS 내의 보안 상태에 대한 포괄적인 보기를 제공합니다. Security Hub는 보안 컨트롤을 사용하여 AWS리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 [Security Hub 제어 참조](#)를 참조하세요.
- [Amazon GuardDuty](#) - 이 AWS 서비스는 의심스럽고 악의적인 활동이 있는지 환경을 모니터링하여 AWS 계정, 워크로드, 컨테이너 및 데이터에 대한 잠재적 위협을 탐지합니다. GuardDuty는 특정 규정 준수 프레임워크에서 요구하는 침입 탐지 요구 사항을 충족하여 PCI DSS와 같은 다양한 규정 준수 요구 사항을 따르는 데 도움을 줄 수 있습니다.
- [AWS Audit Manager](#) – 이 AWS 서비스는 AWS 사용량을 지속적으로 감사하여 위험과 규정 및 산업 표준의 준수를 관리하는 방법을 간소화하는 데 도움이 됩니다.

Amazon Elastic Kubernetes Service에 대한 보안 고려 사항

다음은 Amazon EKS에 영향을 미치는 클라우드 보안 고려 사항입니다.

주제

- [Amazon EKS의 인프라 보안](#)
- [Amazon EKS 클러스터에서 복원력 이해](#)

Amazon EKS의 인프라 보안

관리형 서비스인 Amazon Elastic Kubernetes Service는 AWS 글로벌 네트워크 보안으로 보호됩니다. AWS 보안 서비스와 AWS의 인프라 보호 방법에 대한 자세한 내용은 [AWS 클라우드 보안](#)을 참조하세요. 인프라 보안에 대한 모범 사례를 사용하여 AWS 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 [인프라 보호](#)를 참조하세요.

AWS에서 게시한 API 호출을 사용하여 네트워크를 통해 Amazon EKS에 액세스합니다. 고객은 다음을 지원해야 합니다.

- Transport Layer Security(TLS) TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군 Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 위탁자와 관련된 보안 암호 액세스 키를 사용하여 서명해야 합니다. [AWS 보안 토큰 서비스](#)(AWS STS)를 사용하여 요청에 서명하기 위한 임시 보안 자격 증명을 생성할 수도 있습니다.

Amazon EKS 클러스터를 생성할 때 클러스터가 사용할 VPC 서브넷을 지정합니다. Amazon EKS에는 최소 2개의 가용 영역에 있는 서브넷이 필요합니다. Kubernetes가 프라이빗 서브넷에 있는 노드에서 실행되는 포드로 트래픽을 로드 밸런싱하는 퍼블릭 서브넷에 퍼블릭 로드 밸런서를 생성할 수 있도록 퍼블릭 및 프라이빗 서브넷이 있는 VPC를 사용하는 것이 좋습니다.

VPC 고려 사항에 대한 자세한 내용은 [the section called “VPC 및 서브넷 요구 사항”](#) 섹션을 참조하세요.

[Amazon EKS 시작하기](#) 연습에 제공된 AWS CloudFormation 템플릿을 사용하여 VPC 및 노드 그룹을 생성하면 컨트롤 플레인 및 노드 보안 그룹이 권장 설정으로 구성됩니다.

보안 그룹 고려 사항에 대한 자세한 내용은 [the section called “보안 그룹 요구 사항”](#) 섹션을 참조하세요.

새 클러스터를 생성할 때 Amazon EKS에서는 클러스터와 통신하는 데 사용하는 관리형 Kubernetes API 서버에 대한 엔드포인트를 생성합니다(kubectl과 같은 Kubernetes 관리 도구 사용). 기본적으로 이 API 서버 엔드포인트는 인터넷에 공개되어 있으며, API 서버에 대한 액세스는 AWS Identity and Access Management(IAM) 및 기본 Kubernetes [역할 기반 액세스 제어\(RBAC\)](#)의 조합을 통해 보호됩니다.

노드와 API 서버 간의 모든 통신이 VPC 내에 유지되도록 Kubernetes API 서버에 대한 프라이빗 액세스를 활성화할 수 있습니다. 인터넷에서 API 서버로 액세스하는 IP 주소를 제한하거나 API 서버로의 인터넷 액세스를 완전히 비활성화할 수 있습니다.

클러스터 엔드포인트 액세스 수정에 대한 자세한 내용은 [the section called “클러스터 엔드포인트 액세스 수정”](#) 섹션을 참조하세요.

Amazon VPC CNI 또는 [Project Calico](#)와 같은 타사 도구로 Kubernetes 네트워크 정책을 구현할 수 있습니다. 네트워크 정책에 맞는 Amazon VPC CNI 사용에 관한 자세한 내용은 [the section called “Kubernetes 정책”](#) 섹션을 참조하세요. Project Calico는 타사 오픈소스 프로젝트입니다. 자세한 내용은 [Project Calico 설명서](#)를 참조하세요.

AWS PrivateLink를 사용하여 Amazon EKS에 액세스

AWS 프라이빗 링크를 사용하여 VPC와 Amazon Elastic Kubernetes Service 사이에 프라이빗 연결을 생성할 수 있습니다. 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 AWS Direct Connect를 사용하지 않고 VPC에 있는 것처럼 Amazon EKS에 액세스할 수 있습니다. VPC의 인스턴스에서 Amazon EKS에 액세스하는 데 퍼블릭 IP 주소가 필요하지 않습니다.

AWS 프라이빗 링크에서 제공되는 인터페이스 엔드포인트를 생성하여 이 프라이빗 연결을 설정합니다. 인터페이스 엔드포인트에 대해 사용 설정하는 각 서브넷에서 엔드포인트 네트워크 인터페이스를

생성합니다. 이는 Amazon EKS로 향하는 트래픽의 진입점 역할을 하는 요청자 관리형 네트워크 인터페이스입니다.

자세한 내용은 AWS PrivateLink 가이드의 [AWS PrivateLink를 통한 AWS 서비스 액세스](#)를 참조하세요.

Amazon EKS 고려 사항

- Amazon EKS에 대한 인터페이스 엔드포인트를 설정하려면 먼저 AWS 프라이빗 링크 가이드의 [Considerations](#)를 검토합니다.
- Amazon EKS에서는 인터페이스 엔드포인트를 통해 모든 API 작업에 대한 직접 호출 수행을 지원하지만, Kubernetes API는 해당하지 않습니다. Kubernetes API 서버에서는 이미 [프라이빗 엔드포인트](#)를 지원합니다. Kubernetes API 서버 프라이빗 엔드포인트에서는 클러스터와 통신하는 데 사용하는 Kubernetes API 서버에 대한 프라이빗 엔드포인트를 생성합니다(kubectl과 같은 Kubernetes 관리 도구 사용). 노드와 API 서버 간의 모든 통신이 VPC 내에 유지되도록 Kubernetes API 서버에 대한 [프라이빗 액세스](#)를 활성화할 수 있습니다. AWS Amazon EKS API용 PrivateLink는 퍼블릭 인터넷에 트래픽을 노출하지 않고 VPC에서 Amazon EKS APIs를 직접 호출하는 데 도움이 됩니다.
- 인터페이스 엔드포인트를 통해서만 액세스하도록 Amazon EKS를 구성할 수 없습니다.
- AWS 프라이빗 링크의 표준 요금이 Amazon EKS의 인터페이스 엔드포인트에 적용됩니다. 각 가용 영역에서 인터페이스 엔드포인트가 프로비저닝된 각 시간과 인터페이스 엔드포인트를 통해 처리된 데이터에 대해 요금이 청구됩니다. 자세한 내용은 [AWS PrivateLink 요금](#)을 참조하세요.
- Amazon EKS에는 VPC 엔드포인트 정책이 지원됩니다. 이러한 정책을 사용하여 인터페이스 엔드포인트를 통해 Amazon EKS에 대한 액세스를 제어할 수 있습니다. 또한 보안 그룹을 엔드포인트 네트워크 인터페이스와 연결하여 인터페이스 엔드포인트를 통해 Amazon EKS로 향하는 트래픽을 제어할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 [엔드포인트 정책을 사용하여 VPC 엔드포인트에 대한 액세스 제어](#)를 참조하세요.
- VPC 흐름 로그를 사용하여 네트워크 인터페이스에서 송수신되는 IP 트래픽에 대한 정보를 캡처할 수 있습니다(인터페이스 엔드포인트 포함). 흐름 로그 데이터는 Amazon CloudWatch 또는 Amazon S3에 게시할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서의 [VPC 흐름 로그를 사용하여 IP 트래픽 로깅](#)을 참조하세요.
- Amazon EKS API를 인터페이스 엔드포인트가 있는 VPC에 연결하여 온프레미스 데이터 센터에서 Amazon EKS API에 액세스할 수 있습니다. AWS Direct Connect 또는 AWS Site-to-Site VPN을 사용하여 온프레미스 사이트를 VPC에 연결할 수 있습니다.
- AWS Transit 게이트웨이 또는 VPC 피어링을 사용하여 인터페이스 엔드포인트가 있는 VPC에 다른 VPC를 연결할 수 있습니다. VPC 피어링은 두 VPC 간의 네트워킹 연결입니다. 사용자의 VPC 사이 또는 다른 계정의 VPC와 VPC 피어링 연결을 설정할 수 있습니다. VPC는 두 개의 서로 다른 AWS

리전에 있을 수 있습니다. 피어링된 VPC 간 트래픽은 AWS 네트워크에 유지됩니다. 트래픽은 퍼블릭 인터넷을 통과하지 않습니다. 전송 게이트웨이는 VPC를 상호 연결하는 데 사용할 수 있는 네트워크 전송 허브입니다. VPC와 Transit Gateway 간 트래픽은 AWS 글로벌 프라이빗 네트워크에 남아 있습니다. 트래픽은 퍼블릭 인터넷에 노출되지 않습니다.

- 2024년 8월 이전에는 Amazon EKS용 VPC 인터페이스 엔드포인트는 `eks.region.amazonaws.com`을 사용하여 IPv4를 통해서만 액세스할 수 있었습니다. 2024년 8월 이후에 생성된 새로운 VPC 인터페이스 엔드포인트는 IPv4 및 IPv6 IP 주소와 두 DNS 이름 (`eks.region.amazonaws.com` 및 `eks.region.api.aws`)의 이중 스택을 사용합니다.
- EKS API에 대한 AWS PrivateLink 지원은 아시아 태평양(말레이시아)(`ap-southeast-5`), 아시아 태평양(태국)(`ap-southeast-7`), 멕시코(중부)(`mx-central-1`), 아시아 태평양(타이페이)(`ap-east-2`) AWS 리전에서 사용할 수 없습니다. AWS EKS Pod Identity에 대한 `eks-auth`의 PrivateLink 지원은 아시아 태평양(말레이시아)(`ap-southeast-5`) 리전에서 사용할 수 있습니다.

Amazon EKS용 인터페이스 엔드포인트 생성

Amazon VPC 콘솔 또는 AWS 명령줄 인터페이스(AWS CLI)를 사용하여 Amazon EKS의 인터페이스 엔드포인트를 생성할 수 있습니다. 자세한 정보는 AWS 프라이빗 링크 가이드의 [VPC 엔드포인트 생성](#)을 참조하세요.

다음과 같은 서비스 이름을 사용하여 Amazon EKS의 인터페이스 엔드포인트를 생성합니다.

- EKS API

```
com.amazonaws.region-code.eks
```

- EKS 인증 API(EKS Pod Identity)

```
com.amazonaws.region-code.eks-auth
```

프라이빗 DNS 기능은 Amazon EKS 및 기타 AWS 서비스의 인터페이스 엔드포인트를 생성할 때 기본적으로 활성화됩니다. 프라이빗 DNS 기능을 사용하려면 다음 VPC 속성이 `true`: `enableDnsHostnames` 및 `enableDnsSupport`로 설정되어 있는지 확인해야 합니다. 자세한 내용은 Amazon VPC 사용 설명서의 [VPC에 대한 DNS 속성 보기 및 업데이트](#)를 참조하세요. 인터페이스 엔드포인트에 프라이빗 DNS 기능이 활성화된 경우:

- 기본 리전 DNS 이름을 사용하여 Amazon EKS에 API를 요청할 수 있습니다. 2024년 8월 이후 Amazon EKS API에 대한 모든 새로운 VPC 인터페이스 엔드포인트에는 두 개의 기본 리전 DNS 이름이 있으며 IP 주소 유형으로 dualstack을 선택할 수 있습니다. 첫 번째 DNS 이름은 eks.**region**.api.aws이며 이중 스택입니다. IPv4 주소와 IPv6 주소 모두로 확인됩니다. 2024년 8월 이전에는 Amazon EKS는 IPv4 주소로 확인되는 eks.**region**.amazonaws.com만 사용했습니다. 기존 VPC 인터페이스 엔드포인트와 함께 IPv6 및 이중 스택 IP 주소를 사용하려는 경우 dualstack 유형의 IP 주소를 사용하도록 엔드포인트를 업데이트할 수 있지만 eks.**region**.amazonaws.com DNS 이름만 가지게 됩니다. 이 구성에서는 기존 엔드포인트가 해당 이름이 IPv4 및 IPv6 IP 주소를 모두 가리키도록 업데이트됩니다. API 목록은 Amazon EKS API 참조의 [Actions](#)(작업)를 참조하세요.
- EKS API를 직접 호출하는 애플리케이션은 변경할 필요가 없습니다.

그러나 AWS CLI를 이중 스택 엔드포인트와 함께 사용하려면 AWS SDK 및 도구 참조 안내서의 [이중 스택 및 FIPS 엔드포인트 구성](#)을 참조하세요.

- Amazon EKS 기본 서비스 엔드포인트에 적용된 직접 호출은 프라이빗 AWS 네트워크에서 인터페이스 엔드포인트를 통해 자동으로 라우팅됩니다.

Amazon EKS 클러스터에서 복원력 이해

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전은 물리적으로 분리되고 격리된 다수의 가용 리전을 제공하며 이러한 가용 리전은 짧은 지연 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크에 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 복수 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

Amazon EKS는 여러 AWS 가용 영역에 걸쳐 Kubernetes 제어 플레인을 실행하고 크기 조정하여 높은 가용성을 보장합니다. Amazon EKS는 로드 에 따라 제어 영역 인스턴스의 크기를 자동으로 조정하고, 비정상 제어 영역 인스턴스를 감지하고 교체하며, 자동으로 제어 영역을 패치합니다. 버전 업데이트를 시작하면 Amazon EKS가 제어 영역을 업데이트하여 업데이트 중에 제어 영역의 고가용성을 유지합니다.

이 제어 플레인은 2개 이상의 API 서버 인스턴스와 AWS 리전 내 3개의 가용 영역에서 실행되는 3개의 etcd 인스턴스로 구성됩니다. Amazon EKS:

- 제어 플레인 인스턴스의 로드를 적극적으로 모니터링하고 자동으로 확장하여 높은 성능을 보장합니다.

- 비정상 제어 플레인 인스턴스를 자동으로 감지하고 교체하여 필요에 따라 AWS 리전 내의 가용 영역에서 다시 시작합니다.
- AWS 리전의 아키텍처를 활용하여고가용성을 유지합니다. 따라서 Amazon EKS는 [API 서버 엔드포인트 가용성을 위한 SLA](#)를 제공할 수 있습니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 [AWS 글로벌 인프라](#)를 참조하세요.

Kubernetes의 보안 고려 사항

다음은 Amazon EKS 클러스터에서 Kubernetes에 영향을 미치는 클라우드 보안 고려 사항입니다. Kubernetes의 보안 제어 및 관행에 대한 심층 검토는 Kubernetes Documentation의 [Cloud Native Security and Kubernetes](#)를 참조하세요.

주제

- [Kubernetes 인증서로 워크로드 보안](#)
- [Amazon EKS 기반 RBAC 역할 및 사용자 이해](#)
- [기존 클러스터에서 KMS를 사용하여 Kubernetes 비밀 암호화](#)
- [Amazon EKS 포드와 함께 AWS Secrets Manager 보안 암호 사용](#)
- [모든 Kubernetes API 데이터에 대한 기본 봉투 암호화](#)

Kubernetes 인증서로 워크로드 보안

Kubernetes Certificates API를 통해 [X.509](#) 자격 증명 프로비저닝이 자동화됩니다. API는 Kubernetes API 클라이언트가 인증 기관(CA)에서 [X.509 인증서](#)를 요청하고 받을 수 있는 명령줄 인터페이스를 제공합니다. 표시된 서명자가 인증서에 서명하도록 요청하는 데 CertificateSigningRequest(CSR) 리소스를 사용할 수 있습니다. 요청은 서명되기 전에 승인되거나 거부됩니다. Kubernetes는 잘 정의된 동작을 통해 기본 제공 서명자와 사용자 지정 서명자를 모두 지원합니다. 이렇게 하면 클라이언트는 CSR에 어떤 일이 일어나는지 예측할 수 있습니다. 인증서 서명에 대한 자세한 내용을 알아보려면 [서명 요청](#)을 참조하세요.

기본 제공 서명자 중 하나는 `kubernetes.io/legacy-unknown`입니다. CSR 리소스의 `v1beta1` API는 이 `legacy-unknown` 서명자를 준수했습니다. 그러나 CSR의 안정적인 `v1` API에서는 `signerName`이 `kubernetes.io/legacy-unknown`으로 설정되도록 허용하지 않습니다.

Amazon EKS CA를 사용하여 클러스터에서 인증서를 생성하려는 경우 사용자 지정 서명자를 사용해야 합니다. CSR `v1` API 버전을 사용하여 새 인증서를 생성하려면 기존 매니페스트 및 API 클라이언트

를 마이그레이션해야 합니다. 기존 v1beta1 API를 사용하여 생성된 기존 인증서는 유효하며 인증서가 만료될 때까지 기능합니다. 다음 내용이 포함됩니다:

- 신뢰 배포: 없음. Kubernetes 클러스터에는 이 서명자에 대한 표준 트러스트나 배포가 없습니다.
- 허용된 과목: 모두
- 허용된 x509 확장 기능: subjectAltName 및 키 사용 확장 기능을 준수하고 다른 확장 기능을 폐기합니다.
- 허용된 키 사용: ["키 암호화", "디지털 서명", "서버 인증"] 이외의 용도는 포함하지 않아야 합니다.

Note

클라이언트 인증서 서명은 지원되지 않습니다.

- 만료/인증서 수명: 1년(기본 및 최대)
- CA 비트 허용됨/허용되지 않음: 허용되지 않음

signerName을 사용한 CSR 생성 예

다음 단계에서는 `signerName: beta.eks.amazonaws.com/app-serving`을 사용하여 `myserver.default.svc` DNS 이름에 대한 서빙 인증서를 생성하는 방법을 보여줍니다. 이 정보를 사용자 환경에 대한 가이드로 사용합니다.

1. `openssl genrsa -out myserver.key 2048` 명령을 실행하여 RSA 프라이빗 키를 생성합니다.

```
openssl genrsa -out myserver.key 2048
```

2. 다음 명령을 실행하여 인증 요청을 생성합니다.

```
openssl req -new -key myserver.key -out myserver.csr -subj "/CN=myserver.default.svc"
```

3. CSR 요청에 대한 base64 값을 생성하고 이후 단계에서 사용할 수 있도록 변수에 저장합니다.

```
base_64=$(cat myserver.csr | base64 -w 0 | tr -d "
")
```

4. 다음 명령을 실행하여 `mycsr.yaml`이라는 파일을 생성합니다. 다음 예에서 `beta.eks.amazonaws.com/app-serving`은 `signerName`입니다.

```
cat >mycsr.yaml <<EOF
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: myserver
spec:
  request: $base_64
  signerName: beta.eks.amazonaws.com/app-serving
  usages:
    - digital signature
    - key encipherment
    - server auth
EOF
```

5. CSR을 제출합니다.

```
kubectl apply -f mycsr.yaml
```

6. 서빙 인증서를 승인합니다.

```
kubectl certificate approve myserver
```

7. 인증서가 발급되었는지 확인합니다.

```
kubectl get csr myserver
```

예제 출력은 다음과 같습니다.

NAME	AGE	SIGNERNAME	REQUESTOR	CONDITION
myserver	3m20s	beta.eks.amazonaws.com/app-serving	kubernetes-admin	Approved, Issued

8. 발급된 인증서를 내보냅니다.

```
kubectl get csr myserver -o jsonpath='{.status.certificate}' | base64 -d >
myserver.crt
```

Amazon EKS 기반 RBAC 역할 및 사용자 이해

Kubernetes 클러스터를 생성하면 Kubernetes가 제대로 작동하도록 해당 클러스터에 여러 개의 기본 Kubernetes ID가 생성됩니다. Amazon EKS는 각 기본 구성 요소에 대한 Kubernetes ID를 생성합니다. ID에서는 클러스터 구성 요소에 대한 Kubernetes RBAC(역할 기반 권한 부여 제어)가 제공됩니다. 자세한 내용은 Kubernetes 문서의 [Using RBAC Authorization\(RBAC 승인 사용\)](#)을 참조하십시오.

클러스터에 선택적 [추가 기능](#)을 설치하면 클러스터에 추가 Kubernetes ID가 추가될 수도 있습니다. 이 주제에서 다루지 않는 자격 증명에 대한 자세한 내용은 추가 기능에 대한 설명서를 참조하세요.

Amazon EKS에서 클러스터에 생성한 Kubernetes ID 목록은 AWS Management Console 또는 kubectl 명령줄 도구를 사용하여 볼 수 있습니다. 모든 사용자 자격 증명은 Amazon CloudWatch를 통해 사용할 수 있는 kube 감사 로그에 표시됩니다.

AWS Management Console

전제 조건

사용하는 [IAM 위탁자](#)는 [필수 권한](#)에 설명된 권한을 가지고 있어야 합니다.

AWS Management Console을 사용하여 Amazon EKS에서 생성한 자격 증명을 보는 방법

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 보려는 자격 증명이 포함된 클러스터를 클러스터 목록에서 선택합니다.
3. 리소스 탭을 선택합니다.
4. 리소스 유형에서 권한 부여를 선택합니다.
5. ClusterRoles, ClusterRoleBindings, Roles 또는 RoleBindings를 선택합니다. 앞에 eks로 시작하는 모든 리소스는 Amazon EKS에서 생성합니다. Amazon EKS에서 생성한 추가 자격 증명 리소스는 다음과 같습니다.
 - ClusterRole 및 aws-node라는 ClusterRoleBinding. aws-node 리소스에서는 Amazon EKS에서 모든 클러스터에 설치하는 [Kubernetes용 Amazon VPC CNI 플러그인](#)을 지원합니다.
 - vpc-resource-controller-role이라는 ClusterRole 및 vpc-resource-controller-rolebinding이라는 ClusterRoleBinding. 이러한 리소스에서는 Amazon EKS에서 모든 클러스터에 설치하는 [Amazon VPC 리소스 컨트롤러](#)를 지원합니다.

콘솔에 표시되는 리소스 외에 다음과 같은 특별한 사용자 자격 증명이 클러스터에 있지만, 클러스터의 구성에는 표시되지 않습니다.

- **eks:cluster-bootstrap** – 클러스터 부트스트랩 중 kubectl 작업에 사용됩니다.

- **eks:support-engineer** – 클러스터 관리 작업에 사용됩니다.
6. 특정 리소스를 선택하면 해당 세부 정보를 볼 수 있습니다. 기본적으로 구조화된 뷰에 정보가 표시됩니다. 세부 정보 페이지의 오른쪽 상단에서 Raw view(원시 뷰)를 선택하여 리소스에 대한 모든 정보를 볼 수 있습니다.

kubectl

전제 조건

클러스터에 Kubernetes 리소스를 나열하는 데 사용하는 엔터티(AWS Identity and Access Management(IAM) 또는 OpenID Connect (OIDC))는 IAM 또는 OIDC ID 제공업체에 의해 인증되어야 합니다. 엔터티를 연동하려는 클러스터의 Role, ClusterRole, RoleBinding 및 ClusterRoleBinding 리소스에 get Kubernetes 및 list 동사를 사용하는 권한이 엔터티에 부여되어야 합니다. IAM 엔터티에 클러스터 액세스 권한을 부여하는 자세한 내용은 [the section called “Kubernetes API 액세스”](#) 섹션을 참조하세요. 자체 OIDC 제공자에서 인증한 엔터티에 클러스터 액세스 권한을 부여하는 것에 대한 자세한 내용은 [the section called “OIDC 제공자 연결”](#) 섹션을 참조하세요.

kubectl을 사용하여 Amazon EKS에서 생성한 자격 증명을 보는 방법

표시하려는 리소스 유형에 대한 명령을 실행합니다. eks로 시작하는 반환된 모든 리소스는 Amazon EKS에서 생성합니다. 출력에서 반환되는 리소스 외에 다음과 같은 특별한 사용자 자격 증명에 클러스터에 있지만, 클러스터의 구성에는 표시되지 않습니다.

- **eks:cluster-bootstrap** – 클러스터 부트스트랩 중 kubectl 작업에 사용됩니다.
- **eks:support-engineer** – 클러스터 관리 작업에 사용됩니다.

ClusterRoles – ClusterRoles의 범위가 클러스터로 지정되므로 역할에 부여된 모든 권한이 클러스터의 모든 Kubernetes 네임스페이스에 있는 리소스에 적용됩니다.

다음 명령은 클러스터의 Amazon EKS 생성 Kubernetes ClusterRoles을 모두 반환합니다.

```
kubectl get clusterroles | grep eks
```

출력에서 반환되어 앞에 붙는 ClusterRoles 외에 다음과 같은 ClusterRoles가 있습니다.

- **aws-node** - 이 ClusterRole에서는 Amazon EKS가 모든 클러스터에 설치하는 [Kubernetes용 Amazon VPC CNI 플러그인](#)을 지원합니다.

- **vpc-resource-controller-role** – 이 ClusterRole에서는 Amazon EKS에서 모든 클러스터에 설치하는 [Amazon VPC 리소스 컨트롤러](#)를 지원합니다.

ClusterRole의 사양을 표시하려면 다음과 같은 명령의 *eks:k8s-metrics*를 이전 명령의 출력에서 반환된 ClusterRole로 바꿉니다. 다음 예시에서는 *eks:k8s-metrics* ClusterRole에 대한 사양을 반환합니다.

```
kubectl describe clusterrole eks:k8s-metrics
```

예제 출력은 다음과 같습니다.

```
Name:          eks:k8s-metrics
Labels:        <none>
Annotations:   <none>
PolicyRule:
  Resources      Non-Resource URLs  Resource Names  Verbs
  -----
  endpoints      [ ]                [ ]             [list]
  nodes          [ ]                [ ]             [list]
  pods           [ ]                [ ]             [list]
  deployments.apps [ ]                [ ]             [list]
```

ClusterRoleBindings – ClusterRoleBindings의 범위가 클러스터로 지정됩니다.

다음 명령은 클러스터의 Amazon EKS 생성 Kubernetes ClusterRoleBindings을 모두 반환합니다.

```
kubectl get clusterrolebindings | grep eks
```

출력에서 반환된 ClusterRoleBindings 외에 다음과 같은 ClusterRoleBindings가 있습니다.

- **aws-node** - 이 ClusterRoleBinding에서는 Amazon EKS가 모든 클러스터에 설치하는 [Kubernetes용 Amazon VPC CNI 플러그인](#)을 지원합니다.
- **vpc-resource-controller-rolebinding** – 이 ClusterRoleBinding에서는 Amazon EKS에서 모든 클러스터에 설치하는 [Amazon VPC 리소스 컨트롤러](#)를 지원합니다.

ClusterRoleBinding의 사양을 표시하려면 다음과 같은 명령의 *eks:k8s-metrics*를 이전 명령의 출력에서 반환된 ClusterRoleBinding로 바꿉니다. 다음 예시에서는 *eks:k8s-metrics* ClusterRoleBinding에 대한 사양을 반환합니다.

```
kubectl describe clusterrolebinding eks:k8s-metrics
```

예제 출력은 다음과 같습니다.

```
Name:          eks:k8s-metrics
Labels:        <none>
Annotations:   <none>
Role:
  Kind: ClusterRole
  Name:  eks:k8s-metrics
Subjects:
  Kind  Name          Namespace
  ----  ---
  User  eks:k8s-metrics
```

Roles – Roles의 범위가 Kubernetes 네임스페이스로 지정됩니다. Amazon EKS에서 생성한 모든 Roles의 범위가 kube-system 네임스페이스로 지정됩니다.

다음 명령은 클러스터의 Amazon EKS 생성 Kubernetes Roles을 모두 반환합니다.

```
kubectl get roles -n kube-system | grep eks
```

Role의 사양을 표시하려면 다음과 같은 명령의 *eks:k8s-metrics*를 이전 명령의 출력에서 반환된 Role의 이름으로 변경합니다. 다음 예시에서는 *eks:k8s-metrics* Role에 대한 사양을 반환합니다.

```
kubectl describe role eks:k8s-metrics -n kube-system
```

예제 출력은 다음과 같습니다.

```
Name:          eks:k8s-metrics
Labels:        <none>
Annotations:   <none>
PolicyRule:
  Resources      Non-Resource URLs  Resource Names      Verbs
  -----
  daemonsets.apps  []                [aws-node]          [get]
  deployments.apps  []                [vpc-resource-controller] [get]
```

RoleBindings – RoleBindings의 범위가 Kubernetes 네임스페이스로 지정됩니다. Amazon EKS에서 생성한 모든 RoleBindings의 범위가 kube-system 네임스페이스로 지정됩니다.

다음 명령은 클러스터의 Amazon EKS 생성 Kubernetes RoleBindings을 모두 반환합니다.

```
kubectl get rolebindings -n kube-system | grep eks
```

RoleBinding의 사양을 표시하려면 다음과 같은 명령의 *eks:k8s-metrics*를 이전 명령의 출력에서 반환된 RoleBinding로 바꿉니다. 다음 예시에서는 *eks:k8s-metrics* RoleBinding에 대한 사양을 반환합니다.

```
kubectl describe rolebinding eks:k8s-metrics -n kube-system
```

예제 출력은 다음과 같습니다.

```
Name:          eks:k8s-metrics
Labels:        <none>
Annotations:   <none>
Role:
  Kind:  Role
  Name:  eks:k8s-metrics
Subjects:
  Kind  Name          Namespace
  ----  ---          -
  User  eks:k8s-metrics
```

기존 클러스터에서 KMS를 사용하여 Kubernetes 비밀 암호화

Important

이 절차는 Kubernetes 버전 1.27 이하가 실행되는 EKS 클러스터에만 적용됩니다. Kubernetes 버전 1.28 이상을 실행하는 경우 Kubernetes 비밀은 기본적으로 봉투 암호화로 보호됩니다. 자세한 내용은 [the section called “모든 Kubernetes API 데이터에 대한 기본 봉투 암호화”](#) 단원을 참조하십시오.

[비밀 암호화](#)를 사용 설정하는 경우 Kubernetes 비밀은 선택한 AWS KMS 키를 사용하여 암호화됩니다. KMS 키는 다음 조건을 충족해야 합니다.

- 대칭
- 데이터 암호화 및 해독 가능
- 클러스터와 같은 AWS 리전에 생성됨

- KMS 키가 다른 계정에서 생성된 경우 [IAM 보안 주체](#)는 KMS 키에 액세스할 수 있어야 합니다.

자세한 내용은 [AWS 키 관리 서비스 개발자 가이드](#)에서 [다른 계정의 IAM 주체가 KMS 키를 사용하도록 허용하기](#)를 참조하세요.

 Warning

비밀 암호화를 사용 설정한 후에는 사용 중지할 수 없습니다. 이 작업은 되돌릴 수 없습니다.

eksctl

이 절차는 Kubernetes 버전 1.27 이하가 실행되는 EKS 클러스터에만 적용됩니다. 자세한 내용은 [the section called “모든 Kubernetes API 데이터에 대한 기본 봉투 암호화”](#) 단원을 참조하십시오.

다음 두 가지 방법으로 암호화를 활성화할 수 있습니다.

- 단일 명령으로 클러스터에 암호화를 추가합니다.

비밀을 자동으로 다시 암호화하려면 다음 명령을 실행합니다.

```
eksctl utils enable-secrets-encryption \
  --cluster my-cluster \
  --key-arn arn:aws:kms:region-code:account:key/key
```

비밀을 자동으로 다시 암호화하는 것을 옵트아웃하려면 다음 명령을 실행합니다.

```
eksctl utils enable-secrets-encryption
  --cluster my-cluster \
  --key-arn arn:aws:kms:region-code:account:key/key \
  --encrypt-existing-secrets=false
```

- kms-cluster.yaml 파일을 사용하여 클러스터에 암호화를 추가합니다.

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
```



```
secretsEncryption:
  keyARN: arn:aws:kms:region-code:account:key/key
```

비밀을 자동으로 다시 암호화하려면 다음 명령을 실행합니다.

```
eksctl utils enable-secrets-encryption -f kms-cluster.yaml
```

비밀을 자동으로 다시 암호화하는 것을 옵트아웃하려면 다음 명령을 실행합니다.

```
eksctl utils enable-secrets-encryption -f kms-cluster.yaml --encrypt-existing-secrets=false
```

AWS Management Console

- 이 절차는 Kubernetes 버전 1.27 이하가 실행되는 EKS 클러스터에만 적용됩니다. 자세한 내용은 [the section called “모든 Kubernetes API 데이터에 대한 기본 봉투 암호화”](#) 단원을 참조하십시오.
- [Amazon EKS 콘솔](#)을 엽니다.
- KMS 암호화를 추가하려는 클러스터를 선택합니다.
- 개요 탭을 선택합니다(기본적으로 선택되어 있음).
- 아래로 스크롤하여 비밀 암호화 섹션으로 이동하고 활성화를 선택합니다.
- 드롭다운 목록에서 키를 선택하고 활성화 버튼을 선택합니다. 나열된 키가 없으면 먼저 키를 생성해야 합니다. 자세한 내용은 [키 생성](#)을 참조하세요.
- 확인 버튼을 선택하고 선택한 키를 사용합니다.

AWS CLI

- 이 절차는 Kubernetes 버전 1.27 이하가 실행되는 EKS 클러스터에만 적용됩니다. 자세한 내용은 [the section called “모든 Kubernetes API 데이터에 대한 기본 봉투 암호화”](#) 단원을 참조하십시오.
- 다음 AWS CLI 명령을 사용하여 [비밀 암호화](#) 구성을 클러스터와 연결합니다. **##** **#**을 사용자의 값으로 바꿉니다.

```
aws eks associate-encryption-config \
  --cluster-name my-cluster \
  --encryption-config '[{"resources":["secrets"],"provider":\
  {"keyArn":"arn:aws:kms:region-code:account:key/key"}}]'
```

예제 출력은 다음과 같습니다.

```
{
  "update": {
    "id": "3141b835-8103-423a-8e68-12c2521ffa4d",
    "status": "InProgress",
    "type": "AssociateEncryptionConfig",
    "params": [
      {
        "type": "EncryptionConfig",
        "value": "[{\"resources\":[\"secrets\"],\"provider\":{\"keyArn\":\"arn:aws:kms:region-code:account:key/key\"}}]"
      }
    ],
    "createdAt": 1613754188.734,
    "errors": []
  }
}
```

- c. 다음 명령을 사용하여 암호화 업데이트의 상태를 모니터링할 수 있습니다. 이전 출력에서 반환된 특정 cluster name 및 update ID를 사용합니다. Successful 상태가 표시되면 업데이트가 완료됩니다.

```
aws eks describe-update \
  --region region-code \
  --name my-cluster \
  --update-id 3141b835-8103-423a-8e68-12c2521ffa4d
```

예제 출력은 다음과 같습니다.

```
{
  "update": {
    "id": "3141b835-8103-423a-8e68-12c2521ffa4d",
    "status": "Successful",
    "type": "AssociateEncryptionConfig",
    "params": [
      {
        "type": "EncryptionConfig",
        "value": "[{\"resources\":[\"secrets\"],\"provider\":{\"keyArn\":\"arn:aws:kms:region-code:account:key/key\"}}]"
      }
    ]
  }
}
```

```

    ],
    "createdAt": 1613754188.734>,
    "errors": []
  }
}

```

- d. 클러스터에서 암호화가 활성화되었는지 확인하려면 `describe-cluster` 명령을 실행합니다. 응답에는 `EncryptionConfig` 문자열이 포함됩니다.

```
aws eks describe-cluster --region region-code --name my-cluster
```

클러스터에서 암호화를 사용 설정한 후에는 기존 비밀을 모두 새 키로 암호화해야 합니다.

Note

`eksctl`을 사용하면 비밀을 자동으로 다시 암호화하는 것을 옵트아웃하는 경우에만 다음 명령을 실행해야 합니다.

```
kubectl get secrets --all-namespaces -o json | kubectl annotate --overwrite -f - kms-encryption-timestamp="time value"
```

Warning

기존 클러스터에 대해 [비밀 암호화](#)를 사용 설정하고 사용하는 KMS 키가 삭제된 경우에는 클러스터 복구 경로가 없습니다. KMS 키를 삭제하면 클러스터가 영구적으로 성능 저하 상태가 됩니다. 자세한 내용은 [AWS KMS 키](#)를 참조하세요.

Note

기본적으로 `create-key` 명령은 AWS KMS 작업 및 리소스에 대한 액세스 권한을 계정 루트 관리자에게 부여하는 키 정책으로 [대칭 암호화 KMS 키](#)를 생성합니다. 권한을 축소하려면 `create-cluster` API를 호출할 보안 주체의 정책에서 `kms:DescribeKey` 및 `kms:CreateGrant` 작업이 허용되는지 확인합니다.

KMS 봉투 암호화를 사용하는 클러스터의 경우 `kms:CreateGrant` 권한이 필요합니다. 조건 `kms:GrantIsForAWSResource`는 `CreateCluster` 작업에 대해 지원되지 않으며,

CreateCluster를 수행하는 사용자에게 `kms:CreateGrant` 권한을 제어하기 위해 KMS 정책을 사용해서는 안 됩니다.

Amazon EKS 포드와 함께 AWS Secrets Manager 보안 암호 사용

Secrets Manager의 보안 암호 및 파라미터 스토어의 파라미터를 Amazon EKS 포드에 탑재된 파일로 표시하려면 [Kubernetes 보안 암호 스토어 CSI 드라이버](#)에 대한 AWS 보안 암호 및 구성 제공업체(ASCP)를 사용할 수 있습니다.

ASCP를 사용하면 Secrets Manager 보안 암호를 저장 및 관리한 후 Amazon EKS에서 실행되는 워크로드를 통해 해당 검색할 수 있습니다. IAM 역할 및 정책을 사용하여 보안 암호에 대한 액세스 권한을 클러스터의 특정 Kubernetes 포드로 제한할 수 있습니다. ASCP는 Pod Identity를 검색하고 IAM 역할에 대한 ID를 교환합니다. ASCP는 포드의 IAM 역할을 가정한 후 해당 역할에 대해 인증된 Secrets Manager에서 보안 암호를 검색할 수 있습니다.

보안 암호에 대해 Secrets Manager 자동 교체를 사용하는 경우 보안 암호 스토어 CSI 드라이버 교체 조정자 기능을 사용하여 Secrets Manager에서 최신 보안 암호를 검색할 수도 있습니다.

Note

AWS Fargate(Fargate) 노드 그룹은 지원되지 않습니다.

자세한 내용은 AWS Secrets Manager 사용 설명서의 [Amazon EKS에서 Secrets Manager 보안 암호 사용](#)을 참조하세요.

모든 Kubernetes API 데이터에 대한 기본 봉투 암호화

Amazon Elastic Kubernetes Service(Amazon EKS)는 Kubernetes 버전 1.28 이상을 실행하는 EKS 클러스터의 모든 Kubernetes API 데이터에 기본 봉투 암호화를 제공합니다.

봉투 암호화는 Kubernetes API 서버에 저장하는 데이터를 보호합니다. 예를 들어, 봉투 암호화는 ConfigMaps와 같은 Kubernetes 클러스터의 구성에 적용됩니다. 노드 또는 EBS 볼륨의 데이터에는 봉투 암호화가 적용되지 않습니다. EKS에서는 이전에 Kubernetes 비밀 암호화를 지원했으며, 이제 이 봉투 암호화는 모든 Kubernetes API 데이터로 확장됩니다.

이를 통해 Kubernetes 애플리케이션에 심층 방어를 구현하고 사용자 측에서 별도의 조치가 필요하지 않은 관리형 기본 환경을 제공합니다.

Amazon EKS는 [Amazon Web Services 소유 키](#)를 사용하는 이 추가 보안 계층을 위해 [Kubernetes KMS 공급자 v2](#)에 AWS [Key Management Service\(KMS\)](#)를 사용하고, AWS KMS에서 자체 [고객 관리 형 키](#)(CMK)를 가져올 수 있는 옵션을 사용합니다.

봉투 암호화 이해

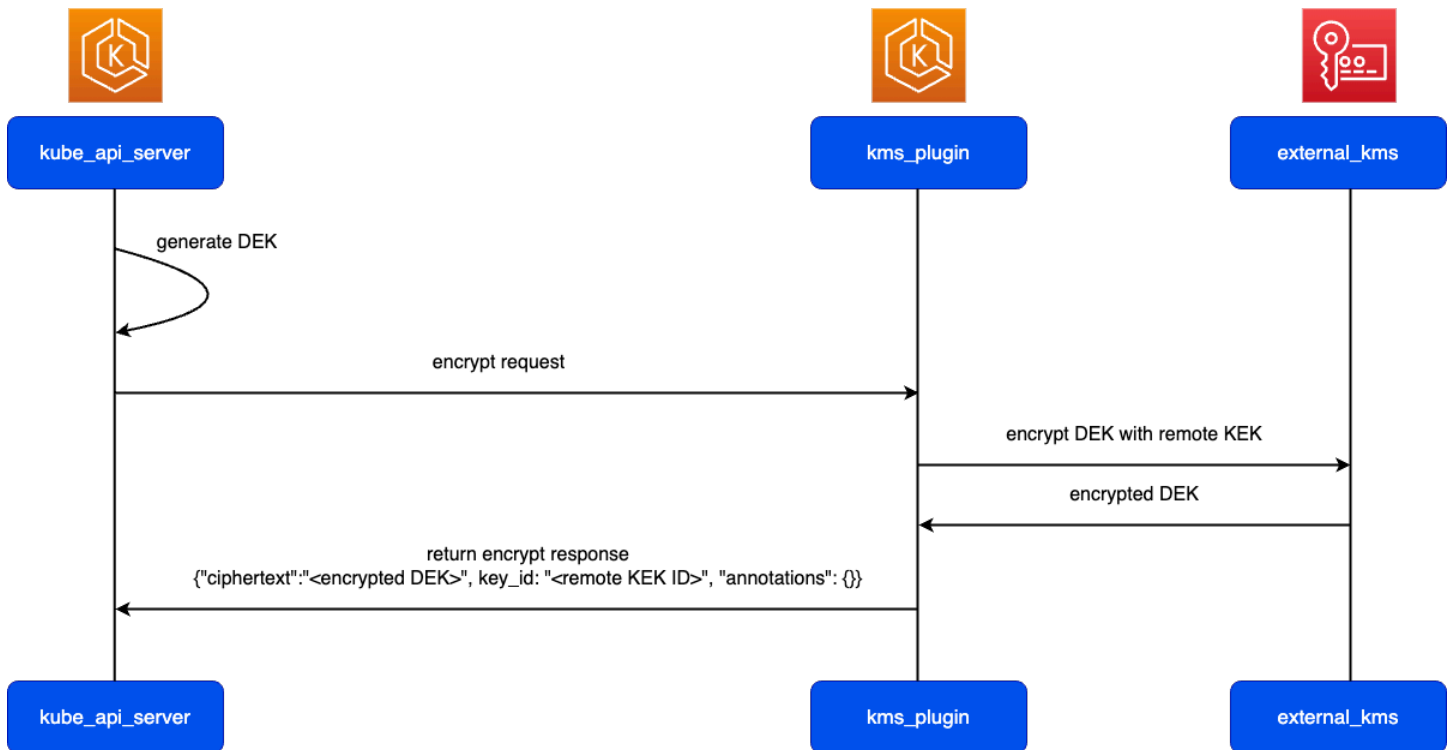
봉투 암호화는 데이터 암호화 키(DEK)를 사용하여 일반 텍스트 데이터를 데이터 스토어(etcd)로 전송하기 전에 암호화한 다음, 원격 중앙 관리형 KMS 시스템(AWS KMS)에 저장된 루트 KMS 키로 DEK를 암호화하는 프로세스입니다. 이는 암호화 키(DEK)로 데이터를 보호한 다음 키 암호화 키(KEK)라고 하는 별도의 안전하게 저장된 암호화 키로 해당 DEK를 보호하여 보안 계층을 더하는 심층 방어 전략입니다.

Amazon EKS가 KMS v2 및 AWS KMS를 사용하여 기본 봉투 암호화를 활성화하는 방법

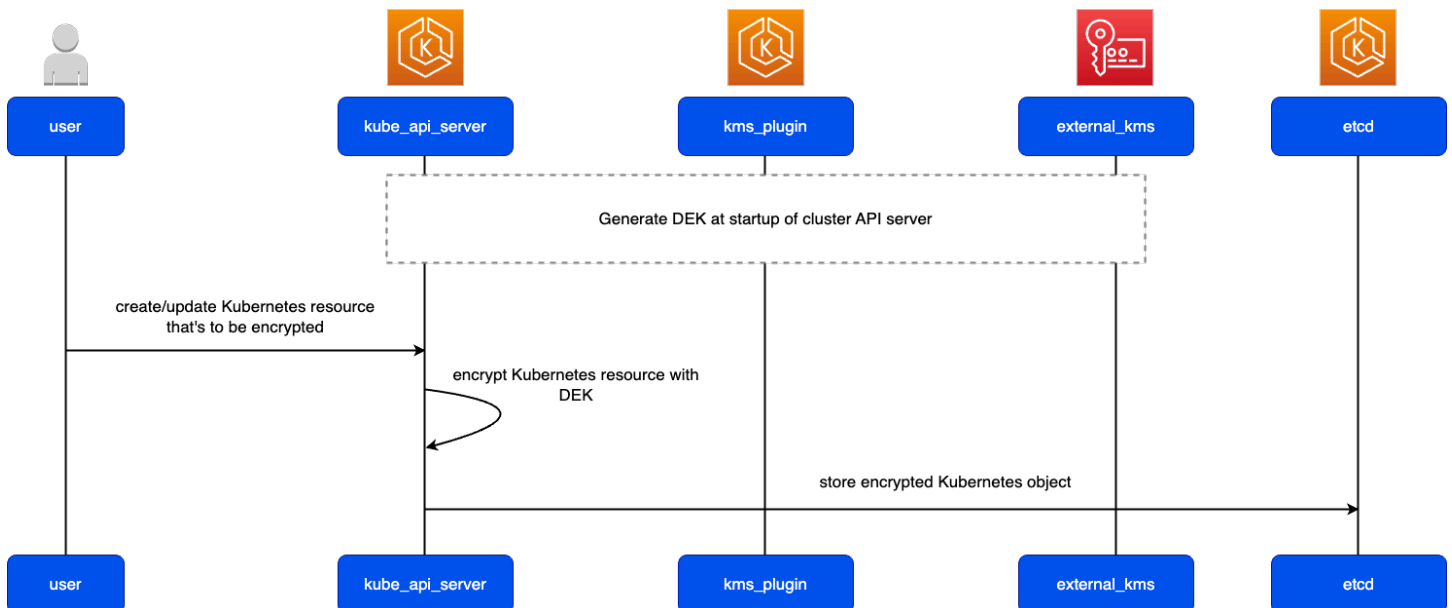
Amazon EKS는 [KMS v2](#)를 사용하여 관리형 Kubernetes 컨트롤 플레인의 모든 API 데이터가 [etcd](#) 데이터베이스에 유지되기 전에 기본 봉투 암호화를 구현합니다. 시작 시 클러스터 API 서버는 임의로 생성된 데이터와 결합된 보안 암호 시드로부터 데이터 암호화 키(DEK)를 생성합니다. 또한 시작 시 API 서버는 KMS 플러그인을 호출하여 AWS KMS의 원격 키 암호화 키(KEK)를 사용하여 DEK 시드를 암호화합니다. 이는 API 서버 시작 및 KEK 교체 시 실행되는 일회성 호출입니다. 이후 API 서버는 암호화된 DEK 시드를 캐싱합니다. 그런 다음 API 서버는 캐싱된 DEK 시드를 사용하여 키 파생 함수(KDF)를 기반으로 다른 일회용 DEK를 생성합니다. 생성된 각 DEK는 etcd에 저장되기 전에 단일 Kubernetes 리소스를 암호화하는 데 한 번만 사용됩니다. KMS v2에서 암호화된 캐싱된 DEK 시드를 사용하면 API 서버에서 Kubernetes 리소스를 암호화하는 프로세스의 성능과 비용 효율성이 개선됩니다.

기본적으로 이 KEK는 AWS에서 소유하지만 선택적으로 AWS KMS에서 직접 가져올 수 있습니다.

아래 다이어그램은 API 서버를 시작할 때 DEK의 생성 및 암호화를 보여줍니다.



아래의 개략적인 다이어그램은 Kubernetes 리소스가 etcd에 저장되기 전에 해당 리소스의 암호화를 보여줍니다.



자주 묻는 질문(FAQ)

기본 봉투 암호화는 EKS 클러스터의 보안 태세를 어떻게 개선하나요?

이 기능은 메타데이터와 고객 콘텐츠가 암호화되지 않은 표면적과 기간을 줄입니다. 기본 봉투 암호화를 사용하면 메타데이터와 고객 콘텐츠가 etcd에 저장되기 전에 kube-apiserver의 메모리에서 일시적으로 암호화되지 않은 상태에 불과합니다. kube-apiserver의 메모리는 [Nitro 시스템](#)을 통해 보호됩니다. Amazon EKS는 관리형 Kubernetes 컨트롤 플레인에 [Nitro 기반 EC2 인스턴스](#)만 사용합니다. 이러한 인스턴스에는 시스템이나 사용자가 메모리에 액세스하지 못하도록 하는 보안 제어 설계가 마련되어 있습니다.

이 기능을 사용하려면 어떤 버전의 Kubernetes를 실행해야 하나요?

기본 봉투 암호화를 활성화하려면 Amazon EKS 클러스터에서 Kubernetes 버전 1.28 이상을 실행해야 합니다.

이 기능을 지원하지 않는 Kubernetes 클러스터 버전을 실행하는 경우에도 데이터는 여전히 보호되나요?

예. AWS에서 [가장 우선순위가 높은 것은 보안](#)입니다. 모든 디지털 트랜스포메이션과 혁신을 가장 높은 수준의 보안 운영 관행의 기반으로 하고, 이러한 기준을 높이기 위해 최선을 다합니다.

etcd에 저장된 모든 데이터는 실행 중인 Kubernetes 버전과 무관하게 모든 EKS 클러스터의 디스크 수준에서 암호화됩니다. EKS는 EKS 서비스에서 관리하는 볼륨 암호화 키를 생성하는 루트 키를 사용합니다. 추가로 모든 Amazon EKS 클러스터는 클러스터별 가상 머신을 사용하여 격리된 VPC에서 실행됩니다. 이 아키텍처와 운영 보안에 대한 관행으로 인해 Amazon EKS는 SOC 1,2,3, PCI-DSS, ISO 및 HIPAA 자격을 포함하여 [여러 규정 준수 등급 및 표준을 달성](#)했습니다. 이러한 규정 준수 등급 및 표준은 기본 봉투 암호화가 활성화 또는 활성화되지 않은 모든 EKS 클러스터에 대해 유지됩니다.

Amazon EKS에서 봉투 암호화는 어떻게 작동하나요?

시작 시 클러스터 API 서버는 임의로 생성된 데이터와 결합된 보안 암호 시드로부터 데이터 암호화 키(DEK)를 생성합니다. 또한 시작 시 API 서버는 KMS 플러그인을 호출하여 AWS KMS의 원격 키 암호화 키(KEK)를 사용하여 DEK를 암호화합니다. 이는 API 서버 시작 및 KEK 교체 시 실행되는 일회성 호출입니다. 이후 API 서버는 암호화된 DEK 시드를 캐싱합니다. 그런 다음 API 서버는 캐싱된 DEK 시드를 사용하여 키 파생 함수(KDF)를 기반으로 다른 일회용 DEK를 생성합니다. 생성된 각 DEK는 etcd에 저장되기 전에 단일 Kubernetes 리소스를 암호화하는 데 한 번만 사용됩니다.

AWS KMS 통합의 상태와 정상 기능을 확인하기 위해 API 서버에서 추가로 호출하는 것이 중요합니다. 이러한 추가 상태 확인은 AWS CloudTrail에 표시됩니다.

이 기능이 EKS 클러스터에서 작동하려면 어떤 작업을 하거나 어떤 권한을 변경해야 하나요?

아니요, 별도의 조치를 할 필요는 없습니다. Amazon EKS의 봉투 암호화는 이제 Kubernetes 버전 1.28 이상을 실행하는 모든 클러스터에서 활성화된 기본 구성입니다. AWS KMS 통합은 AWS에서 관리하는 Kubernetes API 서버에 의해 설정됩니다. 즉, 클러스터에 KMS 암호화 사용을 시작하기 위해 권한을 구성할 필요가 없습니다.

클러스터에서 기본 봉투 암호화가 활성화되어 있는지 어떻게 알 수 있나요?

자체 CMK를 사용하도록 마이그레이션하면 클러스터와 연결된 KMS 키의 ARN이 표시됩니다. 추가로 클러스터의 CMK 사용과 관련된 AWS CloudTrail 이벤트 로그를 볼 수 있습니다.

클러스터가 AWS 소유 키를 사용하는 경우 EKS 콘솔에서 자세히 설명합니다(키의 ARN 제외).

AWS가 Amazon EKS에서 기본 봉투 암호화에 사용되는 AWS 소유 키에 액세스할 수 있나요?

아니요. AWS의 Amazon EKS의 엄격한 보안 제어 기능이 있어 모든 사용자가 etcd 데이터베이스의 데이터를 보호하는 데 사용되는 일반 텍스트 암호화 키에 액세스하지 못하도록 합니다. 이러한 보안 조치는 AWS 소유 KMS 키에도 적용됩니다.

기존 EKS 클러스터에서 기본 봉투 암호화가 활성화되나요?

Kubernetes 버전 1.28 이상에서 Amazon EKS 클러스터를 실행하는 경우 모든 Kubernetes API 데이터의 봉투 암호화가 활성화됩니다. 기존 클러스터의 경우 Amazon EKS는 eks:kms-storage-migrator RBAC ClusterRole을 사용하여 이전에 etcd에서 봉투 암호화되지 않은 데이터를 이 새 암호화 상태로 마이그레이션합니다.

EKS 클러스터에서 비밀에 대한 봉투 암호화를 이미 활성화한 경우 이는 무엇을 의미하나요?

Kubernetes 비밀을 봉투 암호화하는 데 사용된 KMS의 기존 고객 관리형 키(CMK)가 있는 경우 클러스터의 모든 Kubernetes API 데이터 유형의 봉투 암호화를 위한 KEK로 동일한 키가 사용됩니다.

기본 봉투 암호화를 사용하여 EKS 클러스터를 실행하는 데 추가 비용이 발생하나요?

기본 봉투 암호화에 [Amazon Web Services 소유 키](#)를 사용하는 경우 관리형 Kubernetes 컨트롤 플레인과 관련된 추가 비용이 없습니다. 기본적으로 Kubernetes 버전 1.28 이상을 실행하는 모든 EKS 클러스터는 [Amazon Web Service 소유 키](#)를 사용합니다. 하지만 자체 AWS KMS 키를 사용하는 경우 일반 [KMS 요금](#)이 적용됩니다.

자체 AWS KMS 키를 사용하여 클러스터의 Kubernetes API 데이터를 암호화하는 데 드는 비용은 얼마 인가요?

생성하거나 KMS로 가져오는 사용자 지정 키를 저장하기 위해 매월 1 USD를 지불합니다. KMS는 암호화 및 복호화 요청에 대해 요금을 부과합니다. 계정당 매달 20,000개의 요청으로 구성된 프리 티어가 있으며, 사용자는 매달 프리 티어를 초과하는 요청 10,000개당 0.03 USD를 지불합니다. 이는 계정의 모든 KMS 사용량에 적용되므로 클러스터에서 자체 AWS KMS 키를 사용하는 비용은 계정 내 다른 클러스터 또는 AWS 리소스에서 이 키를 사용하는 데 영향을 받습니다.

이제 고객 관리형 키(CMK)가 비밀뿐만 아니라 모든 Kubernetes API 데이터를 봉투 암호화하는 데 사용되므로 KMS 요금이 증가하나요?

아니요. KMS v2를 사용하여 구현하면 AWS KMS에 대한 호출 수가 크게 감소합니다. 그러면 EKS 클러스터에서 암호화 또는 복호화되는 추가 Kubernetes 데이터와 무관하게 CMK와 관련된 비용이 절감됩니다.

위에서 자세히 설명한 대로 Kubernetes 리소스의 암호화에 사용되는 생성된 DEK 시드는 원격 KEK로 암호화된 후 Kubernetes API 서버의 캐시에 로컬로 저장됩니다. 암호화된 DEK 시드가 API 서버의 캐시에 없는 경우 API 서버는 AWS KMS를 호출하여 DEK 시드를 암호화합니다. 이후 API 서버는 KMS를 호출하지 않고 클러스터에서 나중에 사용할 수 있도록 암호화된 DEK 시드를 캐싱합니다. 마찬가지로 복호화 요청의 경우 API 서버는 첫 번째 복호화 요청을 위해 AWS KMS를 호출하고, 이후에는 복호화된 DEK 시드가 캐싱되어 향후 복호화 작업에 사용됩니다.

자세한 내용은 GitHub의 Kubernetes 개선 사항에서 [KEP-3299: KMS v2 개선 사항](#)을 참조하세요.

여러 개의 Amazon EKS 클러스터에 동일한 CMK 키를 사용할 수 있나요?

예. 키를 다시 사용하려면 생성 중에 ARN을 클러스터와 연결하여 동일한 리전의 클러스터에 연결할 수 있습니다. 하지만 여러 EKS 클러스터에 동일한 CMK를 사용하는 경우 CMK의 임의 비활성화를 방지하기 위해 필요한 조치를 취해야 합니다. 그렇지 않으면 여러 EKS 클러스터와 연결된 비활성화된 CMK는 해당 키에 따라 클러스터에 더 광범위한 영향을 미칩니다.

기본 봉투 암호화가 활성화된 후 CMK를 사용할 수 없게 되면 EKS 클러스터는 어떻게 되나요?

KMS 키를 비활성화하면 다시 활성화할 때까지 어떠한 [암호화 작업](#)에서도 사용할 수 없습니다. 기존 CMK에 액세스하지 않은 상태에서 API 서버는 새로 생성된 Kubernetes 객체를 암호화 및 유지하는 것은 물론 앞서 암호화되어 etcd에서 저장된 Kubernetes 객체를 복호화할 수 없습니다. CMK가 비활성화된 경우 클러스터는 즉시 비정상/저하 상태로 전환되고, 이때 연결된 CMK를 다시 활성화할 때까지 [서비스 약정](#)을 이행할 수 없습니다.

CMK가 비활성화되면 EKS 클러스터가 저하된 상태이며 Kubernetes 컨트롤 플레인 리소스를 성공적으로 복원하기 위해 비활성화한 후 30일 이내에 CMK를 다시 활성화해야 한다는 알림을 받게 됩니다.

비활성화/삭제된 CMK의 영향으로부터 EKS 클러스터를 보호하려면 어떻게 해야 하나요?

이러한 문제로부터 EKS 클러스터를 보호하기 위해 키 관리자는 최소 권한 원칙이 있는 IAM 정책을 사용하여 KMS 키 작업에 대한 액세스를 관리하여 EKS 클러스터와 연결된 키의 임의의 비활성화 또는 삭제 위험을 줄여야 합니다. 추가로 CMK 상태에 대한 알림을 받도록 [CloudWatch 경보](#)를 설정할 수 있습니다.

CMK를 다시 활성화하면 EKS 클러스터가 복원되나요?

EKS 클러스터를 성공적으로 복원하려면 비활성화된 후 30일 이내에 CMK를 다시 활성화하는 것이 좋습니다. 하지만 EKS 클러스터의 성공적인 복원은 클러스터가 비정상/저하 상태인 동안 발생할 수 있는 자동 Kubernetes 업그레이드로 인해 API 중단 변경이 발생하는지 여부에 따라 달라집니다.

CMK를 비활성화한 후 EKS 클러스터가 비정상/저하된 상태로 전환되는 이유는 무엇인가요?

EKS 컨트롤 플레인의 API 서버는 암호화되고 API 서버의 메모리에 캐싱되는 DEK 키를 사용하여 생성/업데이트 작업 중에 모든 객체를 암호화한 후 etcd에 저장합니다. 기존 객체가 etcd에서 검색되는 경우 API 서버는 동일한 캐싱된 DEK 키를 사용하고 Kubernetes 리소스 객체를 복호화합니다. CMK를 비활성화하면 API 서버 메모리에 캐싱된 DEK 키로 인해 API 서버에 즉각적으로 미치는 영향이 보이지 않습니다. 하지만 API 서버 인스턴스가 다시 시작되면 캐싱된 DEK가 없으며 암호화 및 복호화 작업을 위해 AWS KMS를 호출해야 합니다. CMK가 없으면 이 프로세스가 KMS_KEY_DISABLED 오류 코드와 함께 실패하여 API 서버가 부팅되지 않습니다.

CMK를 삭제하면 EKS 클러스터는 어떻게 되나요?

EKS 클러스터와 연결된 CMK 키를 삭제하면 복구 이후 상태가 저하됩니다. 클러스터의 CMK가 없으면 API 서버는 더 이상 새로운 Kubernetes 객체를 암호화 및 유지하는 것은 물론 앞서 암호화되어 etcd 데이터베이스에서 저장된 Kubernetes 객체를 복호화할 수 없습니다. EKS 클러스터를 더 이상 사용할 필요가 없다고 확인된 경우에만 EKS 클러스터에 대한 CMK 키 삭제를 진행해야 합니다.

CMK를 찾을 수 없거나(KMS_KEY_NOT_FOUND) 클러스터와 연결된 CMK에 대한 권한 부여가 취소된 경우(KMS_GRANT_REVOKED) 클러스터를 복구할 수 없습니다. 클러스터 상태 및 오류 코드에 대한 자세한 내용은 [클러스터 상태 FAQ 및 해결 경로 포함 오류 코드](#)를 참조하세요.

CMK를 비활성화 또는 삭제했기 때문에 성능이 저하되거나 비정상인 EKS 클러스터에도 요금이 계속 부과되나요?

예. 비활성화된 CMK의 경우 EKS 컨트롤 플레인을 사용할 수 없지만 AWS는 고객이 삭제할 때까지 EKS 클러스터에 할당된 전용 인프라 리소스를 계속 실행합니다. 또한 EKS 클러스터의 정상적인 상태

및 작동을 방해하는 고객의 자발적인 조치 또는 비조치이므로 이러한 상황에서는 [서비스 약정](#)이 적용되지 않습니다.

비활성화된 CMK로 인해 EKS 클러스터가 비정상/저하 상태일 때 자동으로 업그레이드할 수 있나요?

예. 하지만 클러스터에 비활성화된 CMK가 있는 경우 30일 이내에 클러스터를 다시 활성화해야 합니다. 이 30일 기간 동안에는 Kubernetes 클러스터가 자동으로 업그레이드되지 않습니다. 하지만 이 기간이 경과하고 CMK를 다시 활성화하지 않으면 EKS의 Kubernetes 버전 수명 주기에 따라 클러스터가 표준 지원을 받는 다음 버전(n+1)으로 자동 업그레이드됩니다.

영향을 받는 클러스터를 인식하면 비활성화된 CMK를 빠르게 다시 활성화하는 것이 좋습니다. EKS는 영향을 받는 클러스터를 자동으로 업그레이드하지만, 특히 클러스터가 여러 차례 자동으로 업그레이드되는 경우 Kubernetes API에 대한 변경 사항과 API 서버의 부트스트랩 프로세스에서 예기치 않은 동작이 포함될 수 있으므로 성공적인 복구가 보장되지 않습니다.

KMS 키 별칭을 사용할 수 있나요?

예. Amazon EKS는 [KMS 키 별칭 사용을 지원합니다](#). 별칭은 [Amazon Web Service KMS 키](#)의 친숙한 이름입니다. 예를 들어 별칭을 사용하면 KMS 키를 **1234abcd-12ab-34cd-56ef-1234567890ab** 대신 **my-key**로 지칭할 수 있습니다.

자체 Kubernetes 백업 솔루션을 사용하여 클러스터 리소스를 계속 백업하고 복원할 수 있나요?

예. Kubernetes 클러스터 재해 복구, 데이터 마이그레이션 및 데이터 보호를 위해 Kubernetes 백업 솔루션(예: [Velero](#))을 사용할 수 있습니다. API 서버를 통해 클러스터 리소스에 액세스하는 Kubernetes 백업 솔루션을 실행하는 경우 애플리케이션에서 검색하는 모든 데이터가 클라이언트에 도달하기 전에 복호화됩니다. 이렇게 하면 다른 Kubernetes 클러스터의 클러스터 리소스를 복구할 수 있습니다.

Amazon EKS Auto Mode의 보안 고려 사항

이 주제에서는 Amazon EKS Auto Mode의 보안 아키텍처, 제어 및 모범 사례를 설명합니다. 조직이 대규모로 컨테이너화된 애플리케이션을 배포함에 따라 강력한 보안 태세를 유지하기가 점점 복잡해지고 있습니다. EKS Auto Mode는 자동화된 보안 제어를 구현하고 AWS 보안 서비스와 통합하여 클러스터 인프라, 워크로드, 데이터를 보호합니다. 강제 노드 수명 주기 관리, 자동 패치 배포와 같은 기본 제공 보안 기능으로 EKS Auto Mode는 운영 오버헤드를 줄이면서 보안 모범 사례를 유지하는 데 도움이 됩니다.

이 주제를 진행하기 전에 기본 EKS Auto Mode 개념을 숙지하고 클러스터에서 EKS Auto Mode를 활성화하기 위한 사전 조건을 검토하세요. Amazon EKS 보안에 대한 일반적인 내용은 [보안](#) 섹션을 참조하세요.

Amazon EKS Auto Mode는 Amazon EKS의 기존 보안 기반을 바탕으로 하며 EC2 관리형 인스턴스에 대한 추가 자동 보안 제어를 도입합니다.

API 보안 및 인증

Amazon EKS Auto Mode는 AWS 플랫폼 보안 메커니즘을 사용하여 Amazon EKS API에 대한 호출을 보호하고 인증합니다.

- Kubernetes API에 대한 액세스는 AWS IAM 자격 증명과 통합되는 EKS 액세스 항목을 통해 보호됩니다.
- 자세한 내용은 [the section called “권한 부여”](#) 단원을 참조하십시오.
- 고객은 EKS 액세스 항목 구성을 통해 Kubernetes API 엔드포인트에 대한 세분화된 액세스 제어를 구현할 수 있습니다.

네트워크 보안

Amazon EKS Auto Mode는 여러 계층의 네트워크 보안을 지원합니다.

- VPC 통합
 - Amazon Virtual Private Cloud(VPC) 내에서 작동
 - 사용자 지정 VPC 구성 및 서브넷 레이아웃 지원
 - 클러스터 구성 요소 간의 프라이빗 네트워킹 활성화
 - 자세한 내용은 [Managing security responsibilities for Amazon Virtual Private Cloud](#)를 참조하세요.
- 네트워크 정책
 - Kubernetes 네트워크 정책에 대한 기본 지원
 - 세분화된 네트워크 트래픽 규칙을 정의하는 기능
 - 자세한 내용은 [the section called “Kubernetes 정책”](#) 단원을 참조하세요.

EC2 관리형 인스턴스 보안

Amazon EKS Auto Mode는 다음과 같은 보안 제어를 사용하여 EC2 관리형 인스턴스를 운영합니다.

EC2 보안

- EC2 관리형 인스턴스는 Amazon EC2의 보안 기능을 유지합니다.

- EC2 관리형 인스턴스에 대한 자세한 내용은 [Security in Amazon EC2](#)를 참조하세요.

인스턴스 수명 주기 관리

EKS Auto Mode에서 운영하는 EC2 관리형 인스턴스의 최대 수명은 21일입니다. Amazon EKS Auto Mode는 이 수명을 초과하는 인스턴스를 자동으로 종료합니다. 이 수명 주기 제한은 구성 드리프트를 방지하고 보안 태세를 유지하는 데 도움이 됩니다.

데이터 보호

- Amazon EC2 인스턴스 스토리지는 암호화되며 인스턴스에 직접 연결된 스토리지입니다. 자세한 내용은 [Amazon EC2의 데이터 보호](#)를 참조하세요.
- EKS Auto Mode는 루트 및 데이터 볼륨을 포함하여 생성 시 EC2 인스턴스에 연결된 볼륨을 관리합니다. EKS Auto Mode는 Kubernetes 영구 스토리지 기능을 사용하여 생성된 EBS 볼륨을 완전히 관리하지 않습니다.

패치 관리

- Amazon EKS Auto Mode는 관리형 인스턴스에 패치를 자동으로 적용합니다.
- 패치에는 다음이 포함됩니다.
 - 운영 체제 업데이트
 - 보안 패치
 - Amazon EKS Auto Mode 구성 요소

Note

고객은 이러한 인스턴스에서 실행되는 워크로드를 보호하고 업데이트할 책임이 있습니다.

액세스 제어

- 직접 인스턴스 액세스는 제한됩니다.
 - SSH 액세스를 사용할 수 없습니다.
 - AWS Systems Manager Session Manager(SSM) 액세스를 사용할 수 없습니다.
- 관리 작업은 Amazon EKS API 및 Kubernetes API를 통해 수행됩니다.

리소스 관리 자동화

Amazon EKS Auto Mode는 Kubernetes 영구 스토리지 기능을 사용하여 생성된 Amazon Elastic Block Store(Amazon EBS) 볼륨을 완전히 관리하지 않습니다. 또한 EKS Auto Mode는 Elastic Load Balancer(ELB)를 관리하지 않습니다. Amazon EKS Auto Mode는 이러한 리소스에 대한 일상적인 작업을 자동화합니다.

스토리지 보안

- AWS에서는 Kubernetes 영구 스토리지 기능으로 프로비저닝된 EBS 볼륨에 대해 암호화를 활성화할 것을 권장합니다. 자세한 내용은 [the section called “StorageClass 생성”](#) 단원을 참조하십시오.
- AWS KMS를 사용하여 저장 시 암호화
- 사용자의 AWS 계정을 구성하여 생성하는 새 EBS 볼륨 및 스냅샷 사본의 암호화를 적용할 수 있습니다. 자세한 내용은 Amazon EBS 사용 설명서의 [Enable Amazon EBS encryption by default](#)을 참조하세요.
- 자세한 내용은 [Security in Amazon EBS](#)을 참조하세요.

로드 밸런서 보안

- Elastic Load Balancer의 자동 구성
- AWS Certificate Manager 통합을 통한 SSL/TLS 인증서 관리
- 로드 밸런서 액세스 제어를 위한 보안 그룹 자동화
- 자세한 내용은 [Security in Elastic Load Balancing](#)을 참조하세요.

보안 모범 사례

다음 섹션에서는 Amazon EKS Auto Mode의 보안 모범 사례를 설명합니다.

- AWS IAM 정책 및 EKS 액세스 항목을 정기적으로 검토합니다.
- 워크로드에 대한 최소 권한 액세스 패턴을 구현합니다.
- AWS CloudTrail 및 Amazon CloudWatch를 통해 클러스터 활동을 모니터링합니다. 자세한 내용은 [the section called “AWS CloudTrail”](#) 및 [the section called “Amazon CloudWatch”](#) 단원을 참조하세요.
- 보안 태세 평가를 위해 AWS Security Hub를 사용합니다.
- 워크로드에 적합한 포드 보안 표준을 구현합니다.

Amazon EKS용 자격 증명 및 액세스 관리

AWS Identity and Access Management(IAM)는 AWS 리소스에 대한 관리자의 액세스를 안전하게 제어하는 데 도움이 되는 AWS 서비스입니다. IAM 관리자는 어떤 사용자가 Amazon EKS 리소스를 사용할 수 있는 인증(로그인) 및 권한(권한 있음)을 받을 수 있는지를 제어합니다. IAM은 추가 비용 없이 사용할 수 있는 AWS 서비스입니다.

대상

AWS ID 및 액세스 관리(IAM)를 사용하는 방법은 Amazon EKS에서 수행하는 작업에 따라 다릅니다.

서비스 사용자 – Amazon EKS 서비스를 사용하여 작업을 수행하는 경우 필요한 자격 증명과 권한을 관리자가 제공합니다. 다른 Amazon EKS 기능을 사용하여 작업을 수행한다면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방법을 이해하면 관리자에게 올바른 권한을 요청하는 데 도움이 됩니다. Amazon EKS의 기능에 액세스할 수 없다면 [the section called “문제 해결”](#) 부분을 참조하세요.

서비스 관리자 – 사내 Amazon EKS 리소스를 책임지고 있다면 Amazon EKS에 대한 완전한 액세스 권한이 있을 것입니다. 서비스 관리자는 서비스 사용자가 액세스해야 하는 Amazon EKS 기능과 리소스를 결정합니다. 그런 다음 IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해합니다. 기사에서 Amazon EKS와 함께 IAM을 사용하는 방법에 대해 자세히 알아보려면 [the section called “Amazon EKS 및 IAM”](#) 부분을 참조하세요.

IAM 관리자 - IAM 관리자라면 Amazon EKS에 대한 액세스 관리 정책 작성 방법을 자세히 알고 싶을 수도 있습니다. IAM에서 사용할 수 있는 Amazon EKS 자격 증명 기반 정책 예제를 보려면 [the section called “ID 기반 정책”](#) 부분을 참조하세요.

ID를 통한 인증

인증은 ID 자격 증명을 사용하여 AWS에 로그인하는 방식입니다. AWS 계정 루트 사용자나 IAM 사용자 또는 IAM 역할을 수임하여 인증(AWS에 로그인)되어야 합니다.

ID 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 ID로 AWS에 로그인할 수 있습니다. AWS IAM ID 센터 사용자, 회사의 통합 인증, Google 또는 Facebook 자격 증명은 페더레이션 ID의 예입니다. 페더레이션형 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 연동을 사용하여 AWS에 액세스하면 간접적으로 역할을 수임합니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. AWS 로그인에 대한 자세한 내용은 AWS 로그인 사용 설명서에서 [AWS 계정에 로그인하는 방법](#)을 참조하세요.

AWS에 프로그래밍 방식으로 액세스하는 경우, AWS에서는 보안 인증 정보를 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트(SDK) 및 명령줄 인터페이스(CLI)를 제공합니다. AWS 도구를 사용하지 않는 경우 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 요청에 직접 서명하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 [AWS API 요청에 서명](#)을 참조하세요.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS는 다중 인증(MFA)을 사용하여 계정의 보안을 강화하는 것을 권장합니다. 자세한 내용은 AWS IAM ID 센터 사용 설명서의 [Multi-factor 인증](#)과 IAM 사용 설명서의 [AWS에서 다단계 인증\(MFA\) 사용](#)을 참조하세요.

AWS 계정 루트 사용자

AWS 계정을 생성할 때는 해당 계정의 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 단일 로그인 ID로 시작합니다. 이 자격 증명은 AWS 계정 루트 사용자라고 하며, 계정을 생성할 때 사용한 이메일 주소와 암호로 로그인하여 액세스합니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명을 보호하고 루트 사용자만 수행할 수 있는 작업을 수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 [루트 사용자 자격 증명에 필요한 작업](#)을 참조하세요.

IAM 사용자 및 그룹

[IAM 사용자](#)는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가지고 있는 AWS 계정 내 ID입니다. 가능하면 암호 및 액세스 키와 같은 장기 자격 증명에 있는 IAM 사용자를 생성하는 대신 임시 자격 증명을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명에 필요한 특정 사용 사례가 있는 경우, 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 [장기 보안 인증이 필요한 사용 사례의 경우, 정기적으로 액세스 키 교체](#)를 참조하세요.

[IAM 그룹](#)은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수입할 수 있습니다. 사용자는 영구적인 장기 보안 인증 정보를 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 정보는 IAM 사용 설명서의 [IAM 사용자를 만들어야 하는 경우\(역할이 아님\)](#)를 참조하세요.

IAM 역할

[IAM 역할](#)은 특정 권한을 가지고 있는 AWS 계정 내 ID입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. [역할 전환](#)하여 AWS Management Console에서 IAM 역할을 임시로 수입할 수 있

습니다. AWS CLI 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 사용하여 역할을 수입할 수 있습니다. 역할 사용 방법에 대한 자세한 정보는 IAM 사용 설명서의 [IAM 역할 사용](#)을 참조하세요.

임시 보안 인증이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 ID가 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 역할에 대한 자세한 내용은 IAM 사용 설명서의 [서드 파티 ID 공급자의 역할 만들기](#)를 참조하세요. IAM Identity Center를 사용하는 경우, 권한 집합을 구성합니다. 인증 후 ID가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 집합을 IAM의 역할과 연관짓습니다. 권한 집합에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 [권한 집합](#)을 참조하세요.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수입하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 교차 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 교차 계정 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 AWS 서비스를 사용하면 역할을 (프록시로 사용하는 대신) 리소스에 정책을 직접 연결할 수 있습니다. 교차 계정 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 [IAM의 교차 계정 리소스 액세스](#)를 참조하세요.
- 교차 서비스 액세스 - 일부 AWS 서비스는 다른 AWS 서비스의 기능을 사용합니다. 예를 들어, 서비스에서 호출하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 호출하는 보안 주체의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
- 전달 액세스 세션(FAS) - IAM 사용자 또는 역할을 사용하여 AWS에서 작업을 수행하는 사람은 보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우, 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 AWS 서비스를 호출하는 주체의 권한을 요청하는 AWS 서비스와 결합하여 다운스트림 서비스에 요청합니다. FAS 요청은 서비스에서 완료를 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 받은 경우에만 이루어집니다. 이 경우, 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 [전달 액세스 세션](#)을 참조하세요.
- 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 맡는 [IAM 역할](#)입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [AWS서비스에 대한 권한을 위임할 역할 생성](#)을 참조하세요.
- 서비스 연결 역할 - 서비스 연결 역할은 AWS 서비스에 연결된 서비스 역할의 한 유형입니다. 서비스는 사용자를 대신하여 태스크를 수행하기 위해 역할을 수입할 수 있습니다. 서비스 연결 역할은 AWS 계정에 나타나고, 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

- Amazon EC2에서 실행 중인 애플리케이션 – IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 AWS CLI 또는 AWS API 요청을 수행하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 해당 역할을 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로필을 생성합니다. 인스턴스 프로필에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 [IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여](#)를 참조하세요.

IAM 역할을 사용할지 또는 IAM 사용자를 사용할지를 알아보려면 [IAM 사용 설명서](#)의 IAM 역할(사용자 대신)을 생성하는 경우,를 참조하세요.

정책을 사용한 액세스 관리

정책을 생성하고 AWS ID 또는 리소스에 연결하여 AWS에서 내 액세스를 제어합니다. 정책은 ID 또는 리소스와 연결될 때 해당 권한을 정의하는 AWS의 객체입니다. AWS는 보안 주체(사용자, 루트 사용자 또는 역할 세션)가 요청을 보낼 때 이러한 정책을 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은 AWS에 JSON 문서로 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용 설명서의 [JSON 정책 개요](#)를 참조하세요.

관리자는 AWSJSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자 및 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, iam:GetRole 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management Console, AWS CLI 또는 AWS API에서 역할 정보를 가져올 수 있습니다.

ID 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 [IAM 정책 생성](#)을 참조하세요.

보안 인증 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 AWS 계정에 속한 다수의 사용자, 그룹

및 역할에게 독립적으로 추가할 수 있는 정책입니다. 관리형 정책에는 AWS관리형 정책과 고객 관리형 정책이 포함되어 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 [관리형 정책과 인라인 정책의 선택](#)을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 [위탁자를 지정](#)해야 합니다. 보안 주체에는 계정, 사용자, 역할, 연합된 사용자 또는 AWS 서비스가 포함될 수 있습니다.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3, AWS WAF 및 Amazon VPC는 ACL을 지원하는 대표적인 서비스입니다. ACL에 관한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 [액세스 제어 목록\(ACL\) 개요](#)를 참조하세요.

기타 정책 타입

AWS는 비교적 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 ID 기반 정책에 따라 IAM 엔티티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 객체의 자격 증명 기반 정책과 그 권한 경계의 교집합입니다. Principal 필드에서 사용자나 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용 설명서의 [IAM 엔티티에 대한 권한 경계](#)를 참조하세요.
- 서비스 제어 정책(SCPs) – SCP는 AWS조직에서 조직 또는 조직 단위(OU)에 대한 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations는 비즈니스가 소유하는 여러 AWS 계정을 그룹화

고 중앙에서 관리할 수 있는 서비스입니다. 조직에서 모든 기능을 활성화할 경우, 서비스 제어 정책 (SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각 AWS 계정 루트 사용자를 비롯하여 멤버 계정의 엔터티에 대한 권한을 제한합니다. SCP에 대한 자세한 내용은 AWS 조직 사용 설명서에서 [서비스 제어 정책](#)을 참조하세요.

- 세션 정책 – 세션 정책은 역할 또는 페더레이션 사용자에게 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 자격 증명 기반 정책과 세션 정책의 교집합입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용 설명서의 [세션 정책](#)을 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련될 때 AWS가 요청을 허용할지를 결정하는 방법을 알아보려면 IAM 사용 설명서의 [정책 평가 로직](#)을 참조하세요.

Amazon EKS가 IAM과 작동하는 방식

IAM을 사용하여 Amazon EKS에 대한 액세스를 관리하기 전에 Amazon EKS에서 사용할 수 있는 IAM 기능을 이해해야 합니다. Amazon EKS 및 기타 AWS 서비스에서 IAM을 사용하는 방법을 자세히 알아보려면 IAM 사용 설명서의 [IAM으로 작업하는 AWS 서비스](#)를 참조하세요.

주제

- [Amazon EKS 자격 증명 기반 정책](#)
- [Amazon EKS 리소스 기반 정책](#)
- [Amazon EKS 태그를 기반으로 권한 부여](#)
- [Amazon EKS IAM 역할](#)

Amazon EKS 자격 증명 기반 정책

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. Amazon EKS는 특정 작업, 리소스 및 조건 키를 지원합니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 [IAM JSON 정책 요소 참조](#)를 참조하세요.

작업

관리자는 AWSJSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 Action 요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 일반적으로 정책 작업의 이름은 연결된 AWSAPI 작업의 이름과 동일합니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

Amazon EKS의 정책 작업은 작업 앞에 eks: 접두사를 사용합니다. 예를 들어, Amazon EKS 클러스터에 대한 기술 정보를 다운로드할 수 있는 권한을 부여하려면 정책에 DescribeCluster 작업을 포함합니다. 정책 문에는 Action 또는 NotAction 요소가 포함되어야 합니다.

명령문 하나에 여러 태스크를 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": ["eks:action1", "eks:action2"]
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, Describe라는 단어로 시작하는 모든 태스크를 지정하려면 다음 태스크를 포함합니다.

```
"Action": "eks:Describe*"
```

Amazon EKS 작업의 목록을 보려면 서비스 인증 참조의 [Amazon Elastic Kubernetes Service에서 정의한 작업](#)을 참조하세요.

리소스

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

Resource JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 문에는 Resource 또는 NotResource 요소가 반드시 추가되어야 합니다. 모범 사례에 따라 [Amazon 리소스 이름\(ARN\)](#)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 명령문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": ""
```

Amazon EKS 클러스터 리소스의 ARN은 다음과 같습니다.

```
arn:aws:eks:region-code:account-id:cluster/cluster-name
```

ARN 형식에 대한 자세한 내용은 [Amazon 리소스 이름\(ARN\) 및 AWS 서비스 네임스페이스](#)를 참조하세요.

예를 들어, 문에서 이름이 *my-cluster*인 클러스터를 지정하려면 다음 ARN을 사용하세요.

```
"Resource": "arn:aws:eks:region-code:111122223333:cluster/my-cluster"
```

특정 계정 및 AWS 리전에 속한 모든 클러스터를 지정하려면 와일드카드(*)를 사용합니다.

```
"Resource": "arn:aws:eks:region-code:111122223333:cluster/*"
```

리소스 생성 작업과 같은 일부 Amazon EKS 작업은 특정 리소스에서 수행할 수 없습니다. 이러한 경우, 와일드카드(*)를 사용해야 합니다.

```
"Resource": ""
```

Amazon EKS 리소스 유형 및 해당 ARN의 목록을 보려면 서비스 인증 참조의 [Amazon Elastic Kubernetes Service에서 정의한 리소스](#)를 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 [Amazon Elastic Kubernetes Service에서 정의한 작업](#)을 참조하세요.

조건 키

Amazon EKS는 자체 조건 키 세트를 정의하며 일부 글로벌 조건 키 사용도 지원합니다. 모든 AWS전역 조건 키를 보려면 IAM 사용 설명서의 [AWS전역 조건 컨텍스트 키](#)를 참조하세요.

OpenID Connect 공급자를 클러스터에 연결할 때 조건 키를 설정할 수 있습니다. 자세한 내용은 [the section called "IAM; 정책 예"](#) 단원을 참조하십시오.

모든 Amazon EC2 작업은 `aws:RequestedRegion` 및 `ec2:Region` 조건 키를 지원합니다. 자세한 내용은 [예제: 특정 AWS 리전으로 액세스 제한](#)을 참조하세요.

Amazon EKS 조건 키 목록을 보려면 서비스 인증 참조의 [Amazon Elastic Kubernetes Service의 조건 키](#)를 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 [Amazon Elastic Kubernetes Service에서 정의한 작업](#)을 참조하세요.

예시

Amazon EKS 자격 증명 기반 정책 예제를 보려면 [the section called “ID 기반 정책”](#) 섹션을 참조하세요.

Amazon EKS 클러스터를 생성할 경우, 클러스터를 생성하는 [IAM 보안 주체](#)에게는 Amazon EKS 제어 영역의 클러스터 역할 기반 액세스 제어(RBAC) 구성에 `system:masters` 권한이 자동으로 부여됩니다. 이 보안 주체는 표시되는 구성에 나타나지 않으므로 클러스터를 원래 생성한 보안 주체를 추적해야 합니다. 추가 IAM 위탁자에 클러스터와 상호 작용할 수 있는 기능을 부여하려면 Kubernetes 내에서 `aws-auth ConfigMap`을 편집하고 `aws-auth ConfigMap`에 지정하는 `group`의 이름으로 `Kubernetes rolebinding` 또는 `clusterrolebinding`을 생성합니다.

`ConfigMap` 작업에 대한 자세한 내용은 [the section called “Kubernetes API 액세스”](#) 세션을 참조하세요.

Amazon EKS 리소스 기반 정책

Amazon EKS는 리소스 기반 정책을 지원하지 않습니다.

Amazon EKS 태그를 기반으로 권한 부여

Amazon EKS 리소스에 태그를 연결하거나 Amazon EKS에 대한 요청에서 태그를 전달할 수 있습니다. 태그에 근거하여 액세스를 제어하려면 `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 [조건 요소](#)에 태그 정보를 제공합니다. Amazon EKS 리소스 태깅에 대한 자세한 내용은 [the section called “리소스에 태깅”](#) 섹션을 참조하세요. 조건 키의 태그를 사용할 수 있는 작업에 대한 자세한 내용은 [Service Authorization Reference](#)에서 [Amazon EKS가 정의한 작업](#)을 참조하세요.

Amazon EKS IAM 역할

[IAM 역할](#)은 특정 권한을 가지고 있는 AWS계정 내 엔터티입니다.

Amazon EKS에서 임시 자격 증명 사용

임시 보안 인증을 사용하여 페더레이션을 통해 로그인하거나, IAM 역할을 맡거나, 교차 계정 역할을 맡을 수 있습니다. [AssumeRole](#) 또는 [GetFederationToken](#) 같은 [AWS STS API](#) 작업을 호출하여 임시 보안 자격 증명을 가져옵니다.

Amazon EKS는 임시 자격 증명 사용을 지원합니다.

서비스 연결 역할

```
link:IAM/latest/UserGuide/id_roles.html#iam-term-service-linked-role[Service-linked roles,type="documentation"] allow {aws} services to access resources in other services to complete an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An administrator can view but can't edit the permissions for service-linked roles.
```

Amazon EKS는 서비스 연결 역할을 지원합니다. Amazon EKS 서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 [the section called “서비스 연결 역할”](#) 섹션을 참조하세요.

서비스 역할

이 기능을 사용하면 서비스가 사용자를 대신하여 [서비스 역할](#)을 수임할 수 있습니다. 이 역할을 사용하면 서비스가 다른 서비스의 리소스에 액세스해 사용자를 대신해 작업을 완료할 수 있습니다. 서비스 역할은 IAM 계정에 나타나고, 해당 계정이 소유합니다. 즉, IAM 관리자가 이 역할에 대한 권한을 변경할 수 있습니다. 그러나 권한을 변경하면 서비스의 기능이 손상될 수 있습니다.

Amazon EKS는 서비스 역할을 지원합니다. 자세한 내용은 [the section called “클러스터 IAM 역할”](#) 및 [the section called “노드 IAM 역할”](#) 섹션을 참조하세요.

Amazon EKS에서 IAM 역할 선택

Amazon EKS에서 클러스터 리소스를 생성할 경우 Amazon EKS가 사용자 대신 다른 여러 AWS 리소스에 액세스하도록 허용하는 역할을 선택해야 합니다. 이전에 서비스 역할을 생성한 경우 Amazon EKS는 선택할 수 있는 역할 목록을 제공합니다. Amazon EKS 관리형 정책이 연결된 역할을 선택하는 것이 중요합니다. 자세한 내용은 [the section called “기존 클러스터 역할 확인”](#) 및 [the section called “기존 노드 역할 확인”](#) 섹션을 참조하세요.

Amazon EKS 자격 증명 기반 정책 예제

기본적으로 IAM 사용자 및 역할은 Amazon EKS 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS CLI 또는 AWS API를 사용해 작업을 수행할 수 없습니다. IAM 관리자는 지정된 리소스에서 특정 API 작업을 수행할 수 있는 권한을 사용자와 역할에게 부여하는 IAM 정책을 생성해야 합니다. 그런 다음 관리자는 해당 권한이 필요한 IAM 사용자 또는 그룹에 이러한 정책을 연결해야 합니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 [JSON 탭에서 정책 생성](#)을 참조하세요.

Amazon EKS 클러스터를 생성할 경우, 클러스터를 생성하는 [IAM 보안 주체](#)에게는 Amazon EKS 제어 영역의 클러스터 역할 기반 액세스 제어(RBAC) 구성에 `system:masters` 권한이 자동으로 부여됩니다. 이 보안 주체는 표시되는 구성에 나타나지 않으므로 클러스터를 원래 생성한 보안 주체를 추적해야 합니다. 추가 IAM 위탁자에 클러스터와 상호 작용할 수 있는 기능을 부여하려면 Kubernetes 내에서 `aws-auth ConfigMap`을 편집하고 `aws-auth ConfigMap`에 지정하는 `group`의 이름으로 `Kubernetes rolebinding` 또는 `clusterrolebinding`을 생성합니다.

`ConfigMap` 작업에 대한 자세한 내용은 [the section called “Kubernetes API 액세스”](#) 세션을 참조하세요.

주제

- [정책 모범 사례](#)
- [Amazon EKS 콘솔 사용](#)
- [IAM 사용자가 자체 권한을 볼 수 있도록 허용](#)
- [AWS Cloud에서 Kubernetes 클러스터 생성](#)
- [Outpost에서 로컬 Kubernetes 클러스터 생성](#)
- [Kubernetes 클러스터 업데이트](#)
- [모든 클러스터 나열 또는 설명](#)

정책 모범 사례

ID 기반 정책에 따라 계정에서 사용자가 Amazon EKS 리소스를 생성, 액세스 또는 삭제할 수 있는 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책으로 시작하고 최소 권한을 향해 나아가기 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. AWS 계정에서 사용할 수 있습니다. 사용 사례에 고유한 AWS 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다. 자세한 정보는 IAM 사용 설명서의 [AWS 관리형 정책](#) 또는 [AWS 직무에 대한 관리형 정책](#)을 참조하세요.
- 최소 권한 적용 - IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 [IAM의 정책 및 권한](#)을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 - 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정

책 조건을 작성할 수 있습니다. AWS CloudFormation과 같이, 특정 AWS 서비스를 통해 사용되는 경우에만 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다. 자세한 정보는 IAM 사용 설명서의 [IAM JSON 정책 요소: 조건](#)을 참조하세요.

- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 정보는 IAM 사용 설명서의 [IAM Access Analyzer 정책 검증](#)을 참조하세요.
- 다중 인증(MFA) 필요 - AWS 계정에 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우, 추가 보안을 위해 MFA를 설정합니다. API 작업을 직접적으로 호출할 때 MFA가 필요하다면 정책에 MFA 조건을 추가합니다. 자세한 정보는 IAM 사용 설명서의 [MFA 보호 API 액세스 구성](#)을 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 [IAM의 보안 모범 사례](#)를 참조하세요.

Amazon EKS 콘솔 사용

Amazon EKS 콘솔에 액세스하려면 [IAM 보안 주체](#)는 최소한의 권한 집합이 있어야 합니다. 이러한 권한은 보안 주체가 AWS 계정에서 Amazon EKS 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용합니다. 최소 필수 권한보다 더 제한적인 보안 인증 정보 기반 정책을 만들면 콘솔이 해당 정책이 연결된 보안 주체에 대해 의도대로 작동하지 않습니다.

이러한 IAM 보안 주체가 Amazon EKS 콘솔을 계속 사용할 수 있도록 하려면 AmazonEKSAAdminPolicy와 같은 고유한 자체 이름으로 정책을 생성합니다. 정책을 보안 주체에게 연결하세요. 자세한 정보는 IAM 사용 설명서의 [IAM 자격 증명 권한 추가 및 제거](#) 섹션을 참조하세요.

Important

다음 정책 예를 사용하면 보안 주체는 콘솔의 구성 탭에 표시되는 정보를 볼 수 있습니다. AWS Management Console에 있는 개요와 리소스 탭에서 정보를 보려면 위탁자에게 Kubernetes 권한도 필요합니다. 자세한 내용은 [the section called “필수 권한”](#) 단원을 참조하십시오.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```

        "Action": [
            "eks:*"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": "iam:PassRole",
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "iam:PassedToService": "eks.amazonaws.com"
            }
        }
    }
]
}

```

AWS CLI 또는 AWS API만 호출하는 보안 주체에는 최소 콘솔 권한을 허용할 필요가 없습니다. 그 대신, 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

IAM 사용자가 자체 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI나 AWS API를 사용하여 프로그래밍 방식으로 이 작업을 완료할 수 있는 권한이 포함됩니다.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        }
    ]
}

```

```

        "Sid": "NavigateInConsole",
        "Effect": "Allow",
        "Action": [
            "iam:GetGroupPolicy",
            "iam:GetPolicyVersion",
            "iam:GetPolicy",
            "iam:ListAttachedGroupPolicies",
            "iam:ListGroupPolicies",
            "iam:ListPolicyVersions",
            "iam:ListPolicies",
            "iam:ListUsers"
        ],
        "Resource": "*"
    }
]
}

```

AWS Cloud에서 Kubernetes 클러스터 생성

이 예제 정책에는 *us-west-2* AWS 리전에서 *my-cluster*라는 Amazon EKS 클러스터를 생성하는 데 필요한 최소 권한이 포함되어 있습니다. AWS 리전을 클러스터를 생성할 AWS 리전으로 바꿀 수 있습니다. AWS Management Console에 정책의 작업이 리소스 수준 권한을 지원하지 않으므로 **All resources**를 선택해야 합니다.라는 경고가 표시되면 무시해도 됩니다. 계정에 이미 *AWSServiceRoleForAmazonEKS* 역할이 있는 경우 정책에서 `iam:CreateServiceLinkedRole` 작업을 제거할 수 있습니다. 계정에 Amazon EKS 클러스터를 생성한 적이 있다면 삭제하지 않는 한 이 역할이 이미 존재합니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "eks:CreateCluster",
      "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::111122223333:role/aws-service-role/eks.amazonaws.com/AWSServiceRoleForAmazonEKS",
      "Condition": {
        "ForAnyValue:StringEquals": {

```

```

        "iam:AWSServiceName": "eks"
      }
    },
    {
      "Effect": "Allow",
      "Action": "iam:PassRole",
      "Resource": "arn:aws:iam::111122223333:role/cluster-role-name"
    }
  ]
}

```

Outpost에서 로컬 Kubernetes 클러스터 생성

이 예제 정책에는 *us-west-2* AWS 리전의 Outposts에서 *my-cluster*라는 Amazon EKS 로컬 클러스터를 만드는 데 필요한 최소 권한이 포함되어 있습니다. AWS 리전을 클러스터를 생성할 AWS 리전으로 바꿀 수 있습니다. AWS Management Console에 정책의 작업이 리소스 수준 권한을 지원하지 않으므로 **All resources**를 선택해야 합니다.라는 경고가 표시되면 무시해도 됩니다. 계정에 이미 `AWSServiceRoleForAmazonEKSLocalOutpost` 역할이 있는 경우 정책에서 `iam:CreateServiceLinkedRole` 작업을 제거할 수 있습니다. 계정의 Outpost에 Amazon EKS 로컬 클러스터를 생성한 적이 있다면 삭제하지 않는 한 이 역할이 이미 존재합니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "eks:CreateCluster",
      "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
    },
    {
      "Action": [
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "iam:GetRole"
      ],
      "Resource": "*",
      "Effect": "Allow"
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",

```

```

        "Resource": "arn:aws:iam::111122223333:role/aws-service-role/outposts.eks-
local.amazonaws.com/AWSServiceRoleForAmazonEKSLocalOutpost"
    },
    {
        "Effect": "Allow",
        "Action": [
            "iam:PassRole",
            "iam:ListAttachedRolePolicies"
        ],
        "Resource": "arn:aws:iam::111122223333:role/cluster-role-name"
    },
    {
        "Action": [
            "iam:CreateInstanceProfile",
            "iam:TagInstanceProfile",
            "iam:AddRoleToInstanceProfile",
            "iam:GetInstanceProfile",
            "iam>DeleteInstanceProfile",
            "iam:RemoveRoleFromInstanceProfile"
        ],
        "Resource": "arn:aws:iam::*:instance-profile/eks-local-*",
        "Effect": "Allow"
    },
]
}

```

Kubernetes 클러스터 업데이트

이 예제 정책에는 us-west-2 AWS 리전에서 *my-cluster*라는 클러스터를 업데이트하는 데 필요한 최소 권한이 포함되어 있습니다.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "eks:UpdateClusterVersion",
            "Resource": "arn:aws:eks:us-west-2:111122223333:cluster/my-cluster"
        }
    ]
}

```

모든 클러스터 나열 또는 설명

이 예제 정책에는 계정의 모든 클러스터를 나열하고 설명하는 데 필요한 최소 권한이 포함되어 있습니다. [IAM 보안 주체](#)는 update-kubeconfig AWS CLI 명령을 사용하기 위해 클러스터를 나열하고 설명할 수 있어야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks:DescribeCluster",
        "eks:ListClusters"
      ],
      "Resource": "*"
    }
  ]
}
```

Amazon EKS에 대해 서비스 연결 역할 사용

Amazon Elastic Kubernetes Service는 AWS ID 및 액세스 관리(IAM) [서비스 연결 역할](#)을 사용합니다. 서비스 연결 역할은 Amazon EKS에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Amazon EKS에서 사전 정의하며 서비스에서 다른 AWS 서비스를 자동으로 직접 호출하기 위해 필요한 모든 권한을 포함합니다.

주제

- [Amazon EKS 클러스터에 대한 역할 사용](#)
- [Amazon EKS 노드 그룹에 대한 역할 사용](#)
- [Amazon EKS Fargate 프로필에 역할 사용](#)
- [역할을 사용하여 Amazon EKS에 Kubernetes 클러스터 연결](#)
- [Outpost에서 Amazon EKS 로컬 클러스터에 대한 역할 사용](#)
- [Amazon EKS 대시보드에 역할 사용](#)

Amazon EKS 클러스터에 대한 역할 사용

Amazon Elastic Kubernetes Service는 AWS ID 및 액세스 관리(IAM) [서비스 연결 역할](#)을 사용합니다. 서비스 연결 역할은 Amazon EKS에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Amazon EKS에서 사전 정의하며 서비스에서 다른 AWS 서비스를 자동으로 호출하기 위해 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 Amazon EKS를 더 쉽게 설정할 수 있습니다. Amazon EKS에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한, Amazon EKS만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔티티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제한 후에만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 리소스에 대한 액세스 권한을 실수로 삭제할 수 없기 때문에 Amazon EKS 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 [IAM으로 작업하는 AWS 서비스](#)를 참조하고 서비스 연결 역할 열에 예가 있는 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 링크가 있는 예를 선택합니다.

Amazon EKS에 대한 서비스 연결 역할 권한

Amazon EKS는 `AWSServiceRoleForAmazonEKS`이라는 서비스 연결 역할을 사용합니다. 이 역할을 통해 Amazon EKS는 계정의 클러스터를 관리할 수 있습니다. 연결된 정책은 이 역할이 네트워크 인터페이스, 보안 그룹, 로그 및 VPC와 같은 리소스를 관리하도록 허용합니다.

Note

`AWSServiceRoleForAmazonEKS` 서비스 연결 역할은 클러스터 생성에 필요한 역할과 다릅니다. 자세한 내용은 [the section called “클러스터 IAM 역할”](#) 섹션을 참조하세요.

`AWSServiceRoleForAmazonEKS` 서비스 연결 역할은 역할을 위임하기 위해 다음 서비스를 신뢰합니다.

- `eks.amazonaws.com`

역할 권한 정책은 Amazon EKS가 지정된 리소스에서 다음 작업을 완료하도록 허용합니다.

- [AmazonEKSServiceRolePolicy](#)

IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 권한](#)을 참조하세요.

Amazon EKS에 대한 서비스 연결 역할 생성

서비스 링크 역할은 수동으로 생성할 필요가 없습니다. AWS Management Console, AWS CLI 또는 AWS API에서 클러스터를 생성하면 Amazon EKS에서 서비스 연결 역할이 자동으로 생성됩니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 클러스터를 생성할 때 Amazon EKS에서는 서비스 연결 역할을 다시 생성합니다.

Amazon EKS에 대한 서비스 연결 역할 편집

Amazon EKS에서는 AWSServiceRoleForAmazonEKS 서비스 연결 역할을 편집하도록 허용하지 않습니다. 서비스 링크 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 편집](#)을 참조하세요.

Amazon EKS에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제하는 것이 좋습니다. 따라서 적극적으로 모니터링하거나 유지하지 않는 미사용 엔터티가 없도록 합니다. 단, 서비스 연결 역할을 정리해야 수동으로 삭제할 수 있습니다.

서비스 연결 역할을 정리

IAM을 사용하여 서비스 연결 역할을 삭제하기 전에 먼저 역할에서 사용되는 리소스를 삭제해야 합니다.

Note

리소스를 삭제하려고 할 때 Amazon EKS 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 좌측 탐색 창에서 클러스터를 선택합니다.
3. 클러스터에 노드 그룹 또는 Fargate 프로필이 있는 경우 클러스터를 삭제하기 전에 이를 삭제해야 합니다. 자세한 내용을 알아보려면 [the section called “삭제”](#) 및 [the section called “프로필 삭제”](#) 섹션을 참조하세요.

4. 클러스터 페이지에서 삭제하려는 클러스터를 선택하고 삭제를 선택합니다.
5. 삭제 확인 창에 클러스터 이름을 입력한 다음 삭제를 선택합니다.
6. 계정의 다른 클러스터에 대해 이 절차를 반복합니다. 모든 삭제 작업이 완료될 때까지 기다립니다.

수동으로 서비스 연결 역할 삭제

IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 `AWSServiceRoleForAmazonEKS` 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 [서비스에 연결 역할 삭제](#)를 참조하세요.

Amazon EKS 서비스 연결 역할을 지원하는 리전

Amazon EKS는 서비스가 제공되는 모든 리전에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용을 알아보려면 [Amazon EKS 엔드포인트 및 할당량](#)을 참조하세요.

Amazon EKS 노드 그룹에 대한 역할 사용

Amazon EKS는 AWS ID 및 액세스 관리(IAM) [service-linked 역할](#)을 사용합니다. 서비스 연결 역할은 Amazon EKS에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Amazon EKS에서 사전 정의하며 서비스에서 다른 AWS 서비스를 자동으로 호출하기 위해 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 Amazon EKS를 더 쉽게 설정할 수 있습니다. Amazon EKS에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한, Amazon EKS만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔티티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제한 후에만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 리소스에 대한 액세스 권한을 실수로 삭제할 수 없기 때문에 Amazon EKS 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 [IAM으로 작업하는 AWS 서비스](#)를 참조하고 서비스 연결 역할 열에 예가 있는 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 링크가 있는 예를 선택합니다.

Amazon EKS에 대한 서비스 연결 역할 권한

Amazon EKS는 `AWSServiceRoleForAmazonEKSNodegroup`이라는 서비스 연결 역할을 사용합니다. 이 역할을 통해 Amazon EKS는 계정의 노드 그룹을 관리할 수 있습니다. 연결된 `AWSServiceRoleForAmazonEKSNodegroup` 정책은 이 역할이 Auto Scaling 그룹, 보안 그룹, 시작 템플릿, IAM 인스턴스 프로파일 등의 리소스를 관리하도록 허용합니다. 자세한 내용은 [the section called “AWS 관리형 정책: AWSServiceRoleForAmazonEKSNodegroup”](#) 단원을 참조하십시오.

AWSServiceRoleForAmazonEKSNodegroup 서비스 연결 역할은 역할을 위임하기 위해 다음 서비스를 신뢰합니다.

- `eks-nodegroup.amazonaws.com`

역할 권한 정책은 Amazon EKS가 지정된 리소스에서 다음 작업을 완료하도록 허용합니다.

- [AWSServiceRoleForAmazonEKSNodegroup](#)

IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 권한](#)을 참조하세요.

Amazon EKS에 대한 서비스 연결 역할 생성

서비스 링크 역할은 수동으로 생성할 필요가 없습니다. AWS Management Console, AWS CLI 또는 AWS API에서 관리형 노드 그룹을 생성하면 Amazon EKS에서 서비스 연결 역할이 자동으로 생성됩니다.

Important

이러한 서비스 연결 역할은 해당 역할이 지원하는 기능을 사용하는 다른 서비스에서 작업을 완료했을 경우 계정에 나타날 수 있습니다. Amazon EKS 서비스가 서비스 연결 역할을 지원하기 시작한 2017년 1월 1일 이전에 이 서비스를 사용했다면 Amazon EKS가 사용자 계정에 AWSServiceRoleForAmazonEKSNodegroup 역할을 이미 생성했습니다. 자세한 내용은 [내 IAM 계정에 표시되는 새 역할](#)을 참조하세요.

Amazon EKS(AWS API)에서 서비스 연결 역할 생성

서비스 링크 역할은 수동으로 생성할 필요가 없습니다. AWS Management Console, AWS CLI 또는 AWS API에서 관리형 노드 그룹을 생성하면 Amazon EKS에서 서비스 연결 역할이 자동으로 생성됩니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 다른 관리형 노드 그룹을 생성할 때 Amazon EKS가 서비스 연결 역할을 다시 생성합니다.

Amazon EKS에 대한 서비스 연결 역할 편집

Amazon EKS에서는 `AWSServiceRoleForAmazonEKSNodegroup` 서비스 연결 역할을 편집하도록 허용하지 않습니다. 서비스 링크 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 편집](#)을 참조하세요.

Amazon EKS에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제하는 것이 좋습니다. 따라서 적극적으로 모니터링하거나 유지하지 않는 미사용 엔터티가 없도록 합니다. 단, 서비스 연결 역할을 정리해야 수동으로 삭제할 수 있습니다.

서비스 연결 역할을 정리

IAM을 사용하여 서비스 연결 역할을 삭제하기 전에 먼저 역할에서 사용되는 리소스를 삭제해야 합니다.

Note

리소스를 삭제하려고 할 때 Amazon EKS 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 좌측 탐색 창에서 클러스터를 선택합니다.
3. 컴퓨팅 탭을 선택합니다.
4. 노드 그룹 섹션에서 삭제할 노드 그룹을 선택합니다.
5. 삭제 확인 창에 노드 그룹 이름을 입력한 다음 삭제를 선택합니다.
6. 클러스터의 다른 노드 그룹에 대해 이 절차를 반복합니다. 모든 삭제 작업이 완료될 때까지 기다립니다.

수동으로 서비스 연결 역할 삭제

IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 `AWSServiceRoleForAmazonEKSNodegroup` 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 [서비스에 연결 역할 삭제](#)를 참조하세요.

Amazon EKS 서비스 연결 역할을 지원하는 리전

Amazon EKS는 서비스가 제공되는 모든 리전에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용을 알아보려면 [Amazon EKS 엔드포인트 및 할당량](#)을 참조하세요.

Amazon EKS Fargate 프로필에 역할 사용

Amazon EKS는 AWS ID 및 액세스 관리(IAM) [service-linked 역할](#)을 사용합니다. 서비스 연결 역할은 Amazon EKS에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Amazon EKS에서 사전 정의하며 서비스에서 다른 AWS 서비스를 자동으로 호출하기 위해 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 Amazon EKS를 더 쉽게 설정할 수 있습니다. Amazon EKS에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한, Amazon EKS만 해당 역할을 수입할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔티티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제한 후에만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 리소스에 대한 액세스 권한을 실수로 삭제할 수 없기 때문에 Amazon EKS 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 [IAM으로 작업하는 AWS 서비스](#)를 참조하고 서비스 연결 역할 열에 예가 있는 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 링크가 있는 예를 선택합니다.

Amazon EKS에 대한 서비스 연결 역할 권한

Amazon EKS는 `AWSServiceRoleForAmazonEKSFargate`이라는 서비스 연결 역할을 사용합니다. 이 역할을 통해 Amazon EKS Fargate는 Fargate 포드에 필요한 VPC 네트워킹을 구성할 수 있습니다. 연결된 정책을 통해 역할은 탄력적 네트워크 인터페이스를 생성 및 삭제하고 탄력적 네트워크 인터페이스 및 리소스를 설명할 수 있습니다.

`AWSServiceRoleForAmazonEKSFargate` 서비스 연결 역할은 역할을 수입하기 위해 다음 서비스를 신뢰합니다.

- `eks-fargate.amazonaws.com`

역할 권한 정책은 Amazon EKS가 지정된 리소스에서 다음 작업을 완료하도록 허용합니다.

- [AmazonEKSFargateServiceRolePolicy](#)

IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 권한](#)을 참조하세요.

Amazon EKS에 대한 서비스 연결 역할 생성

서비스 링크 역할은 수동으로 생성할 필요가 없습니다. AWS Management Console, AWS CLI 또는 AWS API에서 Fargate 프로필을 생성하면 Amazon EKS에서 서비스 연결 역할이 자동으로 생성됩니다.

Important

이러한 서비스 연결 역할은 해당 역할이 지원하는 기능을 사용하는 다른 서비스에서 작업을 완료했을 경우 계정에 나타날 수 있습니다. Amazon EKS 서비스가 서비스 연결 역할을 지원하기 시작한 2019년 12월 13일 이전에 이 서비스를 사용했다면 Amazon EKS가 사용자 계정에 AWSServiceRoleForAmazonEKSFargate 역할을 이미 생성했습니다. 자세한 내용을 알아보려면 [내 IAM 계정에 표시되는 새 역할](#)을 참조하세요.

Amazon EKS(AWS API)에서 서비스 연결 역할 생성

서비스 링크 역할은 수동으로 생성할 필요가 없습니다. AWS Management Console, AWS CLI 또는 AWS API에서 Fargate 프로필을 생성하면 Amazon EKS에서 서비스 연결 역할이 자동으로 생성됩니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 다른 관리형 노드 그룹을 생성할 때 Amazon EKS가 서비스 연결 역할을 다시 생성합니다.

Amazon EKS에 대한 서비스 연결 역할 편집

Amazon EKS에서는 AWSServiceRoleForAmazonEKSFargate 서비스 연결 역할을 편집하도록 허용하지 않습니다. 서비스 링크 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 편집](#)을 참조하세요.

Amazon EKS에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제하는 것이 좋습니다. 따라서 적극적으로 모니터링하거나 유지하지 않는 미사용 엔터티가 없도록 합니다. 단, 서비스 연결 역할을 정리해야 수동으로 삭제할 수 있습니다.

서비스 연결 역할을 정리

IAM을 사용하여 서비스 연결 역할을 삭제하기 전에 먼저 역할에서 사용되는 리소스를 삭제해야 합니다.

Note

리소스를 삭제하려고 할 때 Amazon EKS 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 좌측 탐색 창에서 클러스터를 선택합니다.
3. 클러스터 페이지에서 클러스터를 선택합니다.
4. 컴퓨팅 탭을 선택합니다.
5. Fargate 프로필 섹션에 Fargate 프로필이 있는 경우 각각 개별적으로 선택한 다음 삭제를 선택합니다.
6. 삭제 확인 창에 프로필 이름을 입력한 다음 삭제를 선택합니다.
7. 클러스터의 다른 Fargate 프로필과 계정의 다른 클러스터에 대해 이 절차를 반복합니다.

수동으로 서비스 연결 역할 삭제

IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 `AWSServiceRoleForAmazonEKSFargate` 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 [서비스에 연결 역할 삭제](#)를 참조하세요.

Amazon EKS 서비스 연결 역할을 지원하는 리전

Amazon EKS는 서비스가 제공되는 모든 리전에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용을 알아보려면 [Amazon EKS 엔드포인트 및 할당량](#)을 참조하세요.

역할을 사용하여 Amazon EKS에 Kubernetes 클러스터 연결

Amazon EKS는 AWS ID 및 액세스 관리(IAM) [service-linked 역할](#)을 사용합니다. 서비스 연결 역할은 Amazon EKS에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Amazon EKS에서 사전 정의하며 서비스에서 다른 AWS 서비스를 자동으로 호출하기 위해 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 Amazon EKS를 더 쉽게 설정할 수 있습니다. Amazon EKS에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한,

Amazon EKS만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔티티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제한 후에만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 리소스에 대한 액세스 권한을 실수로 삭제할 수 없기 때문에 Amazon EKS 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 [IAM으로 작업하는 AWS 서비스](#)를 참조하고 서비스 연결 역할 열에 예가 있는 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 링크가 있는 예를 선택합니다.

Amazon EKS에 대한 서비스 연결 역할 권한

Amazon EKS는 `AWSServiceRoleForAmazonEKSCluster`이라는 서비스 연결 역할을 사용합니다. 이 역할을 통해 Amazon EKS는 Kubernetes 클러스터를 연결할 수 있습니다. 연결된 정책을 사용하면 역할이 등록된 Kubernetes 클러스터에 연결하는 데 필요한 리소스를 관리할 수 있습니다.

`AWSServiceRoleForAmazonEKSCluster` 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- `eks-connector.amazonaws.com`

역할 권한 정책은 Amazon EKS가 지정된 리소스에서 다음 작업을 완료하도록 허용합니다.

- [AmazonEKSClusterServiceRolePolicy](#)

IAM 엔티티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 권한](#)을 참조하세요.

Amazon EKS에 대한 서비스 연결 역할 생성

클러스터 연결을 위해 서비스 연결 역할을 수동으로 생성할 필요가 없습니다. AWS Management Console, AWS CLI, `eksctl` 또는 AWS API에서 클러스터를 연결하면 Amazon EKS에서 서비스 연결 역할이 자동으로 생성됩니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 클러스터를 연결할 때 Amazon EKS에서는 서비스 연결 역할을 다시 생성합니다.

Amazon EKS에 대한 서비스 연결 역할 편집

Amazon EKS에서는 `AWSServiceRoleForAmazonEKSCollector` 서비스 연결 역할을 편집하도록 허용하지 않습니다. 서비스 링크 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 편집](#)을 참조하세요.

Amazon EKS에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제하는 것이 좋습니다. 따라서 적극적으로 모니터링하거나 유지하지 않는 미사용 엔터티가 없도록 합니다. 단, 서비스 연결 역할을 정리해야 수동으로 삭제할 수 있습니다.

서비스 연결 역할을 정리

IAM을 사용하여 서비스 연결 역할을 삭제하기 전에 먼저 역할에서 사용되는 리소스를 삭제해야 합니다.

Note

리소스를 삭제하려고 할 때 Amazon EKS 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 좌측 탐색 창에서 클러스터를 선택합니다.
3. 클러스터 페이지에서 클러스터를 선택합니다.
4. 등록 취소 탭을 선택한 다음 확인 탭을 선택합니다.

수동으로 서비스 연결 역할 삭제

IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 `AWSServiceRoleForAmazonEKSCollector` 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 [서비스에 연결 역할 삭제](#)를 참조하세요.

Outpost에서 Amazon EKS 로컬 클러스터에 대한 역할 사용

Amazon Elastic Kubernetes Service는 AWS ID 및 액세스 관리(IAM) [서비스 연결 역할](#)을 사용합니다. 서비스 연결 역할은 Amazon EKS에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은

Amazon EKS에서 사전 정의하며 서비스에서 다른 AWS 서비스를 자동으로 호출하기 위해 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 Amazon EKS를 더 쉽게 설정할 수 있습니다. Amazon EKS에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한, Amazon EKS만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔티티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제한 후에만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 리소스에 대한 액세스 권한을 실수로 삭제할 수 없기 때문에 Amazon EKS 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 [IAM으로 작업하는 AWS 서비스](#)를 참조하고 서비스 연결 역할 열에 예가 있는 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 링크가 있는 예를 선택합니다.

Amazon EKS에 대한 서비스 연결 역할 권한

Amazon EKS는 `AWSServiceRoleForAmazonEKSLocalOutpost`이라는 서비스 연결 역할을 사용합니다. 이 역할을 통해 Amazon EKS는 계정의 로컬 클러스터를 관리할 수 있습니다. 연결된 정책은 이 역할이 네트워크 인터페이스, 보안 그룹, 로그 및 Amazon EC2 인스턴스와 같은 리소스를 관리하도록 허용합니다.

Note

`AWSServiceRoleForAmazonEKSLocalOutpost` 서비스 연결 역할은 클러스터 생성에 필요한 역할과 다릅니다. 자세한 내용은 [the section called “클러스터 IAM 역할”](#) 섹션을 참조하세요.

`AWSServiceRoleForAmazonEKSLocalOutpost` 서비스 연결 역할은 역할을 위임하기 위해 다음 서비스를 신뢰합니다.

- `outposts.eks-local.amazonaws.com`

역할 권한 정책은 Amazon EKS가 지정된 리소스에서 다음 작업을 완료하도록 허용합니다.

- [AmazonEKSServiceRolePolicy](#)

IAM 엔티티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 권한](#)을 참조하세요.

Amazon EKS에 대한 서비스 연결 역할 생성

서비스 링크 역할은 수동으로 생성할 필요가 없습니다. AWS Management Console, AWS CLI 또는 AWS API에서 클러스터를 생성하면 Amazon EKS에서 서비스 연결 역할이 자동으로 생성됩니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 클러스터를 생성할 때 Amazon EKS에서는 서비스 연결 역할을 다시 생성합니다.

Amazon EKS에 대한 서비스 연결 역할 편집

Amazon EKS에서는 `AWSServiceRoleForAmazonEKSLocalOutpost` 서비스 연결 역할을 편집하도록 허용하지 않습니다. 서비스 연결 역할을 생성한 후에는 다양한 엔터티가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 편집](#)을 참조하세요.

Amazon EKS에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제하는 것이 좋습니다. 따라서 적극적으로 모니터링하거나 유지하지 않는 미사용 엔터티가 없도록 합니다. 단, 서비스 연결 역할을 정리해야 수동으로 삭제할 수 있습니다.

서비스 연결 역할을 정리

IAM을 사용하여 서비스 연결 역할을 삭제하기 전에 먼저 역할에서 사용되는 리소스를 삭제해야 합니다.

Note

리소스를 삭제하려고 할 때 Amazon EKS 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 왼쪽 탐색 창에서 Amazon EKS 클러스터를 선택합니다.
3. 클러스터에 노드 그룹 또는 Fargate 프로필이 있는 경우 클러스터를 삭제하기 전에 이를 삭제해야 합니다. 자세한 내용을 알아보려면 [the section called “삭제”](#) 및 [the section called “프로필 삭제”](#) 섹션을 참조하세요.
4. 클러스터 페이지에서 삭제하려는 클러스터를 선택하고 삭제를 선택합니다.
5. 삭제 확인 창에 클러스터 이름을 입력한 다음 삭제를 선택합니다.

6. 계정의 다른 클러스터에 대해 이 절차를 반복합니다. 모든 삭제 작업이 완료될 때까지 기다립니다.

수동으로 서비스 연결 역할 삭제

IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 `AWSServiceRoleForAmazonEKSLocalOutpost` 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 [서비스에 연결 역할 삭제](#)를 참조하세요.

Amazon EKS 서비스 연결 역할을 지원하는 리전

Amazon EKS는 서비스가 제공되는 모든 리전에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용을 알아보려면 [Amazon EKS 엔드포인트 및 할당량](#)을 참조하세요.

Amazon EKS 대시보드에 역할 사용

Amazon EKS 대시보드는 이 서비스 연결 역할을 사용하여 여러 리전 및 계정의 클러스터 리소스에 대한 정보를 집계합니다. 대시보드는 AWS Organizations와 통합되어 조직의 여러 계정에 대한 정보를 안전하게 읽습니다.

Amazon EKS 대시보드에 대한 자세한 내용은 [the section called “Amazon EKS 대시보드”](#) 섹션을 참조하세요.

배경

Amazon EKS는 AWS ID 및 액세스 관리(IAM) [service-linked 역할](#)을 사용합니다. 서비스 연결 역할은 Amazon EKS에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Amazon EKS에서 사전 정의하며 서비스에서 다른 AWS 서비스를 자동으로 직접 호출하기 위해 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 Amazon EKS를 더 쉽게 설정할 수 있습니다. Amazon EKS에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한, Amazon EKS만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔티티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제한 후에만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 리소스에 대한 액세스 권한을 실수로 삭제할 수 없기 때문에 Amazon EKS 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 [IAM으로 작업하는 AWS 서비스](#)를 참조하고 서비스 연결 역할 열에 예가 있는 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 링크가 있는 예를 선택합니다.

Amazon EKS에 대한 서비스 연결 역할 권한

Amazon EKS는 `AWSServiceRoleForAmazonEKSDashboard`이라는 서비스 연결 역할을 사용합니다. 이 역할을 통해 Amazon EKS는 클러스터 리소스에 대한 집계된 정보를 포함하여 EKS 대시보드를 생성하고 표시할 수 있습니다. 연결된 `AmazonEKSDashboardServiceRolePolicy` 정책은 이 역할이 Auto Scaling 그룹, 보안 그룹, 시작 템플릿, IAM 인스턴스 프로파일 등의 리소스를 관리하도록 허용합니다. 자세한 내용은 [the section called “AWS 관리형 정책: AmazonEKSDashboardServiceRolePolicy”](#) 섹션을 참조하세요.

`AWSServiceRoleForAmazonEKSDashboard` 서비스 연결 역할은 역할을 위임하기 위해 다음 서비스를 신뢰합니다.

- `dashboard.eks.amazonaws.com`

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의 [AmazonEKSDashboardServiceRolePolicy](#)를 참조하세요.

IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 권한](#)을 참조하세요.

Amazon EKS에 대한 서비스 연결 역할 생성

서비스 링크 역할은 수동으로 생성할 필요가 없습니다. AWS 콘솔에서 대시보드를 활성화하면 Amazon EKS가 서비스 연결 역할을 생성합니다.

EKS 대시보드 활성화에 대한 자세한 내용은 [the section called “Organizations 구성”](#) 섹션을 참조하세요.

Important

이러한 서비스 연결 역할은 해당 역할이 지원하는 기능을 사용하는 다른 서비스에서 작업을 완료했을 경우 계정에 나타날 수 있습니다.

Amazon EKS에 대한 서비스 연결 역할 편집

Amazon EKS에서는 `AWSServiceRoleForAmazonEKSDashboard` 서비스 연결 역할을 편집하도록 허용하지 않습니다. 서비스 링크 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 편집](#)을 참조하세요.

Amazon EKS에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제하는 것이 좋습니다. 따라서 적극적으로 모니터링하거나 유지하지 않는 미사용 엔터티가 없도록 합니다. 단, 서비스 연결 역할을 정리해야 수동으로 삭제할 수 있습니다.

서비스 연결 역할을 정리

IAM을 사용하여 서비스 연결 역할을 삭제하기 전에 먼저 역할에서 사용되는 리소스를 삭제해야 합니다.

Note

리소스를 삭제하려고 할 때 Amazon EKS 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

EKS 대시보드 비활성화에 대한 자세한 내용은 [the section called “Organizations 구성”](#) 섹션을 참조하세요.

수동으로 서비스 연결 역할 삭제

IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 `AWSServiceRoleForAmazonEKSDashboard` 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 [서비스에 연결 역할 삭제](#)를 참조하세요.

Amazon EKS 서비스 연결 역할을 지원하는 리전

Amazon EKS는 서비스가 제공되는 모든 리전에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용을 알아보려면 [Amazon EKS 엔드포인트 및 할당량](#)을 참조하세요.

Amazon EKS 포드 실행 IAM 역할

AWS Fargate 인프라에서 포드를 실행하려면 Amazon EKS 포드 실행 역할이 필요합니다.

클러스터가 AWS Fargate 인프라에서 포드를 생성하는 경우 Fargate 인프라에서 실행되는 구성 요소는 사용자를 대신하여 AWS API를 직접적으로 호출해야 합니다. 이를 통해 Amazon ECR에서 컨테이너 이미지를 가져오거나 로그를 다른 AWS 서버로 라우팅하는 등의 작업을 수행할 수 있습니다. Amazon EKS 포드 실행 역할은 이 작업을 수행할 수 있는 IAM 권한을 제공합니다.

Fargate 프로필을 생성할 때 프로필을 사용하여 Fargate 인프라에서 실행되는 Amazon EKS 구성 요소의 포드 실행 역할을 지정해야 합니다. 이 역할은 권한 부여를 위해 클러스터의 Kubernetes [역할 기반](#)

[액세스 제어](#)(RBAC)에 추가됩니다. 이렇게 하면 Fargate 인프라에서 실행 중인 kubelet이 Amazon EKS 클러스터에 등록되어 클러스터에 노드로 표시될 수 있습니다.

 Note

Fargate 프로파일에는 Amazon EC2 노드 그룹과 다른 IAM 역할이 있어야 합니다.

 Important

Fargate 포드에서 실행 중인 컨테이너는 포드 실행 역할과 연결된 IAM 권한을 수임할 수 없습니다. Fargate 포드의 컨테이너에 다른 AWS 서비스에 대한 액세스 권한을 부여하려면 [서비스 계정에 IAM 역할](#)을 사용해야 합니다.

Fargate 프로파일을 생성하기 전에 [AmazonEKSFargatePodExecutionRolePolicy](#)를 사용하여 IAM 역할을 생성해야 합니다.

올바로 구성된 기존 포드 실행 역할 확인

다음 절차를 사용하여 계정에 이미 올바로 구성된 Amazon EKS 포드 실행 역할이 있는지 확인할 수 있습니다. 대리인 보안 문제의 혼동을 방지하려면 역할이 SourceArn에 따라 액세스를 제한하는 것이 중요합니다. 필요에 따라 실행 역할을 수정하여 다른 클러스터의 Fargate 프로파일에 대한 지원을 포함할 수 있습니다.

1. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 역할을 선택합니다.
3. 역할 페이지에서 AmazonEKSFargatePodExecutionRole에 대한 역할 목록을 검색합니다. 역할이 존재하지 않을 경우 [the section called “Amazon EKS 포드 실행 역할 생성”](#)을 참조하여 역할을 생성합니다. 역할이 존재하지 않을 경우 역할을 선택합니다.
4. AmazonEKSFargatePodExecutionRole 페이지에서 다음을 수행합니다.
 - a. 권한을 선택합니다.
 - b. AmazonEKSFargatePodExecutionRolePolicy Amazon 관리형 정책이 역할에 연결되어 있는지 확인합니다.
 - c. 신뢰 관계를 선택합니다.
 - d. 신뢰 정책 편집을 선택합니다.

5. 신뢰 정책 편집 페이지에서 신뢰 관계에 다음 정책이 포함되어 있고 클러스터의 Fargate 프로필에 대한 줄이 있는지 확인합니다. 그렇다면 취소를 선택합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:eks:region-code:111122223333:fargateprofile/my-cluster/*"
        }
      },
      "Principal": {
        "Service": "eks-fargate-pods.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

정책이 일치하지만 클러스터의 Fargate 프로필을 지정하는 줄이 없는 경우 다음 줄을 ArnLike 객체의 상단에 추가할 수 있습니다. *region-code*를 클러스터가 있는 AWS 리전으로, *111122223333*을 계정 ID로, *my-cluster*를 클러스터 이름으로 바꿉니다.

```
"aws:SourceArn": "arn:aws:eks:region-code:111122223333:fargateprofile/my-cluster/*",
```

정책이 일치하지 않는 경우 이전 정책 전체를 양식에 복사하고 업데이트 정책을 선택합니다. *region-code*를 클러스터가 있는 AWS 리전으로 바꿉니다. 계정의 모든 AWS 리전에서 동일한 역할을 사용하려면 *region-code*를 *로 바꾸세요. *my-cluster*를 실제 클러스터의 이름으로 바꾸고 *111122223333*을 계정 ID로 바꿉니다. 계정의 모든 클러스터에 대해 동일한 역할을 사용하려는 경우 *my-cluster*를 *로 바꾸세요.

Amazon EKS 포드 실행 역할 생성

클러스터에 대한 Amazon EKS 포드 실행 역할이 아직 없는 경우 AWS Management Console 또는 AWS CLI를 사용하여 생성할 수 있습니다.

AWS Management Console

- a. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
- b. 왼쪽 탐색 창에서 역할을 선택합니다.
- c. 역할 페이지에서 역할 생성을 선택합니다.
- d. 신뢰할 수 있는 엔터티 선택 페이지에서 다음을 수행합니다.
 - i. 신뢰할 수 있는 엔터티 유형 섹션에서 AWS 서비스를 선택합니다.
 - ii. 다른 AWS 서비스의 사용 사례 드롭다운 목록에서 EKS를 선택합니다.
 - iii. EKS - Fargate 포드를 선택합니다.
 - iv. Next(다음)를 선택합니다.
- e. 권한 추가 페이지에서 다음을 선택합니다.
- f. 이름, 검토 및 생성 페이지에서 다음을 수행합니다.
 - i. 역할 이름에 역할의 고유한 이름(예: AmazonEKSFargatePodExecutionRole)을 입력합니다.
 - ii. 태그 추가(선택사항)에서 태그를 키-값 페어로 연결하여 메타데이터를 역할에 추가합니다. IAM에서 태그 사용에 대한 자세한 내용을 알아보려면 IAM 사용 설명서의 [IAM 리소스에 태그](#)를 참조하세요.
 - iii. 역할 생성을 선택합니다.
- g. 역할 페이지에서 AmazonEKSFargatePodExecutionRole에 대한 역할 목록을 검색합니다. 역할을 선택합니다.
- h. AmazonEKSFargatePodExecutionRole 페이지에서 다음을 수행합니다.
 - i. 신뢰 관계를 선택합니다.
 - ii. 신뢰 정책 편집을 선택합니다.
- i. 신뢰 정책 편집 페이지에서 다음 작업을 수행합니다.
 - i. 다음 내용을 복사하여 신뢰 정책 편집 양식에 붙여 넣습니다. *region-code*를 클러스터가 있는 AWS 리전으로 바꿉니다. 계정의 모든 AWS 리전에서 동일한 역할을 사용하려면 *region-code*를 *로 바꾸세요. *my_cluster*를 실제 클러스터의 이름으로 바꾸고 *111122223333*을 계정 ID로 바꿉니다. 계정의 모든 클러스터에 대해 동일한 역할을 사용하려는 경우 *my-cluster*를 *로 바꾸세요.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```

    "Effect": "Allow",
    "Condition": {
      "ArnLike": {
        "aws:SourceArn": "arn:aws:eks:region-
code:111122223333:fargateprofile/my-cluster/*"
      }
    },
    "Principal": {
      "Service": "eks-fargate-pods.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
  }
]
}

```

ii. 정책 업데이트를 선택합니다.

AWS CLI

- a. 다음 내용을 복사하여 `pod-execution-role-trust-policy.json`라는 파일에 붙여넣습니다. **region-code**를 클러스터가 있는 AWS 리전으로 바꿉니다. 계정의 모든 AWS 리전에서 동일한 역할을 사용하려면 **region-code**를 *로 바꾸세요. **my_cluster**를 실제 클러스터의 이름으로 바꾸고 **111122223333**을 계정 ID로 바꿉니다. 계정의 모든 클러스터에 대해 동일한 역할을 사용하려는 경우 **my-cluster**를 *로 바꾸세요.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:eks:region-code:111122223333:fargateprofile/
my-cluster/*"
        }
      },
      "Principal": {
        "Service": "eks-fargate-pods.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}

```

b. 포드 실행 IAM 역할을 생성합니다.

```
aws iam create-role \
  --role-name AmazonEKSFargatePodExecutionRole \
  --assume-role-policy-document file://"pod-execution-role-trust-policy.json"
```

c. 필요한 Amazon EKS 관리형 IAM 정책을 역할에 연결합니다.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSFargatePodExecutionRolePolicy \
  --role-name AmazonEKSFargatePodExecutionRole
```

Amazon EKS Connector IAM 역할

Kubernetes 클러스터를 연결하여 AWS Management Console에서 볼 수 있습니다. Kubernetes 클러스터에 연결하려면 IAM 역할을 생성합니다.

기존 EKS 커넥터 역할 확인

다음 절차를 사용하여 계정에 이미 Amazon EKS Connector 역할이 있는지 확인할 수 있습니다.

1. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 역할을 선택합니다.
3. 역할 목록에서 AmazonEKSCoordinatorAgentRole(을)를 검색합니다.
AmazonEKSCoordinatorAgentRole을 포함하는 역할이 존재하지 않을 경우 [the section called “Amazon EKS Connector 에이전트 역할 생성”](#) 섹션을 참조하여 역할을 생성합니다.
AmazonEKSCoordinatorAgentRole을 포함하는 역할이 존재하지 않을 경우, 연결된 정책을 볼 역할을 선택합니다.
4. 권한을 선택합니다.
5. AmazonEKSCoordinator에이전트Policy 관리형 정책이 역할에 연결되었는지 확인합니다. 정책이 연결된 경우 Amazon EKS 커넥터 역할이 적절히 구성된 것입니다.
6. 신뢰 관계를 선택한 후 신뢰 정책 편집을 선택합니다.
7. 신뢰 관계에 다음 정책이 포함되어 있는지 확인합니다. 신뢰 관계가 다음 정책과 일치하는 경우, 취소를 선택합니다. 신뢰 관계가 일치하지 않으면 정책을 신뢰 정책 편집 창에 복사하고 정책 업데이트를 선택합니다.

```
{
```

```

    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
          "Service": [
            "ssm.amazonaws.com"
          ]
        },
        "Action": "sts:AssumeRole"
      }
    ]
  }
}

```

Amazon EKS Connector 에이전트 역할 생성

AWS Management Console 또는 AWS CloudFormation을 사용하여 커넥터 에이전트 역할을 생성할 수 있습니다.

AWS CLI

- a. IAM 역할에 사용할 다음과 같은 JSON이 포함된 `eks-connector-agent-trust-policy.json`이라는 이름의 파일을 생성합니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "ssm.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}

```

- b. IAM 역할에 사용할 다음과 같은 JSON이 포함된 `eks-connector-agent-policy.json`이라는 이름의 파일을 생성합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SsmControlChannel",
      "Effect": "Allow",
      "Action": [
        "ssmmessages:CreateControlChannel"
      ],
      "Resource": "arn:aws:eks:*:*:cluster/*"
    },
    {
      "Sid": "ssmDataplaneOperations",
      "Effect": "Allow",
      "Action": [
        "ssmmessages:CreateDataChannel",
        "ssmmessages:OpenDataChannel",
        "ssmmessages:OpenControlChannel"
      ],
      "Resource": "*"
    }
  ]
}
```

- c. 이전 목록 항목에서 생성한 신뢰 정책 및 정책을 사용하여 Amazon EKS Connector 에이전트 역할을 생성합니다.

```
aws iam create-role \
  --role-name AmazonEKSCollectorAgentRole \
  --assume-role-policy-document file://eks-collector-agent-trust-policy.json
```

- d. Amazon EKS Connector 에이전트 역할에 정책을 연결합니다.

```
aws iam put-role-policy \
  --role-name AmazonEKSCollectorAgentRole \
  --policy-name AmazonEKSCollectorAgentPolicy \
  --policy-document file://eks-collector-agent-policy.json
```

AWS CloudFormation

- a. 다음 AWS CloudFormation 템플릿을 로컬 시스템의 텍스트 파일에 저장합니다.

Note

이 템플릿은 또한 registerCluster API가 호출될 때 생성되는 서비스 링크 역할도 생성합니다. 세부 정보는 [the section called “클러스터 커넥터 역할”](#) 섹션을 참조하세요.

```

---
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Provisions necessary resources needed to register clusters in EKS'
Parameters: {}
Resources:
  EKSConectorSLR:
    Type: AWS::IAM::ServiceLinkedRole
    Properties:
      AWSServiceName: eks-connector.amazonaws.com

  EKSConectorAgentRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Action: [ 'sts:AssumeRole' ]
            Principal:
              Service: 'ssm.amazonaws.com'

  EKSConectorAgentPolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyName: EKSConectorAgentPolicy
      Roles:
        - {Ref: 'EKSConectorAgentRole'}
      PolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: 'Allow'
            Action: [ 'ssmmessages:CreateControlChannel' ]
            Resource:
              - Fn::Sub: 'arn:${AWS::Partition}:eks:*:*:cluster/*'
          - Effect: 'Allow'

```

```

    Action: [ 'ssmmessages:CreateDataChannel',
              'ssmmessages:OpenDataChannel', 'ssmmessages:OpenControlChannel' ]
    Resource: "*"
Outputs:
  EKSConectorAgentRoleArn:
    Description: The agent role that EKS connector uses to communicate with AWS
    services.
    Value: !GetAtt EKSConectorAgentRole.Arn

```

- b. [AWS CloudFormation 콘솔](#)을 엽니다.
- c. 새 리소스를 사용한 스택 생성을 선택합니다(표준).
- d. 템플릿 지정에서 템플릿 파일 업로드를 선택한 다음 파일 선택을 선택합니다.
- e. 이전에 생성한 파일을 선택한 후 다음을 선택합니다.
- f. 스택 이름에 eksConnectorAgentRole과 같은 역할 이름을 입력하고 다음을 선택합니다.
- g. 스택 옵션 구성 페이지에서 다음을 선택합니다.
- h. 검토 페이지에서 정보를 검토하고, 스택이 IAM 리소스를 생성할 수 있음을 확인한 다음 스택 생성을 선택합니다.

Amazon Elastic Kubernetes Service에 대한 AWS 관리형 정책

AWS 관리형 정책은 AWS에 의해 생성되고 관리되는 독립 실행형 정책입니다. AWS 관리형 정책은 사용자, 그룹 및 역할에 권한 할당을 시작할 수 있도록 많은 일반 사용 사례에 대한 권한을 제공하도록 설계되었습니다.

AWS 관리형 정책은 모든 AWS 고객이 사용할 수 있기 때문에 특정 사용 사례에 대해 최소 권한을 부여하지 않을 수 있습니다. 사용 사례에 고유한 [고객 관리형 정책](#)을 정의하여 권한을 줄이는 것이 좋습니다.

AWS 관리형 정책에서 정의한 권한은 변경할 수 없습니다. AWS에서 AWS 관리형 정책에 정의된 권한을 업데이트할 경우, 정책이 연결되어 있는 모든 보안 주체 엔터티(사용자, 그룹 및 역할)에도 업데이트가 적용됩니다. 새로운 AWS 서비스를 시작하거나 새로운 API 작업을 기존 서비스에 이용하는 경우, AWS가 AWS 관리형 정책을 업데이트할 가능성이 높습니다.

자세한 내용은 IAM 사용 설명서의 [AWS 관리형 정책](#)을 참조하세요.

AWS 관리형 정책: AmazonEKS_CNI_Policy

AmazonEKS_CNI_Policy를 IAM 엔터티에 연결할 수 있습니다. Amazon EC2 노드 그룹을 생성하기 전에 이 정책을 [노드 IAM 역할](#)에 연결하거나 Kubernetes용 Amazon VPC CNI 플러그인에 명시적으

로 사용되는 IAM 역할에 연결해야 합니다. 이는 사용자를 대신하여 작업을 수행할 수 있도록 합니다. 플러그 인에서만 사용되는 역할에 정책을 연결하는 것이 좋습니다. 자세한 내용은 [the section called “Amazon VPC CNI”](#) 및 [the section called “IRSA에 대한 구성”](#)(을)를 참조하세요.

권한 세부 정보

이 정책에는 Amazon EKS가 다음 태스크를 완료할 수 있도록 허용하는 다음과 같은 권한이 포함되어 있습니다.

- **ec2:*NetworkInterface** 및 **ec2:*PrivateIpAddresses** - Amazon VPC CNI 플러그인이 Amazon EKS에서 실행되는 애플리케이션에 네트워킹을 제공하기 위해 포드의 탄력적 네트워크 인터페이스 및 IP 주소 프로비저닝과 같은 작업을 수행하도록 허용합니다.
- **ec2** 읽기 작업 - Amazon VPC CNI 플러그인이 Amazon VPC 서브넷의 사용 가능한 IP 주소 양을 확인하기 위해 인스턴스 및 서브넷 설명과 같은 작업을 수행하도록 허용합니다. VPC CNI는 각 서브넷의 사용 가능한 IP 주소를 사용하여 탄력적 네트워크 인터페이스를 생성할 때 사용 가능한 IP 주소가 가장 많은 서브넷을 선택할 수 있습니다.

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의 [AmazonEKS_CNI_Policy](#)를 참조하세요.

AWS 관리형 정책: AmazonEKSClusterPolicy

AmazonEKSClusterPolicy를 IAM 엔터티에 연결할 수 있습니다. 클러스터를 생성하기 전에 먼저 이 정책이 연결된 [클러스터 IAM 역할](#)이 있어야 합니다. Amazon EKS에서 관리하는 Kubernetes 클러스터는 사용자 대신 다른 AWS 서비스를 직접 호출합니다. 이러한 작업을 수행하여 서비스에 사용하는 리소스를 관리합니다.

이 정책에는 Amazon EKS가 다음 태스크를 완료할 수 있도록 허용하는 다음과 같은 권한이 포함되어 있습니다.

- **autoscaling** - Auto Scaling 그룹의 구성을 읽고 업데이트합니다. 이러한 권한은 Amazon EKS에서 사용되지 않지만 이전 버전과의 호환성을 위해 정책에 그대로 유지됩니다.
- **ec2** - Amazon EC2 노드에 연결된 볼륨 및 네트워크 리소스로 작업합니다. 이는 Kubernetes 컨트롤 플레인 인스턴스를 클러스터에 조인하고 Kubernetes 영구 볼륨에서 요청한 Amazon EBS 볼륨을 동적으로 프로비저닝 및 관리할 수 있도록 하기 위해 필요합니다.
- **ec2** - VPC CNI에 의해 생성된 탄력적 네트워크 인터페이스를 삭제합니다. 이는 VPC CNI가 예기치 않게 종료되는 경우 EKS가 남아 있는 탄력적 네트워크 인터페이스를 정리할 수 있도록 하는 데 필요합니다.

- **elasticloadbalancing** - Elastic 로드 밸런서를 사용하여 노드를 대상으로 추가합니다. 이는 Kubernetes 제어 플레인인 Kubernetes 서비스에서 요청한 Elastic Load Balancer를 동적으로 프로비저닝할 수 있도록 하기 위해 필요합니다.
- **iam** - 서비스 연결 역할 생성 이는 Kubernetes 컨트롤 플레인인 Kubernetes 서비스에서 요청한 Elastic Load Balancer를 동적으로 프로비저닝할 수 있도록 하기 위해 필요합니다.
- **kms** - AWS KMS에서 키를 읽습니다. 이는 Kubernetes 컨트롤 플레인인 etcd에 저장된 Kubernetes 비밀의 [비밀 암호화](#)를 지원하는 데 필요합니다.

최신 버전의 JSON 정책 문서를 보려면AWS 관리형 정책 참조 안내서의 [AmazonEKSClusterPolicy](#)를 참조하세요.

AWS 관리형 정책: AmazonEKSFargatePodExecutionRolePolicy

AmazonEKSFargatePodExecutionRolePolicy를 IAM 엔터티에 연결할 수 있습니다. Fargate 프로필을 생성하려면 먼저 Fargate 포드 실행 역할을 생성하고 이 정책을 이 역할에 연결해야 합니다. 자세한 내용은 [the section called “2단계: Fargate 포드 실행 역할 생성”](#) 및 [the section called “프로필 정의”](#)(을)를 참조하세요.

이 정책은 Fargate에서 Amazon EKS 포드를 실행하는 데 필요한 다른 AWS 서비스 리소스에 대한 액세스를 제공하는 권한을 역할에 부여합니다.

권한 세부 정보

이 정책에는 Amazon EKS가 다음 태스크를 완료할 수 있도록 허용하는 다음과 같은 권한이 포함되어 있습니다.

- **ecr** - Fargate에서 실행 중인 포드가 Amazon ECR에 저장된 컨테이너 이미지를 가져오도록 허용합니다.

최신 버전의 JSON 정책 문서를 보려면AWS 관리형 정책 참조 안내서의 [AmazonEKSFargatePodExecutionRolePolicy](#)를 참조하세요.

AWS 관리형 정책: AmazonEKSFargateServiceRolePolicy

AmazonEKSFargateServiceRolePolicy를 IAM 엔터티에 연결할 수 없습니다. 이 정책은 Amazon EKS에 사용자를 대신하여 작업을 수행할 수 있도록 서비스 연결 역할에 연결됩니다. 자세한 내용은 [AWSServiceRoleforAmazonEKSFargate](#) 섹션을 참조하세요.

이 정책은 Amazon EKS에 Fargate 태스크를 실행하는 데 필요한 권한을 부여합니다. 이 정책은 Fargate 노드가 있는 경우에만 사용됩니다.

권한 세부 정보

이 정책에는 Amazon EKS가 다음 태스크를 완료할 수 있도록 허용하는 다음과 같은 권한이 포함되어 있습니다.

- **ec2** – Elastic 네트워크 인터페이스를 생성 및 삭제하고 탄력적 네트워크 인터페이스 및 리소스를 설명합니다. 이는 Amazon EKS Fargate 서비스가 Fargate 포드에 필요한 VPC 네트워킹을 구성할 수 있도록 하기 위해 필요합니다.

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의 [AmazonEKSFargateServiceRolePolicy](#)를 참조하세요.

AWS 관리형 정책: AmazonEKSCoordinatePolicy

AmazonEKSCoordinatePolicy을(를) IAM 엔티티에 연결할 수 있습니다. 이 정책을 [클러스터 IAM 역할](#)에 연결하여 EKS가 계정에서 관리할 수 있는 리소스를 확장할 수 있습니다.

이 정책은 Amazon EKS가 EKS 클러스터에 대한 EC2 인스턴스 생성 및 관리에 필요한 권한과 EC2 구성에 필요한 IAM 권한을 부여합니다.

권한 세부 정보

이 정책에는 Amazon EKS가 다음 태스크를 완료할 수 있도록 허용하는 다음과 같은 권한이 포함되어 있습니다.

- **ec2** 권한:
 - `ec2:CreateFleet` 및 `ec2:RunInstances`-EC2 인스턴스를 생성하고 EKS 클러스터 노드에 대한 특정 EC2 리소스(이미지, 보안 그룹, 서브넷)를 사용할 수 있습니다.
 - `ec2:CreateLaunchTemplate`-EKS 클러스터 노드에 대한 EC2 시작 템플릿을 생성할 수 있습니다.
 - 이 정책에는 이러한 EC2 권한 사용을 EKS 클러스터 이름 및 기타 관련 태그가 지정된 리소스로 제한하는 조건도 포함되어 있습니다.
 - `ec2:CreateTags`-`CreateFleet`, `RunInstances`, `CreateLaunchTemplate` 작업에서 생성한 EC2 리소스에 태그를 추가할 수 있습니다.
- **iam** 권한:

- `iam:AddRoleToInstanceProfile-EKS` 컴퓨팅 인스턴스 프로파일에 IAM 역할을 추가할 수 있습니다.
- `iam:PassRole`-필요한 IAM 역할을 EC2 서비스에 전달할 수 있습니다.

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의

[AmazonEKSCoordinatePolicy](#)를 참조하세요.

AWS 관리형 정책: AmazonEKSNetworkPolicy

AmazonEKSNetworkPolicy을(를) IAM 엔티티에 연결할 수 있습니다. 이 정책을 [클러스터 IAM 역할](#)에 연결하여 EKS가 계정에서 관리할 수 있는 리소스를 확장할 수 있습니다.

이 정책은 Amazon EKS가 EKS 클러스터에 대한 네트워크 인터페이스를 생성하고 관리하는 데 필요한 권한을 부여하여 제어 영역과 작업자 노드가 통신하고 제대로 작동할 수 있도록 설계되었습니다.

권한 세부 정보

이 정책은 Amazon EKS가 클러스터의 네트워크 인터페이스를 관리할 수 있도록 허용하는 다음과 같은 권한을 부여합니다.

- **ec2 네트워크 인터페이스 권한 ID**
 - `ec2:CreateNetworkInterface`-EC2 네트워크 인터페이스를 생성할 수 있습니다.
 - 이 정책에는 EKS 클러스터 이름 및 Kubernetes CNI 노드 이름으로 태그가 지정된 네트워크 인터페이스에 대한 이 권한 사용을 제한하는 조건이 포함되어 있습니다.
 - `ec2:CreateTags-CreateNetworkInterface` 작업에서 생성한 네트워크 인터페이스에 태그를 추가할 수 있습니다.
- **ec2 네트워크 인터페이스 관리 권한:**
 - `ec2:AttachNetworkInterface`, `ec2:DetachNetworkInterface`-EC2 인스턴스에 네트워크 인터페이스를 연결하고 분리할 수 있습니다.
 - `ec2:UnassignPrivateIpAddresses`, `ec2:UnassignIpv6Addresses`, `ec2:AssignPrivateIpAddresses`, `ec2:AssignIpv6Addresses`-네트워크 인터페이스의 IP 주소 할당을 관리할 수 있습니다.
 - 이러한 권한은 EKS 클러스터 이름으로 태그가 지정된 네트워크 인터페이스로 제한됩니다.

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의

[AmazonEKSNetworkPolicy](#)를 참조하세요.

AWS 관리형 정책: AmazonEKSBlockStoragePolicy

AmazonEKSBlockStoragePolicy을(를) IAM 엔티티에 연결할 수 있습니다. 이 정책을 [클러스터 IAM 역할](#)에 연결하여 EKS가 계정에서 관리할 수 있는 리소스를 확장할 수 있습니다.

이 정책은 Amazon EKS가 EKS 클러스터에 대한 EC2 볼륨 및 스냅샷을 생성, 관리 및 유지하는 데 필요한 권한을 부여하여 컨트롤 플레인과 워커 노드가 Kubernetes 워크로드에서 요구하는 영구 스토리지를 프로비저닝하고 사용할 수 있게 합니다.

권한 세부 정보

이 IAM 정책은 Amazon EKS가 EC2 볼륨 및 스냅샷을 관리할 수 있도록 다음과 같은 권한을 부여합니다.

- **ec2 볼륨 관리 권한:**
 - ec2:AttachVolume, ec2:DetachVolume, ec2:ModifyVolume, ec2:EnableFastSnapshotRestores-EC2 볼륨에 대한 빠른 스냅샷 복원을 연결, 분리, 수정, 활성화할 수 있습니다.
 - 이러한 권한은 EKS 클러스터 이름으로 태그가 지정된 볼륨으로 제한됩니다.
 - ec2:CreateTags-CreateVolume 및 CreateSnapshot 작업에서 생성된 EC2 볼륨 및 스냅샷에 태그를 추가할 수 있습니다.
- **ec2 볼륨 생성 권한:**
 - ec2:CreateVolume-새 EC2 볼륨을 생성할 수 있습니다.
 - 이 정책에는 이 권한의 사용을 EKS 클러스터 이름 및 기타 관련 태그가 지정된 볼륨으로 제한하는 조건이 포함되어 있습니다.
 - ec2:CreateSnapshot-새 EC2 볼륨 스냅샷을 생성할 수 있습니다.
 - 이 정책에는 이 권한의 사용을 EKS 클러스터 이름 및 기타 관련 태그가 지정된 스냅샷으로 제한하는 조건이 포함되어 있습니다.

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의 [AmazonEKSBlockStoragePolicy](#)를 참조하세요.

AWS 관리형 정책: AmazonEKSLoadBalancingPolicy

AmazonEKSLoadBalancingPolicy을(를) IAM 엔티티에 연결할 수 있습니다. 이 정책을 [클러스터 IAM 역할](#)에 연결하여 EKS가 계정에서 관리할 수 있는 리소스를 확장할 수 있습니다.

이 IAM 정책은 Amazon EKS가 다양한 AWS 서비스를 함께 사용하여 Elastic Load Balancing(ELB) 및 관련 리소스를 관리하는 데 필요한 권한을 부여합니다.

권한 세부 정보

이 정책에서 부여하는 주요 권한은 다음과 같습니다.

- **elasticloadbalancing** : Elastic Load Balancer 및 대상 그룹을 생성, 수정, 관리할 수 있습니다. 여기에는 로드 밸런서, 대상 그룹, 리스너, 규칙을 생성, 업데이트, 삭제하는 권한이 포함됩니다.
- **ec2** : Kubernetes 컨트롤 플레인이 인스턴스를 클러스터에 조인하고 Amazon EBS 볼륨을 관리하는 데 필요한 보안 그룹을 생성하고 관리할 수 있습니다. 또한 인스턴스, VPC, 서브넷, 보안 그룹 및 기타 네트워킹 리소스와 같은 EC2 리소스를 설명하고 나열할 수 있습니다.
- **iam** : Kubernetes 컨트롤 플레인이 ELB를 동적으로 프로비저닝하는 데 필요한 Elastic Load Balancing에 대한 서비스 연결 역할을 생성할 수 있습니다.
- **kms** : AWS KMS에서 키를 읽을 수 있습니다. 이 키는 Kubernetes 컨트롤 플레인이 etcd에 저장된 Kubernetes 보안 암호의 암호화를 지원하기 위해 필요합니다.
- **wafv2** 및 **shield** : 웹 ACL을 연결 및 연결 해제하고 Elastic Load Balancer에 대한 AWS Shield 보호를 생성/삭제할 수 있습니다.
- **cognito-idp**, **acm**, **elasticloadbalancing** : 사용자 풀 클라이언트를 설명하고, 인증서를 나열 및 설명하고, Kubernetes 컨트롤 플레인이 Elastic Load Balancer를 관리하는 데 필요한 대상 그룹을 설명할 수 있는 권한을 부여합니다.

또한 이 정책에는 `eks:eks-cluster-name` 태그를 사용하여 관리 중인 특정 EKS 클러스터에 대한 권한 범위를 확인하기 위한 여러 조건 확인이 포함되어 있습니다.

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의 [AmazonEKSLoadBalancingPolicy](#)를 참조하세요.

AWS 관리형 정책: AmazonEKSServicePolicy

AmazonEKSServicePolicy을(를) IAM 엔티티에 연결할 수 있습니다. 2020년 4월 16일 이전에 생성된 클러스터에서는 IAM 역할을 만들어 이 정책을 역할에 연결해야 했습니다. 2020년 4월 16일 이후에 생성된 클러스터에서는 역할을 생성할 필요가 없으며 이 정책을 할당할 필요가 없습니다. `iam:CreateServiceLinkedRole` 권한이 있는 IAM 보안 주체를 사용하여 클러스터를 생성하는 경우 [AWSServiceRoleforAmazonEKS](#) 서비스 연결 역할이 자동으로 생성됩니다. 서비스 연결 역할에는 [관리형 정책: AmazonEKSServiceRolePolicy](#)가 연결되어 있습니다.

이 정책을 통해 Amazon EKS는 Amazon EKS 클러스터를 운영하는 데 필요한 리소스를 생성하고 관리할 수 있습니다.

권한 세부 정보

이 정책에는 Amazon EKS가 다음 태스크를 완료할 수 있도록 허용하는 다음과 같은 권한이 포함되어 있습니다.

- **eks** - 사용자가 업데이트를 시작한 후 클러스터의 Kubernetes 버전을 업데이트합니다. 이 권한은 Amazon EKS에 사용되지 않지만 이전 버전과의 호환성을 위해 정책에 그대로 유지됩니다.
- **ec2** - Elastic 네트워크 인터페이스 및 기타 네트워크 리소스와 태그를 사용합니다. 이는 Amazon EKS에서 노드와 Kubernetes 제어부 간의 통신을 용이하게 하는 네트워킹을 구성하는 데 필요합니다. 보안 그룹에 대한 정보를 읽습니다. 보안 그룹의 태그를 업데이트합니다.
- **route53** - VPC를 호스팅 영역에 연결합니다. 이는 Amazon EKS에서 Kubernetes 클러스터 API 서버에 프라이빗 엔드포인트 네트워킹을 활성화하는 데 필요합니다.
- **logs** - 로그 이벤트 이는 Amazon EKS가 Kubernetes 제어 영역 로그를 CloudWatch로 전송하기 위해 필요합니다.
- **iam** - 서비스 연결 역할 생성 이는 Amazon EKS가 사용자를 대신하여 [the section called “Amazon EKS에 대한 서비스 연결 역할 권한”](#) 서비스 연결 역할을 생성하는 데 필요합니다.

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의 [AmazonEKSServicePolicy](#)를 참조하세요.

AWS 관리형 정책: AmazonEKSServiceRolePolicy

AmazonEKSServiceRolePolicy를 IAM 엔터티에 연결할 수 없습니다. 이 정책은 Amazon EKS에 사용자를 대신하여 작업을 수행할 수 있도록 서비스 연결 역할에 연결됩니다. 자세한 내용은 [the section called “Amazon EKS에 대한 서비스 연결 역할 권한”](#) 섹션을 참조하세요.

iam:CreateServiceLinkedRole 권한이 있는 IAM 보안 주체를 사용하여 클러스터를 생성하는 경우 [AWSServiceRoleforAmazonEKS](#) 서비스 연결 역할이 자동으로 생성되며 이 정책이 연결됩니다.

이 정책을 사용하면 서비스 연결 역할이 사용자를 대신하여 AWS 서비스를 직접 호출할 수 있습니다.

권한 세부 정보

이 정책에는 Amazon EKS가 다음 태스크를 완료할 수 있도록 허용하는 다음과 같은 권한이 포함되어 있습니다.

- **ec2** - 탄력적 네트워크 인터페이스와 Amazon EC2 인스턴스, 클러스터 보안 그룹 및 클러스터를 생성하는 데 필요한 VPC를 생성하고 설명합니다. 자세한 내용은 [the section called “보안 그룹 요구 사항”](#) 섹션을 참조하세요. 보안 그룹에 대한 정보를 읽습니다. 보안 그룹의 태그를 업데이트합니다.
- **ec2 Auto Mode** - EKS Auto Mode에서 생성된 EC2 인스턴스를 종료합니다. 자세한 내용은 [EKS Auto Mode](#) 섹션을 참조하세요.
- **iam** - IAM 역할에 연결된 모든 관리형 정책을 나열합니다. 이는 Amazon EKS가 클러스터 생성에 필요한 모든 관리형 정책 및 권한을 나열하고 검증하는 데 필요합니다.
- **호스팅 영역과 VPC 연결** - 이는 Amazon EKS에서 Kubernetes 클러스터 API 서버에 프라이빗 엔드포인트 네트워킹을 사용 설정하는 데 필요합니다.
- **로그 이벤트** - 이는 Amazon EKS가 Kubernetes 제어 영역 로그를 CloudWatch로 전송하기 위해 필요합니다.
- **지표 입력** - Amazon EKS가 Kubernetes 컨트롤 플레인 로그를 CloudWatch로 전송하기 위해 필요합니다.
- **eks** -클러스터 액세스 항목 및 정책을 관리하여 EKS 리소스에 액세스할 수 있는 사용자와 수행 가능한 작업을 세밀하게 제어할 수 있습니다. 여기에는 컴퓨팅, 네트워킹, 로드 밸런싱, 스토리지 작업에 대한 표준 액세스 정책 연결이 포함됩니다.
- **elasticloadbalancing** -EKS 클러스터와 연결된 로드 밸런서 및 해당 구성 요소(리스너, 대상 그룹, 인증서)를 생성, 관리, 삭제합니다. 로드 밸런서 속성 및 상태를 봅니다.
- **events** -EKS 클러스터와 관련된 EC2 및 AWS 상태 이벤트를 모니터링하기 위한 EventBridge 규칙을 생성하고 관리하여 인프라 변경 및 상태 알림에 대한 자동 응답을 활성화합니다.
- **iam** -EKS 노드 관리에 필요한 생성, 삭제, 역할 연결을 포함하여 'eks' 접두사를 사용하여 EC2 인스턴스 프로파일을 관리합니다.
- **pricing** 및 **shield** -AWS요금 정보 및 Shield 보호 상태에 액세스하여 EKS 리소스에 대한 비용 관리 및 고급 보안 기능을 활성화합니다.
- **리소스 정리** -클러스터 정리 작업 중에 볼륨, 스냅샷, 시작 템플릿, 네트워크 인터페이스를 포함하여 EKS 태그가 지정된 리소스를 안전하게 삭제합니다.

최신 버전의 JSON 정책 문서를 보려면AWS 관리형 정책 참조 안내서의 [AmazonEKSServiceRolePolicy](#)를 참조하세요.

AWS 관리형 정책: AmazonEKSVPCResourceController

AmazonEKSVPCResourceController 정책을 IAM 보안 인증에 연결할 수 있습니다. [포드에 보안 그룹](#)을 사용하는 경우, 사용자를 대신하여 작업을 수행하려면 이 정책을 [Amazon EKS 클러스터 IAM 역할](#)에 첨부해야 합니다.

이 정책은 노드의 탄력적 네트워크 인터페이스 및 IP 주소를 관리하기 위한 클러스터 역할에 권한을 부여합니다.

권한 세부 정보

이 정책에는 Amazon EKS가 다음 태스크를 완료할 수 있도록 허용하는 다음과 같은 권한이 포함되어 있습니다.

- **ec2** - 탄력적 네트워크 인터페이스 및 IP 주소를 관리하여 포드 보안 그룹 및 Windows 노드를 지원합니다.

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의 [AmazonEKSVPCResourceController](#)를 참조하세요.

AWS 관리형 정책: AmazonEKSWorkerNodePolicy

AmazonEKSWorkerNodePolicy를 IAM 엔터티에 연결할 수 있습니다. 이 정책을 Amazon EC2 노드 생성 시에 지정한, Amazon EKS에서 사용자를 대신하여 작업을 수행하도록 허용하는 [IAM 역할](#)에 연결해야 합니다. eksctl을 사용하여 노드 그룹을 생성하는 경우, 노드 IAM 역할을 생성하고 이 정책을 역할에 자동으로 연결합니다.

이 정책은 Amazon EKS Amazon EC2 노드에 Amazon EKS 클러스터에 연결할 수 있는 권한을 부여합니다.

권한 세부 정보

이 정책에는 Amazon EKS가 다음 태스크를 완료할 수 있도록 허용하는 다음과 같은 권한이 포함되어 있습니다.

- **ec2** - 인스턴스 볼륨 및 네트워크 정보를 읽습니다. 이는 Kubernetes 노드가 Amazon EKS 클러스터에 가입하는 데 필요한 Amazon EC2 리소스에 대한 정보를 설명하는 데 필요합니다.
- **eks** - 선택적으로 클러스터를 노드 부트스트랩의 일부로 설명합니다.
- **eks-auth:AssumeRoleForPodIdentity** - 노드에서 EKS 워크로드에 대한 보안 인증 정보 검색을 허용합니다. EKS Pod Identity가 제대로 작동하려면 필요합니다.

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의 [AmazonEKSWorkerNodePolicy](#)를 참조하세요.

AWS 관리형 정책: AmazonEKSWorkerNodeMinimalPolicy

AmazonEKSWorkerNodeMinimalPolicy를 IAM 엔티티에 연결할 수 있습니다. 이 정책을 Amazon EC2 노드 생성 시에 지정한, Amazon EKS에서 사용자를 대신하여 작업을 수행하도록 허용하는 IAM 역할에 연결할 수 있습니다.

이 정책은 Amazon EKS Amazon EC2 노드에 Amazon EKS 클러스터에 연결할 수 있는 권한을 부여합니다. 이 정책은 AmazonEKSWorkerNodePolicy에 비해 권한이 적습니다.

권한 세부 정보

이 정책에는 Amazon EKS가 다음 태스크를 완료할 수 있도록 허용하는 다음과 같은 권한이 포함되어 있습니다.

- `eks-auth:AssumeRoleForPodIdentity` - 노드에서 EKS 워크로드에 대한 보안 인증 정보 검색을 허용합니다. EKS Pod Identity가 제대로 작동하려면 필요합니다.

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의 [AmazonEKSWorkerNodePolicy](#)를 참조하세요.

AWS 관리형 정책: AWSServiceRoleForAmazonEKSNodegroup

AWSServiceRoleForAmazonEKSNodegroup를 IAM 엔티티에 연결할 수 없습니다. 이 정책은 Amazon EKS에 사용자를 대신하여 작업을 수행할 수 있도록 서비스 연결 역할에 연결됩니다. 자세한 내용은 [the section called “Amazon EKS에 대한 서비스 연결 역할 권한”](#) 섹션을 참조하세요.

이 정책은 계정에서 Amazon EC2 노드 그룹을 생성하고 관리하도록 허용하는 AWSServiceRoleForAmazonEKSNodegroup 역할 권한을 부여합니다.

권한 세부 정보

이 정책에는 Amazon EKS가 다음 태스크를 완료할 수 있도록 허용하는 다음과 같은 권한이 포함되어 있습니다.

- **ec2** - 보안 그룹, 태그, 용량 예약, 시작 템플릿을 사용합니다. 이는 Amazon EKS 관리형 노드 그룹이 원격 액세스 구성을 활성화하고 관리형 노드 그룹에서 사용할 수 있는 용량 예약을 설명하는 데 필요합니다. 또한 Amazon EKS 관리 노드 그룹은 사용자를 대신하여 시작 템플릿을 생성합니다. 이는 각 관리형 노드 그룹을 백업하는 Amazon EC2 Auto Scaling 그룹을 구성하기 위한 것입니다.
- **iam** - 서비스 연결 역할을 생성하고 역할을 전달합니다. 이는 Amazon EKS 관리형 노드 그룹이 관리형 노드 그룹을 생성할 때 전달되는 역할의 인스턴스 프로필을 관리하는 데 필요합니다. 이 인스턴

스 프로파일은 관리형 노드 그룹의 일부로 시작된 Amazon EC2 인스턴스에 사용됩니다. Amazon EKS 는 Amazon EC2 Auto Scaling 그룹과 같은 다른 서비스에 대한 서비스 연결 역할을 생성해야 합니다. 이러한 권한은 관리 노드 그룹 생성에 사용됩니다.

- **autoscaling** - 보안 Auto Scaling 그룹을 사용합니다. 이는 Amazon EKS 관리 노드 그룹이 각 관리 노드 그룹을 백업하는 Amazon EC2 Auto Scaling 그룹을 관리하는 데 필요합니다. 또한 노드 그룹 업데이트 중에 노드가 종료되거나 재활용될 때 포드 제거와 같은 기능을 지원하는 데에도 사용됩니다.

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의 [AWSServiceRoleForAmazonEKSNodegroup](#)를 참조하세요.

AWS 관리형 정책: AmazonEKSDashboardServiceRolePolicy

AmazonEKSDashboardServiceRolePolicy를 IAM 엔터티에 연결할 수 없습니다. 이 정책은 Amazon EKS에 사용자를 대신하여 작업을 수행할 수 있도록 서비스 연결 역할에 연결됩니다. 자세한 내용은 [the section called “Amazon EKS에 대한 서비스 연결 역할 권한”](#) 섹션을 참조하세요.

이 정책은 계정에서 Amazon EC2 노드 그룹을 생성하고 관리하도록 허용하는 AWSServiceRoleForAmazonEKSDashboard 역할 권한을 부여합니다.

권한 세부 정보

이 정책에는 이러한 작업을 완료하기 위해 액세스를 허용하는 다음 권한이 포함되어 있습니다.

- **organizations** - AWS Organizations 구조 및 계정에 대한 정보를 봅니다. 여기에는 조직의 계정을 나열하고, 조직 단위 및 루트를 보고, 위임 관리자를 나열하고, 조직에 액세스할 수 있는 서비스를 보고, 조직 및 계정에 대한 세부 정보를 검색할 수 있는 권한이 포함됩니다.

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의 [AmazonEKSDashboardServiceRolePolicy](#)를 참조하세요.

AWS 관리형 정책: AmazonEBSCSIDriverPolicy

AmazonEBSCSIDriverPolicy 정책은 Amazon EBS Container Storage Interface(CSI) 드라이버가 사용자를 대신하여 볼륨을 생성, 수정, 연결, 분리 및 삭제할 수 있도록 허용합니다. 여기에는 기존 볼륨에서 태그를 수정하고 EBS 볼륨에서 빠른 스냅샷 복원(FSR)을 활성화하는 작업이 포함됩니다. 또한 EBS CSI 드라이버에 스냅샷을 생성, 복원 및 삭제하고 인스턴스, 볼륨 및 스냅샷을 나열할 수 있는 권한을 부여합니다.

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의 [AmazonEBSCSIDriverServiceRolePolicy](#)를 참조하세요.

AWS 관리형 정책: AmazonEFSCSIDriverPolicy

AmazonEFSCSIDriverPolicy 정책을 통해 Amazon EFS 컨테이너 스토리지 인터페이스(CSI)가 사용자를 대신하여 액세스 포인트를 생성하고 삭제할 수 있습니다. 또한 Amazon EFS CSI 드라이버에 액세스 포인트, 파일 시스템, 탑재 대상 및 Amazon EC2 가용 영역을 나열할 수 있는 권한을 부여합니다.

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의 [AmazonEFSCSIDriverServiceRolePolicy](#)를 참조하세요.

AWS 관리형 정책: AmazonEKSLocalOutpostClusterPolicy

IAM 엔터티에 이 정책을 연결할 수 있습니다. 로컬 클러스터를 생성하기 전에 이 정책을 [클러스터 역할](#)에 연결해야 합니다. Amazon EKS에서 관리하는 Kubernetes 클러스터는 사용자 대신 다른 AWS 서비스를 직접 호출합니다. 이러한 작업을 수행하여 서비스에 사용하는 리소스를 관리합니다.

AmazonEKSLocalOutpostClusterPolicy에는 다음 권한이 포함되어 있습니다.

- **ec2** 읽기 작업-컨트롤 플레인 인스턴스가 가용 영역, 라우팅 테이블, 인스턴스, 네트워크 인터페이스 속성을 설명하도록 허용합니다. Amazon EC2 인스턴스가 성공적으로 클러스터에 컨트롤 플레인 인스턴스로 조인하는 데 필요한 권한입니다.
- **ssm** - Amazon EKS가 계정의 로컬 클러스터를 통신하고 관리하는 데 사용하는 컨트롤 플레인 인스턴스에 대한 Amazon EC2 Systems Manager 연결을 허용합니다.
- **logs** - 인스턴스가 Amazon CloudWatch에 로그를 푸시하도록 허용합니다.
- **secretsmanager** - 인스턴스가 AWS Secrets Manager에서 안전하게 컨트롤 플레인 인스턴스에 대한 부트스트랩 데이터를 가져오고 삭제하도록 허용합니다.
- **ecr** - 컨트롤 플레인 인스턴스에서 실행 중인 포드와 컨테이너가 Amazon Elastic Container Registry에 저장된 컨테이너 이미지를 끌어오도록 허용합니다.

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의 [AmazonEKS_CNI_Policy](#)를 참조하세요.

AWS 관리형 정책: AmazonEKSLocalOutpostServiceRolePolicy

IAM 엔터티에 이 정책을 연결할 수 없습니다. iam:CreateServiceLinkedRole 권한이 있는 IAM 보안 주체를 사용하여 클러스터를 생성하면 Amazon EKS가 자동으로

[AWSServiceRoleforAmazonEKSLocalOutpost](#) 서비스 연결 역할을 생성하고 이 정책을 연결합니다. 이 정책을 사용하면 서비스 연결 역할이 로컬 클러스터에 대해 사용자를 대신하여 AWS 서비스를 직접 호출할 수 있습니다.

AmazonEKSLocalOutpostServiceRolePolicy에는 다음 권한이 포함되어 있습니다.

- **ec2** - Amazon EKS가 보안, 네트워크 및 기타 리소스와 함께 작동하여 계정에서 컨트롤 플레인 인스턴스를 성공적으로 시작하고 관리하도록 허용합니다.
- **ssm** - Amazon EKS가 계정의 로컬 클러스터를 통신하고 관리하는 데 사용하는 컨트롤 플레인 인스턴스에 대한 Amazon EC2 Systems Manager 연결을 허용합니다.
- **iam** - Amazon EKS가 컨트롤 플레인 인스턴스와 연결된 인스턴스 프로필을 관리하도록 허용합니다.
- **secretsmanager** - 인스턴스 부트스트랩 중 안전하게 참조할 수 있게 Amazon EKS가 컨트롤 플레인 인스턴스에 대한 부트스트랩 데이터를 AWS Secrets Manager에 저장하도록 허용합니다.
- **outposts** - Amazon EKS가 계정에서 Outpost 정보를 가져와 Outpost에서 로컬 클러스터를 성공적으로 시작할 수 있도록 합니다.

최신 버전의 JSON 정책 문서를 보려면 AWS 관리형 정책 참조 안내서의 [AmazonEKS_CNI_Policy](#)를 참조하세요.

AWS 관리형 정책에 대한 Amazon EKS 업데이트

이 서비스에서 이러한 변경 사항 추적을 시작한 이후 Amazon EKS에 대한 AWS 관리형 정책 업데이트에 대한 세부 정보를 확인합니다.

이 특정 설명서 페이지의 모든 소스 파일 변경 사항에 대한 알림을 받으려면 RSS 리더를 사용하여 다음 URL을 구독하세요.

<https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/security/iam-reference/security-iam-awsmanpol.adoc.atom>

변경 사항	설명	날짜
the section called “AWS 관리형 정책: AmazonEKSDashboardServiceRolePolicy” 를 소개합니다.	새 AmazonEKSDashboardServiceRolePolicy 정책이 도입되었습니다.	2025년 5월 21일

변경 사항	설명	날짜
AmazonEKSClusterPolicy 에 권한을 추가했습니다.	VPC CNI가 예기치 않게 종료되는 경우 Amazon EKS가 남아 있는 탄력적 네트워크 인터페이스를 삭제할 수 있도록 <code>ec2:DeleteNetworkInterfaces</code> 권한이 추가되었습니다.	2025년 4월 16일
AmazonEKSServiceRolePolicy 에 권한이 추가되었습니다.	EKS 1.33 버전 릴리스의 일부로 EKS AI/ML 고객이 EFA와 호환되는 기본 EKS 클러스터 SG에 보안 그룹 송신 규칙을 추가할 수 있도록 <code>ec2:RevokeSecurityGroupEgress</code> 및 <code>ec2:AuthorizeSecurityGroupEgress</code> 권한이 추가되었습니다.	2025년 4월 14일
AmazonEKSServiceRolePolicy 에 권한이 추가되었습니다.	EKS Auto Mode에서 생성된 EC2 인스턴스를 종료할 수 있는 권한이 추가되었습니다.	2025년 2월 28일

변경 사항	설명	날짜
AmazonEBSCSIDriverPolicy 에 권한을 추가했습니다.	EBS CSI 드라이버에 모든 스냅샷을 복원할 수 있는 권한을 부여하는 새 명령문을 추가했습니다. 이전에 기존 정책에서 허용되었지만, CreateVolume에 대한 IAM 처리가 변경되어 새로운 명시적 명령문이 필요합니다. EBS CSI 드라이버가 기존 볼륨에서 태그를 수정할 수 있는 기능이 추가되었습니다. EBS CSI 드라이버는 Kubernetes VolumeAttributesClasses의 파라미터를 통해 기존 볼륨의 태그를 수정할 수 있습니다. EBS CSI 드라이버가 EBS 볼륨에서 빠른 스냅샷 복원(FSR)을 활성화할 수 있는 기능을 추가했습니다. EBS CSI 드라이버는 Kubernetes 스토리지 클래스의 파라미터를 통해 새 볼륨에서 FSR을 활성화할 수 있습니다.	2025년 1월 13일
the section called “AWS 관리형 정책: AmazonEKSLoadBalancingPolicy” 에 권한이 추가되었습니다.	네트워킹 및 IP 주소 리소스를 나열하고 기술할 수 있도록 AmazonEKSLoadBalancingPolicy를 업데이트했습니다.	2024년 12월 26일

변경 사항	설명	날짜
the section called “AWS 관리형 정책: AWSServiceRoleForAmazonEKSNodegroup” 에 권한이 추가되었습니다.	중국 리전과의 호환성을 위해 AWSServiceRoleForAmazonEKSNodegroup 이 업데이트되었습니다.	2024년 11월 22일
the section called “AWS 관리형 정책: AmazonEKSLocalOutpostClusterPolicy” 에 권한 추가	클러스터 컨트롤 플레인 의 AWS Cloud Controller Manager가 각 노드가 있는 가용 영역을 식별할 수 있도록 AmazonEKSLocalOutpostClusterPolicy 에 대한 ec2:DescribeAvailabilityZones 권한이 추가되었습니다.	2024년 11월 21일
the section called “AWS 관리형 정책: AWSServiceRoleForAmazonEKSNodegroup” 에 권한이 추가되었습니다.	Amazon EKS 관리형 노드 그룹에서 생성한 인스턴스에서 ec2:RebootInstances 를 허용하도록 AWSServiceRoleForAmazonEKSNodegroup 정책이 업데이트되었습니다. Amazon EC2 리소스에 대한 ec2:CreateTags 권한이 제한되었습니다.	2024년 11월 20일
the section called “AWS 관리형 정책: AmazonEKSServiceRolePolicy” 에 권한이 추가되었습니다.	EKS가 AWS 관리형 정책 AmazonEKSServiceRolePolicy 를 업데이트했습니다. EKS 액세스 정책, 로드 밸런서 관리, 자동 클러스터 리소스 정리에 대한 권한이 추가되었습니다.	2024년 11월 16일

변경 사항	설명	날짜
the section called “AWS 관리형 정책: AmazonEKS ComputePolicy” 를 소개합니다.	EKS가 AWS 관리형 정책 AmazonEKSComputePolicy 를 업데이트했습니다. iam:AddRoleToInstanceProfile 작업에 대한 리소스 권한이 업데이트되었습니다.	2024년 11월 7일
the section called “AWS 관리형 정책: AmazonEKS ComputePolicy” 를 소개합니다.	AWS에서는 AmazonEKS ComputePolicy 를 도입했습니다.	2024년 11월 1일
AmazonEKSClusterPolicy 에 권한 추가	Amazon EKS가 노드에 토폴로지 정보를 레이블로 연결할 수 있는 ec2:DescribeInstanceTopology 권한이 추가되었습니다.	2024년 11월 1일
the section called “AWS 관리형 정책: AmazonEKS BlockStoragePolicy” 를 소개합니다.	AWS에서는 AmazonEKS BlockStoragePolicy 를 도입했습니다.	2024년 10월 30일
the section called “AWS 관리형 정책: AmazonEKS LoadBalancingPolicy” 를 소개합니다.	AWS에서는 AmazonEKS LoadBalancingPolicy 를 도입했습니다.	2024년 10월 30일
AmazonEKSServiceRolePolicy 에 권한이 추가되었습니다.	Amazon EKS가 Amazon CloudWatch에 지표를 게시할 수 있는 cloudwatch:PutMetricData 권한이 추가되었습니다.	2024년 10월 29일

변경 사항	설명	날짜
the section called “AWS 관리형 정책: AmazonEKS NetworkingPolicy” 를 소개합니다.	AWS에서는 AmazonEKS NetworkingPolicy 를 도입했습니다.	2024년 10월 28일
AmazonEKSServicePolicy 및 AmazonEKS ServiceRolePolicy 에 권한을 추가했습니다	EKS가 보안 그룹 정보를 읽고 관련 태그를 업데이트할 수 있도록 ec2:GetSecurityGroupsForVpc 및 연결된 태그 권한이 추가되었습니다.	2024년 10월 10일
AmazonEKSWorkerNodeMinimalPolicy 를 도입했습니다.	AWS에서는 AmazonEKS WorkerNodeMinimalPolicy 를 도입했습니다.	2024년 10월 3일
AWSServiceRoleForAmazonEKSNodegroup 에 권한을 추가했습니다.	Amazon EKS가 Amazon EKS 관리형 Auto Scaling 그룹에서 AZRebalance 를 일시 중단 및 다시 재개할 수 있도록 autoscaling:ResumeProcesses 및 autoscaling:SuspendProcesses 권한이 추가되었습니다.	2024년 8월 21일
AWSServiceRoleForAmazonEKSNodegroup 에 권한을 추가했습니다.	사용자 계정의 용량 예약을 설명할 수 있는 ec2:DescribeCapacityReservations 권한을 Amazon EKS에 추가했습니다. CAPACITY_BLOCK 노드 그룹에서 예약된 규모 조정을 설정할 수 있는 autoscaling:PutScheduledUpdateGroupAction 권한이 추가되었습니다.	2024년 6월 27일

변경 사항	설명	날짜
AmazonEKS_CNI_Policy – 기존 정책 업데이트	Amazon EKS에 Kubernetes용 Amazon VPC CNI 플러그인에서 Amazon VPC 서브넷의 사용 가능한 IP 주소 양을 볼 수 있도록 허용하는 <code>ec2:DescribeSubnets</code> 권한이 새로 추가되었습니다. VPC CNI는 각 서브넷의 사용 가능한 IP 주소를 사용하여 탄력적 네트워크 인터페이스를 생성할 때 사용 가능한 IP 주소가 가장 많은 서브넷을 선택할 수 있습니다.	2024년 3월 4일
AmazonEKSWorkerNodePolicy - 기존 정책에 대한 업데이트	Amazon EKS에서 EKS Pod Identity를 허용하는 새로운 권한을 추가했습니다. Amazon EKS Pod Identity 에이전트는 노드 역할을 사용합니다.	2023년 11월 26일
AmazonEFSCSIDriverPolicy 를 도입했습니다.	AWS에서는 AmazonEFS CSIDriverPolicy 를 도입했습니다.	2023년 7월 26일
AmazonEKSClusterPolicy 에 권한을 추가했습니다.	Amazon EKS에서 로드 밸런서를 생성하는 동안 서브넷 자동 검색 중에 AZ 세부 정보를 가져올 수 있는 <code>ec2:DescribeAvailabilityZones</code> 권한이 추가되었습니다.	2023년 2월 7일

변경 사항	설명	날짜
AmazonEBSCSIDriverPolicy 의 정책 조건을 업데이트했습니다.	StringLike 키 필드에 와 일드카드 문자가 있는 잘못된 정책 조건을 제거했습니다. 또한 in-tree 플러그인에서 생성된 볼륨을 EBS CSI 드라이버에서 삭제할 수 있는 새 조건 <code>ec2:ResourceTag/kubernetes.io/created-for/pvc/name: "*" </code> 를 <code>ec2:DeleteVolume</code> 에 추가했습니다.	2022년 11월 17일
AmazonEKSLocalOutpostServiceRolePolicy 에 권한이 추가되었습니다.	더 나은 사전 조건 검증 및 관리형 수명 주기 제어를 허용하기 위해 <code>ec2:DescribeVPCAttribute</code> , <code>ec2:GetConsoleOutput</code> 및 <code>ec2:DescribeSecret</code> 을 추가했습니다. 또한 Outposts의 컨트롤 플레인 Amazon EC2 인스턴스의 배치 제어를 지원하는 <code>ec2:RunInstances</code> 에 <code>ec2:DescribePlacementGroups</code> 및 <code>"arn:aws:ec2:*:*:placement-group/*" </code> 을 추가했습니다.	2022년 10월 24일

변경 사항	설명	날짜
AmazonEKSLocalOutpostClusterPolicy 의 Amazon Elastic Container Registry 권한을 업데이트합니다.	모든 리소스 부분에서 범위가 지정된 부분으로 <code>ecr:GetDownloadUrlForLayer</code> 작업을 이동했습니다. <code>arn:aws:ecr:*:*:repository/eks/</code> 리소스를 추가했습니다. <code>arn:aws:ecr:</code> 리소스를 제거했습니다. 이 리소스에는 추가한 <code>arn:aws:ecr:*:*:repository/eks/*</code> 리소스가 적용됩니다.	2022년 10월 20일
AmazonEKSLocalOutpostClusterPolicy 에 권한이 추가되었습니다.	클러스터 컨트롤 플레인 인스턴스가 일부 kubelet 인수를 업데이트할 수 있도록 <code>arn:aws:ecr:*:*:repository/kubelet-config-updater</code> Amazon Elastic Container Registry 리포지토리가 추가되었습니다.	2022년 8월 31일
AmazonEKSLocalOutpostClusterPolicy 가 도입되었습니다.	AWS에서는 AmazonEKSLocalOutpostClusterPolicy 를 도입했습니다.	2022년 8월 24일
AmazonEKSLocalOutpostServiceRolePolicy 가 도입되었습니다.	AWS에서는 AmazonEKSLocalOutpostServiceRolePolicy 를 도입했습니다.	2022년 8월 23일
AmazonEBSCSIDriverPolicy 를 도입했습니다.	AWS에서는 AmazonEBSCSIDriverPolicy 를 도입했습니다.	2022년 4월 4일

변경 사항	설명	날짜
AmazonEKSWorkerNodePolicy 에 권한을 추가했습니다.	인스턴스 수준 속성을 자동으로 검색할 수 있는 Amazon EKS 최적화 AMI를 사용 설정하도록 <code>ec2:DescribeInstanceTypes</code> 를 추가했습니다.	2022년 3월 21일
AWSServiceRoleForAmazonEKSNodegroup 에 권한을 추가했습니다.	Amazon EKS가 지표 수집을 사용할 수 있도록 <code>autoscaling:EnableMetricsCollection</code> 권한이 추가되었습니다.	2021년 12월 13일
AmazonEKSClusterPolicy 에 권한을 추가했습니다.	Amazon EKS가 Network Load Balancer에 대한 서비스 연결 역할을 생성하도록 허용하는 <code>ec2:DescribeAccountAttributes</code> , <code>ec2:DescribeAddresses</code> 및 <code>ec2:DescribeInternetGateways</code> 권한이 추가됨.	2021년 6월 17일
Amazon EKS가 변경 사항 추적 시작.	Amazon EKS가 AWS 관리형 정책에 대한 변경 내용 추적을 시작했습니다.	2021년 6월 17일

IAM 문제 해결

이 주제에서는 IAM과 함께 Amazon EKS를 사용하는 동안 발생할 수 있는 몇 가지 일반적 오류와 이를 해결하는 방법을 다룹니다.

AccessDeniedException

AWS API 작업을 직접 호출할 때 AccessDeniedException이 발생하면 사용하고 있는 [IAM 보안 주체](#) 자격 증명에 해당 직접 호출을 수행할 필수 권한이 없는 것입니다.

```
An error occurred (AccessDeniedException) when calling the DescribeCluster operation:
User: arn:aws:iam::111122223333:user/user_name is not authorized to perform:
eks:DescribeCluster on resource: arn:aws:eks:region:111122223333:cluster/my-cluster
```

이전 예제 메시지에서 사용자는 Amazon EKS DescribeCluster API 작업을 직접 호출할 권한이 없습니다. IAM 보안 주체에게 Amazon EKS 관리 권한을 부여하려면 [the section called “ID 기반 정책”](#) 섹션을 참조하세요.

IAM에 자세한 내용은 IAM 사용 설명서에서 [정책을 사용하여 액세스 제어](#)를 참조하세요.

컴퓨팅 탭에서 노드 또는 리소스 탭에서 아무것도 볼 수 없으며 AWS Management Console에서 오류가 발생합니다.

Your current user or role does not have access to Kubernetes objects on this EKS cluster라는 콘솔 오류 메시지가 나타날 수 있습니다. AWS Management Console를 함께 사용하는 [IAM 보안 주체](#) 사용자에게 필요한 권한이 있는지 확인하세요. 자세한 내용은 [the section called “필수 권한”](#) 섹션을 참조하세요.

aws-auth **ConfigMap**은 클러스터에 대한 액세스 권한을 부여하지 않습니다.

[AWS IAM Authenticator](#)는 ConfigMap에 사용된 역할 ARN의 경로를 허용하지 않습니다. 따라서 rolearn를 지정하기 전에 경로를 제거하세요. 예를 들면 `arn:aws:iam::111122223333:role/team/developers/eks-admin` 를 `arn:aws:iam::111122223333:role/eks-admin` 로 변경합니다.

iam:PassRole을 수행하도록 인증되지 않음

iam:PassRole 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 Amazon EKS에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신, 해당 서비스에 기존 역할을 전달할 수 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예제 오류는 marymajor라는 IAM 사용자가 콘솔을 사용하여 Amazon EKS에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: {arn-aws}iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

이 경우, Mary가 iam:PassRole 작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 AWS 계정 외부의 사용자가 내 Amazon EKS 리소스에 액세스하도록 허용하려고 합니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우, 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하세요.

- Amazon EKS에서 이러한 기능을 지원하는지 알아보려면 [the section called “Amazon EKS 및 IAM”](#) 섹션을 참조하세요.
- 소유하고 있는 AWS 계정의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 [자신이 소유한 다른 AWS 계정의 IAM 사용자에게 대한 액세스 권한 제공](#)을 참조하세요.
- 리소스에 대한 액세스 권한을 서드 파티 AWS 계정에 제공하는 방법을 알아보려면 IAM 사용 설명서의 [서드 파티](#)가 소유한 AWS 계정에 대한 액세스 제공을 참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 [외부에서 인증된 사용자에게 액세스 권한 제공\(ID 페더레이션\)](#)을 참조하세요.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 [IAM의 크로스 계정 리소스 액세스](#)를 참조하세요.

포드 컨테이너는 다음과 같은 오류가 발생합니다. **An error occurred (SignatureDoesNotMatch) when calling the GetCallerIdentity operation: Credential should be scoped to a valid region**

애플리케이션이 명시적으로 AWS STS 글로벌 엔드포인트(<https://sts.amazonaws.com>)를 요청하고 Kubernetes 서비스 계정이 리전별 엔드포인트를 사용하도록 구성된 경우 컨테이너에 이 오류가 표시됩니다. 다음 옵션 중 하나를 사용하여 문제를 해결할 수 있습니다.

- 애플리케이션 코드를 업데이트하여 AWS STS 전역 엔드포인트에 대한 명시적 직접 호출을 제거합니다.
- 애플리케이션 코드를 업데이트하여 <https://sts.us-west-2.amazonaws.com> 와 같은 리전별 엔드포인트를 명시적으로 직접 호출합니다. 애플리케이션에는 해당 AWS 리전에서 서비스 장애가 발생한 경우 다른 AWS 리전을 선택하기 위해 기본 제공되는 중복성이 있어야 합니다. 자세한 내용은 IAM 사용 설명서의 [AWS 리전에서 AWS STS 관리](#)를 참조하세요.
- 전역 엔드포인트를 사용하도록 서비스 계정을 구성합니다. 클러스터는 기본적으로 리전 엔드포인트를 사용합니다. 자세한 내용은 [the section called “STS 엔드포인트”](#) 섹션을 참조하세요.

Amazon EKS 클러스터 IAM 역할

각 클러스터에는 Amazon EKS 클러스터 IAM 역할이 필요합니다. Amazon EKS에서 관리하는 Kubernetes 클러스터는 이 역할을 사용하여 노드를 관리하고, [레거시 클라우드 제공업체](#)는 이 역할을 사용하여 서비스용 Elastic Load Balancing으로 로드 밸런서를 생성합니다.

Amazon EKS 클러스터를 생성하기 전에 다음 IAM 정책 중 하나를 사용하여 IAM 역할을 생성해야 합니다.

- [AmazonEKSClusterPolicy](#)
- 사용자 지정 IAM 정책. 다음 최소 권한으로는 Kubernetes 클러스터가 노드를 관리할 수 있지만, [레거시 클라우드 제공업체](#)가 Elastic Load Balancing으로 로드 밸런서를 생성하는 것은 허용되지 않습니다. 사용자 지정 IAM 정책에는 적어도 다음과 같은 권한이 있어야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateTags"
      ]
    }
  ]
}
```



```

    ],
    "Resource": "arn:aws:ec2:*:*:instance/*",
    "Condition": {
      "ForAnyValue:StringLike": {
        "aws:TagKeys": "kubernetes.io/cluster/*"
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "ec2:DescribeInstances",
      "ec2:DescribeNetworkInterfaces",
      "ec2:DescribeVpcs",
      "ec2:DescribeDhcpOptions",
      "ec2:DescribeAvailabilityZones",
      "ec2:DescribeInstanceTopology",
      "kms:DescribeKey"
    ],
    "Resource": "*"
  }
]
}

```

Note

2023년 10월 3일 이전에는 각 클러스터의 IAM 역할에 [AmazonEKSClusterPolicy](#)이 필요했습니다.

2020년 4월 16일 이전에는 [AmazonEKSServicePolicy](#) 및 [AmazonEKSClusterPolicy](#)도 필요했으며 제안된 이름은 eksServiceRole이었습니다. AWSServiceRoleForAmazonEKS 서비스 연결 역할을 사용하면 2020년 4월 16일 이후에 생성된 클러스터에 [AmazonEKSServicePolicy](#) 정책이 더 이상 필요하지 않습니다.

기존 클러스터 역할 확인

다음 절차를 사용하여 계정에 이미 Amazon EKS 클러스터 역할이 있는지 확인할 수 있습니다.

1. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 역할을 선택합니다.

3. 역할 목록에서 eksClusterRole(을)를 검색합니다. eksClusterRole을 포함하는 역할이 존재하지 않을 경우 [the section called “Amazon EKS 클러스터 역할 생성”](#) 섹션을 참조하여 역할을 생성합니다. eksClusterRole을 포함하는 역할이 존재하지 않을 경우, 연결된 정책을 볼 역할을 선택합니다.
4. 권한을 선택합니다.
5. AmazonEKSClusterPolicy 관리형 정책이 역할에 연결되었는지 확인합니다. 정책이 연결된 경우 Amazon EKS 클러스터 역할이 적절히 구성된 것입니다.
6. 신뢰 관계를 선택한 후 신뢰 정책 편집을 선택합니다.
7. 신뢰 관계에 다음 정책이 포함되어 있는지 확인합니다. 신뢰 관계가 다음 정책과 일치하는 경우, 취소를 선택합니다. 신뢰 관계가 일치하지 않으면 정책을 신뢰 정책 편집 창에 복사하고 정책 업데이트를 선택합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Amazon EKS 클러스터 역할 생성

클러스터 역할을 생성하려면 AWS Management Console 또는 AWS CLI를 사용할 수 있습니다.

AWS Management Console

- a. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
- b. 역할을 선택한 다음 역할 생성을 선택합니다.
- c. 신뢰할 수 있는 엔터티 유형에서 AWS서비스를 선택합니다.
- d. 다른 AWS 서비스의 사용 사례 드롭다운 목록에서 EKS를 선택합니다.
- e. 사용 사례에 대한 EKS - 클러스터(EKS - Cluster)를 선택한 후 다음을 선택합니다.
- f. 권한 추가 탭에서 다음을 선택합니다.

- g. 역할 이름에 역할의 고유한 이름(예: eksClusterRole)을 입력합니다.
- h. 설명에서 Amazon EKS - Cluster role과 같은 설명 텍스트를 입력합니다.
- i. 역할 생성을 선택합니다.

AWS CLI

- a. 다음 콘텐츠를 *cluster-trust-policy.json*라는 파일에 복사합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

- b. 역할을 생성합니다. *eksClusterRole*을 선택한 모든 이름으로 바꿀 수 있습니다.

```
aws iam create-role \
  --role-name eksClusterRole \
  --assume-role-policy-document file://cluster-trust-policy.json
```

- c. 필요한 IAM 정책을 역할에 연결합니다.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy \
  --role-name eksClusterRole
```

Amazon EKS 노드 IAM 역할

Amazon EKS 노드 kubelet 데몬은 사용자를 대신하여 AWS API를 호출합니다. 노드는 IAM 인스턴스 프로필 및 연결 정책을 통해 이 API 호출에 대한 권한을 수신합니다. 노드를 시작해 클러스터에 등록하려면 시작할 때 노드에서 사용할 IAM 역할을 생성해야 합니다. 이 요구 사항은 Amazon이 제공하는 Amazon EKS 최적화 AMI 또는 사용하려는 다른 노드 AMI를 사용하여 시작된 노드에 적용됩니다. 또한 이 요구 사항은 관리형 노드 그룹과 자체 관리형 노드 모두에 적용됩니다.

Note

클러스터를 생성하는 데 사용된 것과 동일한 역할을 사용할 수 없습니다.

노드를 생성하려면 먼저 다음 권한을 사용하여 IAM 역할을 생성해야 합니다.

- [AmazonEKSWorkerNodePolicy](#) 정책에서 제공하는 것과 같이 VPC의 Amazon EC2 리소스를 설명할 수 있는 `kubelet` 권한. 이 정책은 Amazon EKS Pod Identity 에이전트에 대한 권한도 제공합니다.
- Amazon Elastic Container Registry(Amazon ECR)의 컨테이너 이미지를 사용할 수 있는 `kubelet` 권한([AmazonEC2ContainerRegistryPullOnly](#) 정책에서 제공하는 권한). 네트워킹용 내장 애드온이 Amazon ECR의 컨테이너 이미지를 사용하는 포드를 실행하기 때문에 Amazon Elastic Container Registry(Amazon ECR)의 컨테이너 이미지를 사용할 수 있는 권한이 필요합니다.
- (선택사항) Amazon EKS Pod Identity 에이전트에서 `eks-auth:AssumeRoleForPodIdentity` 작업을 사용하여 포드에 대한 보안 인증 정보를 검색할 수 있는 권한.
[AmazonEKSWorkerNodePolicy](#)를 사용하지 않는 경우 EKS Pod Identity를 사용할 수 있는 EC2 권한 외에도 이 권한을 제공해야 합니다.
- (선택 사항) IRSA 또는 EKS Pod Identity를 사용하여 VPC CNI 포드에 권한을 부여하지 않는 경우 인스턴스 역할에서 VPC CNI에 대한 권한을 제공해야 합니다. [AmazonEKS_CNI_Policy](#) 관리형 정책(IPv4 제품군으로 클러스터를 생성하는 경우) 또는 [사용자가 생성한 IPv6 정책](#)(IPv6 제품군으로 클러스터를 생성하는 경우)을 사용할 수 있습니다. 그러나 이 역할에 정책을 연결하는 대신 Amazon VPC CNI 추가 기능에 특별히 사용되는 별도의 역할에 정책을 연결하는 것이 좋습니다. Amazon VPC CNI 추가 기능에 대한 별도의 역할 생성에 대한 자세한 내용은 [the section called “IRSA에 대한 구성”](#) 섹션을 참조하세요.

Note

Amazon EC2 노드 그룹에는 Fargate 프로필과 다른 IAM 역할이 있어야 합니다. 자세한 내용은 [the section called “포드 실행 IAM 역할”](#) 섹션을 참조하세요.

기존 노드 역할 확인

다음 절차를 사용하여 계정에 이미 Amazon EKS 노드 역할이 있는지 확인할 수 있습니다.

1. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.

2. 왼쪽 탐색 창에서 역할을 선택합니다.
3. 역할 목록에서 eksNodeRole, AmazonEKSNodeRole 또는 NodeInstanceRole을 검색합니다. 이러한 이름 중 하나를 가진 역할이 없으면 [the section called “Amazon EKS 노드 IAM 역할 생성”](#) 섹션을 참조하여 역할을 만듭니다. eksNodeRole, AmazonEKSNodeRole 또는 NodeInstanceRole을 포함하는 역할이 존재하는 경우 역할을 선택하여 연결된 정책을 확인합니다.
4. 권한을 선택합니다.
5. AmazonEKSWorkerNodePolicy 및 AmazonEC2ContainerRegistryPullOnly 관리형 정책이 역할에 연결되어 있는지 또는 사용자 지정 정책이 최소 권한으로 연결되어 있는지 확인합니다.

 Note

AmazonEKS_CNI_Policy 정책이 역할에 연결되어 있는 경우 해당 역할을 제거하고 대신 aws-node Kubernetes 서비스 계정에 매핑된 IAM 역할에 연결하는 것이 좋습니다. 자세한 내용은 [the section called “IRSA에 대한 구성”](#) 단원을 참조하십시오.

6. 신뢰 관계를 선택한 후 신뢰 정책 편집을 선택합니다.
7. 신뢰 관계에 다음 정책이 포함되어 있는지 확인합니다. 신뢰 관계가 다음 정책과 일치하는 경우, 최소를 선택합니다. 신뢰 관계가 일치하지 않으면 정책을 신뢰 정책 편집 창에 복사하고 정책 업데이트를 선택합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Principal": {
        "Service": [
          "ec2.amazonaws.com"
        ]
      }
    }
  ]
}
```

Amazon EKS 노드 IAM 역할 생성

AWS Management Console 또는 AWS CLI를 사용하여 노드 IAM 역할을 생성할 수 있습니다.

AWS Management Console

- a. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
- b. 왼쪽 탐색 창에서 역할을 선택합니다.
- c. 역할 페이지에서 역할 생성을 선택합니다.
- d. 신뢰할 수 있는 엔터티 선택 페이지에서 다음을 수행합니다.
 - i. 신뢰할 수 있는 엔터티 유형 섹션에서 AWS 서비스를 선택합니다.
 - ii. 사용 사례에서 EC2를 선택합니다.
 - iii. 다음을 선택합니다.
- e. 권한 추가 페이지에서 사용자 지정 정책을 연결하거나 다음과 같이 합니다.
 - i. 필터 정책 상자에 AmazonEKSWorkerNodePolicy를 입력합니다.
 - ii. 검색 결과의 AmazonEKSWorkerNodePolicy 왼쪽에 있는 확인란을 선택합니다.
 - iii. 필터 지우기를 선택합니다.
 - iv. 필터 정책 상자에 AmazonEC2ContainerRegistryPullOnly를 입력합니다.
 - v. 검색 결과의 AmazonEC2ContainerRegistryPullOnly 왼쪽에 있는 확인란을 선택합니다.

생성하는 AmazonEKS_CNI_Policy 관리형 정책이나 [IPv6 정책](#)은 또한 이 역할이나 aws-node Kubernetes 서비스 계정에 매핑된 다른 역할에 연결되어야 합니다. 이 역할에 정책을 할당하는 대신 Kubernetes 서비스 계정에 연결된 역할에 정책을 할당하는 것이 좋습니다. 자세한 내용은 [the section called “IRSA에 대한 구성”](#) 단원을 참조하십시오.

- vi. 다음을 선택합니다.
- f. 이름, 검토 및 생성 페이지에서 다음을 수행합니다.
 - i. 역할 이름에 역할의 고유한 이름(예: AmazonEKSNodeRole)을 입력합니다.
 - ii. 설명에서 현재 텍스트를 설명이 포함된 텍스트(예: Amazon EKS - Node role)로 바꿉니다.
 - iii. 태그 추가(선택사항)에서 태그를 카-값 페어로 연결하여 메타데이터를 역할에 추가합니다. IAM에서 태그 사용에 대한 자세한 내용을 알아보려면 IAM 사용 설명서의 [IAM 리소스에 태그](#)를 참조하세요.
 - iv. 역할 생성을 선택합니다.

AWS CLI

- a. 다음 명령을 실행해 node-role-trust-relationship.json 파일을 생성합니다.

```
cat >node-role-trust-relationship.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Principal": {
        "Service": [
          "ec2.amazonaws.com"
        ]
      }
    }
  ]
}
EOF
```

- b. IAM 역할을 생성합니다.

```
aws iam create-role \
  --role-name AmazonEKSNodeRole \
  --assume-role-policy-document file://"node-role-trust-relationship.json"
```

- c. 필요한 2개의 관리형 IAM 정책을 IAM 역할에 연결합니다.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy \
  --role-name AmazonEKSNodeRole
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly \
  --role-name AmazonEKSNodeRole
```

- d. 클러스터를 생성한 IP 패밀리에 따라 다음 IAM 정책 중 하나를 IAM 역할에 연결합니다. 이 정책은 이 역할에 연결되거나, Kubernetes용 Amazon VPC CNI 플러그인에 사용되는 Kubernetes aws-node 서비스 계정에 연결된 역할에 연결되어야 합니다. Kubernetes 서비스 계정에 연결된 역할에 정책을 할당하는 것이 좋습니다. Kubernetes 서비스 계정에 연결된 역할에 정책을 할당하려면 [the section called “IRSA에 대한 구성”](#) 섹션을 참조하세요.

- IPv4

```
aws iam attach-role-policy \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy \  
  --role-name AmazonEKSTaskRole
```

- IPv6

A. 다음 텍스트를 복사해 vpc-cni-ipv6-policy.json 파일에 저장합니다.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "ec2:AssignIpv6Addresses",  
        "ec2:DescribeInstances",  
        "ec2:DescribeTags",  
        "ec2:DescribeNetworkInterfaces",  
        "ec2:DescribeInstanceTypes"  
      ],  
      "Resource": "*"   
    },  
    {  
      "Effect": "Allow",  
      "Action": [  
        "ec2:CreateTags"  
      ],  
      "Resource": [  
        "arn:aws:ec2:*:*:network-interface/*"  
      ]  
    }  
  ]  
}
```

B. IAM 정책을 생성합니다.

```
aws iam create-policy --policy-name AmazonEKS_CNI_IPv6_Policy --policy-  
document file://vpc-cni-ipv6-policy.json
```

C. IAM 정책을 IAM 역할에 연결합니다. **111122223333**을 계정 ID로 바꿉니다.


```
aws iam attach-role-policy \  
  --policy-arn arn:aws:iam::111122223333:policy/AmazonEKS_CNI_IPv6_Policy \  
  --role-name AmazonEKSNodeRole
```

Amazon EKS Auto Mode 클러스터 IAM 역할

클러스터마다 Amazon EKS 클러스터 IAM 역할이 필요합니다. Amazon EKS에서 관리하는 Kubernetes 클러스터는 이 역할을 사용하여 스토리지, 네트워킹, 컴퓨팅 오토 스케일링에 대한 일상적인 태스크를 자동화합니다.

Amazon EKS 클러스터를 생성하려면 먼저 EKS Auto Mode에 필요한 정책을 사용하여 IAM 역할을 만들어야 합니다. 제안된 AWS IAM 관리형 정책을 연결하거나 동등한 권한을 보유한 사용자 지정 정책을 생성할 수 있습니다.

- [AmazonEKSClusterPolicy](#)
- [AmazonEKSClusterPolicy](#)
- [AmazonEKSLoadBalancingPolicy](#)
- [AmazonEKSNetworkingPolicy](#)
- [AmazonEKSClusterPolicy](#)

기존 클러스터 역할 확인

다음 절차를 사용하여 계정에 이미 Amazon EKS 클러스터 역할이 있는지 확인할 수 있습니다.

1. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 역할을 선택합니다.
3. 역할 목록에서 AmazonEKSAutoClusterRole(을)를 검색합니다.
AmazonEKSAutoClusterRole을 포함하는 역할이 존재하지 않을 경우 다음 섹션의 지침을 참조하여 역할을 생성합니다. AmazonEKSAutoClusterRole을 포함하는 역할이 존재하지 않을 경우, 연결된 정책을 볼 역할을 선택합니다.
4. 권한을 선택합니다.
5. AmazonEKSClusterPolicy 관리형 정책이 역할에 연결되었는지 확인합니다. 정책이 연결된 경우 Amazon EKS 클러스터 역할이 적절히 구성된 것입니다.
6. 신뢰 관계를 선택한 후 신뢰 정책 편집을 선택합니다.

7. 신뢰 관계에 다음 정책이 포함되어 있는지 확인합니다. 신뢰 관계가 다음 정책과 일치하는 경우, 추소를 선택합니다. 신뢰 관계가 일치하지 않으면 정책을 신뢰 정책 편집 창에 복사하고 정책 업데이트를 선택합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

Note

AWS에는 이 역할의 이름 AmazonEKSAutoClusterRole이 필요하지 않습니다.

Amazon EKS 클러스터 역할 생성

클러스터 역할을 생성하려면 AWS Management Console 또는 AWS CLI를 사용할 수 있습니다.

AWS Management Console

1. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 역할을 선택한 다음 역할 생성을 선택합니다.
3. 신뢰할 수 있는 엔터티 유형에서 AWS서비스를 선택합니다.
4. 다른 AWS 서비스의 사용 사례 드롭다운 목록에서 EKS를 선택합니다.
5. 사용 사례에 대한 EKS - 클러스터(EKS - Cluster)를 선택한 후 다음을 선택합니다.
6. 권한 추가 탭에서 정책을 선택하고 다음을 선택합니다.
 - [AmazonEKSCoordinatePolicy](#)

- [AmazonEKSBlockStoragePolicy](#)
- [AmazonEKSLoadBalancingPolicy](#)
- [AmazonEKSNetworkingPolicy](#)
- [AmazonEKSClusterPolicy](#)

7. 역할 이름에 역할의 고유한 이름(예: AmazonEKSAutoClusterRole)을 입력합니다.
8. 설명에서 Amazon EKS - Cluster role과 같은 설명 텍스트를 입력합니다.
9. 역할 생성을 선택합니다.

AWS CLI

1. 다음 콘텐츠를 *cluster-trust-policy.json*라는 파일에 복사합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "eks.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ]
    }
  ]
}
```

2. 역할을 생성합니다. *AmazonEKSAutoClusterRole*을 선택한 이름으로 바꿀 수 있습니다.

```
aws iam create-role \
  --role-name AmazonEKSAutoClusterRole \
  --assume-role-policy-document file:///cluster-trust-policy.json
```

3. 필요한 IAM 정책을 역할에 연결합니다.

AmazonEKSClusterPolicy:

```
aws iam attach-role-policy \
```

```
--role-name AmazonEKSAutoClusterRole \  
--policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
```

AmazonEKSClusterPolicy:

```
aws iam attach-role-policy \  
  --role-name AmazonEKSAutoClusterRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSClusterPolicy
```

AmazonEKSBlockStoragePolicy:

```
aws iam attach-role-policy \  
  --role-name AmazonEKSAutoClusterRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSBlockStoragePolicy
```

AmazonEKSLoadBalancingPolicy:

```
aws iam attach-role-policy \  
  --role-name AmazonEKSAutoClusterRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSLoadBalancingPolicy
```

AmazonEKSNetworkingPolicy:

```
aws iam attach-role-policy \  
  --role-name AmazonEKSAutoClusterRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSNetworkingPolicy
```

Amazon EKS Auto Mode 노드 IAM 역할

Note

클러스터를 생성하는 데 사용된 것과 동일한 역할을 사용할 수 없습니다.

노드를 생성하려면 먼저 다음 정책 또는 이와 동등한 권한을 사용하여 IAM 역할을 생성해야 합니다.

- [AmazonEKSWorkerNodeMinimalPolicy](#)
- [AmazonEC2ContainerRegistryPullOnly](#)

기존 노드 역할 확인

다음 절차를 사용하여 계정에 이미 Amazon EKS 노드 역할이 있는지 확인할 수 있습니다.

1. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 역할을 선택합니다.
3. 역할 목록에서 AmazonEKSAutoNodeRole(을)를 검색합니다. 이러한 이름 중 하나를 가진 역할이 없으면 다음 섹션의 지침을 참조하여 역할을 만듭니다. AmazonEKSAutoNodeRole을 포함하는 역할이 존재하지 않을 경우, 연결된 정책을 볼 역할을 선택합니다.
4. 권한을 선택합니다.
5. 위의 필수 정책 또는 이에 상응하는 사용자 지정 정책이 연결되어 있는지 확인합니다.
6. 신뢰 관계를 선택한 후 신뢰 정책 편집을 선택합니다.
7. 신뢰 관계에 다음 정책이 포함되어 있는지 확인합니다. 신뢰 관계가 다음 정책과 일치하는 경우, 취소를 선택합니다. 신뢰 관계가 일치하지 않으면 정책을 신뢰 정책 편집 창에 복사하고 정책 업데이트를 선택합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Amazon EKS 노드 IAM 역할 생성

AWS Management Console 또는 AWS CLI를 사용하여 노드 IAM 역할을 생성할 수 있습니다.

AWS Management Console

1. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 역할을 선택합니다.
3. 역할 페이지에서 역할 생성을 선택합니다.

4. 신뢰할 수 있는 엔터티 선택 페이지에서 다음을 수행합니다.
 - a. 신뢰할 수 있는 엔터티 유형 섹션에서 AWS 서비스를 선택합니다.
 - b. 사용 사례에서 EC2를 선택합니다.
 - c. Next(다음)를 선택합니다.
5. 권한 추가 페이지에서 다음 정책을 연결합니다.
 - [AmazonEKSWorkerNodeMinimalPolicy](#)
 - [AmazonEC2ContainerRegistryPullOnly](#)
6. 이름, 검토 및 생성 페이지에서 다음을 수행합니다.
 - a. 역할 이름에 역할의 고유한 이름(예: AmazonEKSAutoNodeRole)을 입력합니다.
 - b. 설명에서 현재 텍스트를 설명이 포함된 텍스트(예: Amazon EKS - Node role)로 바꿉니다.
 - c. 태그 추가(선택사항)에서 태그를 키-값 페어로 연결하여 메타데이터를 역할에 추가합니다. IAM에서 태그 사용에 대한 자세한 내용을 알아보려면 IAM 사용 설명서의 [IAM 리소스에 태깅](#)을 참조하세요.
 - d. 역할 생성을 선택합니다.

AWS CLI

노드 IAM 역할 생성

이전 단계의 node-trust-policy.json 파일을 사용하여 역할을 수임할 수 있는 엔터티를 정의합니다. 다음 명령을 실행하여 노드 IAM 역할을 만듭니다.

```
aws iam create-role \  
  --role-name AmazonEKSAutoNodeRole \  
  --assume-role-policy-document file://node-trust-policy.json
```

역할 ARN 기록

역할을 생성한 후 노드 IAM 역할의 ARN을 검색하고 저장합니다. 이후 단계에서 이 ARN이 필요합니다. 다음 명령을 사용하여 ARN을 가져옵니다.

```
aws iam get-role --role-name AmazonEKSAutoNodeRole --query "Role.Arn" --output text
```

필요한 정책 연결

다음 AWS 관리형 정책을 노드 IAM 역할에 연결하여 필요한 권한을 제공합니다.

AmazonEKSWorkerNodeMinimalPolicy를 연결하려면:

```
aws iam attach-role-policy \  
  --role-name AmazonEKSAutoNodeRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodeMinimalPolicy
```

AmazonEC2ContainerRegistryPullOnly를 연결하려면:

```
aws iam attach-role-policy \  
  --role-name AmazonEKSAutoNodeRole \  
  --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPullOnly
```

클러스터 성능 모니터링 및 로그 보기

사용 가능한 많은 모니터링 또는 로깅 도구를 사용하여 Amazon EKS에서 데이터를 관찰할 수 있습니다. 데이터 분석을 위해 Amazon EKS 로그 데이터를 AWS 서비스 또는 파트너 도구로 스트리밍할 수 있습니다. Amazon EKS 문제 해결을 위한 데이터를 제공하는 AWS Management Console에서 사용할 수 있는 서비스가 많습니다. [Amazon EKS 인프라 모니터링](#)을 위해 AWS 지원 오픈 소스 솔루션을 사용할 수도 있습니다.

Amazon EKS 콘솔의 왼쪽 탐색 창에서 클러스터를 선택한 후 클러스터 이름과 관찰성 탭을 선택하여 클러스터 상태와 세부 정보를 볼 수 있습니다. 클러스터에 배포된 기존의 모든 Kubernetes 리소스에 대한 세부 정보를 보려면 [the section called “클러스터 리소스 액세스”](#) 섹션을 참조하세요.

모니터링은 Amazon EKS와 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. AWS 솔루션의 모든 부분으로부터 모니터링 데이터를 수집하는 것이 좋습니다. 그렇게 하면 다중 지점 실패가 발생하는 경우 보다 쉽게 디버깅할 수 있습니다. Amazon EKS 모니터링을 시작하기 전에 모니터링 계획이 다음 문제를 해결할 수 있는지 확인합니다.

- 목표가 무엇인가요? 클러스터가 급격히 확장될 경우 실시간 알림이 필요하나요?
- 관찰해야 할 리소스는 무엇인가요?
- 이러한 리소스를 얼마나 자주 관찰해야 하나요? 귀사는 위험에 신속하게 대응하고 싶나요?
- 어떤 도구를 사용하고 싶나요? AWS Fargate를 시작의 일부로 이미 실행한 경우 기본 제공 [로그 라우터](#)를 사용할 수 있습니다.
- 모니터링 작업을 수행하려는 사용자는 누구인가요?
- 문제가 발생하면 누구에게 알림을 보내고 싶나요?

Amazon EKS의 모니터링 및 로깅

Amazon EKS는 모니터링 및 로깅을 위한 기본 도구를 제공합니다. 지원되는 버전의 경우 관찰성 대시보드를 통해 클러스터의 성능을 확인할 수 있습니다. 이를 통해 문제를 신속하게 감지 및 해결할 수 있습니다. 모니터링 기능 외에도 컨트롤 플레인 감사 로그를 기반으로 하는 목록이 포함됩니다. Kubernetes 컨트롤 플레인 콘솔 외부에서 스크레이핑할 수도 있는 여러 지표를 노출합니다.

컨트롤 플레인 로깅은 클러스터에 대한 모든 API 직접 호출, 사용자가 클러스터에 어떤 작업을 수행했는지 캡처하는 감사 정보 및 역할 기반 정보를 기록합니다. 자세한 내용은 AWS Prescriptive Guidance의 [Amazon EKS에서 로깅 및 모니터링](#)을 참조하세요.

Amazon EKS 컨트롤 플레인 로깅은 Amazon EKS 컨트롤 플레인에서 계정의 CloudWatch Logs로 감사 및 진단 로그를 직접 제공합니다. 이러한 로그를 통해 클러스터를 쉽게 보호하고 실행할 수 있습니다. 필요한 로그 유형을 정확하게 선택할 수 있으며, 로그는 CloudWatch의 각 Amazon EKS 클러스터에 대한 그룹에 로그 스트림으로 전송됩니다. 자세한 내용은 [the section called “컨트롤 플레인 로그”](#) 섹션을 참조하세요.

Note

Amazon CloudWatch에서 Amazon EKS 인증자 로그를 확인하면 다음 텍스트 예와 유사한 텍스트가 포함된 항목이 표시됩니다.

```
level=info msg="mapping IAM role" groups="[]"
  role="arn:aws:iam::111122223333:role/XXXXXXXXXXXXXXXXX-NodeManagerRole-XXXXXXXXX" username="eks:node-manager"
```

이 텍스트를 포함하는 항목이 필요합니다. username은 관리형 노드 그룹 및 Fargate에 대해 특정 작업을 수행하는 Amazon EKS 내부 서비스 역할입니다.

낮은 수준의 사용자 지정 가능한 로깅의 경우 [Kubernetes 로깅](#)을 사용할 수 있습니다.

Amazon EKS는 Amazon EKS에서 사용자, 역할 또는 AWS 서비스가 수행한 작업에 대한 레코드를 제공하는 서비스인 AWS CloudTrail과 통합됩니다. CloudTrail은 Amazon EKS에 대한 모든 API 직접 호출을 이벤트로 캡처합니다. 캡처되는 직접 호출에는 Amazon EKS 콘솔로부터의 직접 호출과 Amazon EKS API 작업에 대한 코드 직접 호출이 포함됩니다. 자세한 내용은 [the section called “AWS CloudTrail”](#) 섹션을 참조하세요.

Kubernetes API 서버는 모니터링 및 분석에 유용한 여러 가지 지표를 표시합니다. 자세한 내용은 [the section called “Prometheus 지표”](#) 섹션을 참조하세요.

사용자 지정 Amazon CloudWatch Logs에 대해 Fluent Bit를 구성하려면 Amazon CloudWatch 사용 설명서의 [Fluent Bit 설정](#)을 참조하세요.

Amazon EKS 모니터링 및 로깅 도구

Amazon Web Services는 Amazon EKS를 모니터링하는 데 사용할 수 있는 다양한 도구를 제공합니다. 일부 도구를 구성하여 자동 모니터링을 설정할 수 있지만 일부는 수동 직접 호출이 필요합니다. 사용 중인 환경 및 기존 도구 집합이 허용하는 만큼 모니터링 작업을 자동화하는 것이 좋습니다.

다음 테이블에서는 다양한 모니터링 도구 옵션을 설명합니다.

영역	도구	설명	설정
컨트롤 플레인	관찰성 대시보드	지원되는 버전의 경우 관찰성 대시보드를 통해 클러스터의 성능을 확인할 수 있습니다. 이를 통해 문제를 신속하게 감지 및 해결할 수 있습니다.	설정 절차
애플리케이션/컨트롤 플레인	Prometheus	Prometheus를 사용하여 애플리케이션과 컨트롤 플레인에 대한 지표와 경고를 모니터링할 수 있습니다.	설정 절차
Applications	CloudWatch Container Insights	CloudWatch Container Insights는 컨테이너 애플리케이션 및 마이크로서비스의 지표 및 로그를 수집하고 종합하며 요약합니다.	설정 절차
Applications	AWS Distro for OpenTelemetry(ADOT)	ADOT는 상호 연관된 지표, 추적 데이터, 메타데이터를 수집해서 AWS 모니터링 서비스 또는 파트너로 전송합니다. CloudWatch Container Insights를 통해 설정할 수 있습니다.	설정 절차
Applications	Amazon DevOps Guru	Amazon DevOps Guru는 노드 수준의	설정 절차

영역	도구	설명	설정
		운영 성능 및 가용성을 감지합니다.	
Applications	AWS X-Ray	AWS X-Ray는 애플리케이션에 대한 추적 데이터를 수신합니다. 이 추적 데이터에는 수신 및 발신 요청과 요청에 대한 메타데이터가 포함됩니다. Amazon EKS의 경우 구현하려면 OpenTelemetry 추가 기능이 필요합니다.	설정 절차
Applications	Amazon CloudWatch	CloudWatch는 지원되는 버전에서 몇 가지 기본 Amazon EKS 지표를 무료로 제공합니다. CloudWatch Observability Operator를 사용하여 이 기능을 확장하고 지표, 로그 및 추적 데이터의 수집을 처리할 수 있습니다.	설정 절차

다음 테이블에서는 다양한 로깅 도구 옵션을 설명합니다.

영역	도구	설명	설정
컨트롤 플레인	관찰성 대시보드	지원되는 버전의 경우 관찰성 대시보드에 컨트롤 플레인 감사 로그를 기반으로 한 목록이 표시됩니다. Amazon	설정 절차

영역	도구	설명	설정
		CloudWatch의 컨트롤 플레인 로그에 대한 링크도 포함되어 있습니다.	
Applications	Amazon CloudWatch Container Insights	Amazon CloudWatch Container Insights는 컨테이너 애플리케이션 및 마이크로서비스의 지표 및 로그를 수집하고, 종합하며, 요약합니다.	설정 절차
컨트롤 플레인	Amazon CloudWatch Logs	Amazon EKS 컨트롤 플레인에서 계정의 CloudWatch Logs로 감사 및 진단 로그를 직접 전송할 수 있습니다.	설정 절차
컨트롤 플레인	AWS CloudTrail	사용자, 역할 또는 서비스를 통한 API 직접 호출을 기록합니다.	설정 절차
AWS Fargate 인스턴스의 여러 영역	AWS Fargate 로그 라우터	AWS Fargate 인스턴스의 경우 로그 라우터가 AWS 서비스 또는 파트너 도구로 로그를 스트리밍합니다. Fluent Bit용 AWS 를 사용합니다. 로그를 다른 AWS 서비스 또는 파트너 도구로 스트리밍할 수 있습니다.	설정 절차

관찰성 대시보드를 사용하여 클러스터 모니터링

Amazon EKS 콘솔에는 클러스터의 성능에 대한 가시성을 제공하는 관찰성 대시보드가 포함되어 있습니다. 제공되는 정보는 문제를 신속하게 탐지하고 해결하는 데 도움이 됩니다. 상태 및 성능 요약에서 항목을 선택하여 관찰성 대시보드의 해당 섹션을 열 수 있습니다. 이 요약은 관찰성 탭을 포함하여 여러 곳에 포함되어 있습니다.

관찰성 대시보드는 여러 탭으로 나뉩니다.

요약

상태 및 성능 요약에는 다양한 범주의 항목 수량이 나열됩니다. 각 숫자는 해당 범주의 목록이 있는 관찰성 대시보드의 위치에 대한 하이퍼링크 역할을 합니다.

클러스터 상태

클러스터 상태는 알아야 할 중요한 알림을 제공하며, 그 중 일부는 가능한 한 빨리 조치가 필요할 수 있습니다. 이 목록을 사용하면 설명과 영향을 받는 리소스를 볼 수 있습니다. 클러스터 상태에는 상태 문제와 구성 인사이트라는 2개의 테이블이 포함됩니다. 상태 문제의 상태를 새로 고치려면 새로 고침 버튼(↻)을 선택합니다. 구성 인사이트는 24시간마다 한 번씩 자동으로 업데이트되며 수동으로 새로 고칠 수 없습니다.

상태 문제에 대한 자세한 내용은 [the section called “클러스터 상태 FAQ 및 오류 코드\(해결 경로 포함\)”](#) 섹션을 참조하세요. 구성 인사이트에 대한 자세한 내용은 [the section called “클러스터 인사이트”](#) 섹션을 참조하세요.

컨트롤 플레인 모니터링

컨트롤 플레인 모니터링 탭은 세 개의 섹션으로 나뉘며, 각 섹션은 클러스터의 컨트롤 플레인을 모니터링하고 문제를 해결하는 데 도움이 됩니다.

Metrics

Kubernetes 버전 1.28 이상인 클러스터의 경우 지표 섹션에는 다양한 컨트롤 플레인 구성 요소에 대해 수집된 여러 지표의 그래프가 표시됩니다.

섹션 상단에서 선택하여 모든 그래프의 X축에 사용되는 기간을 설정할 수 있습니다. 새로 고침 버튼(↻)을 사용하여 데이터를 새로 고칠 수 있습니다. 개별 그래프에 대해 세로 줄임표 버튼(:)은 CloudWatch의 옵션이 있는 메뉴를 엽니다.

이러한 지표 등은 CloudWatch의 AWS/EKS 네임스페이스에서 기본 모니터링 지표로 자동으로 사용 가능합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서에서 [기본 모니터링 및 세부 모니터링](#)을 참조하세요. 자세한 지표, 시각화, 인사이트는 Amazon CloudWatch 사용 설명서에서 [Container Insights](#)를 참조하세요. 또는 Prometheus 기반 모니터링을 선호하는 경우 [the section called “Prometheus 지표”](#) 섹션을 참조하세요.

다음 표에서는 이용 가능한 지표를 설명합니다.

지표	설명
APIServer 요청	API 서버에 대한 분당 요청 수입입니다.
APIServer 총 요청 4XX	HTTP 4XX 응답 코드(클라이언트 측 오류)가 있는 분당 API 서버 요청 수입입니다.
APIServer 총 요청 5XX	HTTP 5XX 응답 코드(서버 측 오류)가 있는 분당 API 서버 요청 수입입니다.
APIServer 총 요청 429	HTTP 429 응답 코드(요청이 너무 많음)가 있는 분당 API 서버 요청 수입입니다.
스토리지 크기	스토리지 데이터베이스(etcd) 크기입니다.
스케줄러 시도	'예약 불가' '오류', '예약됨' 결과를 기준으로 포드를 예약하려는 시도 횟수입니다.
보류 중인 포드	'활성', '백오프', '예약 불가', '제한적'의 대기 유형별로 구분한 대기 중인 포드 수입입니다.
API 서버 요청 지연 시간	API 서버 요청의 지연 시간입니다.
API 서버 현재 진행 중인 요청	API 서버에 대한 현재 전송 중 요청입니다.
웹훅 요청	분당 웹훅 요청 수입입니다.
웹훅 요청 거부	거부된 웹훅 요청 수입입니다.
웹훅 요청 지연 시간 P99	외부 타사 웹훅 요청의 99번째 백분위수 지연 시간입니다.

CloudWatch Log Insights

CloudWatch Log Insights 섹션에는 컨트롤 플레인 감사 로그를 기반으로 하는 다양한 목록이 표시됩니다. 이 기능을 사용하려면 CloudWatch의 컨트롤 플레인 로그 보기 섹션에서 Amazon EKS 컨트롤 플레인 로그를 활성화해야 합니다.

데이터를 수집하는 데 충분한 시간이 경과하면 모든 쿼리를 실행하거나 한 번에 단일 목록에 대한 쿼리 실행을 선택할 수 있습니다. 쿼리를 실행할 때마다 CloudWatch에서 추가 비용이 발생합니다. 섹션 상단에서 확인할 결과의 기간을 선택합니다. 쿼리에 대한 고급 제어를 원하는 경우 CloudWatch에서 보기를 선택할 수 있습니다. 이렇게 하면 필요에 따라 CloudWatch에서 쿼리를 업데이트할 수 있습니다.

자세한 내용은 Amazon CloudWatch Logs 사용 설명서에서 [CloudWatch Logs Insights를 사용한 로그 데이터 분석](#)을 참조하세요.

CloudWatch에서 컨트롤 플레인 로그 확인

로깅 관리를 선택하여 사용 가능한 로그 유형을 업데이트합니다. 로깅을 활성화한 후 CloudWatch Logs에 로그가 표시되는 데 몇 분 정도 걸립니다. 충분한 시간이 지나면 섹션의 보기 링크 중 하나를 선택하여 해당 로그로 이동합니다.

자세한 내용은 [the section called “컨트롤 플레인 로그”](#) 섹션을 참조하세요.

클러스터 인사이트

업그레이드 인사이트 표는 문제를 모두 표시하고 수정 조치를 권장하며, 새 Kubernetes 버전으로 업그레이드하기 위한 검증 프로세스를 가속화합니다. Amazon EKS는 문제에 영향을 미치는 가능한 Kubernetes 버전 업그레이드 목록을 대상으로 클러스터를 자동으로 스캔합니다. 업그레이드 인사이트 표에는 이 클러스터에 대해 Amazon EKS에서 수행한 인사이트 확인과 관련 상태가 나열됩니다.

Amazon EKS는 Kubernetes 프로젝트의 변경 사항 및 새 버전과 연결된 Amazon EKS 서비스 변경 사항에 대한 평가를 기반으로 수행할 인사이트 검사 목록을 유지 관리하고 주기적으로 새로 고칩니다. Amazon EKS 콘솔은 마지막 새로 고침 시간 열에서 볼 수 있는 각 인사이트의 상태를 자동으로 새로 고칩니다.

자세한 내용은 [the section called “클러스터 인사이트”](#) 섹션을 참조하세요.

노드 상태 문제

Amazon EKS 노드 모니터링 에이전트는 노드 로그를 자동으로 읽어 상태 문제를 감지합니다. 자동 복구 설정과 관계없이 필요에 따라 조사할 수 있도록 모든 노드 상태 문제가 보고됩니다. 문제 유형이 설명 없이 나열된 경우 해당 팝오버 요소에서 설명을 읽을 수 있습니다.

페이지를 새로 고치면 해결된 문제가 목록에서 사라집니다. 자동 복구가 활성화된 경우 일부 상태 문제가 일시적으로 표시될 수 있으며, 이 문제는 사용자의 조치 없이 해결됩니다. 자동 복구로 지원되지 않는 문제는 유형에 따라 수동 작업이 필요할 수 있습니다.

노드 상태 문제를 보고하려면 클러스터가 Amazon EKS Auto Mode를 사용하거나 노드 모니터링 에이전트 추가 기능이 있어야 합니다. 자세한 내용은 [the section called “노드 상태”](#) 섹션을 참조하세요.

Prometheus를 사용한 클러스터 지표 모니터링

[Prometheus](#)는 엔드포인트를 스크레이프하는 모니터링 및 시계열 데이터베이스입니다. 수집된 데이터를 쿼리, 집계 및 저장하는 기능을 제공합니다. 알림 및 알림 집계에도 사용할 수 있습니다. 이 주제에서는 관리형 또는 오픈 소스 옵션으로 Prometheus를 설정하는 방법을 설명합니다. Amazon EKS 컨트롤 플레인 지표 모니터링은 일반적인 사용 사례입니다.

Amazon Managed Service for Prometheus는 컨테이너화된 애플리케이션과 인프라를 대규모로 쉽게 모니터링할 수 있는 Prometheus 호환 모니터링 및 알림 서비스입니다. 지표의 수집, 스토리지, 쿼리 및 알림을 자동으로 확장하는 완전관리형 서비스입니다. 또한 AWS 보안 서비스와 통합되어 데이터에 빠르고 안전하게 액세스할 수 있습니다. 오픈 소스 PromQL 쿼리 언어를 사용하여 지표를 쿼리하고 지표에 대해 알릴 수 있습니다. 또한 Prometheus용 Amazon 관리형 서비스의 알림 관리자를 사용하여 중요한 알림에 대한 알림 규칙을 설정할 수 있습니다. 그런 다음 이러한 중요한 알림을 Amazon SNS 주제에 대한 알림으로 보낼 수 있습니다.

Amazon EKS에서 Prometheus를 사용하는 데는 여러 가지 옵션이 있습니다.

- Amazon EKS 클러스터를 처음 생성할 때 Prometheus 지표를 켜거나 기존 클러스터에 대해 자체 Prometheus 스크레이퍼를 생성할 수 있습니다. 이 주제에서는 이 두 가지 옵션을 모두 다룹니다.
- Helm을 사용하여 Prometheus를 배포할 수 있습니다. 자세한 내용은 [the section called “Helm을 사용하여 배포”](#) 섹션을 참조하세요.
- 컨트롤 플레인 원시 지표를 Prometheus 형식으로 볼 수 있습니다. 자세한 내용은 [the section called “컨트롤 플레인”](#) 섹션을 참조하세요.

1단계: Prometheus 지표 켜기

Important

Prometheus용 Amazon 관리형 서비스 리소스는 클러스터 수명 주기를 벗어나며 클러스터에 독립적으로 유지 관리되어야 합니다. 클러스터를 삭제할 때 해당 스크레이퍼도 모두 삭제하여

관련 비용을 줄여야 합니다. 자세한 내용은 Prometheus용 Amazon 관리형 서비스 사용자 가이드의 [스크레이퍼 찾기 및 삭제](#)를 참조하세요.

Prometheus는 스크레이핑이라는 풀 기반 모델을 통해 클러스터에서 지표를 검색하고 수집합니다. Scraper는 클러스터 인프라 및 컨테이너화된 애플리케이션에서 데이터를 수집하도록 설정됩니다. Prometheus 지표 전송 옵션을 켜면 Amazon Managed Service for Prometheus에서 에이전트 없는 완전 관리형 스크레이퍼를 제공합니다.

아직 클러스터를 생성하지 않은 경우 클러스터를 처음 생성할 때 Prometheus에 지표를 보내는 옵션을 켤 수 있습니다. Amazon EKS 콘솔에서 이 옵션은 새 클러스터를 생성하는 통합 관찰성 구성 단계에 있습니다. 자세한 내용은 [the section called “클러스터 생성”](#) 섹션을 참조하세요.

이미 기존 클러스터가 있는 경우 자체 Prometheus 스크레이퍼를 생성할 수 있습니다. Amazon EKS 콘솔에서 이렇게 하려면 클러스터의 관찰성 탭으로 이동하여 Scraper 추가 버튼을 선택합니다. AWS API 또는 AWS CLI로 Prometheus 스크레이퍼를 생성하려면 Amazon Managed Service for Prometheus User Guide의 [Create a scraper](#)를 참조하세요.

Amazon EKS 콘솔을 사용하여 Scraper를 생성할 때 다음 옵션을 사용할 수 있습니다.

Scraper 별칭

(선택사항) Scraper의 고유한 별칭을 입력합니다.

대상

Amazon Managed Service for Prometheus 작업 영역을 선택합니다. 워크스페이스는 Prometheus 지표 보관 및 쿼리를 위한 전용 논리 공간입니다. 이 작업 영역을 사용하면 액세스 권한이 있는 계정에서 Prometheus 지표를 볼 수 있습니다. 새 작업 영역 생성 옵션은 Amazon EKS에 제공한 작업 영역 별칭을 사용하여 대신 작업 영역을 생성하도록 지시합니다. 기존 작업 영역 선택 옵션을 사용하면 드롭다운 목록에서 기존 작업 영역을 선택할 수 있습니다. 작업 영역에 대한 자세한 내용은 Prometheus용 Amazon 관리형 서비스 사용 설명서의 [작업 영역 관리](#)를 참조하세요.

서비스 액세스

이 섹션에는 Prometheus 지표를 전송할 때 부여하는 권한이 요약되어 있습니다.

- Prometheus용 Amazon 관리형 서비스에 스크레이프된 Amazon EKS 클러스터 설명 허용
- Amazon Managed Prometheus 작업 영역에 원격 쓰기 허용

AmazonManagedScraperRole이 이미 있는 경우 Scraper가 이를 사용합니다.

AmazonManagedScraperRole 링크를 선택하여 권한 세부 정보를 확인합니다.

AmazonManagedScraperRole이 아직 없는 경우 권한 세부 정보 보기 링크를 선택하여 Prometheus 지표를 전송하여 부여하는 특정 권한을 확인합니다.

서브넷

필요에 따라 Scraper가 상속할 서브넷을 수정합니다. 회색으로 표시된 서브넷 옵션을 추가해야 하는 경우 클러스터 만들기 네트워킹 지정 단계로 돌아가세요.

Scraper 구성

필요에 따라 YAML 형식으로 Scraper 구성을 수정합니다. 이렇게 하려면 양식을 사용하거나 대체 YAML 파일을 업로드합니다. 자세한 내용은 Prometheus용 Amazon 관리형 서비스 사용 설명서의 [Scraper 구성](#)을 참조하세요.

Prometheus용 Amazon 관리형 서비스는 클러스터와 함께 AWS 관리형 컬렉터로 생성되는 에이전트 없는 Scraper를 말합니다. AWS 관리형 컬렉터에 대한 자세한 내용은 Prometheus용 Amazon Managed Service 사용 설명서의 [AWS 관리형 컬렉터](#)를 참조하세요.

Important

- AWS CLI 또는 AWS API를 사용하여 Prometheus 스크레이퍼를 생성하는 경우 스크레이퍼에 클러스터 내 권한을 부여하도록 구성을 조정해야 합니다. 자세한 내용은 Prometheus용 Amazon 관리형 서비스 사용 설명서의 [Amazon EKS 클러스터 구성](#)을 참조하세요.
- 액세스 항목 대신 aws-auth ConfigMap을 사용하는 2024년 11월 11일 이전에 생성된 Prometheus 스크레이퍼가 있는 경우 Amazon EKS 클러스터 컨트롤 플레인에서 추가 지표에 액세스하려면 스크레이퍼를 업데이트해야 합니다. 업데이트된 구성은 Amazon Managed Service for Prometheus 사용 설명서에서 [스크레이퍼 액세스를 위한 Amazon EKS 수동 구성](#)을 참조하세요.

2단계: Prometheus 지표 사용

클러스터에 대해 Prometheus 지표를 컨 후 사용하는 방법에 대한 자세한 내용은 [Amazon Managed Service for Prometheus User Guide](#)를 참조하세요.

3단계: Prometheus 스크레이퍼 관리

Scraper를 관리하려면 Amazon EKS 콘솔에서 관찰성 탭을 선택합니다. 테이블에는 Scraper ID, 별칭, 상태 및 생성 날짜와 같은 정보를 포함하여 클러스터의 Scraper 목록이 표시됩니다. 스크레이퍼를 더

추가하거나, 스크레이퍼를 편집하거나, 스크레이퍼를 삭제하거나, 현재 스크레이퍼에 대한 자세한 정보를 볼 수 있습니다.

Scraper에 관한 추가 세부 정보를 확인하려면 Scraper ID 링크를 선택합니다. 예를 들어, ARN, 환경, 작업 영역 ID, IAM 역할, 구성 및 네트워킹 정보를 볼 수 있습니다. 스크레이퍼 ID를 [DescribeScraper](#), [UpdateScraper](#), [DeleteScraper](#) 등의 Prometheus용 Amazon 관리형 서비스 API 작업의 입력으로 사용할 수 있습니다. Prometheus API 사용에 대한 자세한 내용은 [Amazon Managed Service for Prometheus API Reference](#)를 참조하세요.

Helm을 사용하여 Prometheus 배포

Amazon Managed Service for Prometheus를 사용하는 대신 헬름을 사용하여 Prometheus를 클러스터에 배포할 수 있습니다. 이미 Helm이 설치되어 있는 경우 `helm version` 명령을 사용하여 버전을 확인할 수 있습니다. Helm은 Kubernetes 클러스터를 위한 패키지 관리자입니다. Helm 및 Helm 설치 방법에 대한 자세한 내용은 [the section called “Helm을 사용하여 앱 배포”](#) 단원을 참조하십시오.

Amazon EKS 클러스터에 대해 Helm을 구성하면 이 Helm을 사용하여 다음 단계에 따라 Prometheus를 배포할 수 있습니다.

1. Prometheus 네임스페이스를 생성합니다.

```
kubectl create namespace prometheus
```

2. prometheus-community 차트 리포지토리를 추가합니다

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
```

3. Prometheus를 배포합니다.

```
helm upgrade -i prometheus prometheus-community/prometheus \
  --namespace prometheus \
  --set alertmanager.persistence.storageClass="gp2" \
  --set server.persistentVolume.storageClass="gp2"
```

Note

이 명령을 실행할 때 `Error: failed to download "stable/prometheus" (hint: running helm repo update may help)` 오류가 발생하면

helm repo update prometheus-community를 실행한 다음 2단계 명령을 다시 실행 해봅니다.

Error: rendered manifests contain a resource that already exists 오류가 발생하면 helm uninstall *your-release-name* -n *namespace* 를 실행한 다음 3단계 명령을 다시 실행해봅니다.

4. prometheus 네임스페이스의 모든 포드가 READY 상태인지 확인합니다.

```
kubectl get pods -n prometheus
```

예제 출력은 다음과 같습니다.

NAME	READY	STATUS	RESTARTS	AGE
prometheus-alertmanager-59b4c8c744-r7bgp	1/2	Running	0	48s
prometheus-kube-state-metrics-7cfd87cf99-jkz2f	1/1	Running	0	48s
prometheus-node-exporter-jcjzq	1/1	Running	0	48s
prometheus-node-exporter-jxv2h	1/1	Running	0	48s
prometheus-node-exporter-vbdks	1/1	Running	0	48s
prometheus-pushgateway-76c444b68c-82tnw	1/1	Running	0	48s
prometheus-server-775957f748-mmht9	1/2	Running	0	48s

5. kubectl을 사용하여 포트를 통해 로컬 시스템에 Prometheus 콘솔을 전송합니다.

```
kubectl --namespace=prometheus port-forward deploy/prometheus-server 9090
```

6. 웹 브라우저에서 <http://localhost:9090> 으로 이동하여 Prometheus 콘솔을 봅니다.
7. - 커서에 지표 삽입 메뉴에서 지표를 선택한 후 실행을 선택합니다. 그래프 탭을 선택하여 시간에 따른 지표를 표시합니다. 다음 이미지는 시간에 따른 container_memory_usage_bytes를 표시합니다.

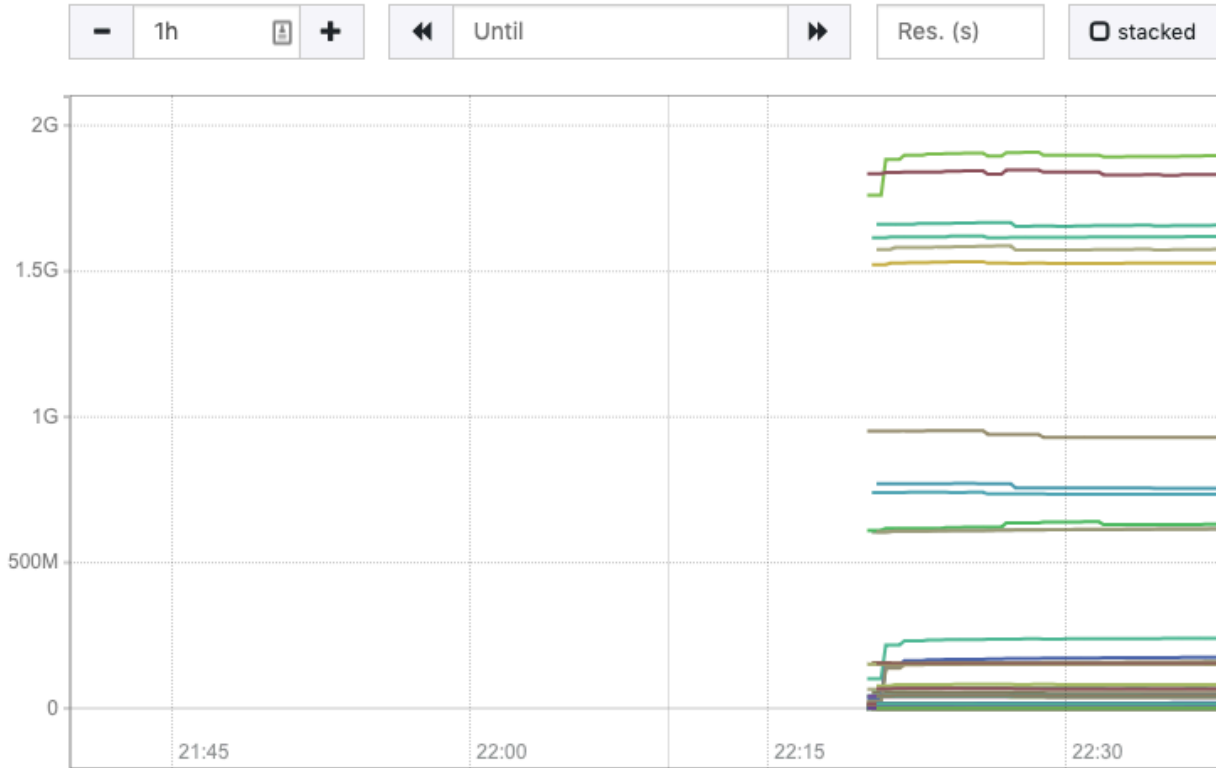
container_memory_usage_bytes

Load time: 216ms
Resolution: 14s
Total time series: 209

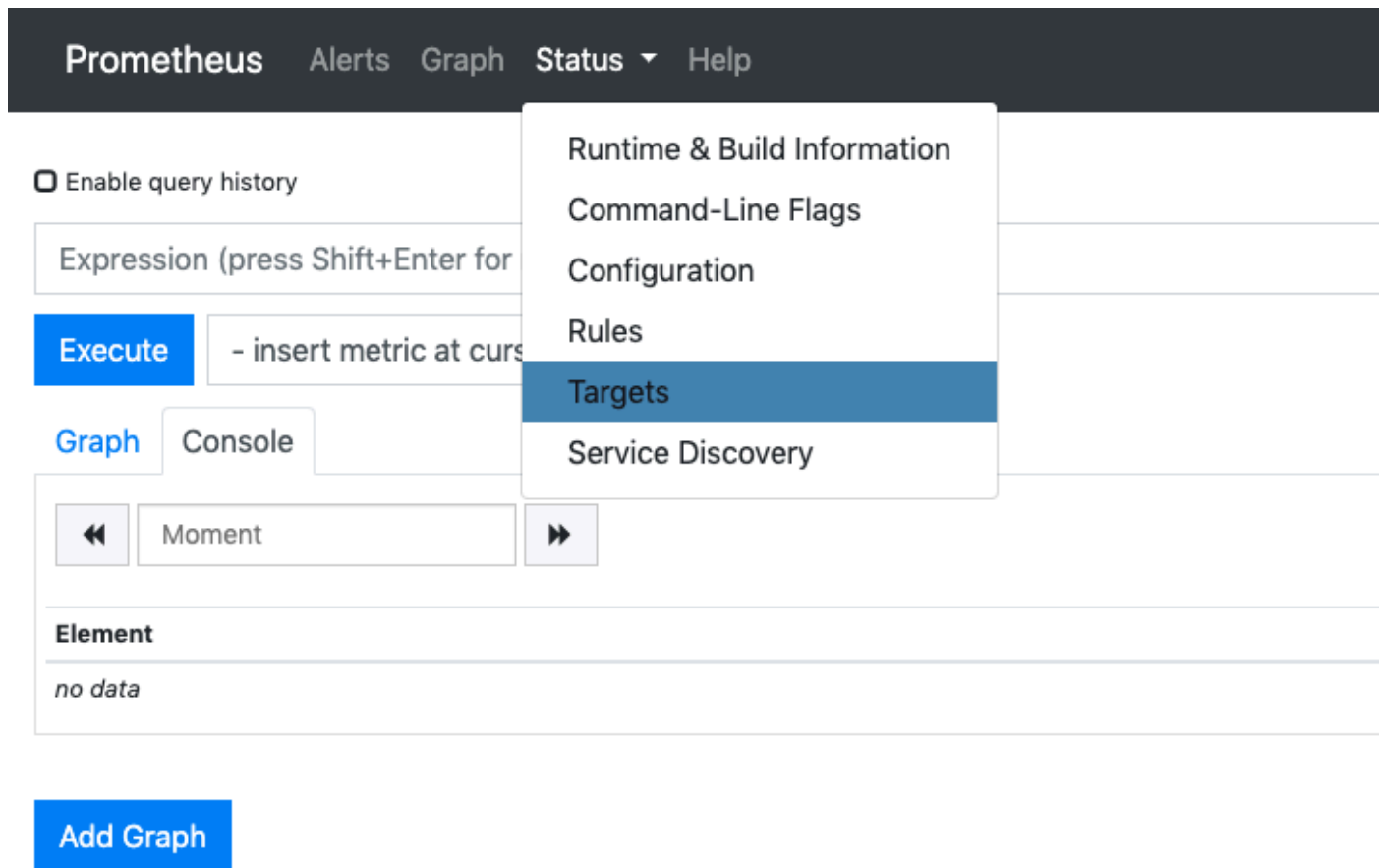
Execute

container_memory_usage 

Graph Console



8. 상단 탐색 모음에서 상태를 선택한 후 대상을 선택합니다.



서비스 검색을 사용하여 Prometheus에 연결된 모든 Kubernetes 엔드포인트가 표시됩니다.

컨트롤 플레인 원시 지표를 Prometheus 형식으로 가져오기

Kubernetes 컨트롤 플레인 [Prometheus 형식](#)으로 표시되는 여러 지표를 노출합니다. 이러한 지표는 모니터링 및 분석에 유용합니다. 지표 엔드포인트를 통해 내부적으로 노출되며 Prometheus를 완전히 배포하지 않고도 액세스할 수 있습니다. 그러나 Prometheus를 더 쉽게 배포하면 시간 경과에 따른 지표를 분석할 수 있습니다.

원시 지표 출력을 보려면 endpoint를 바꾸고 다음 명령을 실행합니다.

```
kubectl get --raw endpoint
```

이 명령을 사용하면 엔드포인트 경로를 전달할 수 있으며 원시 응답이 반환됩니다. 출력에 다양한 지표가 줄별로 나열되며 각 줄에는 지표 이름, 태그, 값이 포함됩니다.

```
metric_name{tag="value"[,...]} value
```

API 서버에서 지표 가져오기

일반 API 서버 엔드포인트는 Amazon EKS 컨트롤 플레인에 노출됩니다. 이 엔드포인트는 주로 특정 지표를 살펴보는 데 유용합니다.

```
kubectl get --raw /metrics
```

예제 출력은 다음과 같습니다.

```
[...]
# HELP rest_client_requests_total Number of HTTP requests, partitioned by status code,
# method, and host.
# TYPE rest_client_requests_total counter
rest_client_requests_total{code="200",host="127.0.0.1:21362",method="POST"} 4994
rest_client_requests_total{code="200",host="127.0.0.1:443",method="DELETE"} 1
rest_client_requests_total{code="200",host="127.0.0.1:443",method="GET"} 1.326086e+06
rest_client_requests_total{code="200",host="127.0.0.1:443",method="PUT"} 862173
rest_client_requests_total{code="404",host="127.0.0.1:443",method="GET"} 2
rest_client_requests_total{code="409",host="127.0.0.1:443",method="POST"} 3
rest_client_requests_total{code="409",host="127.0.0.1:443",method="PUT"} 8
# HELP ssh_tunnel_open_count Counter of ssh tunnel total open attempts
# TYPE ssh_tunnel_open_count counter
ssh_tunnel_open_count 0
# HELP ssh_tunnel_open_fail_count Counter of ssh tunnel failed open attempts
# TYPE ssh_tunnel_open_fail_count counter
ssh_tunnel_open_fail_count 0
```

이 원시 출력은 API 서버가 표시하는 축자를 반환합니다.

metrics.eks.amazonaws.com의 컨트롤 플레인 지표 가져오기

Kubernetes 버전 1.28 이상인 클러스터의 경우 Amazon EKS는 API 그룹 metrics.eks.amazonaws.com에도 지표를 공개합니다. 이러한 지표에는 kube-scheduler 및 kube-controller-manager와 같은 컨트롤 플레인 구성 요소가 포함됩니다.

Note

클러스터에서 새 APIService 리소스 v1.metrics.eks.amazonaws.com의 생성을 차단할 수 있는 웹훅 구성이 있는 경우 지표 엔드포인트 기능을 사용하지 못할 수 있습니다.

v1.metrics.eks.amazonaws.com 키워드를 검색하여 kube-apiserver 감사 로그에서 확인할 수 있습니다.

kube-scheduler 지표 가져오기

다음 명령을 사용하여 kube-scheduler 지표를 검색합니다.

```
kubectl get --raw "/apis/metrics.eks.amazonaws.com/v1/ksh/container/metrics"
```

예제 출력은 다음과 같습니다.

```
# TYPE scheduler_pending_pods gauge
scheduler_pending_pods{queue="active"} 0
scheduler_pending_pods{queue="backoff"} 0
scheduler_pending_pods{queue="gated"} 0
scheduler_pending_pods{queue="unschedulable"} 18
# HELP scheduler_pod_scheduling_attempts [STABLE] Number of attempts to successfully
  schedule a pod.
# TYPE scheduler_pod_scheduling_attempts histogram
scheduler_pod_scheduling_attempts_bucket{le="1"} 79
scheduler_pod_scheduling_attempts_bucket{le="2"} 79
scheduler_pod_scheduling_attempts_bucket{le="4"} 79
scheduler_pod_scheduling_attempts_bucket{le="8"} 79
scheduler_pod_scheduling_attempts_bucket{le="16"} 79
scheduler_pod_scheduling_attempts_bucket{le="+Inf"} 81
[...]
```

kube-controller-manager 지표 가져오기

다음 명령을 사용하여 kube-controller-manager 지표를 검색합니다.

```
kubectl get --raw "/apis/metrics.eks.amazonaws.com/v1/kcm/container/metrics"
```

예제 출력은 다음과 같습니다.

```
[...]
workqueue_work_duration_seconds_sum{name="pvprotection"} 0
workqueue_work_duration_seconds_count{name="pvprotection"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-08"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-07"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="1e-06"} 0
```



```
workqueue_work_duration_seconds_bucket{name="replicaset",le="9.999999999999999e-06"} 0
workqueue_work_duration_seconds_bucket{name="replicaset",le="9.999999999999999e-05"} 19
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.001"} 109
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.01"} 139
workqueue_work_duration_seconds_bucket{name="replicaset",le="0.1"} 181
workqueue_work_duration_seconds_bucket{name="replicaset",le="1"} 191
workqueue_work_duration_seconds_bucket{name="replicaset",le="10"} 191
workqueue_work_duration_seconds_bucket{name="replicaset",le="+Inf"} 191
workqueue_work_duration_seconds_sum{name="replicaset"} 4.265655885000002
[...]
```

스케줄러 및 컨트롤러 관리자 지표 이해

다음 표에서는 Prometheus 스타일 스크레이핑에 사용할 수 있는 스케줄러 및 컨트롤러 관리자 지표를 설명합니다. 이러한 지표에 대한 자세한 내용은 Kubernetes Documentation의 [Kubernetes Metrics Reference](#)를 참조하세요.

지표	컨트롤 플레인 구성 요소	설명
<code>scheduler_pending_pods</code>	<code>scheduler</code>	실행을 위해 노드에 예약되기를 기다리는 포드 수입니다.
<code>scheduler_schedule_attempts_total</code>	<code>scheduler</code>	포드를 예약하기 위해 시도한 횟수입니다.
<code>scheduler_preemption_attempts_total</code>	<code>scheduler</code>	우선순위가 낮은 포드를 제거하여 우선순위가 높은 포드를 예약하기 위해 스케줄러가 시도한 횟수입니다.
<code>scheduler_preemption_victims</code>	<code>scheduler</code>	우선순위가 더 높은 포드를 위한 공간을 만들기 위해 제거하기로 선택한 포드 수입니다.
<code>scheduler_pod_scheduling_attempts</code>	<code>scheduler</code>	포드를 성공적으로 예약하기 위해 시도한 횟수입니다.
<code>scheduler_scheduling_attempt_duration_seconds</code>	<code>scheduler</code>	스케줄러가 리소스 가용성 및 예약 규칙과 같은 다양한 요인을 기반으로 포드를 실행할 적

지표	컨트롤 플레인 구성 요소	설명
		절한 위치를 얼마나 빨리 또는 느리게 찾을 수 있는지를 나타냅니다.
scheduler_pod_scheduling_slack_duration_seconds	scheduler	포드가 예약 대기열에 진입한 시점부터 예약 중인 포드의 엔드 투 엔드 지연 시간입니다. 여기에는 여러 번의 예약 시도가 포함될 수 있습니다.
cronjob_controller_job_creation_skew_duration_seconds	컨트롤러 관리자	cronjob이 실행되도록 예약된 시점과 해당 작업이 생성된 시점 사이의 시간입니다.
workqueue_depth	컨트롤러 관리자	현재 대기열 깊이입니다.
workqueue_adds_total	컨트롤러 관리자	작업 대기열에서 처리한 총 추가 수입니다.
workqueue_queue_duration_seconds	컨트롤러 관리자	항목이 요청되기 전에 작업 대기열에 남아 있는 시간(초)입니다.
workqueue_work_duration_seconds	컨트롤러 관리자	작업 대기열에서 항목을 처리하는 데 걸리는 시간(초)입니다.

Prometheus 스크레이퍼를 배포하여 지표를 일관되게 스크레이핑

Prometheus 스크레이퍼를 배포하여 지표를 일관되게 스크레이핑하려면 다음 구성을 사용합니다.

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-conf
data:
```

```
prometheus.yml: |-
  global:
    scrape_interval: 30s
  scrape_configs:
    # apiserver metrics
    - job_name: apiserver-metrics
      kubernetes_sd_configs:
        - role: endpoints
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
        - source_labels:
            [
              __meta_kubernetes_namespace,
              __meta_kubernetes_service_name,
              __meta_kubernetes_endpoint_port_name,
            ]
          action: keep
          regex: default;kubernetes;https
    # Scheduler metrics
    - job_name: 'ksh-metrics'
      kubernetes_sd_configs:
        - role: endpoints
      metrics_path: /apis/metrics.eks.amazonaws.com/v1/ksh/container/metrics
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      relabel_configs:
        - source_labels:
            [
              __meta_kubernetes_namespace,
              __meta_kubernetes_service_name,
              __meta_kubernetes_endpoint_port_name,
            ]
          action: keep
          regex: default;kubernetes;https
    # Controller Manager metrics
    - job_name: 'kcm-metrics'
      kubernetes_sd_configs:
```

```

- role: endpoints
metrics_path: /apis/metrics.eks.amazonaws.com/v1/kcm/container/metrics
scheme: https
tls_config:
  ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
  insecure_skip_verify: true
bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
relabel_configs:
- source_labels:
  [
    __meta_kubernetes_namespace,
    __meta_kubernetes_service_name,
    __meta_kubernetes_endpoint_port_name,
  ]
  action: keep
  regex: default;kubernetes;https
---
apiVersion: v1
kind: Pod
metadata:
  name: prom-pod
spec:
  containers:
  - name: prom-container
    image: prom/prometheus
    ports:
    - containerPort: 9090
    volumeMounts:
    - name: config-volume
      mountPath: /etc/prometheus/
  volumes:
  - name: config-volume
    configMap:
      name: prometheus-conf

```

포드가 새 지표 엔드포인트에 액세스하려면 다음 권한이 필요합니다.

```

{
  "effect": "allow",
  "apiGroups": [
    "metrics.eks.amazonaws.com"
  ],
  "resources": [

```

```

    "kcm/metrics",
    "ksh/metrics"
  ],
  "verbs": [
    "get"
  ] },

```

사용 중인 역할을 패치하려면 다음 명령을 사용할 수 있습니다.

```

kubectl patch clusterrole <role-name> --type=json -p='[
  {
    "op": "add",
    "path": "/rules/-",
    "value": {
      "verbs": ["get"],
      "apiGroups": ["metrics.eks.amazonaws.com"],
      "resources": ["kcm/metrics", "ksh/metrics"]
    }
  }
]'

```

그런 다음 Prometheus 스크레이퍼의 포트를 로컬 포트에 프록시하여 Prometheus 대시보드를 볼 수 있습니다.

```

kubectl port-forward pods/prom-pod 9090:9090

```

Amazon EKS 클러스터의 경우 코어 Kubernetes 컨트롤 플레인 지표도 AWS/EKS 네임스페이스 아래의 Amazon CloudWatch 지표에 수집됩니다. 이를 보려면 [CloudWatch 콘솔](#)을 열고 왼쪽 탐색 창에서 모든 지표를 선택합니다. 지표 선택 페이지에서 클러스터의 AWS/EKS 네임스페이스와 지표 차원을 선택합니다.

Amazon CloudWatch를 사용한 클러스터 데이터 모니터링

Amazon CloudWatch는 클라우드 리소스에서 지표와 로그를 수집하는 모니터링 서비스입니다. CloudWatch는 버전 1.28 이상의 새 클러스터를 사용할 때 몇 가지 기본 Amazon EKS 지표를 무료로 제공합니다. 그러나 CloudWatch Observability Operator를 Amazon EKS 추가 기능으로 사용하면 향상된 관찰성 기능을 이용할 수 있습니다.

Amazon CloudWatch의 기본 지표

Kubernetes 버전 1.28 이상인 클러스터의 경우 AWS/EKS 네임스페이스에서 CloudWatch 변환 지표를 무료로 얻을 수 있습니다. 다음 표에는 지원되는 버전에서 사용할 수 있는 기본 지표 목록이 나와 있습니다. 나열된 모든 지표의 빈도는 1분입니다.

메트릭 이름	설명
<code>scheduler_schedule_attempts_total</code>	지정된 기간 동안 스케줄러가 클러스터에서 포드를 예약하려는 총 시도 횟수입니다. 이 지표는 스케줄러의 워크로드를 모니터링하는 데 도움이 되며 예약 압력이나 포드 배치의 잠재적 문제를 나타낼 수 있습니다. 단위: 개 유효 통계: Sum
<code>scheduler_schedule_attempts_SCHEDULED</code>	지정된 기간 동안 스케줄러가 클러스터의 노드에 포드를 예약하려는 시도가 성공한 횟수입니다. 단위: 개 유효 통계: Sum
<code>scheduler_schedule_attempts_UNSCHEDULABLE</code>	노드의 CPU나 메모리가 부족한 등 유효한 제약 조건으로 인해 지정된 기간 동안 예약할 수 없었던 포드의 예약 시도 횟수입니다. 단위: 개 유효 통계: Sum
<code>scheduler_schedule_attempts_ERROR</code>	API 서버 연결 문제와 같은 스케줄러 자체의 내부 문제로 인해 지정된 기간 동안 실패한 포드 예약 시도 횟수입니다. 단위: 개

메트릭 이름	설명
	유효 통계: Sum
scheduler_pending_pods	지정된 기간 동안 클러스터에서 스케줄러가 예약할 보류 중인 총 포드 수입니다. 단위: 개 유효 통계: Sum
scheduler_pending_pods_ACTIVEQ	지정된 기간 동안 클러스터에서 예약 대기 중인 activeQ의 보류 중인 포드 수입니다. 단위: 개 유효 통계: Sum
scheduler_pending_pods_UNSCHEДУLABLE	스케줄러가 예약을 시도했지만 실패하여 재시도를 위해 예약 불가 상태로 유지되는 보류 중인 포드 수입니다. 단위: 개 유효 통계: Sum
scheduler_pending_pods_BACKOFF	백오프 상태에서 백오프 기간이 만료되기를 기다리는 backoffQ의 보류 중인 포드 수입니다. 단위: 개 유효 통계: Sum
scheduler_pending_pods_GATED	필요한 조건을 충족할 때까지 예약할 수 없어 비공개 상태에서 현재 대기하고 있는 보류 중인 포드 수입니다. 단위: 개 유효 통계: Sum

메트릭 이름	설명
apiserver_request_total	클러스터의 모든 API 서버에서 이루어진 HTTP 요청 횟수입니다. 단위: 개 유효 통계: Sum
apiserver_request_total_4XX	4XX(클라이언트 오류) 상태 코드를 초래한 클러스터의 모든 API 서버에 대한 HTTP 요청 횟수입니다. 단위: 개 유효 통계: Sum
apiserver_request_total_429	클라이언트가 비율 제한 임계값을 초과할 때 발생하는 429 상태 코드를 초래한 클러스터의 모든 API 서버에 대한 HTTP 요청 횟수입니다. 단위: 개 유효 통계: Sum
apiserver_request_total_5XX	5XX(서버 오류) 상태 코드를 초래한 클러스터의 모든 API 서버에 대한 HTTP 요청 횟수입니다. 단위: 개 유효 통계: Sum
apiserver_request_total_LIST_PODS	클러스터의 모든 API 서버에 대한 LIST 포트 요청 횟수입니다. 단위: 개 유효 통계: Sum

메트릭 이름	설명
apiserver_request_duration_seconds_PUT_P99	클러스터의 모든 API 서버의 모든 요청에서 계산된 PUT 요청에 대한 지연 시간의 99번째 백분위수입니다. 모든 PUT 요청의 99%가 완료되는 응답 시간을 나타냅니다. 단위: 초 유효 통계: Average
apiserver_request_duration_seconds_PATCH_P99	클러스터의 모든 API 서버의 모든 요청에서 계산된 PATCH 요청에 대한 지연 시간의 99번째 백분위수입니다. 모든 PATCH 요청의 99%가 완료되는 응답 시간을 나타냅니다. 단위: 초 유효 통계: Average
apiserver_request_duration_seconds_POST_P99	클러스터의 모든 API 서버의 모든 요청에서 계산된 POST 요청에 대한 지연 시간의 99번째 백분위수입니다. 모든 POST 요청의 99%가 완료되는 응답 시간을 나타냅니다. 단위: 초 유효 통계: Average
apiserver_request_duration_seconds_GET_P99	클러스터의 모든 API 서버의 모든 요청에서 계산된 GET 요청에 대한 지연 시간의 99번째 백분위수입니다. 모든 GET 요청의 99%가 완료되는 응답 시간을 나타냅니다. 단위: 초 유효 통계: Average

메트릭 이름	설명
apiserver_request_duration_seconds_LIST_P99	클러스터의 모든 API 서버의 모든 요청에서 계산된 LIST 요청에 대한 지연 시간의 99번째 백분위수입니다. 모든 LIST 요청의 99%가 완료되는 응답 시간을 나타냅니다. 단위: 초 유효 통계: Average
apiserver_request_duration_seconds_DELETE_P99	클러스터의 모든 API 서버의 모든 요청에서 계산된 DELETE 요청에 대한 지연 시간의 99번째 백분위수입니다. 모든 DELETE 요청의 99%가 완료되는 응답 시간을 나타냅니다. 단위: 초 유효 통계: Average
apiserver_current_inflight_requests_MUTATING	클러스터의 모든 API 서버에서 현재 처리 중인 변경 요청(POST, PUT, DELETE, PATCH) 횟수입니다. 이 지표는 진행 중이며 아직 처리가 완료되지 않은 요청을 나타냅니다. 단위: 개 유효 통계: Sum
apiserver_current_inflight_requests_READONLY	클러스터의 모든 API 서버에서 현재 처리 중인 읽기 전용 요청(GET, LIST) 횟수입니다. 이 지표는 진행 중이며 아직 처리가 완료되지 않은 요청을 나타냅니다. 단위: 개 유효 통계: Sum

메트릭 이름	설명
<code>apiserver_admission_webhook_request_total</code>	클러스터의 모든 API 서버에서 수행된 승인 웹훅 요청 횟수입니다. 단위: 개 유효 통계: Sum
<code>apiserver_admission_webhook_request_total_ADMIT</code>	클러스터의 모든 API 서버에서 수행된 변경 승인 웹훅 요청 횟수입니다. 단위: 개 유효 통계: Sum
<code>apiserver_admission_webhook_request_total_VALIDATING</code>	클러스터의 모든 API 서버에서 수행된 승인 웹훅 요청의 검증 횟수입니다. 단위: 개 유효 통계: Sum
<code>apiserver_admission_webhook_rejection_count</code>	클러스터의 모든 API 서버에서 거부된 승인 웹훅 요청 횟수입니다. 단위: 개 유효 통계: Sum
<code>apiserver_admission_webhook_rejection_count_ADMIT</code>	클러스터의 모든 API 서버에서 거부된 변경 승인 웹훅 요청 횟수입니다. 단위: 개 유효 통계: Sum

메트릭 이름	설명
<code>apiserver_admission_webhook_rejection_count_VALIDATING</code>	거부된 클러스터의 모든 API 서버에서 수행된 승인 웹훅 요청의 검증 횟수입니다. 단위: 개 유효 통계: Sum
<code>apiserver_admission_webhook_admission_duration_seconds</code>	클러스터의 모든 API 서버의 모든 요청에서 계산된 타사 승인 웹훅 요청에 대한 지연 시간의 99번째 백분위수입니다. 모든 타사 승인 웹훅 요청의 99%가 완료되는 응답 시간을 나타냅니다. 단위: 초 유효 통계: Average
<code>apiserver_admission_webhook_admission_duration_seconds_ADMIT_P99</code>	클러스터의 모든 API 서버의 모든 요청에서 계산된 타사 변경 승인 웹훅 요청에 대한 지연 시간의 99번째 백분위수입니다. 모든 타사 변경 승인 웹훅 요청의 99%가 완료되는 응답 시간을 나타냅니다. 단위: 초 유효 통계: Average
<code>apiserver_admission_webhook_admission_duration_seconds_VALIDATING_P99</code>	클러스터의 모든 API 서버의 모든 요청에서 계산된 타사 검증 승인 웹훅 요청에 대한 지연 시간의 99번째 백분위수입니다. 모든 타사 검증 승인 웹훅 요청의 99%가 완료되는 응답 시간을 나타냅니다. 단위: 초 유효 통계: Average

메트릭 이름	설명
apiserver_storage_size_bytes	클러스터의 API 서버에서 사용하는 etcd 스토리지 데이터베이스 파일의 물리적 크기(바이트)입니다. 이 지표는 스토리지에 할당된 실제 디스크 공간을 나타냅니다. 단위: 바이트 유효 통계: Maximum

Amazon CloudWatch Observability Operator

Amazon CloudWatch Observability는 실시간 로그, 지표 및 추적 데이터를 수집합니다. 이를 [Amazon CloudWatch](#) 및 [AWS X-Ray](#)로 보냅니다. 이 추가 기능을 설치하여 CloudWatch 에이전트를 설치하고 Amazon EKS의 향상된 관찰성을 통해 CloudWatch Application Signals와 CloudWatch Container Insights를 모두 활성화할 수 있습니다. 이를 통해 인프라 및 컨테이너화된 애플리케이션의 상태와 성능을 모니터링할 수 있습니다. Amazon CloudWatch Observability Operator는 필요한 구성 요소를 설치하고 구성하도록 설계되었습니다.

Amazon EKS는 CloudWatch Observability Operator를 [Amazon EKS 추가 기능](#)으로 지원합니다. 추가 기능을 사용하면 클러스터의 Linux 및 Windows 워커 노드 모두에서 Container Insights를 사용할 수 있습니다. Windows에서 Container Insights를 활성화하려면 Amazon EKS 추가 기능 버전이 1.5.0 이상이어야 합니다. 현재 Amazon EKS Windows에서는 CloudWatch Application Signals가 지원되지 않습니다.

아래 주제에서는 Amazon EKS 클러스터에서 CloudWatch Observability Operator 사용을 시작하는 방법을 설명합니다.

- 이 추가 기능 설치 지침은 Amazon CloudWatch 사용 설명서의 [Amazon CloudWatch Observability EKS 추가 기능 또는 헬름 차트를 사용하여 CloudWatch 에이전트 설치](#)를 참조하세요.
- CloudWatch Application Signals에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 [Application Signals](#)를 참조하세요.
- Container Insights에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 [Container Insights 사용](#) 섹션을 참조하세요.

CloudWatch Logs에 컨트롤 플레인 로그 전송

Amazon EKS 컨트롤 플레인 로깅은 Amazon EKS 컨트롤 플레인에서 계정의 CloudWatch Logs로 감사 및 진단 로그를 직접 제공합니다. 이러한 로그를 통해 클러스터를 쉽게 보호하고 실행할 수 있습니다. 필요한 로그 유형을 정확하게 선택할 수 있으며, 로그는 CloudWatch의 각 Amazon EKS 클러스터에 대한 그룹에 로그 스트림으로 전송됩니다. CloudWatch 구독 필터를 사용하여 로그에 대한 실시간 분석을 수행하거나 이를 다른 서비스로 전달할 수 있습니다(로그는 Base64로 인코딩되고 gzip 형식으로 압축됨). 자세한 내용은 [Amazon CloudWatch 로깅](#)을 참조하세요.

각각의 새로운 또는 기존 Amazon EKS 클러스터에 대해 활성화하려는 로그 유형을 선택하여 Amazon EKS 컨트롤 플레인 로깅의 사용을 시작할 수 있습니다. AWS Management Console, AWS CLI(버전 1.16.139 이상)를 사용하거나 Amazon EKS API를 통해 클러스터 단위로 각 로그 유형을 활성화하거나 비활성화할 수 있습니다. 활성화하면 Amazon EKS 클러스터에서 동일한 계정의 CloudWatch Logs로 로그가 자동으로 전송됩니다.

Amazon EKS 컨트롤 플레인 로깅을 사용하면 실행하는 각 클러스터에 대해 표준 Amazon EKS 요금이 부과됩니다. 클러스터에서 CloudWatch Logs로 전송한 모든 로그에 대해 표준 CloudWatch Logs 데이터 수집 및 스토리지 비용이 부과됩니다. 또한 클러스터의 일부로 프로비저닝한 AWS 리소스(예: Amazon EC2 인스턴스 또는 Amazon EBS 볼륨)에 대해서도 요금이 부과됩니다.

사용할 수 있는 클러스터 컨트롤 플레인 로그 유형은 다음과 같습니다. 각 로그 유형은 Kubernetes 제어 플레인의 구성 요소에 해당합니다. 이러한 구성 요소에 대한 자세한 내용은 Kubernetes 설명서의 [쿠버네티스 컴포넌트](#)를 참조하십시오.

API 서버api()

클러스터의 API 서버는 Kubernetes API가 표시되는 컨트롤 플레인 구성 요소입니다. 클러스터를 시작할 때 또는 직후에 API 서버 로그를 활성화하면 API 서버를 시작하는 데 사용된 API 서버 플러그가 로그에 포함됩니다. 자세한 내용을 알아보려면 Kubernetes 설명서의 [kube-apiserver](#) 및 [audit policy](#)를 참조하세요.

감사audit()

Kubernetes 감사 로그는 클러스터에 영향을 주는 개별 사용자, 관리자 또는 시스템 구성 요소를 제공합니다. 자세한 내용은 Kubernetes 설명서의 [Auditing](#)을 참조하십시오.

인증자authenticator()

인증자 로그는 Amazon EKS에 고유합니다. 이러한 로그는 Amazon EKS가 IAM 자격 증명을 사용한 Kubernetes [역할 기반 액세스 제어](#)(RBAC) 인증에 사용하는 제어 플레인 구성 요소를 나타냅니다. 자세한 내용은 [클러스터 관리](#) 단원을 참조하십시오.

컨트롤러 관리자 `controllerManager()`

컨트롤러 관리자는 Kubernetes와 함께 제공되는 핵심 컨트롤 루프를 관리합니다. 자세한 내용은 Kubernetes 설명서의 [kube-controller-manager](#)를 참조하십시오.

스케줄러 `scheduler()`

스케줄러 구성 요소는 클러스터에서 포드를 실행하는 시기와 위치를 관리합니다. 자세한 내용은 Kubernetes 설명서의 [kube-scheduler](#)를 참조하십시오.

컨트롤 플레인 로그 활성화 또는 비활성화

기본적으로 클러스터 컨트롤 플레인 로그는 CloudWatch Logs로 전송되지 않습니다. 클러스터에 대해 로그를 전송하려면 각 로그 유형을 개별적으로 활성화해야 합니다. CloudWatch Logs 수집, 아카이브 스토리지 및 데이터 검색 요율이 활성화된 컨트롤 플레인 로그에 적용됩니다. 자세한 내용은 [CloudWatch 요금](#)을 참조하세요.

컨트롤 플레인 로깅 구성을 업데이트하려면 Amazon EKS에서 각 서브넷에 최대 5개의 사용 가능한 IP 주소가 필요합니다. 로그 유형을 활성화하면 로그가 로그 세부 수준 2로 전송됩니다.

[AWS Management Console](#) 또는 [AWS CLI](#)를 사용하여 컨트롤 플레인 로그를 활성화하거나 비활성화할 수 있습니다.

AWS Management Console

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 클러스터 이름을 선택하여 클러스터 정보를 표시합니다.
3. 관찰성 탭을 선택합니다.
4. 컨트롤 플레인 로깅 섹션에서 로깅 관리를 선택합니다.
5. 각 개별 로그 유형에 대해 로그 유형이 켜져 있거나 꺼져 있어야 하는지를 선택합니다. 기본적으로 각 로그 유형은 해제되어 있습니다.
6. 변경 사항 저장을 선택하여 종료합니다.

AWS CLI

1. AWS CLI 버전은 다음 명령을 통해 확인할 수 있습니다.

```
aws --version
```

AWS CLI 버전이 1.16.139 이하인 경우 먼저 최신 버전으로 업데이트해야 합니다. AWS CLI를 설치하거나 업그레이드하려면 [AWS 명령줄 인터페이스 사용 설명서](#)에서 AWS 명령줄 인터페이스 설치를 참조하세요.

- 다음 AWS CLI 명령을 사용하여 클러스터의 컨트롤 플레인 로그 내보내기 구성을 업데이트합니다. *my-cluster*를 클러스터 이름으로 바꾸고 원하는 엔드포인트 액세스 값을 지정합니다.

Note

다음 명령은 사용 가능한 모든 로그 유형을 CloudWatch Logs로 전송합니다.

```
aws eks update-cluster-config \
  --region region-code \
  --name my-cluster \
  --logging '{"clusterLogging":[{"types":
["api","audit","authenticator","controllerManager","scheduler"],"enabled":true}]}'
```

예제 출력은 다음과 같습니다.

```
{
  "update": {
    "id": "883405c8-65c6-4758-8cee-2a7c1340a6d9",
    "status": "InProgress",
    "type": "LoggingUpdate",
    "params": [
      {
        "type": "ClusterLogging",
        "value": "{\"clusterLogging\":{\"types\":[\"api\",\"audit\",
\"authenticator\",\"controllerManager\",\"scheduler\"],\"enabled\":true}}}"
      }
    ],
    "createdAt": 1553271814.684,
    "errors": []
  }
}
```

- 이전 명령에서 반환된 클러스터 이름과 업데이트 ID를 사용하여 다음 명령으로 로그 구성 업데이트의 상태를 모니터링합니다. 상태가 `Successful`로 표시되면 업데이트가 완료된 것입니다.


```
aws eks describe-update \
  --region region-code\
  --name my-cluster \
  --update-id 883405c8-65c6-4758-8cee-2a7c1340a6d9
```

예제 출력은 다음과 같습니다.

```
{
  "update": {
    "id": "883405c8-65c6-4758-8cee-2a7c1340a6d9",
    "status": "Successful",
    "type": "LoggingUpdate",
    "params": [
      {
        "type": "ClusterLogging",
        "value": "{\"clusterLogging\": [{\"types\": [\"api\", \"audit\", \"authenticator\", \"controllerManager\", \"scheduler\"], \"enabled\": true}]}"
      }
    ],
    "createdAt": 1553271814.684,
    "errors": []
  }
}
```

클러스터 컨트롤 플레인 로그 보기

Amazon EKS 클러스터에 대한 컨트롤 플레인 로그 유형을 활성화하면 CloudWatch 콘솔에서 해당 유형을 볼 수 있습니다.

CloudWatch에서 로그 보기, 분석 및 관리에 대한 자세한 내용은 [Amazon CloudWatch Logs 사용 설명서](#)를 참조하세요.

1. [CloudWatch 콘솔](#)을 엽니다. 이 링크는 콘솔을 열고 현재 사용 가능한 로그 그룹을 표시하며 /aws/eks 접두사로 해당 로그 그룹을 필터링합니다.
2. 로그를 보려는 클러스터를 선택합니다. 로그 그룹 이름 형식은 /aws/eks/*my-cluster*/cluster입니다.
3. 보려는 로그 스트림을 선택합니다. 다음 목록에는 각 로그 유형의 로그 스트림 이름 형식에 대한 설명이 나와 있습니다.

Note

로그 스트림 데이터가 증가함에 따라 로그 스트림 이름이 교체됩니다. 특정 로그 유형에 대해 여러 로그 스트림이 있는 경우 최신 마지막 이벤트 시간으로 로그 스트림 이름을 찾아 최신 로그 스트림을 볼 수 있습니다.

- Kubernetes API 서버 구성 요소 로그(**api**) - kube-apiserver-*1234567890abcdef01234567890abcde*
- 감사(**audit**) - kube-apiserver-audit-*1234567890abcdef01234567890abcde*
- 인증자(**authenticator**) - authenticator-*1234567890abcdef01234567890abcde*
- 컨트롤러 관리자(**controllerManager**) - kube-controller-manager-*1234567890abcdef01234567890abcde*
- 스케줄러(**scheduler**) - kube-scheduler-*1234567890abcdef01234567890abcde*

4. 로그 스트림의 이벤트를 살펴봅니다.

예를 들어 상위 kube-apiserver-*1234567890abcdef01234567890abcde* 를 볼 때 클러스터의 초기 API 서버 플래그가 표시되어야 합니다.

Note

로그 스트림의 시작 부분에 API 서버 로그가 표시되지 않으면 서버에서 API 서버 로깅을 활성화하기 전에 서버에서 API 서버 로그 파일이 교체되었을 수 있습니다. API 서버 로깅이 활성화되기 전에 교체되는 모든 로그 파일은 CloudWatch로 내보낼 수 없습니다.

그러나 동일한 Kubernetes 버전으로 새 클러스터를 생성하고 클러스터를 생성할 때 API 서버 로깅을 활성화할 수 있습니다. 동일한 플랫폼 버전의 클러스터에는 동일한 플래그가 활성화되어 있으므로 플래그가 새 클러스터의 플래그와 일치해야 합니다. CloudWatch에서 새 클러스터에 대한 플래그 보기를 마치면 새 클러스터를 삭제할 수 있습니다.

API 호출을 AWS CloudTrail 이벤트로 기록

Amazon EKS는 AWS CloudTrail에 통합됩니다. CloudTrail은 Amazon EKS에서 사용자, 역할 또는 AWS 서비스가 수행한 작업의 레코드를 제공하는 서비스입니다. CloudTrail은 Amazon EKS에 대한 모

든 API 호출을 이벤트로 캡처합니다. 여기에는 Amazon EKS 콘솔 호출과 Amazon EKS API 작업에 대한 코드 호출이 포함됩니다.

추적을 생성하면 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 배포할 수 있습니다. 여기에는 Amazon EKS 이벤트가 포함됩니다. 추적을 구성하지 않은 경우에도 CloudTrail 콘솔의 이벤트 기록에서 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집하는 정보를 사용하여 요청에 대한 몇 가지 세부 정보를 확인할 수 있습니다. 예를 들어, 언제 Amazon EKS에 요청했는지, 어떤 IP 주소에서 요청했는지 그리고 누가 요청했는지를 확인할 수 있습니다.

CloudTrail에 대한 자세한 내용은 [AWS CloudTrail 사용 설명서](#)를 참조하세요.

주제

- [AWS CloudTrail에 대한 유용한 참조 보기](#)
- [AWS CloudTrail 로그 파일 항목 분석](#)
- [Amazon EC2 Auto Scaling 그룹을 위한 지표 보기](#)

AWS CloudTrail에 대한 유용한 참조 보기

CloudTrail은 AWS 계정 생성 시 AWS 계정에서 사용 설정합니다. Amazon EKS에서 어떤 활동이 발생하는 경우, 해당 활동이 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 [CloudTrail 이벤트 기록을 사용하여 이벤트 보기](#)를 참조하세요.

Amazon EKS에 대한 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 지역에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 AWS 리전의 이벤트를 로깅하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 또는 CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 처리하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 정보는 다음 리소스를 참조하세요.

- [추적 생성 개요](#)
- [CloudTrail 지원 서비스 및 통합](#)
- [CloudTrail에 대한 Amazon SNS 알림 구성](#)
- [여러 지역에서 CloudTrail 로그 파일 받기](#) 및 [여러 계정에서 CloudTrail 로그 파일 받기](#)

모든 Amazon EKS 작업은 CloudTrail에서 로깅되며 [Amazon EKS API 참조](#)에 설명되어 있습니다. 예를 들어, [CreateCluster](#), [ListClusters](#) 및 [DeleteCluster](#)을 호출하면 CloudTrail 로그 파일에 항목이 생성됩니다.

모든 이벤트 또는 로그 항목에는 요청이 이루어지고 자격 증명이 사용되는 IAM 자격 증명 유형에 관한 정보가 포함됩니다. 임시 자격 증명이 사용되는 경우 요소는 자격 증명을 획득하는 방법을 보여 줍니다.

자세한 내용은 [CloudTrail userIdentity 요소](#)를 참조하세요.

AWS CloudTrail 로그 파일 항목 분석

추적이란 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 입력할 수 있게 하는 구성입니다.

CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스로부터의 단일 요청을 나타내며 요청 작업에 대한 정보가 들어 있습니다. 여기에는 작업 날짜와 시간, 사용된 요청 파라미터에 대한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 호출에 대한 순서가 지정된 스택 추적이 아니므로 특정 순서로 표시되지 않습니다.

다음은 [CreateCluster](#) 작업을 보여 주는 CloudTrail 로그 항목이 나타낸 예제입니다.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AKIAIOSFODNN7EXAMPLE",
    "arn": "arn:aws:iam::111122223333:user/username",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "userName": "username"
  },
  "eventTime": "2018-05-28T19:16:43Z",
  "eventSource": "eks.amazonaws.com",
  "eventName": "CreateCluster",
  "awsRegion": "region-code",
  "sourceIPAddress": "205.251.233.178",
  "userAgent": "PostmanRuntime/6.4.0",
  "requestParameters": {
    "resourcesVpcConfig": {
      "subnetIds": [
        "subnet-a670c2df",
        "subnet-4f8c5004"
      ]
    }
  }
}
```

```

    },
    "roleArn": "arn:aws:iam::111122223333:role/AWSServiceRoleForAmazonEKS-
CAC1G1VH3ZKZ",
    "clusterName": "test"
  },
  "responseElements": {
    "cluster": {
      "clusterName": "test",
      "status": "CREATING",
      "createdAt": 1527535003.208,
      "certificateAuthority": {},
      "arn": "arn:aws:eks:region-code:111122223333:cluster/test",
      "roleArn": "arn:aws:iam::111122223333:role/AWSServiceRoleForAmazonEKS-
CAC1G1VH3ZKZ",
      "version": "1.10",
      "resourcesVpcConfig": {
        "securityGroupIds": [],
        "vpcId": "vpc-21277358",
        "subnetIds": [
          "subnet-a670c2df",
          "subnet-4f8c5004"
        ]
      }
    }
  },
  "requestID": "a7a0735d-62ab-11e8-9f79-81ce5b2b7d37",
  "eventID": "eab22523-174a-499c-9dd6-91e7be3ff8e3",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "recipientAccountId": "111122223333"
}

```

Amazon EKS 서비스 연결 역할에 대한 로그 항목

Amazon EKS 서비스 연결 역할은 AWS 리소스에 대한 API 호출을 수행합니다. Amazon EKS 서비스 연결 역할에서 수행한 호출의 경우 CloudTrail 로그 항목에 username: AWSServiceRoleForAmazonEKS 및 username: AWSServiceRoleForAmazonEKSNodegroup이 표시됩니다. Amazon EKS 및 서비스 연결 역할에 대한 자세한 내용은 [the section called “서비스 연결 역할”](#) 섹션을 참조하세요.

다음 예에서는 sessionContext에 표시된 AWSServiceRoleForAmazonEKSNodegroup 서비스 연결 역할에서 생성된 [DeleteInstanceProfile](#) 작업을 보여주는 CloudTrail 로그 항목을 표시합니다.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "ARO3WHGPEZ7SJ2CW55C5:EKS",
    "arn": "arn:aws:sts::111122223333:assumed-role/AWSServiceRoleForAmazonEKSNodegroup/EKS",
    "accountId": "111122223333",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "ARO3WHGPEZ7SJ2CW55C5",
        "arn": "arn:aws:iam::111122223333:role/aws-service-role/eks-nodegroup.amazonaws.com/AWSServiceRoleForAmazonEKSNodegroup",
        "accountId": "111122223333",
        "userName": "AWSServiceRoleForAmazonEKSNodegroup"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-02-26T00:56:33Z"
      }
    },
    "invokedBy": "eks-nodegroup.amazonaws.com"
  },
  "eventTime": "2020-02-26T00:56:34Z",
  "eventSource": "iam.amazonaws.com",
  "eventName": "DeleteInstanceProfile",
  "awsRegion": "region-code",
  "sourceIPAddress": "eks-nodegroup.amazonaws.com",
  "userAgent": "eks-nodegroup.amazonaws.com",
  "requestParameters": {
    "instanceProfileName": "eks-11111111-2222-3333-4444-abcdef123456"
  },
  "responseElements": null,
  "requestID": "11111111-2222-3333-4444-abcdef123456",
  "eventID": "11111111-2222-3333-4444-abcdef123456",
  "eventType": "AwsApiCall",
}
```

```
"recipientAccountId": "111122223333"
}
```

Amazon EC2 Auto Scaling 그룹을 위한 지표 보기

Amazon EKS 관리형 노드 그룹에는 Amazon EC2 Auto Scaling 그룹 지표가 추가 비용 없이 기본적으로 활성화되어 있습니다. Auto Scaling 그룹은 1분마다 샘플링된 데이터를 Amazon CloudWatch로 전송합니다. 이러한 지표는 Auto Scaling 그룹의 이름으로 조정할 수 있습니다. 시간 경과에 따른 그룹 크기 변화와 같이 관리형 노드 그룹을 지원하는 Auto Scaling 그룹의 기록을 지속적으로 파악할 수 있습니다. Auto Scaling 그룹 지표는 [Amazon CloudWatch](#) 또는 Auto Scaling 콘솔에서 사용할 수 있습니다. 자세한 내용은 [Monitor CloudWatch metrics for your Auto Scaling groups and instances](#)를 참조하세요.

Auto Scaling 그룹 지표 수집을 사용하면 관리형 노드 그룹의 규모 조정을 모니터링할 수 있습니다. Auto Scaling 그룹 지표를 통해 오토 스케일링의 최소, 최대 및 원하는 크기가 보고됩니다. 노드 그룹의 노드 수가 최소 크기 미만으로 감소하면 비정상 노드 그룹을 나타내는 경보를 생성할 수 있습니다. 노드 그룹 크기를 추적하면 데이터 영역의 용량이 부족하지 않도록 최대 개수를 조정하는 데도 유용합니다.

이러한 지표를 수집하지 않으려면 전체 또는 일부 지표만 비활성화하도록 선택할 수 있습니다. 예를 들어, CloudWatch 대시보드에서 소음을 방지하기 위해 이를 수행할 수 있습니다. 자세한 내용은 [Amazon CloudWatch metrics for Amazon EC2 Auto Scaling](#)을 참조하세요.

ADOT Operator를 사용하여 지표 및 추적 데이터 전송

Amazon EKS는 AWS Management Console, AWS CLI 및 Amazon EKS API를 사용하여 [AWS Distro for OpenTelemetry\(ADOT\)](#) 연산자를 설치하고 관리합니다. 이를 통해 Amazon EKS에서 실행되는 애플리케이션이 [Amazon CloudWatch](#), [Prometheus](#), [X-Ray](#)와 같은 여러 모니터링 서비스 옵션으로 메트릭 및 추적 데이터를 쉽게 전송할 수 있습니다.

자세한 내용은 AWS Distro for OpenTelemetry 설명서의 [Getting Started with AWS Distro for OpenTelemetry using EKS Add-Ons](#)를 참조하세요.

EKS 대시보드를 사용하여 클러스터 리소스에 대한 집계된 데이터 보기

image::images/eks-dashboard.png[screenshot of account level cluster metrics]

Amazon EKS 대시보드란 무엇인가요?

Amazon EKS 대시보드는 여러 AWS 리전과 AWS 계정에 걸쳐 Kubernetes 클러스터에 대한 통합 가시성을 제공합니다. 이 대시보드를 사용하면 다음을 수행할 수 있습니다.

- 향후 90일 이내에 지원 종료 자동 업그레이드에 대해 예약된 클러스터를 추적합니다.
- 추가 지원의 클러스터에 대한 EKS 컨트롤 플레인 비용을 예측합니다.
- 업그레이드 전에 주의가 필요한 인사이트가 있는 클러스터를 검토합니다.
- 특정 AMI 버전을 실행하는 관리형 노드 그룹을 식별합니다.
- 클러스터 지원 유형 분포를 모니터링합니다(표준 지원과 추가 지원 비교).

EKS 대시보드는 EKS Cluster Insights와 통합되어 더 이상 사용되지 않는 Kubernetes API 사용과 같은 클러스터 관련 문제를 표시합니다. 자세한 내용은 [the section called “클러스터 인사이트”](#) 섹션을 참조하세요.

Note

EKS 대시보드는 실시간이 아니며 12시간마다 업데이트됩니다. 실시간 클러스터 모니터링은 [클러스터 모니터링](#) 섹션을 참조하세요.

대시보드에서 AWS Organizations를 어떻게 사용하나요?

Amazon EKS 대시보드의 기능을 사용하려면 AWS Organizations 통합이 필요합니다. Amazon EKS 대시보드는 AWS Organizations를 활용하여 계정 전반의 클러스터 정보를 안전하게 수집합니다. 이러한 통합은 AWS 인프라 규모 조정에 따라 중앙 집중식 관리와 거버넌스를 제공합니다.

인프라에 AWS Organizations가 활성화되어 있지 않은 경우 설정 지침은 AWS [Organizations 사용 설명서](#)를 참조하세요.

교차 리전 및 교차 계정 액세스

EKS 대시보드는 AWS Organization의 멤버인 계정의 클러스터 리소스를 볼 수 있습니다. 조직의 AWS 계정 목록을 생성하려면 [Export details for all accounts in an organization](#)을 참조하세요.

us-east-1 AWS 리전에서 대시보드를 생성합니다. 대시보드를 보려면 이 리전에 로그인해야 합니다. 대시보드는 AWS 리전 전반의 데이터를 집계하지만 GovCloud 또는 중국 리전은 여기에 포함되지 않습니다.

주요 용어

- AWS Organization: 여러 AWS 계정에 대한 통합 관리 구조입니다.
- 관리 계정: AWS Organization을 제어하는 기본 계정입니다.
- 멤버 계정: 관리 계정을 제외한 조직 내 모든 계정입니다.
- 위임 관리자: 특정 교차 계정 관리 권한이 부여된 멤버 계정입니다. 관리 계정 내에서 AWS 서비스당 하나의 위임 관리자 계정을 선택할 수 있습니다.
- 신뢰할 수 있는 액세스: 조직 계정 전반에서 클러스터 정보에 액세스할 수 있도록 EKS 대시보드에 대한 권한을 부여합니다.
- 서비스 연결 역할(SLR): AWS 서비스에 직접 연결된 고유한 유형의 IAM 역할입니다. EKS 대시보드는 SLR을 사용하여 계정과 조직에 대한 정보를 읽습니다.
- 자세한 내용은 AWS Organizations 사용 설명서의 [Terminology and concepts](#)를 참조하세요.

일반 개요

1. AWS Organization의 관리 계정에 액세스합니다.
 - 관리 계정에 액세스하는 단계는 AWS Organization을 구성한 방법에 따라 다릅니다. 예를 들어 AWS [Identity Center](#) 또는 [Okta](#)를 통해 관리 계정에 액세스할 수 있습니다.
2. EKS 콘솔을 통해 신뢰할 수 있는 액세스를 활성화합니다.
3. AWS 계정 ID를 사용하여 위임 관리자를 할당합니다.
4. 위임 관리자 계정으로 전환합니다.
5. 조직 전체에 대한 가시성을 갖춘 향상된 EKS 콘솔에 액세스합니다.

AWS 콘솔을 사용하여 EKS 대시보드 활성화

Important

EKS 대시보드를 활성화하려면 AWS Organization의 관리 계정에 로그인해야 합니다.

EKS 대시보드 설정 액세스

1. 다음을 확인합니다.
 - a. AWS Organizations를 활성화하고 구성했습니다.

- b. 조직의 관리 계정에 로그인했습니다.
 - c. us-east-1 리전에서 AWS Management Console을 보고 있습니다.
2. EKS 콘솔로 이동합니다.
3. 왼쪽 사이드바에서 대시보드 설정을 엽니다.

Amazon EKS 대시보드에 대한 액세스 설정

1. EKS 대시보드를 볼 수 있도록 허용하려는 AWS 계정의 AWS 계정 ID를 찾습니다.
 - a. 이 단계는 선택 사항이지만 권장됩니다. 이 단계를 수행하지 않으면 관리 계정에서만 대시보드에 액세스할 수 있습니다. 관리 계정으로 액세스를 제한하는 것이 가장 좋습니다.
2. 신뢰할 수 있는 액세스 활성화를 클릭합니다.
 - a. 이제 관리 계정에서 대시보드를 볼 수 있습니다.
3. 위임 관리자 등록을 클릭하고 대시보드를 보는 데 사용할 AWS 계정의 계정 ID를 입력합니다.
 - a. 이제 위임 관리자 계정이나 관리 계정에서 대시보드를 볼 수 있습니다.

대시보드를 활성화하는 데 필요한 권한에 대한 자세한 내용은 [필요한 최소 IAM 정책](#)을 참조하세요.

EKS 대시보드 보기

1. 위임 관리자 계정(권장) 또는 관리 계정에 로그인합니다.
2. us-east-1 리전에 로그인합니다.
3. EKS 서비스로 이동하고 왼쪽 사이드바에서 대시보드를 선택합니다.

Note

[EKS 대시보드를 보는 데 필요한 IAM 권한을 검토하세요.](#)

대시보드 구성

대시보드 보기를 구성하고 리소스를 필터링할 수 있습니다.

사용 가능한 리소스

- 클러스터: EKS 클러스터의 상태와 위치에 대한 집계 정보를 봅니다.

- 상태 문제가 있는 클러스터
- EKS 추가 지원의 클러스터
- Kubernetes 버전별 클러스터 분석
- 관리형 노드 그룹: 관리형 노드 그룹과 EC2 인스턴스를 검토합니다.
 - Amazon Linux, Bottlerocket 등의 AMI 유형별 노드 그룹
 - 노드 그룹 상태 문제
 - 인스턴스 유형 분포.
- 추가 기능: 설치한 Amazon EKS 추가 기능과 그 상태에 대해 알아봅니다.
 - 추가 기능당 설치 횟수
 - 상태 문제가 있는 추가 기능
 - 추가 기능당 버전 분포

사용 가능한 보기

- 그래프 보기
 - 선택한 리소스의 그래프와 시각화를 표시하는 사용자 지정 가능한 위젯 보기
 - EKS 대시보드의 모든 사용자에게 표시되는 위젯 제거 등의 그래프 보기 변경 사항
- 리소스 보기
 - 필터를 지원하는 선택한 리소스의 목록 보기
- 맵 보기
 - 선택한 리소스의 지리적 분포

EKS 대시보드 필터링

다음은 기준으로 EKS 대시보드를 필터링할 수 있습니다.

- AWS 계정
- AWS Organizations에서 정의한 조직 단위
- AWS 리전

AWS 콘솔을 사용하여 EKS 대시보드 비활성화

1. 다음을 확인합니다.
 - a. AWS Organizations를 활성화하고 구성했습니다.
 - b. 조직의 관리 계정에 로그인했습니다.
 - c. us-east-1 리전에서 AWS Management Console을 보고 있습니다.
2. EKS 콘솔로 이동합니다.
3. 왼쪽 사이드바에서 대시보드 설정을 엽니다.
4. 신뢰할 수 있는 액세스 비활성화를 클릭합니다.

EKS 대시보드 문제 해결

EKS 대시보드 활성화 문제

- AWS Organization의 관리 계정으로 로그인해야 합니다.
 - AWS Organization이 없는 경우 하나 생성합니다. [조직을 생성하고 구성](#)하는 방법을 알아보세요.
 - AWS 계정이 이미 AWS Organization의 멤버인 경우 조직의 관리자를 식별합니다.
- AWS Organizations 리소스를 생성하고 업데이트하려면 충분한 IAM 권한이 있는 AWS 계정에 로그인해야 합니다.

EKS 대시보드 보기 문제

- 다음 AWS 계정 중 하나에 로그인해야 합니다.
 - AWS Organization의 관리 계정
 - 관리 계정의 EKS 대시보드 설정에서 식별된 위임 관리자 계정
- 최근에 EKS 대시보드를 활성화한 경우 초기 데이터 채우기에 12시간까지 걸릴 수 있습니다.

대시보드 위젯이 예기치 않게 이동

- EKS 대시보드는 AWS 계정 수준에서 구성 가능한 위젯 보기를 저장합니다. 위젯 보기를 변경하면 동일한 AWS 계정을 사용하는 다른 사용자에게 변경 사항이 표시됩니다.

AWS Organizations와의 EKS 대시보드 통합 구성

이 섹션에서는 EKS 대시보드와 AWS Organizations의 통합을 구성하기 위한 단계별 지침을 제공합니다. 서비스 간에 신뢰할 수 있는 액세스를 활성화하고 비활성화하는 방법과 위임 관리자 계정을 등록하고 등록 취소하는 방법을 알아봅니다. 각 구성 태스크는 AWS 콘솔 또는 AWS CLI를 사용하여 수행할 수 있습니다.

신뢰할 수 있는 액세스 활성화

신뢰할 수 있는 액세스는 EKS 대시보드가 조직의 모든 계정에서 클러스터 정보에 안전하게 액세스할 수 있는 권한을 부여합니다.

AWS 콘솔 사용

1. AWS Organization의 관리 계정으로 로그인합니다.
2. us-east-1 리전의 EKS 콘솔로 이동합니다.
3. 왼쪽 사이드바에서 대시보드 설정을 선택합니다.
4. 신뢰할 수 있는 액세스 활성화를 클릭합니다.

Note

EKS 콘솔을 통해 신뢰할 수 있는 액세스를 활성화하면 시스템이 `AWSServiceRoleForAmazonEKSDashboard` 서비스 연결 역할을 자동으로 생성합니다. AWS CLI 또는 AWS Organizations 콘솔을 사용하여 신뢰할 수 있는 액세스를 활성화하면 이 자동 생성이 수행되지 않습니다.

AWS CLI 사용

1. AWS Organization의 관리 계정으로 로그인합니다.
2. 다음 명령을 실행합니다.

```
aws iam create-service-linked-role --aws-service-name dashboard.eks.amazonaws.com
aws organizations enable-aws-service-access --service-principal eks.amazonaws.com
```

신뢰할 수 있는 액세스 비활성화

신뢰할 수 있는 액세스를 비활성화하면 조직의 계정 전체에서 클러스터 정보에 액세스할 수 있는 EKS 대시보드의 권한이 취소됩니다.

AWS 콘솔 사용

1. AWS Organization의 관리 계정으로 로그인합니다.
2. us-east-1 리전의 EKS 콘솔로 이동합니다.
3. 왼쪽 사이드바에서 대시보드 설정을 선택합니다.
4. 신뢰할 수 있는 액세스 비활성화를 클릭합니다.

AWS CLI 사용

1. AWS Organization의 관리 계정으로 로그인합니다.
2. 다음 명령 실행:

```
aws organizations disable-aws-service-access --service-principal eks.amazonaws.com
```

위임된 관리자 계정 활성화

위임 관리자는 EKS 대시보드에 액세스할 수 있는 권한이 부여된 멤버 계정입니다.

AWS 콘솔 사용

1. AWS Organization의 관리 계정으로 로그인합니다.
2. us-east-1 리전의 EKS 콘솔로 이동합니다.
3. 왼쪽 사이드바에서 대시보드 설정을 선택합니다.
4. 위임 관리자 등록을 클릭합니다.
5. 위임 관리자로 선택하려는 AWS 계정의 계정 ID를 입력합니다.
6. 등록을 확인합니다.

AWS CLI 사용

1. AWS Organization의 관리 계정으로 로그인합니다.
2. 123456789012를 계정 ID로 바꿔서 다음 명령을 실행합니다.

```
aws organizations register-delegated-administrator --account-id 123456789012 --
service-principal eks.amazonaws.com
```

위임 관리자 계정 비활성화

위임 관리자를 비활성화하면 EKS 대시보드에 액세스할 수 있는 계정의 권한이 제거됩니다.

AWS 콘솔 사용

1. AWS Organization의 관리 계정으로 로그인합니다.
2. us-east-1 리전의 EKS 콘솔로 이동합니다.
3. 왼쪽 사이드바에서 대시보드 설정을 선택합니다.
4. 목록에서 위임 관리자를 찾습니다.
5. 위임 관리자로 제거하려는 계정 옆의 등록 취소를 클릭합니다.

AWS CLI 사용

1. AWS Organization의 관리 계정으로 로그인합니다.
2. 다음 명령을 실행하여 123456789012를 위임 관리자의 계정 ID로 바꿉니다.

```
aws organizations deregister-delegated-administrator --account-id 123456789012 --
service-principal eks.amazonaws.com
```

필요한 최소 IAM 정책

이 섹션에서는 신뢰할 수 있는 액세스를 활성화하고 AWS Organizations와의 EKS 대시보드 통합을 위한 관리자를 위임하는 데 필요한 최소 IAM 정책을 간략하게 설명합니다.

신뢰할 수 있는 액세스를 활성화하기 위한 정책

EKS 대시보드와 AWS Organizations 간에 신뢰할 수 있는 액세스를 활성화하려면 다음 권한이 필요합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```

        "Effect": "Allow",
        "Action": [
            "organizations:EnableAWSServiceAccess",
            "organizations:DescribeOrganization",
            "organizations:ListAWSServiceAccessForOrganization"
        ],
        "Resource": "*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "iam:CreateServiceLinkedRole"
        ],
        "Resource": "arn:aws:iam::*:role/aws-service-role/
dashboard.eks.amazonaws.com/AWSServiceRoleForAmazonEKSDashboard"
    }
]
}

```

관리자 위임 정책

EKS 대시보드에 위임 관리자를 등록하거나 등록 취소하려면 다음 권한이 필요합니다.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "organizations:RegisterDelegatedAdministrator",
        "organizations:DeregisterDelegatedAdministrator",
        "organizations:ListDelegatedAdministrators"
      ],
      "Resource": "*"
    }
  ]
}

```

EKS 대시보드를 보는 정책

```

{
  "Version": "2012-10-17",
  "Statement": [

```



```

    {
      "Sid": "AmazonEKSDashboardReadOnly",
      "Effect": "Allow",
      "Action": [
        "eks:ListDashboardData",
        "eks:ListDashboardResources",
        "eks:DescribeClusterVersions"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AmazonOrganizationsReadOnly",
      "Effect": "Allow",
      "Action": [
        "organizations:DescribeOrganization",
        "organizations:ListAWSServiceAccessForOrganization",
        "organizations:ListRoots",
        "organizations:ListAccountsForParent",
        "organizations:ListOrganizationalUnitsForParent"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AmazonOrganizationsDelegatedAdmin",
      "Effect": "Allow",
      "Action": [
        "organizations:ListDelegatedAdministrators"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "organizations:ServicePrincipal": "eks.amazonaws.com"
        }
      }
    }
  ]
}

```

 Note

이러한 정책은 AWS Organization의 관리 계정의 IAM 위탁자(사용자 또는 역할)에 연결되어야 합니다. 멤버 계정은 신뢰할 수 있는 액세스를 활성화하거나 관리자를 위임할 수 없습니다.

통합 AWS 서비스를 통한 EKS 개선

다른 섹션에서 다루는 서비스 외에도 Amazon EKS는 더 많은 AWS 서비스를 사용하여 추가 솔루션을 제공합니다. 이 주제에서는 Amazon EKS를 사용하여 기능을 추가하는 몇 가지 서비스 또는 Amazon EKS에서 작업을 수행하는 데 사용하는 서비스에 대해 살펴봅니다.

주제

- [AWS CloudFormation을 사용하여 Amazon EKS 리소스 생성](#)
- [Amazon Detective를 통해 EKS에서 보안 이벤트 분석](#)
- [Amazon GuardDuty로 위협 탐지](#)
- [AWS Resilience Hub를 통해 EKS 클러스터 복원력 평가](#)
- [Security Lake를 통해 EKS 보안 데이터 중앙 집중화 및 분석](#)
- [Amazon VPC Lattice를 사용하여 클러스터 간 보안 연결 지원](#)
- [AWS 로컬 영역을 통해 지연 시간이 짧은 EKS 클러스터 시작](#)

AWS CloudFormation을 사용하여 Amazon EKS 리소스 생성

Amazon EKS는 리소스 및 인프라를 생성하고 관리하는 데 소요되는 시간을 줄일 수 있도록 AWS 리소스를 모델링하고 설정하는 데 도움이 되는 서비스인 AWS CloudFormation과 통합됩니다. 필요한 모든 AWS 리소스(예: Amazon EKS 클러스터)를 설명하는 템플릿을 생성하면 AWS CloudFormation에서 해당 리소스의 프로비저닝과 구성을 담당합니다.

AWS CloudFormation을 사용할 때 템플릿을 재사용하여 Amazon EKS 리소스를 일관되고 반복적으로 설정할 수 있습니다. 리소스를 한 번만 설명하고 여러 AWS 계정 및 리전에서 동일한 리소스를 반복적으로 프로비저닝할 수 있습니다.

Amazon EKS 및 AWS CloudFormation 템플릿

Amazon EKS 및 관련 서비스에 대한 리소스를 프로비저닝하고 구성하려면 [AWS CloudFormation 템플릿](#)을 이해해야 합니다. 템플릿은 JSON 또는 YAML로 서식 지정된 텍스트 파일입니다. 이 템플릿은 AWS CloudFormation 스택에서 프로비저닝할 리소스에 대해 설명합니다. JSON 또는 YAML에 익숙하지 않은 경우 AWS CloudFormation Designer를 사용하면 AWS CloudFormation 템플릿을 시작하는 데 도움이 됩니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 [AWS CloudFormation Designer란 무엇입니까?](#)를 참조하세요.

Amazon EKS는 AWS CloudFormation에서 클러스터 및 노드 그룹 생성을 지원합니다. Amazon EKS 리소스에 대한 JSON 및 YAML 템플릿의 예를 포함한 자세한 내용은 AWS 사용 설명서의 [Amazon EKS 리소스 유형 참조](#)를 참조하세요.

AWS CloudFormation에 대해 자세히 알아보기

AWS CloudFormation에 대한 자세한 내용은 다음 리소스를 참조하세요.

- [AWS CloudFormation](#)
- [AWS CloudFormation 사용 설명서](#)
- [AWS CloudFormation 명령줄 인터페이스 사용 설명서](#)

Amazon Detective를 통해 EKS에서 보안 이벤트 분석

[Amazon Detective](#)는 사용자가 보안 조사 결과 또는 의심스러운 활동의 근본 원인을 분석 및 조사하고 신속하게 식별하는 데 도움이 됩니다. Detective는 AWS 리소스에서 로그 데이터를 자동으로 수집합니다. 그런 다음 기계 학습, 통계 분석 및 그래프 이론을 사용하여 더 빠르고 효율적으로 보안 조사를 수행할 수 있도록 시각화를 생성합니다. Detective의 사전 구축된 데이터 집계, 요약 및 컨텍스트는 가능한 보안 문제의 특성과 범위를 신속하게 분석하고 확인하는 데 도움이 됩니다. 자세한 내용은 [Amazon Detective 사용 설명서](#)를 참조하세요.

Detective는 Kubernetes와 AWS 데이터를 다음과 같은 조사 결과로 정리합니다.

- 클러스터를 생성한 IAM ID 및 클러스터의 서비스 역할을 포함한 Amazon EKS 클러스터 세부 정보. Detective를 사용하여 이러한 IAM ID의 AWS 및 Kubernetes API 활동을 조사할 수 있습니다.
- 컨테이너 세부 정보(예: 이미지 및 보안 컨텍스트). 종료된 포드에 대한 세부 정보를 검토할 수도 있습니다.
- API 활동의 전체 추세와 특정 API 직접 호출에 대한 세부 정보를 포함한 Kubernetes API 활동. 예를 들어, 선택한 시간 범위 동안 실행된 성공 및 실패한 Kubernetes API 직접 호출 수를 표시할 수 있습니다. 또한 새로 관찰된 API 호출에 대한 섹션은 의심스러운 활동을 식별하는 데 도움이 될 수 있습니다.

Amazon EKS 감사 로그는 Detective 행동 그래프에 추가할 수 있는 선택적 데이터 소스 패키지입니다. 계정에서 사용 가능한 선택적 소스 패키지와 해당 상태를 볼 수 있습니다. 자세한 내용은 Amazon Detective 사용 설명서의 [Detective에 대한 Amazon EKS 감사 로그](#)를 참조하세요.

Amazon EKS와 함께 Amazon Detective 사용

조사 결과를 검토하기 전에 클러스터가 있는 곳과 같은 AWS 리전에서 최소 48시간 동안 Detective를 활성화해야 합니다. 자세한 내용은 [Amazon Detective 사용 설명서](#)의 Amazon Detective 설정을 참조하세요.

1. <https://console.aws.amazon.com/detective/>에서 Detective 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 검색을 선택합니다.
3. 유형 선택을 선택한 다음 EKS 클러스터를 선택합니다.
4. 클러스터 이름 또는 ARN을 입력한 다음 검색을 선택합니다.
5. 검색 결과에서 활동을 볼 클러스터 이름을 선택합니다. 확인할 수 있는 내용에 대한 자세한 내용은 Amazon Detective 사용 설명서의 [Amazon EKS 클러스터와 관련된 전체 Kubernetes API 활동](#)을 참조하세요.

Amazon GuardDuty로 위협 탐지

Amazon GuardDuty는 AWS 환경에서 계정, 컨테이너, 워크로드 및 데이터를 보호하는 데 도움이 되는 위협 감지서비스입니다. GuardDuty는 기계 학습(ML) 모델, 이상 및 위협 감지 기능을 통해 다양한 로그 소스와 런타임 활동을 지속적으로 모니터링하여 사용자 환경의 잠재적 보안 위험과 악의적 활동을 식별하고 우선순위를 지정합니다.

다른 기능 중에서도 GuardDuty는 EKS 클러스터에 대한 잠재적 위협을 탐지하는 두 가지 기능인 EKS Protection과 런타임 모니터링을 제공합니다.

Note

신규: Amazon EKS Auto Mode는 GuardDuty와 통합됩니다.

EKS 보호

이 기능은 연결된 Kubernetes 감사 로그를 모니터링하여 Amazon EKS 클러스터를 보호하는 데 도움이 되는 위협 탐지 범위를 제공합니다. Kubernetes 감사 로그는 클러스터 내에서 사용자, Kubernetes API를 사용하는 애플리케이션, 컨트롤 플레인의 활동을 포함하여 순차적인 작업을 캡처합니다. 예를 들어, GuardDuty는 Kubernetes 클러스터의 리소스를 잠재적으로 변조하기 위해 직접적으로 호출되는 API가 인증되지 않은 사용자에게 의해 간접적으로 호출되었음을 식별할 수 있습니다.

EKS Protection을 활성화하면 GuardDuty가 지속적인 위협 탐지를 위해서만 Amazon EKS 감사 로그에 액세스할 수 있습니다. GuardDuty가 클러스터에 대한 잠재적인 위협을 식별하면 특정 유형의 관련 Kubernetes 감사 로그 조사 결과를 생성합니다. Kubernetes 감사 로그에서 사용할 수 있는 조사 결과 유형에 대한 자세한 내용은 Amazon GuardDuty 사용 설명서의 [Kubernetes 감사 로그 조사 결과 유형](#)을 참조하세요.

자세한 내용은 Amazon GuardDuty 사용 설명서의 [EKS Protection](#)을 참조하세요.

런타임 모니터링

이 기능은 운영 체제 수준, 네트워킹 및 파일 이벤트를 모니터링하고 분석하여 사용자 환경의 특정 AWS 워크로드에서 잠재적인 위협을 탐지하는 데 도움이 됩니다.

런타임 모니터링을 활성화하고 Amazon EKS 클러스터에 GuardDuty 에이전트를 설치하면 GuardDuty는 이 클러스터와 연결된 런타임 이벤트 모니터링을 시작합니다. Amazon EKS Hybrid Nodes에서 GuardDuty 에이전트 및 런타임 모니터링을 사용할 수 없으므로 하이브리드 노드에서 발생하는 런타임 이벤트에는 런타임 모니터링을 사용할 수 없습니다. GuardDuty가 클러스터에 대한 잠재적인 위협을 식별하면 관련 런타임 모니터링 조사 결과를 생성합니다. 예를 들어, 취약한 웹 애플리케이션을 실행하는 단일 컨테이너를 손상시키는 것으로 위협이 시작될 수 있습니다. 이 웹 애플리케이션에 기본 컨테이너와 워크로드에 대한 액세스 권한이 있을 수 있습니다. 이 시나리오에서 자격 증명을 잘못 구성하면 계정과 그 안에 저장된 데이터에 대한 액세스 권한이 더 광범위해질 수 있습니다.

런타임 모니터링을 구성하려면 GuardDuty 에이전트를 클러스터에 Amazon EKS 추가 기능으로 설치합니다. 추가 기능에 대한 자세한 내용은 [the section called “AWS 추가 기능”](#) 섹션을 참조하세요.

자세한 내용은 Amazon GuardDuty 사용 설명서의 [Runtime Monitoring](#)을 참조하세요.

AWS Resilience Hub를 통해 EKS 클러스터 복원력 평가

AWS Resilience Hub는 인프라를 분석하여 Amazon EKS 클러스터의 복원력을 평가합니다. AWS Resilience Hub는 RBAC(Kubernetes 역할 기반 액세스 제어) 구성을 사용하여 클러스터에 배포된 Kubernetes 워크로드를 평가합니다. 자세한 내용은 AWS Resilience Hub 사용 설명서의 [Amazon EKS 클러스터에 대한 AWS Resilience Hub 액세스 활성화](#)를 참조하세요.

Security Lake를 통해 EKS 보안 데이터 중앙 집중화 및 분석

Amazon Security Lake는 Amazon EKS를 비롯한 다양한 소스의 보안 데이터를 중앙 집중화하는 데 사용할 수 있는 완전 관리형 보안 데이터 레이크 서비스입니다. Amazon EKS와 Security Lake를 통합하면 Kubernetes 리소스에서 수행되는 활동에 대한 심층적인 인사이트를 얻고 Amazon EKS 클러스터의 보안 태세를 강화할 수 있습니다.

Note

Amazon EKS와 함께 Security Lake를 사용하고 데이터 소스를 설정하는 방법에 대한 자세한 내용은 [Amazon Security Lake 설명서](#)를 참조하세요.

Amazon EKS에서 Security Lake를 사용하여 얻을 수 있는 이점

보안 데이터 중앙 집중화 - Security Lake는 다른 AWS 서비스, SaaS 공급자, 온프레미스 소스 및 타사 소스의 데이터와 함께 Amazon EKS 클러스터의 보안 데이터를 자동으로 수집하고 중앙 집중화합니다. 이를 통해 조직 전반의 보안 태세에 대한 포괄적인 뷰를 제공합니다.

데이터 형식 표준화 - Security Lake는 수집된 데이터를 표준 오픈 소스 스키마인 [OCSF\(Open Cyber Security Schema Framework\) 형식](#)으로 변환합니다. 이러한 정규화를 통해 분석과 다른 보안 도구 및 서비스와의 통합이 더 쉬워집니다.

위협 탐지 개선 - Amazon EKS 컨트롤 플레인 로그를 비롯한 중앙 집중식 보안 데이터를 분석하여 Amazon EKS 클러스터 내에서 잠재적으로 의심스러운 활동을 보다 효과적으로 탐지할 수 있습니다. 이를 통해 보안 사고를 즉시 식별하고 이에 대응할 수 있습니다.

데이터 관리 간소화 - Security Lake는 사용자 지정 가능한 보존 및 복제 설정을 통해 보안 데이터의 수명 주기를 관리합니다. 이를 통해 데이터 관리 작업이 간소화되고 규정 준수 및 감사 목적으로 필요한 데이터를 보존할 수 있습니다.

Amazon EKS에서 Security Lake 활성화

1. EKS 클러스터에서 Amazon EKS 컨트롤 플레인 로깅을 활성화합니다. 자세한 지침은 [컨트롤 플레인 로그 활성화 및 비활성화](#)를 참조하세요.
2. [Amazon EKS 감사 로그를 Security Lake의 소스로 추가합니다](#). 그러면 Security Lake가 EKS 클러스터에서 실행되고 있는 Kubernetes 리소스에서 수행된 활동에 대한 심층적인 정보를 수집하기 시작합니다.

- 요구 사항에 따라 Security Lake의 보안 데이터에 대한 [보존 및 복제 설정을 구성](#)합니다.
- Security Lake에 저장된 정규화된 OCSF 데이터를 인시던트 대응, 보안 분석 및 다른 AWS 서비스 또는 타사 도구와의 통합을 위해 사용합니다. 예를 들어, [Amazon OpenSearch Ingestion을 사용하여 Amazon Security Lake 데이터로부터 보안 인사이트를 생성](#)할 수 있습니다.

Security Lake에서 EKS 로그 분석

Security Lake는 EKS 로그 이벤트를 OCSF 형식으로 정규화하여 데이터를 더 쉽게 분석하고 다른 보안 이벤트와 상호 연관시킬 수 있도록 만듭니다. Amazon Athena, Amazon QuickSight 또는 타사 보안 분석 도구와 같은 다양한 도구 및 서비스를 사용하여 정규화된 데이터를 쿼리하고 시각화할 수 있습니다.

EKS 로그 이벤트에 대한 OCSF 매핑에 대한 자세한 내용은 OCSF GitHub 리포지토리의 <https://github.com/ocsf/examples/tree/main/mappings/markdown/AWS/v1.1.0/EKS> 감사 로그[mapping reference]를 참조하세요.

Amazon VPC Lattice를 사용하여 클러스터 간 보안 연결 지원

Amazon VPC Lattice는 AWS 네트워킹 인프라에 직접 구축된 완전관리형 애플리케이션 네트워킹 서비스로, 여러 계정과 Virtual Private Cloud(VPC)에서 서비스를 연결, 보호 및 모니터링하는 데 사용할 수 있습니다. Amazon EKS를 사용하면 Kubernetes [Gateway API](#)의 구현인 AWS Gateway API 컨트롤러를 사용하여 Amazon VPC Lattice를 활용할 수 있습니다. Amazon VPC Lattice를 사용하면 간단하고 일관된 방식으로 표준 Kubernetes 의미 체계를 사용하여 클러스터 간 연결을 설정할 수 있습니다. Amazon EKS에서 Amazon VPC Lattice를 사용하기 시작하려면 [AWS Gateway API 컨트롤러 사용 설명서](#)를 참조하세요.

AWS 로컬 영역을 통해 지연 시간이 짧은 EKS 클러스터 시작

[AWS 로컬 영역](#)은 사용자와 지리적으로 근접한 AWS 리전의 확장입니다. 로컬 영역은 인터넷에 대한 자체 연결을 가지고 있으며 [AWS Direct Connect](#)를 지원합니다. Local Zones에서 생성된 리소스는 대기 시간이 매우 짧은 통신으로 로컬 사용자에게 제공될 수 있습니다. 자세한 내용은 [AWS 로컬 영역 사용 설명서](#) 및 Amazon EC2 사용 설명서의 [로컬 영역](#)을 참조하세요.

Amazon EKS는 Local Zones의 특정 리소스를 지원합니다. 여기에는 [관리형 노드 그룹](#), [자체 관리형 Amazon EC2 노드](#), Amazon EBS 볼륨, Application Load Balancers(ALB)가 포함됩니다. Amazon EKS 클러스터의 일부로 Local Zones를 사용할 때 다음을 고려하는 것이 좋습니다.

- Amazon EKS를 사용하여 로컬 영역에서 Fargate 노드를 생성할 수 없습니다.
- Amazon EKS 관리형 Kubernetes 제어 플레인 은 항상 AWS 리전에서 실행됩니다. Amazon EKS 관리형 Kubernetes 컨트롤 플레인 은 로컬 영역에서 실행될 수 없습니다. Local Zones는 VPC 내에서 서브넷으로 표시되므로 Kubernetes는 로컬 영역 리소스를 해당 서브넷의 일부로 인식합니다.
- Amazon EKS Kubernetes 클러스터는 Amazon EKS 관리형 [탄력적 네트워크 인터페이스](#)를 사용하여 AWS 리전 또는 로컬 영역을 실행하는 Amazon EC2 인스턴스와 통신합니다. Amazon EKS 네트워킹 아키텍처에 대한 자세한 내용은 [네트워킹 구성](#) 섹션을 참조하세요.
- 리전 서브넷과 달리 Amazon EKS는 로컬 영역 서브넷에는 네트워크 인터페이스를 배치할 수 없습니다. 즉, 클러스터를 만들 때 로컬 영역 서브넷을 지정하지 않아야 합니다.

Amazon EKS 클러스터 및 노드 관련 문제 해결

이 장에서는 Amazon EKS를 사용하는 동안 발생할 수 있는 몇 가지 일반 오류와 이를 해결하는 방법을 다룹니다. 특정 Amazon EKS 영역의 문제를 해결해야 하는 경우 별도의 [the section called “문제 해결”](#), [the section called “EKS Connector 문제 해결”](#) 및 [EKS 추가 기능을 사용하여 ADOT 문제 해결](#) 주제를 참조하세요.

기타 문제 해결에 대한 자세한 내용은 AWS re:Post의 [Amazon Elastic Kubernetes 서비스에 대한 지식 센터 콘텐츠](#)를 참조하세요.

용량 부족

Amazon EKS 클러스터를 생성하려는 도중 다음 오류가 표시되는 경우 지정된 가용 영역 중 하나에 클러스터를 지원하는 충분한 용량이 없습니다.

```
Cannot create cluster 'example-cluster' because region-1d, the targeted Availability Zone, does not currently have sufficient capacity to support the cluster. Retry and choose from these Availability Zones: region-1a, region-1b, region-1c
```

이 오류 메시지를 반환한 가용 영역에서 호스팅하는 클러스터 VPC의 서브넷을 사용하여 클러스터를 재생성하세요.

클러스터가 상주할 수 없는 가용 영역이 있습니다. 서브넷이 속한 가용 영역과 [서브넷 요구 사항 및 고려 사항](#)의 가용 영역 목록을 비교합니다.

노드가 클러스터 조인에 실패

노드가 클러스터에 조인하지 못하는 몇 가지 일반적인 원인은 다음과 같습니다.

- 노드가 관리형 노드인 경우 Amazon EKS는 노드 그룹을 생성할 때 항목을 `aws-auth ConfigMap`에 추가합니다. 항목이 제거되거나 수정된 경우 항목을 다시 추가해야 합니다. 자세한 내용을 보려면 터미널에 `eksctl create iamidentitymapping --help`를 입력합니다. 다음 명령에서 `my-cluster`를 클러스터 이름으로 바꾸고 수정된 명령(`eksctl get iamidentitymapping --cluster my-cluster`)을 실행하여 현재 `aws-auth ConfigMap` 항목을 확인할 수 있습니다. 지정하는 역할의 ARN에 / 이외의 [경로](#)를 포함할 수 없습니다. 예를 들어, 역할 이름이 `development/apps/my-role`인 경우 역할에 대한 ARN을 지정할 때 역할 이름을

my-role로 변경해야 합니다. 노드 IAM 역할 ARN(인스턴스 프로파일 ARN이 아님)을 지정해야 합니다.

자체 관리형 노드이고 노드의 IAM 역할 ARN에 대한 [액세스 항목](#)을 생성하지 않은 경우 관리형 노드에 대해 나열된 것과 동일한 명령을 실행합니다. 노드 IAM 역할의 ARN에 대한 액세스 항목을 생성한 경우 액세스 항목에서 올바르게 구성되지 않을 수 있습니다. aws-auth ConfigMap 항목 또는 액세스 항목에서 노드 IAM 역할 ARN(인스턴스 프로파일 ARN이 아님)이 보안 주체 ARN으로 지정되어야 합니다. 액세스 항목에 대한 자세한 내용은 [the section called “권한 부여”](#) 섹션을 참조하세요.

- 노드 AWS CloudFormation 템플릿에서 ClusterName이 조인할 클러스터의 이름과 정확히 일치하지 않습니다. 이 필드에 올바르지 않은 값을 전달하면 노드의 /var/lib/kubelet/kubeconfig 파일의 올바르지 않은 구성이 발생하고, 노드가 클러스터에 조인되지 않습니다.
- 노드가 클러스터에서 소유하는 것으로 태그가 지정되지 않았습니다. 노드에는 다음 태그가 적용되어야 하며, 여기서 *my-cluster*는 클러스터의 이름으로 대체됩니다.

키	값
kubernetes.io/cluster/ <i>my-cluster</i>	owned

- 노드는 퍼블릭 IP 주소를 사용하여 클러스터에 액세스하지 못할 수 있습니다. 퍼블릭 서브넷에 배포된 노드에 퍼블릭 IP 주소가 할당되었는지 확인합니다. 그렇지 않은 경우 시작 후 탄력적 IP 주소를 노드에 연결할 수 있습니다. 자세한 내용은 [실행 중인 인스턴스 또는 네트워크 인터페이스와 탄력적 IP 주소 연결](#)을 참조하세요. 퍼블릭 IP 주소를 배포된 인스턴스에 자동으로 할당하도록 퍼블릭 서브넷이 설정되어 있지 않은 경우, 해당 설정을 사용하도록 설정하는 것이 좋습니다. 자세한 내용은 [서브넷의 퍼블릭 IPv4 주소 지정 속성 수정](#) 섹션을 참조하세요. 노드가 프라이빗 서브넷에 배포된 경우, 서브넷에는 퍼블릭 IP 주소가 할당된 NAT 게이트웨이에 대한 경로가 있어야 합니다.
- 노드를 배포할 AWS 리전의 AWS STS 엔드포인트가 계정에 대해 활성화되지 않았습니다. 리전을 활성화하려면 [AWS 리전에서 AWS STS 활성화 및 비활성화](#)를 참조하세요.
- 노드에 프라이빗 DNS 항목이 없으므로 kubelet 로그에 node "" not found 오류가 포함됩니다. 노드가 생성된 VPC에 대해 DHCP options set에서 Options로 domain-name 및 domain-name-servers 값이 설정되었는지 확인합니다. 기본값은 domain-name:<region>.compute.internal 및 domain-name-servers:AmazonProvidedDNS입니다. 자세한 내용은 Amazon VPC 사용 설명서에서 [DHCP 옵션 세트](#)를 참조하세요.
- 관리형 노드 그룹의 노드가 15분 내에 클러스터에 연결되지 않으면 “NodeCreationFailure”라는 상태 문제가 발생하고 콘솔 상태가 Create failed로 설정됩니다. 시작 시간이 느린 Windows AMI의 경우 [빠른 실행](#)을 사용하여 이 문제를 해결할 수 있습니다.

작업자 노드의 클러스터 조인을 방해하는 일반적인 원인을 식별하고 문제를 해결하기 위해 `AWSSupport-TroubleshootEKSTaskNode` 실행서를 사용할 수 있습니다. 자세한 내용은 AWS 시스템 관리자 자동화 런북 참조에서 [AWSSupport-TroubleshootEKSTaskNode](#) 를 참조하세요.

권한이 없거나 액세스가 거부됨(kubectl)

`kubectl` 명령을 실행할 때 다음 오류 중 하나가 발생하는 경우 `kubectl`이 Amazon EKS에 대해 올바르게 구성되지 않았거나 사용하고 있는 IAM 위탁자(역할 또는 사용자)의 자격 증명이 Amazon EKS 클러스터의 Kubernetes 객체에 대해 충분한 권한을 보유한 Kubernetes 사용자 이름에 매핑되지 않은 것입니다.

- `could not get token: AccessDenied: Access denied`
- `error: You must be logged in to the server (Unauthorized)`
- `error: the server doesn't have a resource type "svc"`

이유는 다음 중 하나 때문일 수 있습니다.

- 클러스터가 하나의 IAM 보안 주체에 대한 보안 인증 정보로 생성되었지만 `kubectl`이 다른 IAM 보안 주체의 보안 인증 정보를 사용하도록 구성되었기 때문입니다. 이 문제를 해결하려면 클러스터를 생성한 보안 인증 정보를 사용하도록 `kube config` 파일을 업데이트합니다. 자세한 내용은 [the section called “kubectl을 사용하여 클러스터에 액세스”](#) 섹션을 참조하세요.
- 클러스터가 [EKS 액세스 항목으로 IAM 사용자에게 Kubernetes 액세스 권한 부여](#)의 전제 조건 섹션에 있는 최소 플랫폼 요구 사항을 충족하는 경우, 액세스 항목이 IAM 위탁자에 존재하지 않습니다. 존재하는 경우 필수 Kubernetes 그룹 이름이 정의되지 않았거나 적절한 액세스 정책이 연결되지 않은 것입니다. 자세한 내용은 [the section called “권한 부여”](#) 섹션을 참조하세요.
- 클러스터가 [EKS 액세스 항목으로 IAM 사용자에게 Kubernetes 액세스 권한 부여](#)의 최소 플랫폼 요구 사항을 충족하지 않는 경우, IAM 위탁자가 있는 항목이 `aws-auth ConfigMap`에 존재하지 않습니다. 존재하는 경우 필요한 권한을 보유한 Kubernetes Role 또는 ClusterRole에 바인딩된 Kubernetes 그룹 이름에 매핑되지 않은 것입니다. Kubernetes 역할 기반 권한 부여 (RBAC) 객체에 대한 자세한 내용은 Kubernetes Documentation의 [Using RBAC authorization](#)을 참조하세요. 다음 명령에서 `my-cluster`를 클러스터 이름으로 바꾸고 수정된 명령(`eksctl get iamidentitymapping --cluster my-cluster`)을 실행하여 현재 `aws-auth ConfigMap` 항목을 확인할 수 있습니다. IAM 위탁자의 ARN이 포함된 항목이 `ConfigMap`에 없는 경우 터미널에 `eksctl create iamidentitymapping --help`를 입력하여 생성 방법을 알아보세요.

AWS CLI를 설치하고 구성하면 사용할 IAM 자격 증명을 구성할 수 있습니다. 자세한 내용은 AWS 명령줄 인터페이스 사용 설명서에서 [AWS CLI 구성](#)을 참조하세요. 클러스터의 Kubernetes 객체에 액세스하기 위해 IAM 역할을 수입하는 경우 IAM 역할을 사용하도록 kubectl을 구성할 수도 있습니다. 자세한 내용은 [the section called “kubectl을 사용하여 클러스터에 액세스”](#) 섹션을 참조하세요.

hostname doesn't match

시스템의 Python 버전이 2.7.9 이상이어야 합니다. 그렇지 않은 경우 Amazon EKS에 대한 AWS CLI 직접 호출과 함께 hostname doesn't match 오류가 표시됩니다. 자세한 내용은 Python 요청 자주 묻는 질문에서 [“호스트 이름이 일치하지 않습니다” 오류란 무엇인가요?](#)를 참조하세요.

getsockopt: no route to host

Docker는 Amazon EKS 클러스터의 172.17.0.0/16 CIDR 범위에서 실행됩니다. 클러스터의 VPC 서브넷이 이 범위와 겹치지 않게 하는 것이 좋습니다. 그렇지 않으면 다음 오류가 발생합니다.

```
Error: : error upgrading connection: error dialing backend: dial tcp
172.17.<nn>.<nn>:10250: getsockopt: no route to host
```

Instances failed to join the Kubernetes cluster

AWS Management Console에서 Instances failed to join the Kubernetes cluster 오류 메시지가 표시되면 클러스터의 프라이빗 엔드포인트 액세스가 활성화되어 있는지 또는 퍼블릭 엔드포인트 액세스에 대해 CIDR 블록을 올바르게 구성했는지 확인합니다. 자세한 내용은 [the section called “클러스터 엔드포인트 액세스”](#) 섹션을 참조하세요.

관리형 노드 그룹 오류

관리형 노드 그룹에 하드웨어 상태 문제가 발생하면 Amazon EKS는 문제 진단에 도움이 되는 오류 메시지를 반환합니다. 이러한 상태 확인은 [Amazon EC2 상태 확인](#)을 기반으로 하기 때문에 소프트웨어 문제를 감지하지 못합니다. 다음 목록에서는 오류 코드에 대해 설명합니다.

AccessDenied

Amazon EKS 또는 하나 이상의 관리형 노드가 Kubernetes 클러스터 API 서버로 인증하거나 인증하지 못합니다. 일반적인 응답 헤더에 대한 자세한 내용은 [the section called “관리형 노드 그룹의](#)

[일반적인AccessDenied 오류 원인 해결](#)을 참조하세요. 프라이빗 Windows AMI로 인해 오류 메시지와 함께 Not authorized for images 오류 코드가 발생할 수도 있습니다. 자세한 내용은 [the section called “Not authorized for images”](#) 섹션을 참조하세요.

AmildNotFound

시작 템플릿과 연결된 AMI ID를 찾을 수 없습니다. AMI가 존재하고 계정과 공유되는지 확인합니다.

AutoScalingGroupNotFound

관리형 노드 그룹과 연결된 Auto Scaling 그룹을 찾을 수 없습니다. 복구하려면 동일한 설정으로 Auto Scaling 그룹을 다시 만들어야 합니다.

ClusterUnreachable

Amazon EKS 또는 하나 이상의 관리형 노드가 Kubernetes 클러스터 API 서버와 통신할 수 없습니다. 네트워크 중단이 있거나 API 서버가 요청 처리 시간을 초과한 경우 이 문제가 발생할 수 있습니다.

Ec2SecurityGroupNotFound

클러스터의 클러스터 보안 그룹을 찾을 수 없습니다. 클러스터를 다시 생성해야 합니다.

Ec2SecurityGroupDeletionFailure

관리형 노드 그룹의 원격 액세스 보안 그룹을 삭제할 수 없습니다. 보안 그룹에서 종속성을 제거합니다.

Ec2LaunchTemplateNotFound

관리형 노드 그룹의 Amazon EC2 시작 템플릿을 찾을 수 없습니다. 복구하려면 노드 그룹을 다시 생성해야 합니다.

Ec2LaunchTemplateVersionMismatch

관리형 노드 그룹의 Amazon EC2 시작 템플릿 버전이 Amazon EKS에서 생성한 버전과 일치하지 않습니다. 복구하려면 Amazon EKS에서 생성한 버전으로 되돌려야 합니다.

IamInstanceProfileNotFound

관리형 노드 그룹의 IAM 인스턴스 프로필을 찾을 수 없습니다. 복구하려면 동일한 설정으로 인스턴스 프로필을 다시 만들어야 합니다.

IamNodeRoleNotFound

관리형 노드 그룹의 IAM 역할을 찾을 수 없습니다. 복구하려면 동일한 설정으로 IAM 역할을 다시 만들어야 합니다.

AsgInstanceLaunchFailures

인스턴스를 시작하려고 시도하는 중에 Auto Scaling 그룹에 오류가 발생했습니다.

NodeCreationFailure

시작된 인스턴스를 Amazon EKS 클러스터에 등록할 수 없습니다. 이러한 오류의 일반적인 원인은 [노드 IAM 역할](#) 권한이 충분하지 않거나 노드에 대한 아웃바운드 인터넷 액세스가 부족하기 때문입니다. 노드는 다음 요구 사항 중 하나를 충족해야 합니다.

- 퍼블릭 IP 주소를 사용하여 인터넷에 액세스할 수 있습니다. 노드가 있는 서브넷에 연결된 보안 그룹은 통신을 허용해야 합니다. 자세한 내용은 [the section called “서브넷 요구 사항 및 고려 사항”](#) 및 [the section called “보안 그룹 요구 사항”](#) 섹션을 참조하세요.
- 노드 및 VPC는 [인터넷 액세스가 제한된 프라이빗 클러스터 배포](#)의 요구 사항을 충족해야 합니다.

InstanceLimitExceeded

AWS 계정에서 해당 인스턴스 유형의 인스턴스를 더 이상 시작할 수 없습니다. 복구하려면 Amazon EC2 인스턴스 제한의 증가를 요청해야 합니다.

InsufficientFreeAddresses

관리형 노드 그룹과 연결된 하나 이상의 서브넷에 새 노드에 사용 가능한 IP 주소가 충분하지 않습니다.

InternalFailure

이러한 오류는 일반적으로 Amazon EKS 서버 측 문제로 인해 발생합니다.

관리형 노드 그룹의 일반적인 **AccessDenied** 오류 원인 해결

AccessDenied 오류의 가장 흔한 원인은 관리형 노드 그룹에서 작업을 수행할 때 `eks:node-manager` ClusterRole 또는 ClusterRoleBinding이 누락되는 것입니다. Amazon EKS는 관리형 노드 그룹을 사용한 온보딩의 일부로 클러스터에 이러한 리소스를 설정하며 이러한 리소스는 노드 그룹을 관리하는 데 필요합니다.

ClusterRole은 시간이 지남에 따라 변경될 수 있지만 다음 예제와 비슷해야 합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
```

```
  name: eks:node-manager
rules:
- apiGroups:
  - ''
  resources:
  - pods
  verbs:
  - get
  - list
  - watch
  - delete
- apiGroups:
  - ''
  resources:
  - nodes
  verbs:
  - get
  - list
  - watch
  - patch
- apiGroups:
  - ''
  resources:
  - pods/eviction
  verbs:
  - create
```

ClusterRoleBinding은 시간이 지남에 따라 변경될 수 있지만 다음 예제와 비슷해야 합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks:node-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:node-manager
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: eks:node-manager
```

eks:node-manager ClusterRole이 존재하는지 확인합니다.


```
kubectl describe clusterrole eks:node-manager
```

존재하는 경우, 출력을 이전 ClusterRole 예와 비교합니다.

eks:node-manager ClusterRoleBinding이 존재하는지 확인합니다.

```
kubectl describe clusterrolebinding eks:node-manager
```

존재하는 경우, 출력을 이전 ClusterRoleBinding 예와 비교합니다.

관리형 노드 그룹 작업을 요청하는 동안 누락되거나 손상된 ClusterRole 또는 ClusterRoleBinding이 AccessDenied 오류의 원인으로 확인된 경우 복원할 수 있습니다. 다음 내용을 *eks-node-manager-role.yaml*이라는 파일에 저장합니다.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: eks:node-manager
rules:
- apiGroups:
  - ''
  resources:
  - pods
  verbs:
  - get
  - list
  - watch
  - delete
- apiGroups:
  - ''
  resources:
  - nodes
  verbs:
  - get
  - list
  - watch
  - patch
- apiGroups:
  - ''
  resources:
  - pods/eviction
  verbs:
```

```

- create
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks:node-manager
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:node-manager
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: User
  name: eks:node-manager

```

파일을 적용합니다.

```
kubectl apply -f eks-node-manager-role.yaml
```

노드 그룹 작업을 다시 시도하여 문제가 해결되었는지 확인합니다.

Not authorized for images

Not authorized for images 오류 메시지의 잠재적 원인 중 하나는 프라이빗 Amazon EKS Windows AMI를 사용하여 Windows 관리형 노드 그룹을 시작하는 것입니다. 새 Windows AMI를 릴리스한 후 AWS에서는 4개월이 지난 AMI를 프라이빗으로 설정하여 더 이상 액세스할 수 없도록 합니다. 관리형 노드 그룹에서 프라이빗 Windows AMI를 사용하는 경우 [Windows 관리형 노드 그룹을 업데이트](#)하는 것이 좋습니다. 프라이빗으로 설정된 AMI에 대한 액세스 제공을 보장할 수는 없지만 AWS Support에 티켓을 제출하여 액세스를 요청할 수 있습니다. 자세한 내용은 Amazon EC2 사용 설명서의 [패치](#)를 참조하세요.

노드가 NotReady 상태임

노드가 NotReady 상태인 경우 해당 노드가 비정상이어서 새 포드를 예약할 수 없을 가능성이 있습니다. 이는 노드에 CPU 리소스, 메모리 또는 사용 가능한 디스크 공간이 충분하지 않은 등의 다양한 이유로 발생할 수 있습니다.

Amazon EKS 최적화 Windows AMI의 경우 kubelet 구성에 기본적으로 지정된 컴퓨팅 리소스를 예약할 수 없습니다. 리소스 문제를 방지하기 위해 kubelet에 [kube-reserved](#) 및/또는 [system-reserved](#)에 대한 구성 값을 제공하여 시스템 프로세스를 위한 컴퓨팅 리소스를 예약할 수 있습니다. 부

트스트랩 스크립트의 `-KubeletExtraArgs` 명령줄 파라미터를 사용하여 이를 수행합니다. 자세한 내용은 Kubernetes Documentation의 [Reserve Compute Resources for System Daemons](#) 및 이 사용자 가이드의 [the section called “부트스트랩 스크립트 구성 파라미터”](#)을 참조하세요.

EKS 로그 수집기

Amazon EKS 노드 관련 문제를 해결하기 위해 노드에서 사용할 수 있는 사전 구축된 스크립트가 있습니다. 이 스크립트는 `/etc/eks/log-collector-script/eks-log-collector.sh`에 있습니다. 이 스크립트를 사용하여 지원 사례 및 일반 문제 해결에 대한 진단 로그를 수집할 수 있습니다.

다음 명령을 사용하여 노드에서 스크립트를 실행합니다.

```
sudo bash /etc/eks/log-collector-script/eks-log-collector.sh
```

Note

스크립트가 해당 위치에 없는 경우 다음 명령을 사용하여 스크립트를 수동으로 다운로드하고 실행할 수 있습니다.

```
curl -O https://amazon-eks.s3.amazonaws.com/support/log-collector-script/linux/eks-log-collector.sh
sudo bash eks-log-collector.sh
```

이 스크립트는 다음 진단 정보를 수집합니다.

```
$ sudo bash /etc/eks/log-collector-script/eks-log-collector.sh
```

```
    This is version 0.7.8. New versions can be found at https://github.com/aws-labs/
amazon-eks-ami/blob/main/log-collector-script/
```

```
Trying to collect common operating system logs...
```

```
Trying to collect kernel logs...
```

```
Trying to collect mount points and volume information...
```

```
...
```

```
...
```

```
Done... your bundled logs are located in /var/log/eks_i-EXAMPLE_2025-03-25_0000-
UTC_0.7.8.tar.gz
```

진단 정보는 다음에 수집되고 저장됩니다.

```
/var/log/eks_i-EXAMPLE_2025-03-25_0000-UTC_0.7.8.tar.gz
```

Bottlerocket 노드의 로그 번들을 검색하려는 경우 자세한 내용은 [Bottlerocket 로그](#)를 참조하세요.

컨테이너 런타임 네트워크가 준비되지 않음

다음과 같은 Container runtime network not ready 오류 및 권한 부여 오류가 나타날 수 있습니다.

```
4191 kubelet.go:2130] Container runtime network not ready: NetworkReady=false
reason:NetworkPluginNotReady message:docker: network plugin is not ready: cni config
uninitialized
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
*v1.Service: Unauthorized
4191 kubelet_node_status.go:106] Unable to register node
"ip-10-40-175-122.ec2.internal" with API server: Unauthorized
4191 reflector.go:205] k8s.io/kubernetes/pkg/kubelet/kubelet.go:452: Failed to list
*v1.Service: Unauthorized
```

이유는 다음 중 하나 때문일 수 있습니다.

1. 클러스터에 aws-auth ConfigMap이 없거나 노드를 구성하는 데 사용한 IAM 역할 항목이 포함되어 있지 않습니다.

이 문제를 해결하기 위해 다음 명령에서 *my-cluster*를 클러스터 이름으로 바꾸고 수정된 명령(`eksctl get iamidentitymapping --cluster my-cluster`)을 실행하여 ConfigMap에서 기존 항목을 확인합니다. 명령에서 오류 메시지를 수신한 경우 클러스터에 aws-auth ConfigMap이 없기 때문일 수 있습니다. 다음 명령은 ConfigMap에 항목을 추가합니다. ConfigMap이 없으면 명령에서도 이를 생성합니다. *111122223333*을 IAM 역할의 AWS 계정 ID로, *myAmazonEKSNoderole*을 노드 역할 이름으로 바꾸세요.

```
eksctl create iamidentitymapping --cluster my-cluster \
  --arn arn:aws:iam::111122223333:role/myAmazonEKSNoderole --group
system:bootstrappers,system:nodes \
  --username system:node:{{EC2PrivateDNSName}}
```

지정하는 역할의 ARN에 / 이외의 [경로](#)를 포함할 수 없습니다. 예를 들어, 역할 이름이 development/apps/my-role인 경우 역할에 대한 ARN을 지정할 때 역할 이름을 my-role로 변경해야 합니다. 노드 IAM 역할 ARN(인스턴스 프로파일 ARN이 아님)을 지정해야 합니다.

2. 자체 관리 노드가 플랫폼 버전이 [EKS 액세스 항목으로 IAM 사용자에게 Kubernetes 액세스 권한 부여](#) 주제의 전제 조건에 나열된 최소 버전인 클러스터에 있지만, 항목이 노드의 IAM 역할에 대한 aws-auth ConfigMap(이전 항목 참조)에 나열되지 않았거나 역할에 대한 액세스 항목이 존재하지 않습니다. 이 문제를 해결하기 위해 다음 명령에서 *my-cluster*를 클러스터 이름으로 바꾸고 수정된 명령(aws eks list-access-entries --cluster-name *my-cluster*)을 실행하여 기존 액세스 항목을 확인합니다. 다음 명령은 노드의 IAM 역할에 대한 액세스 항목을 추가합니다. *111122223333*을 IAM 역할의 AWS 계정 ID로, *myAmazonEKSNodeRole*을 노드 역할 이름으로 바꾸세요. Windows 노드가 있는 경우 *EC2_Linux*를 EC2_Windows로 바꾸세요. 노드 IAM 역할 ARN(인스턴스 프로파일 ARN이 아님)을 지정해야 합니다.

```
aws eks create-access-entry --cluster-name my-cluster --principal-arn
arn:aws:iam::111122223333:role/myAmazonEKSNodeRole --type EC2_LINUX
```

TLS 핸드셰이크 시간 초과

노드에서 퍼블릭 API 서버 엔드포인트에 연결할 수 없는 경우 다음과 유사한 오류가 발생할 수 있습니다.

```
server.go:233] failed to run Kubelet: could not init cloud provider "aws": error
finding instance i-1111f2222f333e44c: "error listing AWS instances: \"RequestError:
send request failed\\ncaused by: Post net/http: TLS handshake timeout\""
```

kubelet 프로세스가 지속적으로 다시 생성되어 API 서버 엔드포인트를 테스트합니다. 이 오류는 구성 변경 또는 버전 업데이트와 같이 컨트롤 플레인에서 클러스터의 롤링 업데이트를 수행하는 절차 중에 일시적으로 발생할 수도 있습니다.

이 문제를 해결하려면 라우팅 테이블과 보안 그룹을 점검하여 노드의 트래픽이 퍼블릭 엔드포인트에 도달할 수 있는지 확인합니다.

InvalidClientTokenId

중국 AWS 리전에 배포된 포드 또는 DaemonSet의 서비스 계정에 IAM 역할을 사용하고 사양에 `AWS_DEFAULT_REGION` 환경 변수를 설정하지 않은 경우 포드 또는 DaemonSet에 다음과 같은 오류가 발생할 수 있습니다.

```
An error occurred (InvalidClientTokenId) when calling the GetCallerIdentity operation:
The security token included in the request is invalid
```

이 문제를 해결하려면 다음 예제 포드 사양에 표시된 것처럼 포드 또는 DaemonSet 사양에 `AWS_DEFAULT_REGION` 환경 변수를 추가해야 합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: envar-demo
  labels:
    purpose: demonstrate-envvars
spec:
  containers:
    - name: envar-demo-container
      image: gcr.io/google-samples/node-hello:1.0
      env:
        - name: AWS_DEFAULT_REGION
          value: "region-code"
```

컨트롤 플레인을 업데이트하기 전에 노드 그룹이 Kubernetes 버전과 일치해야 합니다.

컨트롤 플레인을 새 Kubernetes 버전으로 업그레이드하기 전에 클러스터에 있는 관리형 노드 및 Fargate 노드의 Kubernetes 마이너 버전이 컨트롤 플레인의 현재 버전과 동일해야 합니다. Amazon EKS `update-cluster-version` API는 모든 Amazon EKS 관리형 노드를 현재 클러스터 버전으로 업그레이드할 때까지 요청을 거부합니다. Amazon EKS는 관리형 노드 업그레이드를 위한 API를 제공합니다. 관리형 노드 그룹의 Kubernetes 버전 업그레이드에 대한 자세한 내용은 [the section called “업데이트”](#) 섹션을 참조하세요. Fargate 노드의 버전을 업그레이드하려면 컨트롤 플레인을 업그레이드한 후 노드가 나타내는 포드를 삭제하고 포드를 다시 배포합니다. 자세한 내용은 [the section called “Kubernetes 버전 업데이트”](#) 섹션을 참조하세요.

많은 노드를 시작할 때 Too Many Requests 오류가 있습니다.

많은 노드를 동시에 시작하는 경우 [Amazon EC2 사용자 데이터](#) 실행 로그에 Too Many Requests라는 오류 메시지가 표시될 수 있습니다. 컨트롤 플레인 이 describeCluster 직접 호출로 오버로드되기 때문일 수 있습니다. 오버로드는 제한, 부트스트랩 스크립트를 실행하지 못하는 노드, 클러스터에 참여하지 못하는 노드를 초래합니다.

--apiserver-endpoint, --b64-cluster-ca 및 --dns-cluster-ip 인수가 노드의 부트스트랩 스크립트에 전달되고 있는지 확인합니다. 이러한 인수를 포함하면 부트스트랩 스크립트가 describeCluster를 직접 호출할 필요가 없으므로 컨트롤 플레인 오버로드를 방지하는 데 도움이 됩니다. 자세한 내용은 [the section called “Amazon EKS 최적화 Linux/Bottlerocket AMI에 포함된 bootstrap.sh 파일에 인수를 전달하기 위해 사용자 데이터를 제공함”](#) 섹션을 참조하세요.

Kubernetes API 서버 요청에 대한 HTTP 401 권한 없음 오류 응답

이러한 오류는 포드의 서비스 계정 토큰이 클러스터에서 만료된 경우 표시됩니다.

Amazon EKS 클러스터의 Kubernetes API 서버는 90일이 지난 토큰을 사용한 요청을 거부합니다. 이전 Kubernetes 버전에는 토큰에 만료 기간이 없었습니다. 즉, 이러한 토큰에 의존하는 클라이언트는 1 시간 이내에 토큰을 새로 고쳐야 합니다. Kubernetes API 서버가 유효하지 않은 토큰으로 인해 요청을 거부하는 것을 방지하려면 워크로드에서 사용하는 [Kubernetes 클라이언트 SDK](#) 버전은 다음 버전과 동일하거나 이상 버전이어야 합니다.

- Go 버전 0.15.7 이상
- Python 버전 12.0.0 이상
- Java 버전 9.0.0 이상
- JavaScript 버전 0.10.3 이상
- Ruby master 브랜치
- Haskell 버전 0.3.0.0
- C# 버전 7.0.5 이상

오래된 토큰을 사용하는 클러스터의 모든 기존 포드를 식별할 수 있습니다. 자세한 내용은 [the section called “서비스 계정 토큰”](#) 섹션을 참조하세요.

Amazon EKS 플랫폼 버전이 현재 플랫폼 버전보다 두 버전 이상 뒤떨어져 있음

이는 Amazon EKS가 클러스터의 [플랫폼 버전](#)을 자동으로 업데이트할 수 없는 경우에 발생할 수 있습니다. 여기에는 여러 가지 원인이 있지만 일반적인 원인 중 일부는 다음과 같습니다. 이러한 문제가 클러스터에 적용되는 경우 여전히 작동할 수 있으며 플랫폼 버전은 Amazon EKS에서 업데이트되지 않습니다.

문제

[클러스터 IAM 역할](#)이 삭제됨 – 이 역할은 클러스터가 생성될 때 지정되었습니다. 다음 명령을 사용하여 지정된 역할을 확인할 수 있습니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks describe-cluster --name my-cluster --query cluster.roleArn --output text | cut -d / -f 2
```

예제 출력은 다음과 같습니다.

```
eksClusterRole
```

Solution

동일한 이름으로 새 [클러스터 IAM 역할](#)을 생성합니다.

문제

클러스터 생성 중에 지정된 서브넷이 삭제됨 - 클러스터와 함께 사용할 서브넷이 클러스터 생성 중에 지정되었습니다. 다음 명령을 사용하여 지정된 서브넷을 확인할 수 있습니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks describe-cluster --name my-cluster --query cluster.resourcesVpcConfig.subnetIds
```

예제 출력은 다음과 같습니다.

```
[  
  "subnet-EXAMPLE1",  
  "subnet-EXAMPLE2"  
]
```

Solution

계정에 서브넷 ID가 있는지 확인합니다.

```
vpc_id=$(aws eks describe-cluster --name my-cluster --query
cluster.resourcesVpcConfig.vpcId --output text)
aws ec2 describe-subnets --filters "Name=vpc-id,Values=$vpc_id" --query
"Subnets[*].SubnetId"
```

예제 출력은 다음과 같습니다.

```
[
  "subnet-EXAMPLE3",
  "subnet-EXAMPLE4"
]
```

출력에 반환된 서브넷 ID가 클러스터 생성 시 지정된 서브넷 ID와 일치하지 않는 경우 Amazon EKS가 클러스터를 업데이트하도록 하려면 클러스터에서 사용하는 서브넷을 변경해야 합니다. 클러스터를 생성할 때 서브넷을 3개 이상 지정한 경우 Amazon EKS는 새로운 탄력적 네트워크 인터페이스를 생성하기 위해 지정한 서브넷을 무작위로 선택하기 때문입니다. 이러한 네트워크 인터페이스를 사용하면 컨트롤 플레인 노드와 통신할 수 있습니다. Amazon EKS는 선택한 서브넷이 없는 경우 클러스터를 업데이트하지 않습니다. Amazon EKS가 새 네트워크 인터페이스를 생성하기 위해 선택하는 클러스터 생성 시 지정한 서브넷을 컨트롤할 수 없습니다.

클러스터에 대한 Kubernetes 버전 업데이트를 시작하면 같은 이유로 업데이트가 실패할 수 있습니다.

문제

클러스터 생성 중에 지정된 보안 그룹이 삭제됨 - 클러스터 생성 중에 보안 그룹을 지정한 경우 다음 명령을 사용하여 보안 그룹의 ID를 볼 수 있습니다. *my-cluster*를 해당 클러스터의 이름으로 바꿉니다.

```
aws eks describe-cluster --name my-cluster --query
cluster.resourcesVpcConfig.securityGroupIds
```

예제 출력은 다음과 같습니다.

```
[
  "sg-EXAMPLE1"
]
```

[]이 반환되면 클러스터 생성 시 보안 그룹이 지정되지 않았으며 보안 그룹이 누락된 것이 문제가 아닙니다. 보안 그룹이 반환되면 계정에 보안 그룹이 있는지 확인합니다.

Solution

계정에 이러한 보안 그룹이 있는지 확인합니다.

```
vpc_id=$(aws eks describe-cluster --name my-cluster --query
cluster.resourcesVpcConfig.vpcId --output text)
aws ec2 describe-security-groups --filters "Name=vpc-id,Values=$vpc_id" --query
"SecurityGroups[*].GroupId"
```

예제 출력은 다음과 같습니다.

```
[
  "sg-EXAMPLE2"
]
```

출력에 반환된 보안 그룹 ID가 클러스터 생성 시 지정된 보안 그룹 ID와 일치하지 않는 경우 Amazon EKS가 클러스터를 업데이트하도록 하려면 클러스터에서 사용하는 보안 그룹을 변경해야 합니다. 클러스터 생성 시 지정된 보안 그룹 ID가 없는 경우 Amazon EKS는 클러스터를 업데이트하지 않습니다.

클러스터에 대한 Kubernetes 버전 업데이트를 시작하면 같은 이유로 업데이트가 실패할 수 있습니다.

- 클러스터를 생성할 때 지정한 각 서브넷에 최소 6개(16개 권장)의 사용 가능한 IP 주소가 없습니다. 서브넷에 사용 가능한 IP 주소가 충분하지 않은 경우 서브넷에서 IP 주소를 확보하거나 클러스터에서 사용하는 서브넷을 변경하여 가용 IP 주소가 충분한 서브넷을 사용해야 합니다.
- 클러스터를 생성할 때 [암호 암호화](#)를 활성화했으며 지정한 AWS KMS 키가 삭제되었습니다. Amazon EKS가 클러스터를 업데이트하도록 하려면 새 클러스터를 생성해야 합니다.

클러스터 상태 FAQ 및 오류 코드(해결 경로 포함)

Amazon EKS는 EKS 클러스터 및 클러스터 인프라에서 문제를 감지하여 EKS 클러스터 리소스의 상태 객체에 저장합니다. 클러스터 상태 정보를 사용하여 클러스터 문제를 더 빠르게 감지하고 해결할 수 있습니다. 이를 통해 보다 안전하고 최신 상태인 애플리케이션 환경을 구축할 수 있습니다. 또한 필요한 인프라 또는 클러스터 구성 문제로 인해 Amazon EKS에서 성능이 저하된 클러스터에 보안 업데이트를 설치하거나 Kubernetes의 최신 버전으로 업그레이드할 수 없습니다. Amazon EKS에서 문제 자체 또는 문제가 해결되었음을 감지하는 데 3시간이 걸릴 수 있습니다.

Amazon EKS 클러스터의 상태에 대해서는 Amazon EKS와 해당 사용자가 공동으로 책임을 집니다. 사용자는 IAM 역할 및 Amazon VPC 서브넷의 필수 인프라와 사전에 제공해야 하는 기타 필수 인프라를 책임져야 합니다. Amazon EKS는 이 인프라 및 클러스터의 구성 변경을 감지합니다.

Amazon EKS 콘솔에서 클러스터 상태에 액세스하려면 Amazon EKS 클러스터 세부 정보 페이지에서 액세스하는 관찰성 대시보드의 클러스터 상태 문제 탭에서 상태 문제라는 테이블을 찾습니다. 이 데이터는 EKS API에서 `DescribeCluster` 작업을 직접 호출하여 사용할 수도 있습니다(예: AWS CLI 내에서 수행).

왜 이 기능을 사용해야 하나요?

디버깅하거나 AWS 지원 사례를 여는 데 시간을 소비하지 않고도 Amazon EKS 클러스터의 상태를 더 잘 파악하고 문제를 신속하게 진단 및 수정할 수 있습니다. 예: 실수로 Amazon EKS 클러스터의 서브넷을 삭제한 경우 Amazon EKS는 크로스 계정 네트워크 인터페이스 및 Kubernetes AWS CLI 명령(예: `kubectl exec`) 또는 `kubectl` 로그를 생성할 수 없습니다. 다음과 같은 오류와 함께 실패합니다. `Error from server: error dialing backend: remote error: tls: internal error`. 이제 다음과 같은 Amazon EKS 상태 문제가 표시됩니다. `subnet-da60e280 was deleted: could not create network interface`.

이 기능은 다른 AWS 서비스와 어떻게 관련되었거나 어떻게 작동하나요?

IAM 역할과 Amazon VPC 서브넷은 클러스터 상태에서 문제를 감지하는 필수 인프라의 두 가지 예입니다. 이 기능은 리소스가 제대로 구성되지 않은 경우 자세한 정보를 반환합니다.

상태 문제가 있는 클러스터에서 요금이 발생하나요?

예. 모든 아마존 EKS 클러스터에는 표준 Amazon EKS 요금이 청구됩니다. 클러스터 상태 기능은 추가 비용 없이 사용할 수 있습니다.

이 기능은 AWS Outposts의 Amazon EKS 클러스터에서 작동하나요?

예, 클러스터 문제는 AWS Outpost의 확장 클러스터 및 AWS Outpost의 로컬 클러스터를 포함하여 AWS Cloud의 EKS 클러스터에서 감지됩니다. 클러스터 상태에서는 Amazon EKS Anywhere 또는 Amazon EKS Distro(EKS-D) 관련 문제를 감지하지 못합니다.

새로운 문제가 감지되면 알림을 받을 수 있나요?

예. AWS는 새 클러스터 상태 문제가 감지되면 이메일 및 Personal Health Dashboard 알림을 보냅니다.

콘솔에서 상태 문제에 대한 경고를 제공하나요?

예. 상태 문제가 있는 모든 클러스터에는 콘솔 상단에 배너가 포함됩니다.

처음 두 열은 API 응답 값에 필요한 항목입니다. [Health ClusterIssue](#) 객체의 세 번째 필드는 `resourceIds`이며, 반환되는 내용은 문제 유형에 따라 달라집니다.

코드	메시지	ResourceIds	클러스터를 복구할 수 있나요?
SUBNET_NOT_FOUND	현재 클러스터와 연결된 하나 이상의 서브넷을 찾을 수 없습니다. Amazon EKS update-cluster-config API를 직접적으로 직접 호출하여 서브넷 업데이트	서브넷 ID	예
SECURITY_GROUP_NOT_FOUND	현재 클러스터와 연결된 하나 이상의 보안 그룹을 찾을 수 없습니다. Amazon EKS update-cluster-config API를 직접 호출하여 보안 그룹 업데이트	보안 그룹 ID	예
IP_NOT_AVAILABLE	클러스터와 연결된 하나 이상의 서브넷에서 Amazon EKS가 클러스터 관리 작업을 수행하는 데 사용할 수 있는 IP 주소가 충분하지 않습니다. Amazon EKS update-cluster-config API를 사용하여 서브넷의 주소를 확보하거나 다른 서브넷을 클러스터에 연결합니다.	서브넷 ID	예
VPC_NOT_FOUND	클러스터와 연결된 VPC를 찾을 수 없습니다. 클러스터를 삭제하	VPC ID	아니요

코드	메시지	ResourceIds	클러스터를 복구할 수 있나요?
	고 다시 생성해야 합니다.		
ASSUME_ROLE_ACCESS_DENIED	클러스터는 Amazon EKS 서비스 연결 역할을 사용하지 않습니다. 필수 Amazon EKS 관리 작업을 수행할 클러스터와 연결된 관련된 역할을 수임할 수 없습니다. 역할이 존재하고 필요한 신뢰 정책이 있는지 확인합니다.	클러스터 IAM 역할	예
PERMISSION_ACCESS_DENIED	클러스터는 Amazon EKS 서비스 연결 역할을 사용하지 않습니다. 클러스터와 연결된 역할이 Amazon EKS가 필수 관리 작업을 수행할 수 있는 충분한 권한을 부여하지 않습니다. 클러스터 역할에 연결된 정책을 확인하고 별도의 거부 정책이 적용되어 있는지 확인합니다.	클러스터 IAM 역할	예
ASSUME_ROLE_ACCESS_DENIED_USING_SLR	Amazon EKS 클러스터 관리 서비스 연결 역할을 수임할 수 없습니다. 역할이 존재하고 필요한 신뢰 정책이 있는지 확인합니다.	Amazon EKS 서비스 연결 역할	예

코드	메시지	ResourceIds	클러스터를 복구할 수 있나요?
PERMISSION_ACCESS_DENIED_USING_SLR	Amazon EKS 클러스터 관리 서비스 연결 역할이 Amazon EKS가 필수 관리 작업을 수행할 수 있는 충분한 권한을 부여하지 않습니다. 클러스터 역할에 연결된 정책을 확인하고 별도의 거부 정책이 적용되어 있는지 확인합니다.	Amazon EKS 서비스 연결 역할	예
OPT_IN_REQUIRED	계정에 Amazon EC2 서비스 구독이 없습니다. 계정 설정 페이지에서 계정 구독을 업데이트합니다.	N/A	예
STS_REGIONAL_ENDPOINT_DISABLED	STS 리전별 엔드포인트가 비활성화되었습니다. Amazon EKS의 엔드포인트가 필수 클러스터 관리 작업을 수행할 수 있도록 합니다.	N/A	예
KMS_KEY_DISABLED	클러스터와 연결된 AWS KMS 키가 비활성화되었습니다. 키를 다시 활성화하여 클러스터를 복구합니다.	KMS 키 Arn	예

코드	메시지	ResourceIds	클러스터를 복구할 수 있나요?
KMS_KEY_NOT_FOUND	클러스터와 연결된 AWS KMS 키를 찾을 수 없습니다. 클러스터를 삭제하고 다시 생성해야 합니다.	KMS 키 ARN	아니요
KMS_GRANT_REVOKED	클러스터와 연결된 AWS KMS 키에 부여된 권한이 취소됩니다. 클러스터를 삭제하고 다시 생성해야 합니다.	KMS 키 Arn	아니요

Amazon EKS Connector를 사용하여 Kubernetes 클러스터를 Amazon EKS Management Console에 연결

Amazon EKS Connector를 사용하여 적합한 Kubernetes 클러스터를 AWS에 등록하고 연결할 수 있으며 Amazon EKS 콘솔에서 시각화할 수 있습니다. 클러스터가 연결되면 Amazon EKS 콘솔에서 해당 클러스터의 상태, 구성 및 워크로드를 확인할 수 있습니다. 이 기능을 사용하여 Amazon EKS 콘솔에서 연결된 클러스터를 볼 수 있지만 관리할 수는 없습니다. Amazon EKS Connector에는 [Github의 오픈 소스 프로젝트](#)인 에이전트가 필요합니다. FAQ 및 문제 해결을 포함한 추가 기술 콘텐츠는 [the section called “EKS Connector 문제 해결”](#)을 참조하세요.

Amazon EKS Connector는 다음 유형의 Kubernetes 클러스터를 Amazon EKS에 연결할 수 있습니다.

- 온프레미스 Kubernetes 클러스터
- Amazon EC2에서 실행 중인 자체 관리형 클러스터
- 다른 클라우드 공급자의 관리형 클러스터

Amazon EKS Connector 고려 사항

Amazon EKS Connector를 사용하기 전에 다음 사항에 대한 이해가 필요합니다.

- 클러스터를 Amazon EKS에 연결하려면 Kubernetes 클러스터에 대한 관리 권한이 있어야 합니다.
- Kubernetes 클러스터에는 연결하기 전에 Linux 64비트(x86) 작업자 노드가 있어야 합니다. ARM 작업자 노드는 지원되지 않습니다.
- Kubernetes 클러스터에 `ssm`, 및 `ssmmessages`, Systems Manager 엔드포인트에 대한 아웃바운드 액세스 권한이 있는 작업자 노드가 있어야 합니다. 자세한 내용은 AWS 일반 참조의 [시스템 관리자 엔드포인트](#)를 참조하세요.
- 기본적으로 리전에서 최대 10개의 클러스터를 연결할 수 있습니다. [서비스 할당량 콘솔](#)을 통해 증가를 요청할 수 있습니다. 자세한 내용은 [할당량 증가 요청](#)을 참조하세요.
- Amazon EKS `RegisterCluster`, `ListClusters`, `DescribeCluster`, 및 `DeregisterCluster` API만 외부 Kubernetes 클러스터에 대해 지원됩니다.
- 클러스터를 등록하려면 다음 권한이 있어야 합니다.
 - `eks:RegisterCluster`
 - `ssm:CreateActivation`
 - `ssm>DeleteActivation`

- iam:PassRole
- 클러스터를 등록 취소하려면 다음 권한이 있어야 합니다.
 - eks:DeregisterCluster
 - ssm:DeleteActivation
 - ssm:DeregisterManagedInstance

Amazon EKS Connector에 필요한 IAM 역할

Amazon EKS Connector를 사용하려면 다음 두 가지 IAM 역할이 필요합니다.

- [Amazon EKS Connector](#) 서비스 연결 역할은 클러스터를 처음 등록할 때 생성됩니다.
- Amazon EKS Connector 에이전트 IAM 역할을 생성해야 합니다. 세부 정보는 [the section called “커넥터 IAM 역할”](#) 섹션을 참조하세요.

[IAM 보안 주체](#)에 대한 클러스터 및 워크로드 보기 권한을 사용 설정하려면 클러스터에 eks-connector 및 Amazon EKS Connector 클러스터 역할을 적용해야 합니다. [Amazon EKS 콘솔에서 Kubernetes 클러스터 리소스를 볼 수 있는 액세스 권한 부여](#)의 단계를 따르세요.

외부 Kubernetes 클러스터를 Amazon EKS Management Console에 연결

다음 프로세스에서 여러 방법을 사용하여 외부 Kubernetes 클러스터를 Amazon EKS에 연결할 수 있습니다. 이 프로세스에는 두 단계로 이루어집니다. Amazon EKS에 클러스터를 등록하고 클러스터에 eks-connector 에이전트를 설치하는 것입니다.

Important

첫 번째 단계를 완료한 후 등록이 만료되는 3일 이내에 두 번째 단계를 완료해야 합니다.

고려 사항

에이전트를 설치할 때 YAML 매니페스트를 사용할 수 있습니다. 아니면 AWS Management Console 또는 AWS 명령줄 인터페이스에 클러스터를 등록하는 경우 Helm을 사용할 수 있습니다. 그러나 eksctl에 클러스터를 등록한 경우 Helm을 사용하여 에이전트를 설치할 수 없습니다.

사전 조건

- Amazon EKS Connector 에이전트 역할이 생성되었는지 확인합니다. [Amazon EKS 커넥터 에이전트 역할 생성](#)의 단계를 따릅니다.
- 클러스터를 등록하려면 다음 권한이 있어야 합니다.
 - `eks:RegisterCluster`
 - `ssm:CreateActivation`
 - `ssm>DeleteActivation`
 - `iam:PassRole`

1단계: 클러스터 등록

Amazon EKS 커넥터에 클러스터를 등록하려면 다음 도구 중 하나를 사용할 수 있습니다.

- [the section called “AWS CLI”](#)
- [the section called “AWS Management Console”](#)
- [the section called “eksctl”](#)

AWS CLI

1. AWS CLI가 설치되어 있어야 합니다. 설치하거나 업그레이드하려면 [AWS CLI 설치](#)를 참조하세요.
2. 커넥터 구성의 경우 Amazon EKS Connector 에이전트 IAM 역할을 지정합니다. 자세한 내용은 [the section called “Amazon EKS Connector에 필요한 IAM 역할”](#) 섹션을 참조하세요.

```
aws eks register-cluster \
  --name my-first-registered-cluster \
  --connector-config roleArn=arn:aws:iam::111122223333:role/
AmazonEKSCoordinatorAgentRole,provider="OTHER" \
  --region aws-region
```

예제 출력은 다음과 같습니다.

```
{
  "cluster": {
    "name": "my-first-registered-cluster",
    "arn": "arn:aws:eks:region:111122223333:cluster/my-first-registered-cluster",
```

```

    "createdAt": 1627669203.531,
    "ConnectorConfig": {
      "activationId": "xxxxxxxxACTIVATION_IDxxxxxxxx",
      "activationCode": "xxxxxxxxACTIVATION_CODExxxxxxxx",
      "activationExpiry": 1627672543.0,
      "provider": "OTHER",
      "roleArn": "arn:aws:iam::111122223333:role/AmazonEKSCollectorAgentRole"
    },
    "status": "CREATING"
  }
}

```

다음 단계에서 `aws-region`, `activationId` 및 `activationCode` 값을 사용합니다.

AWS Management Console

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 클러스터 추가를 선택하고 등록을 선택하여 구성 페이지를 불러옵니다.
3. 클러스터 구성 페이지에서 다음 필드를 작성합니다.
 - 이름 - 클러스터에 대한 고유한 이름
 - 공급자(Provider) - 선택하여 Kubernetes 클러스터 공급자의 드롭다운 목록을 표시합니다. 특정 공급자를 모르는 경우 기타를 선택합니다.
 - EKS Connector 역할 - 클러스터 연결에 사용할 역할을 선택합니다.
4. 클러스터 등록을 선택합니다.
5. 클러스터 개요 페이지가 표시됩니다. 차트 Helm을 사용하려면 `helm install` 명령을 복사하고 다음 단계로 넘어갑니다. YAML 매니페스트를 사용하려면 YAML 파일 다운로드를 선택하여 매니페스트 파일을 로컬 드라이브에 다운로드합니다.

Important

`helm install` 명령을 복사하거나 이 파일을 다운로드할 수 있는 유일한 기회입니다. 링크에 액세스할 수 없으며 클러스터를 등록 취소하고 처음부터 단계를 시작해야 하므로 이 페이지에서 벗어나지 마세요.

명령 또는 매니페스트 파일은 등록된 클러스터에 대해 한 번만 사용할 수 있습니다. Kubernetes 클러스터에서 리소스를 삭제하는 경우 클러스터를 다시 등록하고 새 매니페스트 파일을 가져와야 합니다.

매니페스트 파일을 Kubernetes 클러스터에 적용하려면 다음 단계로 이동합니다.

eksctl

1. eksctl 버전 0.68 이상을 설치해야 합니다. 설치하거나 업그레이드하려면 [the section called “클러스터 생성\(eksctl\)”](#) 섹션을 참조하세요.
2. 이름, 공급자 및 리전을 제공하여 클러스터를 등록합니다.

```
eksctl register cluster --name my-cluster --provider my-provider --region region-code
```

출력 예시:

```
2021-08-19 13:47:26 [#] creating IAM role "eksctl-20210819194112186040"
2021-08-19 13:47:26 [#] registered cluster "<name>" successfully
2021-08-19 13:47:26 [#] wrote file eks-connector.yaml to <current directory>
2021-08-19 13:47:26 [#] wrote file eks-connector-clusterrole.yaml to <current directory>
2021-08-19 13:47:26 [#] wrote file eks-connector-console-dashboard-full-access-group.yaml to <current directory>
2021-08-19 13:47:26 [!] note: "eks-connector-clusterrole.yaml" and "eks-connector-console-dashboard-full-access-group.yaml" give full EKS Console access to IAM identity "<aws-arn>", edit if required; read https://eksctl.io/usage/eks-connector for more info
2021-08-19 13:47:26 [#] run `kubectl apply -f eks-connector.yaml,eks-connector-clusterrole.yaml,eks-connector-console-dashboard-full-access-group.yaml` before expiry> to connect the cluster
```

이렇게 하면 로컬 컴퓨터에 파일이 생성됩니다. 이 파일을 3일 이내에 외부 클러스터에 적용해야 하며, 그렇지 않으면 등록이 만료됩니다.

3. 클러스터에 액세스할 수 있는 터미널에서 eks-connector-binding.yaml 파일 적용:

```
kubectl apply -f eks-connector-binding.yaml
```

2단계: eks-connector 에이전트 설치

eks-connector 에이전트를 설치하려면 다음 도구 중 하나를 사용합니다.

- [the section called “Helm”](#)
- [the section called “yaml”](#)

Helm

Note

eksctl에 클러스터를 등록한 경우 차트 Helm 방법 대신 YAML 매니페스트 방법을 사용하세요.

1. 이전 단계에서 AWS CLI를 사용한 경우 다음 명령에서 ACTIVATION_CODE 및 ACTIVATION_ID를 각각 activationId 및 activationCode 값으로 대체합니다. aws-region을 이전 단계에서 사용한 AWS 리전으로 바꿉니다. 그런 다음 명령을 실행하여 등록하는 클러스터에 eks-connector 에이전트를 설치합니다.

```
$ helm install eks-connector \
  --namespace eks-connector \
  oci://public.ecr.aws/eks-connector/eks-connector-chart \
  --set eks.activationCode=ACTIVATION_CODE \
  --set eks.activationId=ACTIVATION_ID \
  --set eks.agentRegion=aws-region
```

이전 단계에서 AWS Management Console을 사용한 경우 이전 단계에서 복사하고 이러한 값이 채워진 명령을 사용합니다.

2. 설치된 eks-connector 배포의 상태를 확인하고 Amazon EKS에 등록된 클러스터의 상태가 ACTIVE가 될 때까지 기다립니다.

yaml

Amazon EKS Connector 매니페스트 파일을 Kubernetes 클러스터에 적용하여 연결 작업을 완료합니다. 이렇게 하려면 앞서 설명한 방법을 사용해야 합니다. 3일 이내에 매니페스트가 적용되지 않으면

Amazon EKS Connector 등록이 만료됩니다. 클러스터 연결이 만료되면 먼저 클러스터의 등록을 취소한 후 클러스터를 다시 연결해야 합니다.

1. Amazon EKS Connector YAML 파일을 다운로드합니다.

```
curl -O https://amazon-eks.s3.us-west-2.amazonaws.com/eks-connector/manifests/eks-connector/latest/eks-connector.yaml
```

2. Amazon EKS Connector YAML 파일을 편집하여 %AWS_REGION%, %EKS_ACTIVATION_ID%, %EKS_ACTIVATION_CODE%의 모든 참조를 이전 단계 출력의 aws-region, activationId 및 activationCode로 바꿉니다.

다음 예제 명령은 이러한 값을 대체할 수 있습니다.

```
sed -i "s~%AWS_REGION%~$aws-region~g; s~%EKS_ACTIVATION_ID%~$EKS_ACTIVATION_ID~g; s~%EKS_ACTIVATION_CODE%~$(echo -n $EKS_ACTIVATION_CODE | base64)~g" eks-connector.yaml
```

 Important

정품 인증 코드가 base64 형식인지 확인합니다.

3. 클러스터에 액세스할 수 있는 터미널에서 다음 명령을 실행하여 업데이트된 매니페스트 파일을 적용할 수 있습니다.

```
kubectl apply -f eks-connector.yaml
```

4. Amazon EKS Connector 매니페스트 및 역할 바인딩 YAML 파일이 Kubernetes 클러스터에 적용된 후 클러스터가 이제 연결되었는지 확인합니다.

```
aws eks describe-cluster \
  --name "my-first-registered-cluster" \
  --region AWS_REGION
```

출력에 status=ACTIVE가 포함되어야 합니다.

5. (선택사항) 클러스터에 태그를 추가합니다. 자세한 내용은 [the section called “리소스에 태깅”](#) 섹션을 참조하세요.

다음 단계

이 단계에서 문제가 있는 경우 [the section called “EKS Connector 문제 해결”](#)의 내용을 참조하세요.

연결된 클러스터에서 Kubernetes 리소스를 볼 수 있도록 추가 [IAM 위탁자](#)에게 Amazon EKS 콘솔에 대한 액세스 권한을 부여하려면 [the section called “클러스터에 액세스 권한 부여”](#) 섹션을 참조하세요.

Amazon EKS 콘솔에서 Kubernetes 클러스터 리소스를 볼 수 있는 액세스 권한 부여

[IAM 위탁자](#)에게 Amazon EKS 콘솔에 대한 액세스 권한을 부여하여 연결된 클러스터에서 실행 중인 Kubernetes 리소스에 대한 정보를 봅니다.

사전 조건

AWS Management Console 액세스에 사용하는 [IAM 보안 주체](#)에서는 다음과 같은 요구 사항을 충족해야 합니다.

- eks:AccessKubernetesApi IAM 권한이 있어야 합니다.
- Amazon EKS Connector 서비스 계정은 클러스터에서 IAM 보안 주체를 가장할 수 있습니다. 그러면 Amazon EKS Connector에서 IAM 위탁자를 Kubernetes 사용자에게 매핑할 수 있습니다.

Amazon EKS Connector 클러스터 역할을 생성하고 적용하려면

1. eks-connector 클러스터 역할 템플릿을 다운로드합니다.

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-clusterrole.yaml
```

2. 클러스터 역할 템플릿 YAML 파일을 편집합니다. %IAM_ARN%의 참조를 IAM 보안 주체의 Amazon 리소스 이름(ARN)으로 바꿉니다.
3. Amazon EKS Connector 클러스터 역할 YAML을 Kubernetes 클러스터에 적용합니다.

```
kubectl apply -f eks-connector-clusterrole.yaml
```

IAM 위탁자가 Amazon EKS 콘솔에서 Kubernetes 리소스를 보려면 위탁자가 이러한 리소스를 읽는 데 필요한 권한을 통해 Kubernetes role 또는 clusterrole과 연결되어 있어야 합니다. 자세한 내용은 Kubernetes 문서의 [Using RBAC Authorization\(RBAC 승인 사용\)](#)을 참조하십시오.

연결된 클러스터에 액세스하도록 IAM 보안 주체를 구성하려면

1. 예제 매니페스트 파일을 다운로드하여 clusterrole과 clusterrolebinding 또는 role과 rolebinding을 각각 생성할 수 있습니다.

모든 네임스페이스의 Kubernetes 리소스 보기

- eks-connector-console-dashboard-full-access-clusterrole 클러스터 역할을 사용하면 콘솔에서 시각화할 수 있는 모든 네임스페이스와 리소스에 액세스할 수 있습니다. 먼저 role, clusterrole 및 해당 바인딩의 이름을 변경해야 이를 클러스터에 적용할 수 있습니다. 다음 명령을 사용하여 샘플 파일을 다운로드합니다.

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-console-dashboard-full-access-group.yaml
```

특정 네임스페이스의 Kubernetes 리소스 보기

- 이 파일의 네임스페이스는 default이므로 다른 네임스페이스를 지정하려면 클러스터에 적용하기 전에 파일을 편집합니다. 다음 명령을 사용하여 샘플 파일을 다운로드합니다.

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/eks-connector/manifests/eks-connector-console-roles/eks-connector-console-dashboard-restricted-access-group.yaml
```

2. 전체 액세스 또는 제한된 액세스 YAML 파일을 편집하고, %IAM_ARN%의 참조를 IAM 보안 주체의 Amazon 리소스 이름(ARN)으로 바꿉니다.
3. Kubernetes 클러스터에 전체 액세스 또는 제한된 액세스 YAML 파일을 적용합니다. YAML 파일 값을 사용자의 고유한 값으로 교체합니다.

```
kubectl apply -f eks-connector-console-dashboard-full-access-group.yaml
```

연결된 클러스터에서 Kubernetes 리소스를 보려면 [the section called “클러스터 리소스 액세스”](#) 섹션을 참조하세요. 리소스 탭에서 일부 리소스 유형에 대한 데이터는 연결된 클러스터에서 사용할 수 없습니다.

Amazon EKS 콘솔에서 Kubernetes 클러스터 등록 취소

연결된 클러스터 사용을 마치면 등록 취소할 수 있습니다. 등록이 취소된 후에는 클러스터가 Amazon EKS 콘솔에 더이상 표시되지 않습니다.

deregisterCluster API를 호출하려면 다음 권한이 있어야 합니다.

- eks:DeregisterCluster
- ssm:DeleteActivation
- ssm:DeregisterManagedInstance

이 프로세스는 두 단계로 이루어집니다. Amazon EKS에서 클러스터 등록을 취소하고 클러스터에서 eks-connector 에이전트를 제거하는 것입니다.

Kubernetes 클러스터 등록 취소

Amazon EKS 커넥터에서 클러스터를 등록 취소하려면 다음 도구 중 하나를 사용할 수 있습니다.

- [the section called “AWS CLI”](#)
- [the section called “AWS Management Console”](#)
- [the section called “eksctl”](#)

AWS CLI

1. AWS CLI가 설치되어 있어야 합니다. 설치하거나 업그레이드하려면 [AWS CLI 설치](#)를 참조하세요.
2. Amazon EKS Connector 에이전트 역할이 생성되었는지 확인합니다.
3. 연결된 클러스터의 등록을 취소합니다.

```
aws eks deregister-cluster \
  --name my-cluster \
  --region region-code
```

AWS Management Console

1. [Amazon EKS 콘솔](#)을 엽니다.
2. 클러스터를 선택하세요.

3. 클러스터 페이지에서 연결된 클러스터를 선택한 다음 등록 취소를 선택합니다.
4. 클러스터 등록을 취소할지 확인합니다.

eksctl

1. eksctl 버전 0.68 이상을 설치합니다. 설치하거나 업그레이드하려면 [the section called “클러스터 생성\(eksctl\)”](#) 섹션을 참조하세요.
2. Amazon EKS Connector 에이전트 역할이 생성되었는지 확인합니다.
3. 연결된 클러스터의 등록을 취소합니다.

```
eksctl deregister cluster --name my-cluster
```

Kubernetes 클러스터의 리소스 정리

eks-connector 에이전트를 삭제하려면 다음 도구 중 하나를 사용합니다.

- [the section called “Helm”](#)
- [the section called “yaml”](#)

Helm

다음 명령을 실행하여 에이전트를 제거합니다.

```
helm -n eks-connector uninstall eks-connector
```

yaml

1. Kubernetes 클러스터에서 Amazon EKS Connector YAML 파일을 삭제합니다.

```
kubectl delete -f eks-connector.yaml
```

2. 추가 [IAM 위탁자](#)가 클러스터에 액세스할 수 있도록 clusterrole 또는 clusterrolebindings를 생성한 경우 Kubernetes 클러스터에서 이를 삭제합니다.

Amazon EKS Connector 이슈 문제 해결

이 주제에서는 해결 방법에 대한 지침과 해결 방법을 포함하여 Amazon EKS 커넥터를 사용하는 동안 발생할 수 있는 몇 가지 일반적인 오류를 설명합니다.

기본 문제 해결

이 섹션에서는 Amazon EKS 커넥터 문제를 진단하는 단계를 설명합니다.

Amazon EKS 커넥터 상태 확인

Amazon EKS 커넥터 상태를 확인하려면 다음과 같이 입력합니다.

```
kubectl get pods -n eks-connector
```

Amazon EKS 커넥터 로그 검사

Amazon EKS 커넥터 포드는 3개의 컨테이너로 구성됩니다. 이러한 컨테이너를 모두 검사할 수 있도록 전체 로그를 검색하려면 다음 명령을 실행합니다.

- connector-init

```
kubectl logs eks-connector-0 --container connector-init -n eks-connector  
kubectl logs eks-connector-1 --container connector-init -n eks-connector
```

- connector-proxy

```
kubectl logs eks-connector-0 --container connector-proxy -n eks-connector  
kubectl logs eks-connector-1 --container connector-proxy -n eks-connector
```

- connector-agent

```
kubectl exec eks-connector-0 --container connector-agent -n eks-connector -- cat /  
var/log/amazon/ssm/amazon-ssm-agent.log  
kubectl exec eks-connector-1 --container connector-agent -n eks-connector -- cat /  
var/log/amazon/ssm/amazon-ssm-agent.log
```

유효 클러스터 이름 가져오기

Amazon EKS 클러스터는 단일 AWS 계정 및 AWS 리전 내에서 `clusterName`으로 고유하게 식별됩니다. Amazon EKS에 연결된 클러스터가 여러 개 있는 경우 현재 Kubernetes 클러스터가 등록된 Amazon EKS 클러스터를 확인할 수 있습니다. 이렇게 하려면 다음을 입력하여 현재 클러스터의 `clusterName`을 찾습니다.

```
kubectl exec eks-connector-0 --container connector-agent -n eks-connector \
  -- cat /var/log/amazon/ssm/amazon-ssm-agent.log | grep -m1 -oE "eks_c:[a-zA-Z0-9_-]+"
| sed -E "s/^.*eks_c:([a-zA-Z0-9_-]+)_[a-zA-Z0-9-]+.*$/\1/"
kubectl exec eks-connector-1 --container connector-agent -n eks-connector \
  -- cat /var/log/amazon/ssm/amazon-ssm-agent.log | grep -m1 -oE "eks_c:[a-zA-Z0-9_-]+"
| sed -E "s/^.*eks_c:([a-zA-Z0-9_-]+)_[a-zA-Z0-9-]+.*$/\1/"
```

기타 명령

다음 명령은 문제를 해결하는 데 필요한 정보를 검색하는 데 유용합니다.

- 다음 명령을 사용하여 Amazon EKS 커넥터의 포드에서 사용하는 이미지를 수집합니다.

```
kubectl get pods -n eks-connector -o jsonpath="{.items[*].spec.containers[*].image}"
| tr -s '[:space:]' '\n'
```

- 다음과 같은 명령을 사용하여 Amazon EKS 커넥터가 실행 중인 노드 이름을 확인합니다.

```
kubectl get pods -n eks-connector -o jsonpath="{.items[*].spec.nodeName}" | tr -s
'[:space:]' '\n'
```

- 다음 명령을 실행하여 Kubernetes 클라이언트 및 서버 버전을 가져옵니다.

```
kubectl version
```

- 다음과 같은 명령을 실행하여 노드에 대한 정보를 가져옵니다.

```
kubectl get nodes -o wide --show-labels
```

Helm 문제: 403 Forbidden

`helm install` 명령을 실행할 때 다음 오류가 발생하는 경우:

```
Error: INSTALLATION FAILED: unexpected status from HEAD request to https://
public.ecr.aws/v2/eks-connector/eks-connector-chart/manifests/0.0.6: 403 Forbidden
```

다음 줄을 실행하여 문제를 해결할 수 있습니다.

```
docker logout public.ecr.aws
```

콘솔 오류: 클러스터가 보류 중 상태에서 멈춰 있음

클러스터를 등록한 후 Amazon EKS 콘솔에서 클러스터가 Pending 상태로 멈추는 경우 Amazon EKS 커넥터가 아직 클러스터를 AWS에 성공적으로 연결하지 않았기 때문일 수 있습니다. 등록된 클러스터의 경우 Pending 상태는 연결이 성공적으로 설정되지 않았음을 의미합니다. 이 문제를 해결하려면 대상 Kubernetes 클러스터에 매니페스트를 적용했는지 확인하세요. 클러스터에 적용했지만 클러스터가 여전히 Pending 상태인 경우에는 eks-connector statefulset가 비정상 상태일 수 있습니다. 이 문제를 해결하려면 이 주제의 [the section called “Amazon EKS 커넥터 포드가 크래시 반복됨”](#) 섹션을 참조하세요.

콘솔 오류: 사용자 system:serviceaccount:eks-connector:eks-connector가 클러스터 범위에서 API 그룹의 리소스 사용자를 가장할 수 없습니다.

Amazon EKS 커넥터는 Kubernetes [사용자 가장](#)을 사용하여 AWS Management Console에서 [IAM 위탁자](#)를 대신합니다. AWS eks-connector 서비스 계정에서 Kubernetes API에 액세스하는 각 위탁자에 IAM ARN을 Kubernetes 사용자 이름으로 사용하여 해당 Kubernetes 사용자를 가장할 수 있는 권한이 부여되어야 합니다. 다음 예에서 IAM ARN은 Kubernetes 사용자에게 매핑됩니다.

- AWS 계정 **111122223333**의 IAM 사용자 **john**은 Kubernetes 사용자에게 매핑됩니다. [IAM 모범 사례](#)에서는 되도록 사용자 대신 역할에 권한을 부여하세요.

```
arn:aws:iam::111122223333:user/john
```

- AWS 계정 **111122223333**의 IAM 역할 **admin**은 Kubernetes 사용자에게 매핑됩니다.

```
arn:aws:iam::111122223333:role/admin
```

결과는 AWS STS 세션 ARN 대신 IAM 역할 ARN입니다.

매핑된 사용자를 가장할 수 있는 eks-connector 서비스 계정 권한을 부여하도록 ClusterRole 및 ClusterRoleBinding을 구성하는 방법에 대한 지침은 [the section called “클러스터에 액세스 권한 부여”](#) 섹션을 참조하세요. 템플릿에서 %IAM_ARN%가 AWS Management Console IAM 보안 주체의 IAM ARN으로 바뀌었는지 확인합니다.

콘솔 오류: [...] 금지됨: 사용자 [...]이(가) 클러스터 범위에서 API 그룹의 [...] 리소스를 나열할 수 없음

다음 문제를 고려하세요. Amazon EKS 커넥터가 대상 Kubernetes 클러스터에서 요청하는 AWS Management Console IAM 위탁자를 성공적으로 가장했습니다. 그러나 가장된 위탁자에게 Kubernetes API 작업에 대한 RBAC 권한이 없습니다.

이 문제를 해결하려면 두 가지 방법으로 추가 사용자에게 권한을 부여할 수 있습니다. 이전에 차트 Helm을 통해 eks-connector를 설치한 경우 다음 명령을 실행하여 사용자에게 액세스 권한을 쉽게 부여할 수 있습니다. userARN1 및 userARN2를 Kubernetes 리소스에 대한 보기 권한을 부여하는 IAM 역할의 ARN 목록으로 대체합니다.

```
helm upgrade eks-connector oci://public.ecr.aws/eks-connector/eks-connector-chart \
  --reuse-values \
  --set 'authentication.allowedUserARNs={userARN1,userARN2}'
```

클러스터 관리자로서 개별 Kubernetes 사용자에게 적절한 수준의 RBAC 권한을 부여하세요. 자세한 정보와 지침은 [the section called “클러스터에 액세스 권한 부여”](#) 섹션을 참조하세요.

콘솔 오류: Amazon EKS가 Kubernetes 클러스터 API 서버와 통신할 수 없습니다. 연결에 성공하려면 클러스터가 ACTIVE 상태여야 합니다. 몇 분 후에 다시 시도하세요.

Amazon EKS 서비스가 대상 클러스터의 Amazon EKS 커넥터와 통신할 수 없는 경우 다음과 같은 사유 중 하나 때문일 수 있습니다.

- 대상 클러스터의 Amazon EKS 커넥터가 비정상입니다.
- 대상 클러스터와 AWS 리전 간의 연결 상태가 좋지 않거나 연결이 중단되었습니다.

이 문제를 해결하려면 [Amazon EKS 커넥터 로그](#)를 확인하세요. Amazon EKS 커넥터에 대한 오류가 표시되지 않으면 몇 분 후에 연결을 다시 시도하세요. 대상 클러스터에 대해 긴 대기 시간 또는 간헐적인 연결이 정기적으로 발생하는 경우 가까운 AWS 리전에 클러스터를 다시 등록하는 것이 좋습니다.

Amazon EKS 커넥터 포드가 크래시 반복됨

여러 가지 사유로 Amazon EKS 커넥터 포드가 CrashLoopBackOff 상태로 전환될 수 있습니다. 이 문제는 connector-init 컨테이너와 관련이 있을 수 있습니다. Amazon EKS 커넥터 포드의 상태를 확인합니다.

```
kubectl get pods -n eks-connector
```

예제 출력은 다음과 같습니다.

NAME	READY	STATUS	RESTARTS	AGE
eks-connector-0	0/2	Init:CrashLoopBackOff	1	7s

출력이 이전 출력과 비슷한 경우 [the section called “Amazon EKS 커넥터 로그 검사”](#)의 내용을 참조하여 문제를 해결합니다.

eks-connector 시작 실패: InvalidActivation

Amazon EKS 커넥터를 처음 시작할 때 Amazon Web Services에 activationId 및 activationCode를 등록합니다. 등록에 실패하여 다음과 같은 오류와 비슷한 오류로 connector-init 컨테이너가 충돌할 수 있습니다.

```
F1116 20:30:47.261469      1 init.go:43] failed to initiate eks-connector:
InvalidActivation:
```

이 문제를 해결하려면 다음 원인과 권장 수정 사항을 고려하세요.

- activationId 및 activationCode가 매니페스트 파일에 없기 때문에 등록에 실패했을 수 있습니다. 이 경우 RegisterCluster API 작업에서 반환된 올바른 값이고 activationCode가 매니페스트 파일에 있는지 확인합니다. activationCode는 Kubernetes 보안 암호에 추가되므로 base64로 인코딩되어야 합니다. 자세한 내용은 [the section called “1단계: 클러스터 등록”](#) 단원을 참조하십시오.
- 정품 인증이 만료되어 등록에 실패했을 수 있습니다. 보안상의 이유로 클러스터를 등록한 후 3일 이내에 Amazon EKS 커넥터를 활성화해야 하기 때문입니다. 이 문제를 해결하려면 만료 날짜 및 시간 전에 Amazon EKS Connector 매니페스트가 대상 Kubernetes 클러스터에 적용되었는지 확인하세요. 정품 인증 만료 날짜를 확인하려면 DescribeCluster API 작업을 호출하세요.

```
aws eks describe-cluster --name my-cluster
```

다음과 같은 예시 응답에서는 만료 날짜 및 시간이 2021-11-12T22:28:51.101000-08:00으로 기록됩니다.

```
{
  "cluster": {
    "name": "my-cluster",
    "arn": "arn:aws:eks:region:111122223333:cluster/my-cluster",
    "createdAt": "2021-11-09T22:28:51.449000-08:00",
    "status": "FAILED",
    "tags": {
    },
    "connectorConfig": {
      "activationId": "000000000-0000-0000-0000-000000000000",
      "activationExpiry": "2021-11-12T22:28:51.101000-08:00",
      "provider": "OTHER",
      "roleArn": "arn:aws:iam::111122223333:role/my-connector-role"
    }
  }
}
```

activationExpiry가 지나면 클러스터를 등록 취소하고 다시 등록합니다. 이렇게 하면 새 정품 인증이 생성됩니다.

클러스터 노드에 아웃바운드 연결이 없습니다.

제대로 작동하려면 Amazon EKS 커넥터에 여러 AWS 엔드포인트에 대한 아웃바운드 연결이 필요합니다. 대상 AWS 리전에 대한 아웃바운드 연결 없이 프라이빗 클러스터를 연결할 수 없습니다. 이 문제를 해결하려면 필요한 아웃바운드 연결을 추가해야 합니다. 커넥터 요구 사항에 대한 자세한 내용은 [the section called “Amazon EKS Connector 고려 사항”](#) 섹션을 참조하세요.

Amazon EKS 커넥터 포드의 상태가 ImagePullBackOff임

get pods 명령을 실행하고 포드의 상태가 ImagePullBackOff이면 제대로 작동하지 않습니다. Amazon EKS 커넥터 포드의 상태가 ImagePullBackOff이면 제대로 작동하지 않습니다. Amazon EKS 커넥터 포드의 상태를 확인합니다.

```
kubectl get pods -n eks-connector
```

예제 출력은 다음과 같습니다.

NAME	READY	STATUS	RESTARTS	AGE
eks-connector-0	0/2	Init:ImagePullBackOff	0	4s

기본 Amazon EKS 커넥터 매니페스트 파일은 [Amazon ECR Public Gallery](#)의 이미지를 참조합니다. 대상 Kubernetes 클러스터가 Amazon ECR 퍼블릭 갤러리에서 이미지를 가져올 수 없는 가능성이 있습니다. Amazon ECR Public Gallery 이미지 풀 문제를 해결하거나 선택한 프라이빗 컨테이너 레지스트리의 이미지 미러링을 고려하세요.

AWS Connector 자주 묻는 질문

Q: Amazon EKS 커넥터의 기본 기술은 어떻게 작동하나요?

A: Amazon EKS 커넥터는 AWS 시스템 관리자(System Manager) 에이전트를 기반으로 합니다. Amazon EKS Connector는 Kubernetes 클러스터에서 StatefulSet로 실행됩니다. 연결을 설정하고 클러스터의 API 서버와 Amazon Web Services 간의 통신을 프록시합니다. EKS 커넥터는 AWS에서 클러스터 연결을 끊을 때까지 Amazon EKS 콘솔에 클러스터 데이터를 표시하기 위해 이 작업을 수행합니다. Systems Manager 에이전트는 오픈 소스 프로젝트입니다. 이 프로젝트에 대한 자세한 내용은 [GitHub 프로젝트 페이지](#)를 참조하세요.

Q: 연결하려는 온프레미스 Kubernetes 클러스터가 있습니다. 연결하려면 방화벽 포트를 열어야 하나요?

A: 아니요, 방화벽 포트를 열 필요가 없습니다. Kubernetes 클러스터는 AWS 리전에 대한 아웃바운드 연결만 필요합니다. AWS 서비스는 온프레미스 네트워크의 리소스에 절대 액세스하지 않습니다. Amazon EKS 커넥터는 클러스터에서 실행되고 AWS에 대한 연결을 시작합니다. 클러스터 등록이 완료되면 AWS는 클러스터에 있는 Kubernetes API 서버의 정보가 필요한 작업을 Amazon EKS 콘솔에서 시작한 후에만 Amazon EKS 커넥터에 명령을 실행합니다.

Q: Amazon EKS 커넥터가 내 클러스터에서 AWS로 어떤 데이터를 전송하나요?

A: Amazon EKS 커넥터는 클러스터를 AWS에 등록하는 데 필요한 기술 정보를 전송합니다. 또한 고객이 요청하는 Amazon EKS 콘솔 기능에 대한 클러스터 및 워크로드 메타데이터를 전송합니다. Amazon EKS 커넥터는 데이터를 AWS로 전송해야 하는 Amazon EKS 콘솔 또는 Amazon EKS API에서 작업을 시작하는 경우에만 이 데이터를 수집하거나 전송합니다. Kubernetes 버전 번호 외에 AWS는 기본적으로 데이터를 저장하지 않습니다. 권한을 부여받은 경우에만 데이터를 저장합니다.

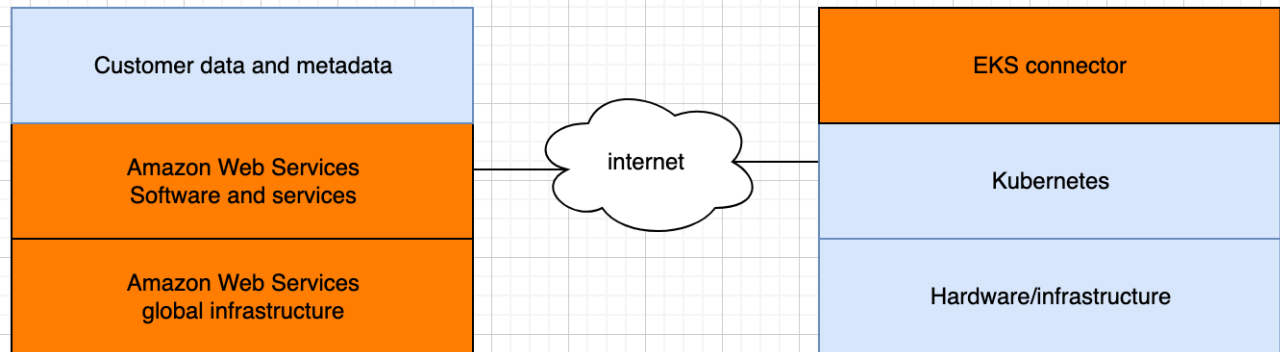
Q: AWS 리전 외부의 클러스터를 연결할 수 있나요?

A: 예. 어느 위치에서든 클러스터를 Amazon EKS에 연결할 수 있습니다. 또한 Amazon EKS 서비스는 모든 AWS 퍼블릭 상용 AWS 리전에 위치할 수 있습니다. 이는 클러스터에서 대상 AWS 리전으로의

유효한 네트워크 연결에서 작동합니다. UI 성능 최적화를 위해 클러스터 위치에서 가장 가까운 AWS 리전을 선택하는 것이 좋습니다. 예를 들어, 도쿄에서 실행 중인 클러스터가 있는 경우, 짧은 대기 시간을 위해 클러스터를 도쿄의 AWS 리전 (즉, ap-northeast-1 AWS 리전)에 연결합니다. 중국 또는 GovCloud AWS 리전을 제외한 모든 퍼블릭 상용 AWS 리전의 Amazon EKS에 모든 위치의 클러스터를 연결할 수 있습니다.

Amazon EKS 커넥터에서 보안 이해

Amazon EKS Connector는 Kubernetes 클러스터에서 실행되는 오픈 소스 구성 요소입니다. 이 클러스터는 AWS 환경의 외부에 위치해 있을 수 있습니다. 이로 인해 보안 책임에 대한 추가 고려 사항이 생깁니다. 다음 다이어그램에서 이 구성을 볼 수 있습니다. 오렌지색은 AWS 책임을 표시하며, 파란색은 고객 책임을 나타냅니다.



이 주제에서는 연결된 클러스터가 AWS의 외부에 있는 경우 책임 모델의 차이점에 대해 설명합니다.

AWS 책임

- 고객의 Kubernetes 클러스터에서 실행되고 AWS와 통신하는 [오픈 소스 구성 요소](#)인 Amazon EKS Connector의 유지 관리, 구축 및 전달.
- 연결된 Kubernetes 클러스터 및 AWS 서비스 간의 전송 및 애플리케이션 계층 통신 보안 유지 관리.

고객 책임

- 특히 다음 줄이 표시되는 Kubernetes 클러스터 특정 보안.
 - Kubernetes 보안 암호는 적절히 암호화되고 보호되어야 합니다.

- eks-connector 네임스페이스에 대한 액세스 잠금.
- AWS에서 [IAM 보안 주체](#)를 관리하기 위한 역할 기반 액세스 제어(RBAC) 권한 구성. 지침은 [the section called “클러스터에 액세스 권한 부여”](#) 섹션을 참조하세요.
- Amazon EKS Connector 설치 및 업그레이드.
- 연결된 Kubernetes 클러스터를 지원하는 하드웨어, 소프트웨어 및 인프라 유지 관리.
- 해당 AWS 계정 보안(예: [루트 사용자 보안 인증 정보](#) 보호를 통해).

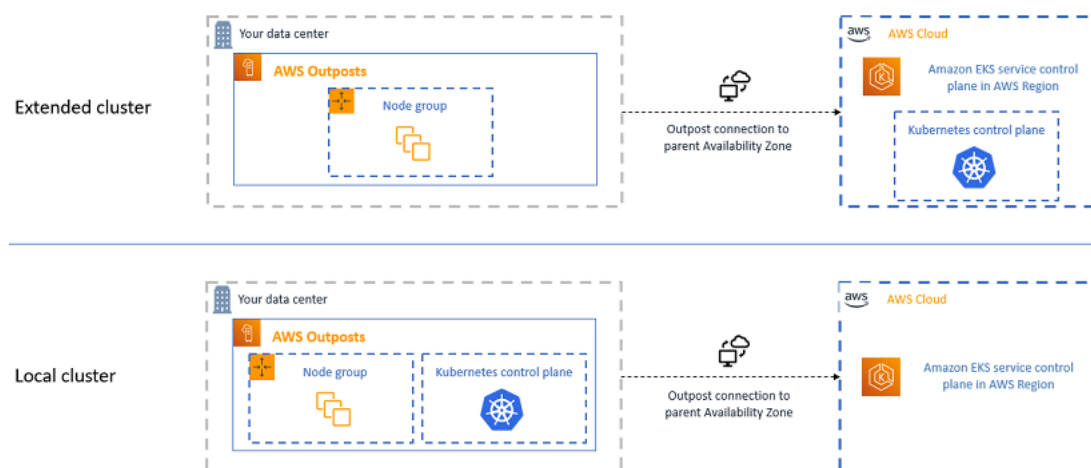
AWS Outposts를 사용한 Amazon EKS 온프레미스 배포

Amazon EKS를 사용하여 AWS Outposts에서 온프레미스 Kubernetes 애플리케이션을 실행할 수 있습니다. 다음과 같은 방법으로 Outposts에 Amazon EKS를 배포할 수 있습니다.

- 확장 클러스터 - AWS 리전에서 Kubernetes 컨트롤 플레인을 실행하고 Outpost에서 노드를 실행합니다.
- 로컬 클러스터 - Outpost에서 Kubernetes 컨트롤 플레인과 노드를 실행합니다.

두 배포 옵션 모두 AWS에서 Kubernetes 컨트롤 플레인을 완전히 관리합니다. 클라우드에서 사용하는 것과 동일한 Amazon EKS API, 도구 및 콘솔을 사용하여 Outposts에서 Amazon EKS를 생성하고 실행할 수 있습니다.

다음 다이어그램은 이러한 배포 옵션을 보여줍니다.



각 배포 옵션을 사용해야 하는 경우

로컬 클러스터와 확장 클러스터 모두 범용 배포 옵션이며 다양한 용도에 사용할 수 있습니다.

로컬 클러스터를 통해 전체 Amazon EKS 클러스터를 Outposts에서 로컬로 실행할 수 있습니다. 이 옵션을 통해 클라우드에 대한 일시적인 네트워크 연결 해제로 인해 발생할 수 있는 애플리케이션 가동 중지 위험을 완화할 수 있습니다. 광케이블 절단 또는 기상 현상이 이러한 네트워크 연결 해제의 원인일 수 있습니다. 전체 Amazon EKS 클러스터가 Outposts에서 로컬로 실행되기 때문에 애플리케이션을 계속 사용할 수 있습니다. 클라우드에 대한 네트워크 연결 해제 중 클러스터 작업을 수행할 수 있습니다. 자세한 내용은 [the section called “연결 해제 준비”](#) 섹션을 참조하세요. Outposts부터 상위 AWS 리전까

지 네트워크 연결 품질이 우려되고 네트워크 연결 해제를 통한 고가용성이 필요한 경우 로컬 클러스터 배포 옵션을 사용합니다.

확장 클러스터가 있으면 Kubernetes 컨트롤 플레인이 상위 AWS 리전에서 실행되기 때문에 Outpost의 용량을 절약할 수 있습니다. Outpost부터 AWS 리전까지 안정적이고 중복된 네트워크 연결에 투자할 수 있으면 이 옵션이 적합합니다. 이 옵션에는 네트워크 연결 품질이 매우 중요합니다. Kubernetes에서 Kubernetes 컨트롤 플레인과 노드 간 네트워크 연결 해제를 처리하는 방법에서는 애플리케이션 가동 중지가 발생할 수도 있습니다. Kubernetes의 동작에 대한 자세한 내용은 쿠버네티스 문서의 [스케줄링](#), [선점\(Preemption\)](#), [축출\(Eviction\)](#)을 참조하세요.

배포 옵션 비교

다음 표에 두 옵션의 차이점이 비교되어 있습니다.

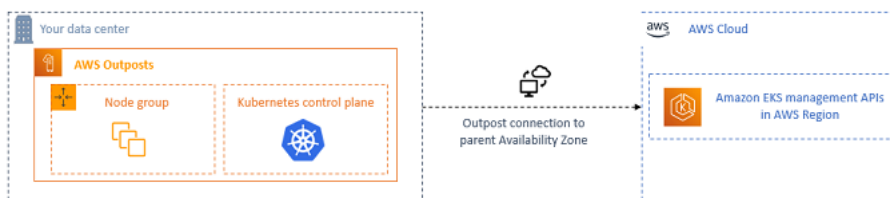
Feature	확장 클러스터	로컬 클러스터
Kubernetes 제어 영역 위치	AWS 리전	Outpost
Kubernetes 컨트롤 플레인 계정	AWS 계정	사용자 계정
지역별 가용성	서비스 엔드포인트 참조	미국 동부(오하이오), 미국 동부(버지니아 북부), 미국 서부(캘리포니아 북부), 미국 서부(오레곤), 아시아 태평양(서울), 아시아 태평양(싱가포르), 아시아 태평양(시드니), 아시아 태평양(도쿄), 캐나다(중부), 유럽(프랑크푸르트), 유럽(아일랜드), 유럽(런던), 중동(바레인) 및 남아메리카(상파울루)
Kubernetes 마이너 버전	지원되는 Amazon EKS 버전	지원되는 Amazon EKS 버전
플랫폼 버전	the section called “플랫폼 버전” 섹션 참조	the section called “EKS 플랫폼 버전” 섹션 참조
Outpost 폼 팩터	Outpost 랙	Outpost 랙

Feature	확장 클러스터	로컬 클러스터
사용자 인터페이스	AWS Management Console, AWS CLI, Amazon EKS API, eksctl, AWS CloudFormation 및 Terraform	AWS Management Console, AWS CLI, Amazon EKS API, eksctl, AWS CloudFormation 및 Terraform
관리형 정책	AmazonEKSClusterPolicy 및 the section called “AWS 관리형 정책: AmazonEKS ServiceRolePolicy”	AmazonEKSLocalOutpostClusterPolicy 및 the section called “AWS 관리형 정책: AmazonEKSLocalOutpostServiceRolePolicy”
클러스터 VPC 및 서브넷	the section called “VPC 및 서브넷 요구 사항” 섹션 참조	the section called “VPC 및 서브넷 생성” 섹션 참조
클러스터 엔드포인트 액세스	퍼블릭이나 프라이빗 또는 둘 다	프라이빗 전용
Kubernetes API 서버 인증	AWS Identity and Access Management(IAM) 및 OIDC	IAM 및 x.509 인증서
노드 유형	자체 관리형만	자체 관리형만
노드 컴퓨팅 유형	Amazon EC2 온디맨드	Amazon EC2 온디맨드
노드 스토리지 유형	Amazon EBS gp2 및 로컬 NVMe SSD	Amazon EBS gp2 및 로컬 NVMe SSD
Amazon EKS 최적화 AMI	Amazon Linux Windows 및 Bottlerocket	Amazon Linux만 해당
IP 버전	IPv4 전용	IPv4 전용
추가 기능	Amazon EKS 추가 기능 또는 자체 관리형 추가 기능	자체 관리형 추가 기능만
기본 컨테이너 네트워크 인터페이스	Kubernetes용 Amazon VPC CNI 플러그인	Kubernetes용 Amazon VPC CNI 플러그인

Feature	확장 클러스터	로컬 클러스터
Kubernetes 컨트롤 플레인 로그	Amazon CloudWatch Logs	Amazon CloudWatch Logs
로드 밸런싱	AWS 로드 밸런서 컨트롤러 를 사용하여 Application Load Balancer만 프로비저닝(Network Load Balancer 없음)	AWS 로드 밸런서 컨트롤러 를 사용하여 Application Load Balancer만 프로비저닝(Network Load Balancer 없음)
보안 암호 봉투 암호화	the section called “비밀 암호화 활성화” 섹션 참조	지원되지 않음
서비스 계정에 대한 IAM 역할	the section called “IRSA를 사용한 자격 증명” 섹션 참조	지원되지 않음
문제 해결	문제 해결 섹션 참조	the section called “클러스터 문제 해결” 부분 참조

고가용성을 위해 AWS Outposts에서 로컬 Amazon EKS 클러스터 생성

로컬 클러스터를 사용하여 전체 Amazon EKS 클러스터를 AWS Outposts에서 로컬로 실행할 수 있습니다. 그러면 클라우드에 대한 일시적인 네트워크 연결 해제로 인해 발생할 수 있는 애플리케이션 가동 중지 위험 완화에 도움이 됩니다. 광케이블 절단 또는 기상 현상이 이러한 연결 해제의 원인일 수 있습니다. 전체 Kubernetes 클러스터가 Outposts에서 로컬로 실행되기 때문에 애플리케이션을 계속 사용할 수 있습니다. 클라우드에 대한 네트워크 연결 해제 중 클러스터 작업을 수행할 수 있습니다. 자세한 내용은 [the section called “연결 해제 준비”](#) 섹션을 참조하세요. 다음 다이어그램은 로컬 클러스터 배포를 보여줍니다.



로컬 클러스터는 일반적으로 Outposts 랙과 함께 사용할 수 있습니다.

지원되는 AWS 리전

미국 동부(오하이오), 미국 동부(버지니아 북부), 미국 서부(캘리포니아 북부), 미국 서부(오레곤), 아시아 태평양(서울), 아시아 태평양(싱가포르), 아시아 태평양(시드니), 아시아 태평양(도쿄), 캐나다(중부), 유럽(프랑크푸르트), 유럽(아일랜드), 유럽(런던), 중동(바레인) 및 남아메리카(상파울루) AWS 리전에서 로컬 클러스터를 생성할 수 있습니다. 지원되는 기능에 대한 자세한 내용을 알아보려면 [the section called “배포 옵션 비교”](#) 섹션을 참조하세요.

AWS Outposts에 Amazon EKS 클러스터 배포

이 주제에서는 Outpost에서 로컬 클러스터를 실행할 때 고려할 내용의 개요를 제공합니다. 이 주제에서는 Outpost에 로컬 클러스터를 배포하는 방법에 대한 지침도 제공합니다.

Important

- 이러한 고려 사항은 관련 Amazon EKS 설명서에 나와 있지 않습니다. 기타 Amazon EKS 설명서 주제가여기의 고려 사항과 충돌하는 경우 여기의 고려 사항을 따릅니다.
- 이러한 고려 사항은 변경될 수 있으며 자주 변경될 수 있습니다. 따라서 이 주제를 주기적으로 검토하는 것이 좋습니다.
- AWS 클라우드에서 클러스터를 생성하기 위한 고려 사항과 많은 고려 사항이 다릅니다.

- 로컬 클러스터는 Outpost 랙만 지원합니다. 단일 로컬 클러스터는 단일 논리적 Outpost를 구성하는 여러 물리적 Outpost 랙에서 실행할 수 있습니다. 단일 로컬 클러스터는 여러 논리적 Outposts에서 실행할 수 없습니다. 각 논리적 Outpost에는 단일 Outpost ARN이 있습니다.
- 로컬 클러스터는 Outpost의 계정에서 Kubernetes 컨트롤 플레인을 실행하고 관리합니다. Kubernetes 컨트롤 플레인 인스턴스에 대한 워크로드를 실행하거나 Kubernetes 컨트롤 플레인 구성 요소를 수정할 수 없습니다. 이러한 노드는 Amazon EKS 서비스에서 관리합니다. Kubernetes 컨트롤 플레인에 대한 변경 사항은 패치와 같은 자동 Amazon EKS 관리 작업 시 유지되지 않습니다.
- 로컬 클러스터는 자체 관리형 추가 기능 및 자체 관리형 Amazon Linux 노드 그룹을 지원합니다. [Kubernetes용 Amazon VPC CNI 플러그인](#), [kube-proxy](#) 및 [CoreDNS](#) 추가 기능은 로컬 클러스터에 자동으로 설치됩니다.
- 로컬 클러스터는 Outposts에서 Amazon EBS를 사용해야 합니다. Outpost에는 Kubernetes 컨트롤 플레인 스토리지에 사용할 수 있는 Amazon EBS가 있어야 합니다.

- 로컬 클러스터는 Outposts에서 Amazon EBS를 사용합니다. Outpost에는 Kubernetes 컨트롤 플레인 스토리지에 사용할 수 있는 Amazon EBS가 있어야 합니다. Outposts는 Amazon EBS gp2 볼륨만 지원합니다.
- Amazon EBS 지원 Kubernetes PersistentVolumes는 Amazon EBS CSI 드라이버를 사용하여 지원됩니다.
- 로컬 클러스터의 컨트롤 플레인 인스턴스는 [가용성이 높은 스택 토폴로지](#)로 설정됩니다. 쿼럼을 유지하려면 컨트롤 플레인 인스턴스 3개 중 2개가 항상 정상 상태여야 합니다. 쿼럼이 손실된 경우 새 관리형 인스턴스를 활성화하려면 일부 서비스 측 작업이 필요하므로 AWS 지원팀에 문의하세요.

사전 조건

- [Outposts 배포 옵션, 용량 고려 사항에 따라 AWS Outposts에서 Amazon EKS 클러스터의 인스턴스 유형 및 배치 그룹 선택하기](#), [VPC 요구 사항 및 고려 사항](#)을 숙지하세요.
- 기존 Outpost. 자세한 내용은 [AWS Outposts이란 무엇입니까?](#) 섹션을 참조하세요.
- 컴퓨터 또는 AWS CloudShell에 설치된 kubectl 명령줄 도구. 버전은 클러스터의 Kubernetes 버전과 동일하거나 최대 하나 이전 또는 이후의 마이너 버전일 수 있습니다. 예를 들어, 클러스터 버전이 1.29인 경우 kubectl 버전 1.28, 1.29 또는 1.30를 함께 사용할 수 있습니다. kubectl을 설치하거나 업그레이드하려면 [the section called “kubectl 및 eksctl 설정”](#) 부분을 참조하세요.
- 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.
- Amazon EKS 클러스터에 대한 create 및 describe 작업을 수행할 수 있는 권한이 있는 IAM 보안 주체(사용자 또는 역할). 자세한 내용은 [the section called “Outpost에서 로컬 Kubernetes 클러스터 생성”](#) 및 [the section called “모든 클러스터 나열 또는 설명”](#) 섹션을 참조하세요.

로컬 Amazon EKS 클러스터가 생성될 때 클러스터를 생성하는 [IAM 보안 주체](#)가 영구적으로 추가됩니다. 위탁자는 특히 Kubernetes RBAC 권한 부여 표에 관리자로 추가됩니다. 이 엔터티에는 `system:masters` 권한이 있습니다. 이 엔터티의 ID는 클러스터 구성에 표시되지 않습니다. 따라서 클러스터를 생성한 엔터티를 기록하고 절대로 삭제하지 않도록 하는 것이 중요합니다. 처음에는 서버를 생성한 위탁자만 kubectl을 사용하여 Kubernetes API 서버를 직접적으로 직접 호출할 수 있습니다.

콘솔을 사용하여 클러스터를 생성하는 경우 클러스터에서 `kubectl` 명령을 실행할 때 동일한 IAM 보안 인증 정보가 AWS SDK 보안 인증 정보 체인에 있어야 합니다. 클러스터가 생성되면 클러스터에 대한 액세스 권한을 다른 IAM 보안 주체에 부여할 수 있습니다.

Amazon EKS 로컬 클러스터 생성

이 페이지에 설명된 다음 도구를 사용하여 로컬 클러스터를 생성할 수 있습니다.

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)

[AWS CLI](#), [Amazon EKS API](#), [AWS SDKs](#), [AWS CloudFormation](#) 또는 [Terraform](#)을 사용하여 Outposts에 클러스터를 생성할 수도 있습니다.

eksctl

eksctl를 사용하여 클러스터를 생성하려면

1. 장치 또는 AWS CloudShell에 eksctl 명령줄 도구의 버전 0.210.0 이상을 설치합니다. eksctl을 설치 또는 업그레이드하려면 eksctl 설명서에서 [설치](#)를 참조하세요.
2. 다음 콘텐츠를 디바이스에 복사합니다. 다음 값을 바꾼 다음 수정된 명령을 실행하여 `outpost-control-plane.yaml` 파일을 생성합니다.
 - **region-code**를 클러스터를 만들려는 [지원되는 AWS 리전](#)으로 바꿉니다.
 - **my-cluster**를 클러스터의 이름으로 바꿉니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다.
 - **vpc-ExampleID1**과 **subnet-ExampleID1**을 기존 VPC 및 서브넷의 ID로 바꿉니다. VPC 및 서브넷은 [AWS Outposts에서 Amazon EKS 클러스터에 대한 VPC 및 서브넷 생성](#)의 요구 사항을 충족해야 합니다.
 - **uniqueid**를 Outpost의 ID로 바꿉니다.
 - **m5.large**를 Outpost에서 사용 가능한 인스턴스 유형으로 바꿉니다. 인스턴스 유형을 선택하기 전에 [the section called “용량 고려 사항”](#) 섹션을 참조하세요. 3개의 컨트롤 플레인 인스턴스가 배포됩니다. 이 수는 변경할 수 없습니다.

```
cat >outpost-control-plane.yaml <<EOF
apiVersion: eksctl.io/v1alpha5
```

```
kind: ClusterConfig

metadata:
  name: my-cluster
  region: region-code
  version: "1.32"

vpc:
  clusterEndpoints:
    privateAccess: true
  id: "vpc-vpc-ExampleID1"
  subnets:
    private:
      outpost-subnet-1:
        id: "subnet-subnet-ExampleID1"

outpost:
  controlPlaneOutpostARN: arn:aws:outposts:region-code:111122223333:outpost/op-uniqueid
  controlPlaneInstanceType: m5.large
EOF
```

사용 가능한 모든 옵션 및 기본값의 전체 목록은 eksctl 설명서의 [AWS Outposts 지원](#)과 [구성 파일 스키마](#)를 참조하세요.

- 이전 단계에서 생성한 구성 파일을 사용하여 클러스터를 생성합니다. eksctl은 클러스터를 배포할 Outpost에 VPC와 서브넷 하나를 생성합니다.

```
eksctl create cluster -f outpost-control-plane.yaml
```

클러스터 프로비저닝에는 몇 분 정도 걸립니다. 클러스터가 생성되는 동안 여러 줄의 출력이 나타납니다. 출력의 마지막 줄은 다음 예제와 유사합니다.

```
[#] EKS cluster "my-cluster" in "region-code" region is ready
```

Tip

eksctl을 사용하여 클러스터를 생성할 때 지정할 수 있는 대부분의 옵션을 보려면 eksctl create cluster --help 명령을 사용합니다. 사용 가능한 옵션을 모두 확인하려면

config 파일을 사용할 수 있습니다. 자세한 내용은 eksctl 설명서에서 [config 파일 사용](#) 및 [config 파일 스키마](#) 부분을 참조하세요. GitHub에서 [구성 파일 예제](#)를 찾을 수 있습니다.

eksctl 명령은 클러스터를 생성한 IAM 위탁자(사용자 또는 역할)에 대한 [액세스 항목](#)을 자동으로 생성하고 IAM 위탁자 관리자에게 클러스터의 Kubernetes 객체에 대한 권한을 부여합니다. 클러스터 생성자가 클러스터의 Kubernetes 객체에 대한 관리자 액세스 권한을 갖지 않도록 하려면 이전 구성 파일에 bootstrapClusterCreatorAdminPermissions: false 텍스트를 추가합니다 (metadata, vpc 및 outpost와 같은 수준). 옵션을 추가한 경우 클러스터를 생성한 후 하나 이상의 IAM 위탁자에 대한 액세스 항목을 생성해야 합니다. 그렇지 않으면 어떤 IAM 위탁자도 클러스터의 Kubernetes 객체에 액세스할 수 없습니다.

AWS Management Console

AWS Management Console를 사용하여 클러스터를 생성하려면

1. Amazon EKS 요구 사항을 충족하는 기존 VPC와 서브넷이 필요합니다. 자세한 내용은 [the section called “VPC 및 서브넷 생성”](#) 섹션을 참조하세요.
2. 이미 로컬 클러스터 IAM 역할이 있거나 eksctl을 사용하여 클러스터를 생성하려는 경우 이 단계를 건너뛸 수 있습니다. 기본적으로 eksctl은 사용자를 위한 역할을 생성합니다.
 - a. 다음 명령을 실행하여 IAM 신뢰 정책 JSON 파일을 생성합니다.

```
cat >eks-local-cluster-role-trust-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

- b. Amazon EKS 클러스터 IAM 역할을 생성합니다. [IAM 보안 주체](#)를 생성하려면 역할을 생성하는 IAM 보안 주체에 iam:CreateRole 작업(권한)을 할당해야 합니다.

```
aws iam create-role --role-name myAmazonEKSLocalClusterRole --assume-role-policy-document file://eks-local-cluster-role-trust-policy.json
```

- c. [AmazonEKSLocalOutpostClusterPolicy](#) Amazon EKS 관리형 정책을 역할에 연결합니다. IAM 정책을 [IAM 보안 주체](#)에 연결하려면 정책을 연결하는 보안 주체에 iam:AttachUserPolicy 또는 iam:AttachRolePolicy의 IAM 작업(권한) 중 하나를 할당해야 합니다.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/AmazonEKSLocalOutpostClusterPolicy --role-name myAmazonEKSLocalClusterRole
```

3. [Amazon EKS 콘솔](#)을 엽니다.

4. 콘솔 화면 상단에서 [지원되는 AWS 리전](#)을 선택했는지 확인합니다.

5. 클러스터 추가를 선택하고 생성을 선택합니다.

6. 클러스터 구성 페이지에서 다음 필드의 값을 입력하거나 선택합니다.

- Kubernetes 컨트롤 플레인 위치 – AWS Outposts를 선택합니다.
- Outpost ID - 컨트롤 플레인을 생성할 Outpost의 ID를 선택합니다.
- 인스턴스 유형 - 인스턴스 유형을 선택합니다. Outpost에서 사용 가능한 인스턴스 유형만 표시됩니다. 드롭다운 목록에서 각 인스턴스 유형은 인스턴스 유형이 권장되는 노드 수를 설명합니다. 인스턴스 유형을 선택하기 전에 [the section called “용량 고려 사항”](#) 부분을 참조하세요. 모든 복제본은 동일한 인스턴스 유형을 사용하여 배포됩니다. 클러스터가 생성된 후에는 인스턴스 유형을 변경할 수 없습니다. 3개의 컨트롤 플레인 인스턴스가 배포됩니다. 이 수는 변경할 수 없습니다.
- 이름 - 클러스터의 이름 AWS 계정에서 고유해야 합니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다.
- Kubernetes 버전 – 클러스터에 사용하려는 Kubernetes 버전을 선택합니다. 이전 버전이 필요한 경우가 아니면 최신 버전을 선택하는 것이 좋습니다.
- 클러스터 서비스 역할 - 이전 단계에서 생성한 Amazon EKS 클러스터 IAM 역할을 선택하여 Kubernetes 컨트롤 플레인에서 AWS 리소스를 관리하게 할 수 있습니다.
- Kubernetes 클러스터 관리자 액세스 – Kubernetes 클러스터를 생성하는 IAM 위탁자(역할 또는 사용자)가 클러스터의 Kubernetes 객체에 대한 관리자 액세스 권한을 갖도록 하려면 기본값(허용)을 그대로 수락합니다. Amazon EKS는 IAM 보안 주체에 대한 액세스 항목을 생성하고 클러스터 관리자에게 액세스 항목에 대한 권한을 부여합니다. 액세스 항목에 대한 자세한 내용은 [the section called “권한 부여”](#) 섹션을 참조하세요.

클러스터를 생성하는 주체가 아닌 다른 IAM 위탁자가 Kubernetes 클러스터 객체에 대한 관리자 액세스 권한을 갖도록 하려면 거부 옵션을 선택합니다. 클러스터를 생성한 후 액세스 항목을 생성할 수 있는 IAM 권한이 있는 IAM 위탁자는 Kubernetes 클러스터 객체에 액세스해야 하는 모든 IAM 위탁자에 대한 액세스 항목을 추가할 수 있습니다. 필수 IAM 권한에 대한 자세한 내용은 서비스 권한 부여 참조에서 [Actions defined by Amazon Elastic Kubernetes Service](#)를 참조하세요. 거부 옵션을 선택하고 액세스 항목을 생성하지 않으면 어떤 IAM 위탁자도 클러스터의 Kubernetes 객체에 액세스할 수 없습니다.

- 태그-(선택사항) 클러스터에 태그를 추가합니다. 자세한 내용은 [the section called “리소스에 태그”](#) 섹션을 참조하세요. 이 페이지를 모두 완료하면 다음을 선택합니다.

7. 네트워킹 지정 페이지에서 다음 필드의 값을 선택합니다.

- VPC – 기존 VPC를 선택합니다. VPC에는 생성하려는 클러스터, 노드 및 기타 Kubernetes 리소스에 사용 가능한 IP 주소가 충분해야 합니다. VPC는 [VPC 요구 사항 및 고려 사항의 요구 사항](#)을 충족해야 합니다.
- 서브넷 - 기본적으로 이전 필드에서 지정한 VPC에서 사용 가능한 모든 서브넷이 사전 선택됩니다. 선택한 서브넷은 [서브넷 요구 사항 및 고려 사항](#)의 요구 사항을 충족해야 합니다.
- 보안 그룹 – (선택사항) 생성하는 네트워크 인터페이스에 Amazon EKS가 연결하려는 보안 그룹을 하나 이상 지정합니다. Amazon EKS에서는 클러스터와 VPC 간 통신을 활성화하는 보안 그룹을 자동으로 생성합니다. Amazon EKS는 이 보안 그룹과 사용자가 선택한 보안 그룹을 생성하는 네트워크 인터페이스에 연결합니다. Amazon EKS에서 생성하는 클러스터 보안 그룹에 대한 자세한 내용을 알아보려면 [the section called “보안 그룹 요구 사항”](#) 부분을 참조하세요. Amazon EKS가 생성하는 클러스터 보안 그룹에서 규칙을 수정할 수 있습니다. 자체 보안 그룹을 추가하도록 선택한 경우 클러스터 생성 후 선택한 보안 그룹은 변경할 수 없습니다. 온프레미스 호스트가 클러스터 엔드포인트와 통신하려면 클러스터 보안 그룹의 인바운드 트래픽을 허용해야 합니다. 수신 및 송신 인터넷 연결이 없는 클러스터(프라이빗 클러스터라고도 함)의 경우, 다음 중 하나를 수행해야 합니다.
 - 필수 VPC 엔드포인트와 연결된 보안 그룹을 추가합니다. 필요한 엔드포인트에 대한 자세한 내용은 [AWS 서비스에 대한 서브넷 액세스](#)의 [the section called “인터페이스 VPC 종단점 사용”](#) 섹션을 참조하세요.
 - VPC 엔드포인트와 연결된 보안 그룹의 트래픽을 허용하려면 Amazon EKS에서 생성한 보안 그룹을 수정합니다. 이 페이지를 모두 완료하면 다음을 선택합니다.

8. 관찰성 구성 페이지에서 설정할 지표 및 컨트롤 플레인 로깅 옵션을 선택할 수 있습니다. 기본적으로 각 로그 유형은 해제되어 있습니다.

- Prometheus 지표 옵션에 자세한 내용은 [the section called “1단계: Prometheus 지표 켜기”](#) 섹션을 참조하세요.

- 컨트롤 플레인 로깅 옵션에 대한 자세한 내용은 [the section called “컨트롤 플레인 로그”](#) 섹션을 참조하세요. 이 페이지를 모두 완료하면 다음을 선택합니다.
9. 검토 및 생성 페이지에서 이전 페이지에서 입력하거나 선택한 정보를 검토합니다. 변경해야 하는 경우 편집을 선택합니다 만족하는 경우 생성을 선택합니다. 클러스터가 프로비저닝되는 동안 상태 필드에는 생성 중이 표시됩니다.

클러스터 프로비저닝에는 몇 분 정도 걸립니다.

Amazon EKS 로컬 클러스터 보기

1. 클러스터가 생성된 후 생성된 Amazon EC2 컨트롤 플레인 인스턴스를 볼 수 있습니다.

```
aws ec2 describe-instances --query 'Reservations[*].Instances[*].{Name:Tags[?Key==`Name`][0].Value}' | grep my-cluster-control-plane
```

예제 출력은 다음과 같습니다.

```
"Name": "my-cluster-control-plane-id1"
"Name": "my-cluster-control-plane-id2"
"Name": "my-cluster-control-plane-id3"
```

컨트롤 플레인 인스턴스에서 워크로드가 예약되지 않도록 각 인스턴스는 `node-role.eks-local.amazonaws.com/control-plane`으로 테인트됩니다. 테인트에 대한 자세한 내용은 쿠버네티스 문서의 [테인트\(Taints\)](#)와 [톨러레이션\(Tolerations\)](#)을 참조하세요. Amazon EKS는 로컬 클러스터의 상태를 지속적으로 모니터링합니다. 보안 패치 및 비정상 인스턴스 복구와 같은 자동 관리 작업을 수행합니다. 로컬 클러스터가 클라우드에서 연결 해제되면 재연결 시 클러스터가 정상 상태로 복구되도록 조치를 완료합니다.

2. `eksctl`을 사용하여 클러스터를 생성한 경우 이 단계를 건너뛸 수 있습니다. `eksctl`에서 자동으로 이 단계를 완료합니다. `kubectl config` 파일에 새 컨텍스트를 추가하여 `kubectl`이 클러스터와 통신하도록 사용 설정합니다. 파일 생성 설치 및 업데이트 방법에 대한 지침은 [the section called “kubectl을 사용하여 클러스터에 액세스”](#) 부분을 참조하세요.

```
aws eks update-kubeconfig --region region-code --name my-cluster
```

예제 출력은 다음과 같습니다.

```
Added new context arn:aws:eks:region-code:111122223333:cluster/my-cluster to /home/username/.kube/config
```

3. 로컬 클러스터의 Kubernetes API 서버에 연결하려면 서브넷의 로컬 게이트웨이에 액세스하거나 VPC 내에서 연결해야 합니다. Outpost 랙을 온프레미스 네트워크에 연결하는 방법을 자세히 알아보려면 AWS Outposts 사용 설명서의 [랙용 로컬 게이트웨이 작동 방식](#)을 참조하세요. Direct VPC Routing을 사용하며 Outpost 서브넷에 로컬 게이트웨이에 대한 경로가 있으면 Kubernetes 컨트롤 플레인 인스턴스의 프라이빗 IP 주소가 로컬 네트워크를 통해 자동으로 브로드캐스트됩니다. 로컬 클러스터의 Kubernetes API 서버 엔드포인트는 Amazon Route 53(Route 53)에서 호스팅됩니다. API 서비스 엔드포인트는 퍼블릭 DNS 서버에서 Kubernetes API 서버의 프라이빗 IP 주소로 확인할 수 있습니다.

로컬 클러스터의 Kubernetes 컨트롤 플레인 인스턴스는 클러스터 수명 주기 동안 변경되지 않는 고정 프라이빗 IP 주소를 사용하여 정적 탄력적 네트워크 인터페이스로 구성됩니다. Kubernetes API 서버와 상호 작용하는 머신은 네트워크 연결이 해제된 동안 Route 53에 연결되지 않을 수도 있습니다. 이 경우 계속되는 작업을 위해 고정 프라이빗 IP 주소로 /etc/hosts를 구성하는 것이 좋습니다. 또한 로컬 DNS 서버를 설정하고 Outpost에 연결하는 것이 좋습니다. 자세한 내용은 [AWS Outposts 문서](#)를 참조하세요. 다음 명령을 실행하여 클러스터와 설정된 통신을 확인합니다.

```
kubectl get svc
```

예제 출력은 다음과 같습니다.

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
kubernetes	ClusterIP	10.100.0.1	<none>	443/TCP	28h

4. (선택사항) 로컬 클러스터가 AWS 클라우드에서 연결 해제된 상태일 때 인증을 테스트합니다. 지침은 [the section called “연결 해제 준비”](#) 섹션을 참조하세요.

내부 리소스

Amazon EKS는 클러스터에 다음 리소스를 생성합니다. 리소스는 Amazon EKS 내부용입니다. 클러스터가 제대로 작동하려면 이러한 리소스를 편집하거나 수정하지 않습니다.

- 다음 [미러 포트](#):
 - aws-iam-authenticator-*node-hostname*
 - eks-certificates-controller-*node-hostname*

- etcd-*node-hostname*
- kube-apiserver-*node-hostname*
- kube-controller-manager-*node-hostname*
- kube-scheduler-*node-hostname*
- 다음 자체 관리형 추가 기능:
 - kube-system/coredns
 - kube-system/ kube-proxy(첫 번째 노드를 추가할 때까지 생성되지 않음)
 - kube-system/aws-node(첫 번째 노드를 추가할 때까지 생성되지 않음). 로컬 클러스터는 클러스터 네트워킹에 Kubernetes용 Amazon VPC CNI 플러그인을 사용합니다. 컨트롤 플레인 인스턴스(aws-node-controlplane-*라는 포드)의 구성을 변경하지 않습니다. 플러그인에서 새 네트워크 인터페이스를 생성할 때 기본값 변경에 사용할 수 있는 구성 변수가 있습니다. 자세한 내용을 알아보려면 GitHub의 [설명서](#)를 참조하세요.
- 다음 서비스:
 - default/kubernetes
 - kube-system/kube-dns
- 이름이 eks.system인 PodSecurityPolicy
- 이름이 eks:system:podsecuritypolicy인 ClusterRole
- 이름이 eks:system인 ClusterRoleBinding
- [클러스터 보안 그룹](#) 외에도 Amazon EKS에서는 AWS 계정에 이름이 eks-local-internal-donot-use-or-edit-*cluster-name-uniqueid* 인 보안 그룹을 생성합니다. 이 보안 그룹을 사용하면 컨트롤 플레인 인스턴스에서 실행되는 Kubernetes 구성 요소 간에 트래픽이 자유롭게 흐를 수 있습니다.

권장되는 다음 단계:

- [클러스터를 생성한 IAM 보안 주체에 AWS Management Console에서 Kubernetes 리소스를 볼 수 있는 필요한 권한 부여](#)
- [클러스터에 IAM 엔터티에 액세스 권한 부여](#) 엔터티가 Amazon EKS 콘솔에서 Kubernetes 리소스를 보도록 하려면 엔터티에 [필수 권한](#)을 부여합니다.
- [클러스터에 대한 로깅 구성](#)
- [네트워크 연결 해제](#) 중 어떤 일이 발생하는지 숙지합니다.
- [클러스터에 노드 추가](#)

- etcd에 대한 백업 계획 설정을 고려합니다. Amazon EKS는 로컬 클러스터에 대한 etcd의 자동 백업 및 복원을 지원하지 않습니다. 자세한 내용은 Kubernetes Documentation의 [Backing up an etcd cluster](#)를 참조하세요. 두 가지 주요 옵션은 etcdctl을 사용하여 스냅샷 생성 또는 Amazon EBS 스토리지 볼륨 백업 사용을 자동화합니다.

Kubernetes 및 AWS Outposts에 대한 Amazon EKS 플랫폼 버전 알아보기

로컬 클러스터 플랫폼 버전은 AWS Outposts에 있는 Amazon EKS 클러스터의 기능을 나타냅니다. 버전에는 Kubernetes API 서버 플러그가 활성화된 Kubernetes 컨트롤 플레인에서 실행되는 구성 요소가 포함됩니다. 현재 Kubernetes 패치 버전도 포함됩니다. 각 Kubernetes 마이너 버전에는 하나 이상의 연결된 플랫폼 버전이 있습니다. 서로 다른 Kubernetes 마이너 버전에 대한 플랫폼 버전은 독립적입니다. 클라우드의 로컬 클러스터와 Amazon EKS 클러스터의 플랫폼 버전은 독립적입니다.

1.31과 같은 로컬 클러스터에 새 Kubernetes 마이너 버전을 사용할 수 있는 경우 해당 Kubernetes 마이너 버전의 초기 플랫폼 버전은 eks-local-outposts.1에서 시작합니다. 그러나 Amazon EKS에서는 새 Kubernetes 제어 영역 설정을 활성화하고 보안 수정 사항을 제공하기 위해 새 플랫폼 버전을 주기적으로 릴리스합니다.

마이너 버전에 대해 새 로컬 클러스터 플랫폼 버전을 사용할 수 있게 된 경우 다음과 같은 사항이 발생합니다.

- 플랫폼 버전 번호가 증가합니다(eks-local-outposts.n+1).
- Amazon EKS에서 기존의 모든 로컬 클러스터를 해당하는 Kubernetes 마이너 버전에 대한 최신 플랫폼 버전으로 자동으로 업데이트합니다. 기존 플랫폼 버전의 자동 업데이트는 증분 방식으로 롤아웃됩니다. 롤아웃 프로세스는 3개의 인스턴스가 모두 새 인스턴스로 교체될 때까지 Outpost에서 실행되는 관리형 Kubernetes 컨트롤 플레인 인스턴스를 한 번에 하나씩 교체합니다.
- 서비스 중단 위험이 있으면 Kubernetes 컨트롤 플레인 인스턴스 교체 프로세스 진행이 중지됩니다. Amazon EKS는 다른 2개의 Kubernetes 컨트롤 플레인 인스턴스가 정상이고 클러스터 노드로서 모든 준비 조건을 통과하는 경우에만 인스턴스 교체를 시도합니다.
- 플랫폼 버전 롤아웃을 완료하는 데 일반적으로 30분 미만이 소요됩니다. 클러스터가 장기간 UPDATING 상태를 유지하는 경우 [the section called “클러스터 문제 해결”](#) 섹션을 참조하고 AWS Support에 도움을 요청하세요. AWS Support의 지시가 없는 한 Kubernetes 컨트롤 플레인 인스턴스를 수동으로 종료하지 마세요.
- Amazon EKS에서는 해당 패치 버전으로 새 AMI를 게시할 수 있습니다. 모든 패치 버전은 단일 Kubernetes 마이너 버전에 대한 노드 AMI와 Kubernetes 컨트롤 플레인 간에 호환됩니다.

새 플랫폼 버전은 주요 변경 사항을 도입하거나 서비스 중단을 초래하지 않습니다.

로컬 클러스터는 항상 지정된 Kubernetes 버전에 사용 가능한 최신 플랫폼 버전(eks-local-outposts.n)으로 생성됩니다.

현재 및 최근 플랫폼 버전은 다음 표에 설명되어 있습니다.

이 특정 설명서 페이지의 모든 소스 파일 변경 사항에 대한 알림을 받으려면 RSS 리더를 사용하여 다음 URL을 구독하세요.

<https://github.com/awsdocs/amazon-eks-user-guide/commits/mainline/latest/ug/outposts/eks-outposts-platform-versions.adoc.atom>

Kubernetes 버전 1.31

다음 승인 컨트롤러는 모든 1.31 플랫폼 버전(CertificateApproval, CertificateSigning, CertificateSubjectRestriction, ClusterTrustBundleAttest, DefaultIngressClass, DefaultStorageClass, DefaultTolerationSeconds, ExtendedResourceToleration, LimitRanger, MutatingAdmissionWebhook, NamespaceLifecycle, NodeRestriction, PersistentVolumeClaimResize, PodSecurity, Priority, ResourceQuota, RuntimeClass, ServiceAccount, StorageObjectInUseProtection, TaintNodesByCondition, ValidatingAdmissionPolicy, ValidatingAdmissionWebhook)에 대해 활성화됩니다.

Kubernetes 버전	Amazon EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.31.6	eks-local-outposts.1	Outposts의 로컬 Amazon EKS 클러스터용 Kubernetes 버전 v1.31의 최초 릴리스입니다.	2025년 4월 9일

Kubernetes 버전 1.30

다음 승인 컨트롤러는 모든 1.30 플랫폼 버전(CertificateApproval, CertificateSigning, CertificateSubjectRestriction, ClusterTrustBundleAttest, DefaultIngressClass, DefaultStorageClass, DefaultTolerationSeconds, ExtendedResourceToleration,

LimitRanger, MutatingAdmissionWebhook, NamespaceLifecycle, NodeRestriction, PersistentVolumeClaimResize, PodSecurity, Priority, ResourceQuota, RuntimeClass, ServiceAccount, StorageObjectInUseProtection, TaintNodesByCondition, ValidatingAdmissionPolicy, ValidatingAdmissionWebhook)에 대해 활성화됩니다.

Kubernetes 버전	Amazon EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.30.10	eks-local-outposts.3	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. kube-proxy는 v1.30.10으로 업데이트되었습니다. AWS IAM Authenticator가 v0.6.29로 업데이트되었습니다. Kubernetes용 Amazon VPC CNI 플러그인은 v1.19.2(으)로 업데이트되었습니다. CoreDNS가 v1.11.4로 업데이트되었습니다. AWS Cloud Controller Manager가 v1.30.8로 업데이트되었습니다. Bottlerocket 버전이 v1.34.0으로 업데이트되었습니다.	2025년 3월 27일
1.30.7	eks-local-outposts.2	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전	2025년 1월 10일

Kubernetes 버전	Amazon EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
		입니다. kube-proxy 는 v1.30.7으로 업데이트되었습니다. AWS IAM Authenticator가 v0.6.28로 업데이트되었습니다. Kubernetes 용 Amazon VPC CNI 플러그인은 v1.19.0(으)로 업데이트되었습니다. Bottlerocket 버전을 v1.29.0으로 업데이트했습니다.	
1.30.5	eks-local-outposts.1	Outposts의 로컬 Amazon EKS 클러스터용 Kubernetes 버전 v1.30의 최초 릴리스입니다.	2024년 11월 13일

Kubernetes 버전 1.29

다음 승인 컨트롤러는 모든 1.29 플랫폼 버전(CertificateApproval, CertificateSigning, CertificateSubjectRestriction, ClusterTrustBundleAttest, DefaultIngressClass, DefaultStorageClass, DefaultTolerationSeconds, ExtendedResourceToleration, LimitRanger, MutatingAdmissionWebhook, NamespaceLifecycle, NodeRestriction, PersistentVolumeClaimResize, PodSecurity, Priority, ResourceQuota, RuntimeClass, ServiceAccount, StorageObjectInUseProtection, TaintNodesByCondition, ValidatingAdmissionPolicy, ValidatingAdmissionWebhook)에 대해 활성화됩니다.

Kubernetes 버전	Amazon EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
v1.29.14	eks-local-outposts.6	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. kube-proxy는 v1.29.14로 업데이트되었습니다. Kubernetes용 Amazon VPC CNI 플러그인은 v1.19.2(으)로 업데이트되었습니다. CoreDNS가 v1.11.4로 업데이트되었습니다. AWS Cloud Controller Manager가 v1.29.8로 업데이트되었습니다. Bottlerocket 버전이 v1.34.0으로 업데이트되었습니다.	2025년 3월 27일
v1.29.11	eks-local-outposts.5	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. kube-proxy는 v1.29.11로 업데이트되었습니다. Kubernetes용 Amazon VPC CNI 플러그인은 v1.19.0(으)로 업데이트되었습니다.	2025년 1월 10일

Kubernetes 버전	Amazon EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
		CoreDNS 이미지가 v1.11.3 버전으로 업데이트되었습니다. Bottlerocket 버전을 v1.29.0으로 업데이트했습니다.	
1.29.9	eks-local-outposts.4	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. kube-proxy는 v1.29.9으로 업데이트되었습니다. AWS IAM Authenticator가 v0.6.26로 업데이트되었습니다. Bottlerocket 버전을 v1.26.0으로 업데이트했습니다.	2024년 11월 8일
1.29.6	eks-local-outposts.3	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전. Bottlerocket 버전을 v1.22.0으로 업데이트했습니다.	2024년 10월 22일
1.29.6	eks-local-outposts.2	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전. Bottlerocket 버전을 v1.21.0으로 업데이트했습니다.	2024년 8월 27일

Kubernetes 버전	Amazon EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.29.6	eks-local-outposts.1	Outposts의 로컬 Amazon EKS 클러스터용 Kubernetes 버전 v1.29의 최초 릴리스입니다.	2024년 8월 20일

Kubernetes 버전 1.28

다음 승인 컨트롤러는 모든 1.28 플랫폼 버전(CertificateApproval, CertificateSigning, CertificateSubjectRestriction, ClusterTrustBundleAttest, DefaultIngressClass, DefaultStorageClass, DefaultTolerationSeconds, ExtendedResourceToleration, LimitRanger, MutatingAdmissionWebhook, NamespaceLifecycle, NodeRestriction, PersistentVolumeClaimResize, PodSecurity, Priority, ResourceQuota, RuntimeClass, ServiceAccount, StorageObjectInUseProtection, TaintNodesByCondition, ValidatingAdmissionPolicy, ValidatingAdmissionWebhook)에 대해 활성화됩니다.

Kubernetes 버전	Amazon EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.28.15	eks-local-outposts.13	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. kube-proxy v1.28.15 빌드가 업데이트되었습니다. Kubernetes용 Amazon VPC CNI 플러그인이 v1.19.0으로 업데이트되었습니다. AWS Cloud Controller Manager가	2025년 3월 27일

Kubernetes 버전	Amazon EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
		v1.28.11로 업데이트되었습니다.	
1.28.15	eks-local-outposts.12	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. kube-proxy는 v1.28.15로 업데이트되었습니다. Kubernetes용 Amazon VPC CNI 플러그인은 v1.19.0(으)로 업데이트되었습니다. Bottlerocket 버전을 v1.29.0으로 업데이트했습니다.	2025년 1월 10일
1.28.14	eks-local-outposts.11	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. kube-proxy는 v1.28.14으로 업데이트되었습니다. AWS IAM Authenticator가 v0.6.26로 업데이트되었습니다. CoreDNS 이미지가 v1.11.1 버전으로 업데이트되었습니다. Bottlerocket 버전을 v1.26.0으로 업데이트했습니다.	2024년 11월 8일

Kubernetes 버전	Amazon EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.28.10	eks-local-outposts.10	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전. Bottlerocket 버전을 v1.22.0으로 업데이트했습니다.	2024년 10월 22일
1.28.10	eks-local-outposts.9	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전. Bottlerocket 버전을 v1.21.0으로 업데이트했습니다.	2024년 8월 27일
1.28.10	eks-local-outposts.8	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 7월 30일
1.28.10	eks-local-outposts.6	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. kube-proxy는 v1.28.10으로 업데이트되었습니다. AWS IAM Authenticator가 v0.6.20로 업데이트되었습니다. Bottlerocket 버전을 v1.20.2으로 업데이트했습니다.	2024년 6월 19일

Kubernetes 버전	Amazon EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.28.6	eks-local-outposts.5	Bottlerocket 버전이 Outposts에서 로컬 부팅을 지원하기 위한 최신 버그 수정이 포함된 v1.19.3으로 업데이트되었습니다.	2024년 4월 18일
1.28.6	eks-local-outposts.4	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전. Outposts에서 지원 또는 로컬 부팅이 복원되었습니다. 호환성을 위해 Bottlerocket 버전이 v1.15.1로 다운그레이드되었습니다.	2024년 4월 2일
1.28.6	eks-local-outposts.3	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 3월 22일

Kubernetes 버전	Amazon EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.28.6	eks-local-outposts.2	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. kube-proxy는 v1.28.6으로 업데이트되었습니다. AWS IAM Authenticator가 v0.6.17로 업데이트되었습니다. 호환성 이유로 Kubernetes용 Amazon VPC CNI 플러그인이 v1.13.2로 다운그레이드되었습니다. Bottlerocket 버전을 v1.19.2으로 업데이트했습니다.	2024년 3월 8일
1.28.1	eks-local-outposts.1	Outposts의 로컬 Amazon EKS 클러스터용 Kubernetes 버전 v1.28의 최초 릴리스입니다.	2023년 10월 4일

Kubernetes 버전 1.27

다음 승인 컨트롤러는 모든 1.27 플랫폼 버전(CertificateApproval, CertificateSigning, CertificateSubjectRestriction, ClusterTrustBundleAttest, DefaultIngressClass, DefaultStorageClass, DefaultTolerationSeconds, ExtendedResourceToleration, LimitRanger, MutatingAdmissionWebhook, NamespaceLifecycle, NodeRestriction, PersistentVolumeClaimResize, PodSecurity, Priority, ResourceQuota, RuntimeClass, ServiceAccount, StorageObjectInUseProtection, TaintNodesByCondition, ValidatingAdmissionPolicy, ValidatingAdmissionWebhook)에 대해 활성화됩니다.

Kubernetes 버전	Amazon EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.27.16	eks-local-outposts.13	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전. Kubernetes용 Amazon VPC CNI 플러그인은 v1.19.2(으)로 업데이트되었습니다. Bottlerocket 버전이 v1.34.0으로 업데이트되었습니다.	2025년 3월 27일
1.27.16	eks-local-outposts.12	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전. Kubernetes용 Amazon VPC CNI 플러그인은 v1.19.0(으)로 업데이트되었습니다. Bottlerocket 버전을 v1.29.0으로 업데이트했습니다.	2025년 1월 10일
1.27.16	eks-local-outposts.11	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. kube-proxy는 v1.27.16으로 업데이트되었습니다. AWS IAM Authenticator가 v0.6.26로 업데이트되었습니다. CoreDNS 이미지가 v1.11.1 버전으로 업데이트되었습니다. Bottlerocket	2024년 11월 8일

Kubernetes 버전	Amazon EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
		버전을 v1.26.0으로 업데이트했습니다.	
1.27.14	eks-local-outposts.10	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전. Bottlerocket 버전을 v1.22.0으로 업데이트했습니다.	2024년 10월 22일
1.27.14	eks-local-outposts.9	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전. Bottlerocket 버전을 v1.21.0으로 업데이트했습니다.	2024년 8월 27일
1.27.14	eks-local-outposts.8	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 7월 30일
1.27.14	eks-local-outposts.6	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. kube-proxy는 v1.27.14으로 업데이트되었습니다. AWS IAM Authenticator가 v0.6.20로 업데이트되었습니다. Bottlerocket 버전을 v1.20.2으로 업데이트했습니다.	2024년 6월 19일

Kubernetes 버전	Amazon EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.27.10	eks-local-outposts.5	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전.	2024년 4월 2일
1.27.10	eks-local-outposts.4	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. kube-proxy는 v1.27.10으로 업데이트되었습니다. AWS IAM Authenticator가 v0.6.17로 업데이트되었습니다. Bottlerocket 버전을 v1.19.2으로 업데이트했습니다.	2024년 3월 22일
1.27.3	eks-local-outposts.3	보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. kube-proxy는 v1.27.3로 업데이트되었습니다. Kubernetes 용 Amazon VPC CNI 플러그인은 v1.13.2(으)로 업데이트되었습니다.	2023년 7월 14일
1.27.1	eks-local-outposts.2	CoreDNS 이미지가 v1.10.1 버전으로 업데이트되었습니다.	2023년 6월 22일

Kubernetes 버전	Amazon EKS 플랫폼 버전	릴리스 정보	릴리스 날짜
1.27.1	eks-local-outposts.1	Outposts의 로컬 Amazon EKS 클러스터용 Kubernetes 버전 1.27의 최초 릴리스입니다.	2023년 5월 30일

AWS 출력에서 Amazon EKS 클러스터를 위한 VPC 및 서브넷 생성

로컬 클러스터를 생성할 때 Outposts에서 실행되는 하나 이상의 프라이빗 서브넷과 VPC를 지정합니다. 이 주제에서는 로컬 클러스터에 대한 VPC 및 서브넷 요구 사항 및 고려 사항을 간략하게 살펴봅니다.

VPC 요구 사항 및 고려 사항

로컬 클러스터를 생성하는 경우 지정하는 VPC는 다음 요구 사항 및 고려 사항을 충족해야 합니다.

- 생성하려는 로컬 클러스터, 노드 및 기타 Kubernetes 리소스에 대한 충분한 IP 주소가 VPC에 있는지 확인합니다. 사용하려는 VPC에 충분한 IP 주소가 없는 경우 사용 가능한 IP 주소를 증량합니다. 이 작업은 해당 VPC를 통해 [추가 CIDR\(Classless Inter-Domain Routing\) 블록을 연결](#)하여 수행할 수 있습니다. 클러스터를 생성하기 전이나 후에 프라이빗(RFC 1918) 및 퍼블릭(비RFC 1918) CIDR 블록을 VPC에 연결할 수 있습니다. 클러스터가 VPC와 연결된 CIDR 블록을 인식하는 데 최대 5시간이 걸릴 수 있습니다.
- VPC에는 할당된 IP 접두사 또는 IPv6 CIDR 블록이 있을 수 없습니다. 이러한 제약으로 인해 [접두사를 사용하여 Amazon EKS 노드에 더 많은 IP 주소 할당](#) 및 [the section called "IPv6"](#)에서 다루는 정보는 VPC에 적용되지 않습니다.
- 활성화된 DNS 호스트 이름과 DNS 확인이 VPC에 있습니다. 이러한 기능이 없으면 로컬 클러스터 생성에 실패하므로 기능을 활성화하고 클러스터를 다시 생성해야 합니다. 자세한 내용을 알아보려면 Amazon VPC 사용 설명서의 [VPC에 대한 DNS 속성](#)을 참조하세요.
- 로컬 네트워크를 통해 로컬 클러스터에 액세스하려면 VPC가 Outpost의 로컬 게이트웨이 라우팅 테이블과 연결되어 있어야 합니다. 자세한 내용을 알아보려면 AWS Outposts 사용 설명서의 [VPC 연결](#)을 참조하세요.

서브넷 요구 사항 및 고려 사항

클러스터를 생성할 때 하나 이상의 프라이빗 서브넷을 지정합니다. 서브넷을 여러 개 지정하면 Kubernetes 컨트롤 플레인 인스턴스가 서브넷 전체에 고르게 분산됩니다. 2개 이상의 서브넷이 지정된 경우 서브넷이 동일한 Outpost에 있어야 합니다. 아울러 서로 통신하는 적절한 경로와 보안 그룹 권한도 서브넷에 있어야 합니다. 로컬 클러스터를 생성할 때 지정하는 서브넷에서 다음과 같은 요구 사항을 충족해야 합니다.

- 서브넷이 모두 동일한 논리적 Outpost에 있습니다.
- 서브넷에 전체적으로 Kubernetes 컨트롤 플레인 인스턴스에 대해 사용 가능한 IP 주소가 3개 이상 있어야 합니다. 3개의 서브넷이 지정된 경우 각 서브넷에 사용 가능한 IP 주소가 1개 이상 있어야 합니다. 2개의 서브넷이 지정된 경우 각 서브넷에 사용 가능한 IP 주소가 2개 이상 있어야 합니다. 1개의 서브넷이 지정된 경우 서브넷에 사용 가능한 IP 주소가 3개 이상 있어야 합니다.
- 로컬 네트워크를 통해 Kubernetes API 서버에 액세스하기 위한 Outpost 랙의 [로컬 게이트웨이](#)에 대한 경로가 서브넷에 있어야 합니다. 서브넷에 Outpost 랙의 로컬 게이트웨이에 대한 경로가 없는 경우 VPC 내에서 Kubernetes API 서버와 통신해야 합니다.
- 서브넷에서는 IP 주소 기반 이름 지정을 사용해야 합니다. Amazon EC2 [리소스 기반 이름 지정](#)은 Amazon EKS에서 지원되지 않습니다.

AWS 서비스에 대한 서브넷 액세스

Outposts에 있는 로컬 클러스터의 사설 서브넷은 리전 AWS 서비스와 통신할 수 있어야 합니다. 아웃바운드 인터넷 액세스를 위해 [NAT 게이트웨이](#)를 사용하거나, VPC 내에서 모든 트래픽을 비공개로 유지하려는 경우 [인터페이스 VPC 엔드포인트](#)를 사용하여 이를 달성할 수 있습니다.

NAT 게이트웨이 태깅

Outpost의 상위 가용 영역에 있는 퍼블릭 서브넷에서 실행되는 NAT 게이트웨이에 대한 경로가 있는 연결된 라우팅 테이블이 서브넷에 있습니다. 퍼블릭 서브넷에 [인터넷 게이트웨이](#)에 대한 경로가 있어야 합니다. 이 경로는 아웃바운드 인터넷 액세스를 활성화하고 인터넷에서 Outpost의 인스턴스로의 원치 않는 인바운드 연결을 방지합니다.

인터페이스 VPC 종단점 사용

Outst에 있는 로컬 클러스터의 사설 서브넷에 송신 인터넷 연결이 없는 경우에는, 또는 송신 인터넷 연결이 없는 경우에는 클러스터를 생성하기 전에 다음과 같은 인터페이스 VPC [엔드포인트와 게이트웨이 엔드포인트](#)를 생성해야 합니다.

엔드포인트	엔드포인트 유형
com.amazonaws. <i>region-code</i> .ssm	인터페이스
com.amazonaws. <i>region-code</i> .ssmmessages	인터페이스
com.amazonaws. <i>region-code</i> .ec2messages	인터페이스
com.amazonaws. <i>region-code</i> .ec2	인터페이스
com.amazonaws. <i>region-code</i> .secretsmanager	인터페이스
com.amazonaws. <i>region-code</i> .logs	인터페이스
com.amazonaws. <i>region-code</i> .sts	인터페이스
com.amazonaws. <i>region-code</i> .ecr.api	인터페이스
com.amazonaws. <i>region-code</i> .ecr.dkr	인터페이스
com.amazonaws. <i>region-code</i> .s3	게이트웨이

이름은 다음 요구 사항을 충족해야 합니다.

- Outpost의 상위 가용 영역에 있는 프라이빗 서브넷에서 생성됨
- 프라이빗 DNS 이름 활성화
- 사설 송신 서브넷의 CIDR 범위에서 오는 인바운드 HTTPS 트래픽을 허용하는 연결된 보안 그룹이 있습니다.

엔드포인트를 생성하면 요금이 부과됩니다. 자세한 내용은 [AWS PrivateLink 요금](#)을 참조하세요. 기타 AWS 서비스에 대한 액세스 권한이 포드에 필요한 경우에는 추가 엔드포인트를 생성해야 합니다. 엔드포인트의 전체 목록은 [AWS PrivateLink와 통합되는 AWS 서비스](#)를 참조하세요.

VPC 생성

다음과 같은 AWS CloudFormation 템플릿 중 하나를 사용하여 이전 요구 사항을 충족하는 VPC를 생성할 수 있습니다.

- [템플릿 1](#) - 이 템플릿에서는 Outpost의 프라이빗 서브넷 1개와 AWS 리전의 퍼블릭 서브넷 1개로 VPC를 생성합니다. 프라이빗 서브넷에는 AWS 리전의 퍼블릭 서브넷에 상주하는 NAT 게이트웨이를 통해 인터넷에 연결되는 경로가 있습니다. 이 템플릿은 송신 인터넷 액세스 권한이 있는 서브넷에 로컬 클러스터를 생성하는 데 사용할 수 있습니다.
- [템플릿 2](#) - 이 템플릿에서는 Outpost의 프라이빗 서브넷 1개와 수신 또는 송신 인터넷 액세스 권한이 없는 서브넷에 로컬 클러스터(프라이빗 서브넷이라고도 함)를 생성하는 데 필요한 최소 VPC 엔드포인트 집합이 있는 VPC를 생성합니다.

네트워크 연결 해제에 대비하여 AWS Outposts에 로컬 Amazon EKS 클러스터 준비

로컬 네트워크와 AWS 클라우드의 연결이 끊어진 경우 Outpost에서 로컬 Amazon EKS 클러스터를 계속 사용할 수 있습니다. 이 주제에서는 로컬 클러스터에서 네트워크 연결 해제에 대비하는 방법과 관련 고려 사항을 설명합니다.

- 로컬 클러스터를 사용하면 계획되지 않은 임시 네트워크 연결 해제 중에 안정성과 지속적인 작업을 수행할 수 있습니다. AWS Outposts는 데이터 센터에서 AWS 클라우드의 확장 역할을 하는 완전 연결형 제품으로 남아 있습니다. Outpost와 AWS 클라우드 간 네트워크 연결이 해제된 경우 연결 복원을 시도하는 것이 좋습니다. 지침은 AWS Outposts 사용 설명서의 [AWS Outposts 랙 네트워크 문제 해결 체크리스트](#)를 참조하세요. 로컬 클러스터 문제를 해결하는 방법에 대한 자세한 내용은 [the section called “클러스터 문제 해결”](#) 섹션을 참조하세요.
- Outposts는 Outpost의 연결 상태를 모니터링하는 데 사용할 수 있는 ConnectedStatus 지표를 내보냅니다. 자세한 내용을 알아보려면 AWS Outposts 사용 설명서에서 [Outposts 지표](#)를 참조하세요.
- 로컬 클러스터는 [Kubernetes용 AWS ID 및 액세스 관리 인증자를 사용하여 IAM을 기본 인증 메커니즘](#)으로 사용합니다. 네트워크 연결 해제 중에는 IAM을 사용할 수 없습니다. 따라서 로컬 클러스터에서는 네트워크 연결 해제 중 클러스터에 연결하는 데 사용할 수 있는 x.509 인증서를 사용하는 대체 인증 메커니즘을 지원합니다. 클러스터에 대한 x.509 인증서를 받고 사용하는 방법에 대한 내용은 [the section called “네트워크 연결 해제 중 로컬 클러스터에 인증”](#) 섹션을 참조하세요.
- 네트워크 연결 해제 중 Route 53에 액세스할 수 없는 경우 온프레미스 환경에서 로컬 DNS 서버를 사용하는 것이 좋습니다. Kubernetes 컨트롤 플레인 인스턴스는 고정 IP 주소를 사용합니다. 로컬 DNS 서버를 사용하는 대신 엔드포인트 호스트 이름과 IP 주소를 사용하여 클러스터에 연결하는 데

사용하는 호스트를 구성할 수 있습니다. 자세한 내용은 AWS Outposts 사용 설명서에서 [DNS](#)를 참조하세요.

- 네트워크 연결 해제 중 애플리케이션 트래픽이 증가할 것으로 예상되는 경우 클라우드에 연결될 때 클러스터에 여분의 컴퓨팅 용량을 프로비저닝할 수 있습니다. Amazon EC2 인스턴스는 AWS Outposts의 가격에 포함됩니다. 따라서 여분의 인스턴스를 실행해도 AWS 사용 비용에 영향을 미치지 않습니다.
- 네트워크 연결 해제 중 워크로드에 대한 생성, 업데이트 및 조정 작업을 활성화하려면 로컬 네트워크를 통해 애플리케이션의 컨테이너 이미지에 액세스할 수 있어야 하고 클러스터에 충분한 용량이 있어야 합니다. 로컬 클러스터는 컨테이너 레지스트리를 호스팅하지 않습니다. 포드가 이전에 해당 노드에서 실행된 경우 컨테이너 이미지가 노드에서 캐시됩니다. 일반적으로 클라우드의 Amazon ECR에서 애플리케이션의 컨테이너 이미지를 가져오는 경우 로컬 캐시 또는 레지스트리를 실행해 보세요. 네트워크 연결 해제 중 워크로드 리소스에 대한 생성, 업데이트 또는 조정 작업이 필요한 경우 로컬 캐시 또는 레지스트리가 도움이 됩니다.
- 로컬 클러스터는 Amazon EBS를 영구 볼륨의 기본 스토리지 클래스로 사용하고 Amazon EBS CSI 드라이버를 사용하여 Amazon EBS 영구 볼륨의 수명 주기를 관리합니다. 네트워크 연결 해제 중 Amazon EBS에서 지원하는 포드는 생성, 업데이트 또는 조정할 수 없습니다. 이러한 작업에는 클라우드의 Amazon EBS API에 대한 호출이 필요하기 때문입니다. 로컬 클러스터에 상태 저장 워크로드를 배포하고 네트워크 연결 해제 중 생성, 업데이트 또는 조정 작업이 필요한 경우 대체 스토리지 메커니즘을 사용하는 것이 좋습니다.
- AWS Outposts에서 관련 AWS 리전 내 API(예: Amazon EBS 또는 Amazon S3용 API)에 액세스할 수 없는 경우, Amazon EBS 스냅샷을 생성하거나 삭제할 수 없습니다.
- ALB(수신)를 AWS 인증 관리(ACM)와 통합하면 인증서가 푸시되어 AWS Outposts ALB Compute 인스턴스의 메모리에 저장됩니다. AWS 리전에서 연결이 해제되는 경우 현재 TLS 종료는 계속 작동합니다. 이 컨텍스트에서는 변형 작업(예: 새 수신 정의, 새 ACM 기반 인증서 API 작업, ALB 컴퓨팅 크기 조정 또는 인증서 교체)에 실패합니다. 자세한 내용은 AWS 사용 설명서의 [관리형 인증서 갱신 문제 해결](#)을 참조하세요.
- Amazon EKS 컨트롤 플레인 로그는 네트워크 연결 해제 중 Kubernetes 컨트롤 플레인 인스턴스에서 로컬로 캐시됩니다. 다시 연결되면 로그가 상위 AWS 리전의 CloudWatch Logs로 전송됩니다. [Prometheus](#), [Grafana](#) 또는 Amazon EKS 파트너 솔루션을 사용하여 Kubernetes API 서버의 지표 엔드포인트를 사용하거나 로그에 Fluent Bit를 사용하여 로컬로 클러스터를 모니터링할 수 있습니다.
- 애플리케이션 트래픽을 위해 Outposts에서 AWS Load Balancer Controller를 사용하는 경우 AWS Load Balancer Controller 뒤에 있는 기존 포드는 네트워크 연결 해제 중 트래픽을 계속 수신합니다. 네트워크 연결 해제 중 생성된 새 포드는 Outpost가 AWS 클라우드에 다시 연결될 때까지 트래픽을 수신하지 않습니다. 네트워크 연결 해제 중 조정 요구 사항을 수용할 수 있도록 AWS 클라우드에 연결되어 있는 동안 애플리케이션의 복제본 수를 설정하는 것이 좋습니다.

- Kubernetes용 Amazon VPC CNI 플러그인은 기본적으로 [보조 IP 모드](#)로 설정됩니다. WARM_ENI_TARGET=1로 구성되어 플러그인이 사용 가능한 IP 주소의 '전체 탄력적 네트워크 인터페이스'를 사용 가능한 상태로 유지할 수 있습니다. 연결 해제된 상태에서 조정 요구 사항에 따라 WARM_ENI_TARGET, WARM_IP_TARGET 및 MINIMUM_IP_TARGET 값을 변경하는 것을 고려하세요. 자세한 내용을 알아보려면 GitHub에서 플러그인의 [readme](#) 파일을 참조하세요. 각 인스턴스 유형별로 지원되는 최대 포트 수 목록은 GitHub의 [eni-max-pods.txt](#) 파일을 참조하세요.

네트워크 연결 해제 중 로컬 클러스터에 인증

네트워크 연결 해제 중에는 AWS ID 및 액세스 관리(IAM)를 사용할 수 없습니다. 연결이 해제된 동안에는 IAM 자격 증명을 사용하여 로컬 클러스터에 인증할 수 없습니다. 그러나 연결이 해제되면 x509 인증서를 사용하여 로컬 네트워크를 통해 클러스터에 연결할 수 있습니다. 연결 해제 중 사용할 클라이언트 X509 인증서를 다운로드하여 저장해야 합니다. 이 주제에서는 연결 해제된 상태일 때 클러스터에 인증하기 위해 인증서를 생성하고 사용하는 방법을 알아봅니다.

1. 인증서 서명 요청을 생성합니다.

a. 인증서 서명 요청을 생성합니다.

```
openssl req -new -newkey rsa:4096 -nodes -days 365 \
-keyout admin.key -out admin.csr -subj "/CN=admin"
```

b. Kubernetes에서 인증서 서명 요청을 생성합니다.

```
BASE64_CSR=$(cat admin.csr | base64 -w 0)
cat << EOF > admin-csr.yaml
apiVersion: certificates.k8s.io/v1
kind: CertificateSigningRequest
metadata:
  name: admin-csr
spec:
  signerName: kubernetes.io/kube-apiserver-client
  request: ${BASE64_CSR}
  usages:
    - client auth
EOF
```

2. kubectl을 사용하여 인증서 서명 요청을 생성합니다.

```
kubectl create -f admin-csr.yaml
```

3. 인증서 서명 요청의 상태를 확인합니다.

```
kubectl get csr admin-csr
```

예제 출력은 다음과 같습니다.

NAME	AGE	REQUESTOR	CONDITION
admin-csr	11m	kubernetes-admin	Pending

Kubernetes에서 인증서 서명 요청을 생성했습니다.

4. 인증서 서명 요청을 승인합니다.

```
kubectl certificate approve admin-csr
```

5. 인증서 서명 요청의 승인 상태를 다시 확인합니다.

```
kubectl get csr admin-csr
```

예제 출력은 다음과 같습니다.

NAME	AGE	REQUESTOR	CONDITION
admin-csr	11m	kubernetes-admin	Approved

6. 인증서를 검색하고 확인합니다.

a. 인증서를 검색합니다.

```
kubectl get csr admin-csr -o jsonpath='{.status.certificate}' | base64 --decode > admin.crt
```

b. 인증서를 확인합니다.

```
cat admin.crt
```

7. admin 사용자에게 대한 클러스터 역할 바인딩을 생성합니다.

```
kubectl create clusterrolebinding admin --clusterrole=cluster-admin \
  --user=admin --group=system:masters
```

8. 연결 해제된 상태에 대한 사용자 범위 kubeconfig를 생성합니다.

다운로드한 admin 인증서를 사용하여 kubeconfig 파일을 생성할 수 있습니다. 다음 명령에서 *my-cluster*와 *apiserver-endpoint*를 바꿉니다.

```
aws eks describe-cluster --name my-cluster \
  --query "cluster.certificateAuthority" \
  --output text | base64 --decode > ca.crt
```

```
kubect1 config --kubeconfig admin.kubeconfig set-cluster my-cluster \
  --certificate-authority=ca.crt --server apiserver-endpoint --embed-certs
```

```
kubect1 config --kubeconfig admin.kubeconfig set-credentials admin \
  --client-certificate=admin.crt --client-key=admin.key --embed-certs
```

```
kubect1 config --kubeconfig admin.kubeconfig set-context admin@my-cluster \
  --cluster my-cluster --user admin
```

```
kubect1 config --kubeconfig admin.kubeconfig use-context admin@my-cluster
```

9. kubeconfig 파일을 봅니다.

```
kubect1 get nodes --kubeconfig admin.kubeconfig
```

10. Outpost에서 서비스가 이미 프로덕션에 있는 경우 이 단계를 건너뛰니다. Amazon EKS가 Outpost에서 실행되는 유일한 서비스이고 Outpost가 현재 프로덕션에 없는 경우 네트워크 연결 해제를 시뮬레이션할 수 있습니다. 로컬 클러스터를 사용하여 프로덕션에 들어가기 전에 연결 해제를 시뮬레이션하여 연결 해제된 상태에서도 클러스터에 계속 액세스할 수 있는지 확인해야 합니다.

- Outpost를 AWS 리전에 연결하는 네트워킹 디바이스에 방화벽 규칙을 적용합니다. 그러면 Outpost의 서비스 링크 연결이 해제됩니다. 새 인스턴스를 생성할 수 없습니다. 현재 실행 중인 인스턴스는 AWS 리전 및 인터넷 연결이 끊어집니다.
- x509 인증서를 사용하여 연결이 해제된 동안 로컬 클러스터 연결을 테스트할 수 있습니다. kubeconfig를 이전 단계에서 생성한 admin.kubeconfig로 변경해야 합니다. *my-cluster*를 로컬 클러스터의 이름으로 바꿉니다.

```
kubect1 config use-context admin@my-cluster --kubeconfig admin.kubeconfig
```

연결 해제된 상태에서 로컬 클러스터에 문제가 있는 경우 지원 티켓을 여는 것이 좋습니다.

용량 고려 사항에 따라 AWS Outposts의 Amazon EKS 클러스터에 대한 인스턴스 유형 및 배치 그룹 선택

이 주제에서는 Kubernetes 컨트롤 플레인 인스턴스 유형 선택 및 Outpost의 로컬 Amazon EKS 클러스터에 대한 고가용성 요구 사항을 충족하기 위한 배치 그룹 사용(선택 사항)에 대한 지침을 제공합니다.

Outposts에서 로컬 클러스터의 Kubernetes 컨트롤 플레인에 사용할 인스턴스 유형(예: m5, c5 또는 r5)을 선택하기 전에 Outpost 구성에서 사용 가능한 인스턴스 유형을 확인합니다. 사용 가능한 인스턴스 유형 식별 후 워크로드에 필요한 노드 수에 따라 인스턴스 크기(예: large, xlarge 또는 2xlarge)를 선택합니다. 다음 표에는 인스턴스 크기 선택에 대한 권장 사항이 나와 있습니다.

Note

인스턴스 크기는 Outposts에 지정되어 있어야 합니다. 로컬 클러스터의 수명 동안 Outposts에서 사용할 수 있는 크기의 인스턴스 3개에 대한 용량이 충분한지 확인합니다. 사용 가능한 Amazon EC2 인스턴스 유형 목록은 [AWS Outposts 랙 기능](#)의 컴퓨팅 및 스토리지 섹션을 참조하세요.

노드 수	Kubernetes 컨트롤 플레인 인스턴스 크기
1~20	large
21~100	xlarge
101~250	2xlarge
251~500	4xlarge

Kubernetes 컨트롤 플레인용 스토리지에는 etcd의 필수 IOPS를 충족하기 위해 로컬 클러스터마다 246GB의 Amazon EBS 스토리지가 필요합니다. 로컬 클러스터가 생성될 때 Amazon EBS 볼륨이 자동으로 프로비저닝됩니다.

컨트롤 플레인 배치

OutpostConfig.ControlPlanePlacement.GroupName 속성으로 배치 그룹을 지정하지 않으면 Kubernetes 컨트롤 플레인용으로 프로비저닝된 Amazon EC2 인스턴스에는 Outpost에서 사용할 수 있는 기본 용량에 대한 특정 하드웨어 배치가 적용되지 않습니다.

배치 그룹을 사용하여 Outpost의 로컬 Amazon EKS 클러스터에 대한 고가용성 요구 사항을 충족할 수 있습니다. 클러스터 생성 중에 배치 그룹을 지정하면 Kubernetes 컨트롤 플레인 인스턴스의 배치에 영향을 줍니다. 인스턴스가 독립적인 기본 하드웨어(랙 또는 호스트) 전체에 분산되어 하드웨어 장애 발생 시 상관 관계가 있는 인스턴스 영향을 최소화합니다.

구성할 수 있는 분산 유형은 배포되어 있는 Outpost 랙의 수에 따라 다릅니다.

- 하나의 논리적 Outpost에 1개 또는 2개의 물리적 랙이 있는 배포 – Kubernetes 컨트롤 플레인 인스턴스에 대해 선택하는 인스턴스 유형으로 구성되는 호스트가 3개 이상 있어야 합니다. 호스트 수준 스프레드를 사용하는 스프레드 배치 그룹은 모든 Kubernetes 컨트롤 플레인 인스턴스가 Outpost 배포에서 사용할 수 있는 기본 랙 내의 개별 호스트에서 실행되도록 합니다.
- 하나의 논리적 Outpost에 3개 이상의 물리적 랙이 있는 배포 – Kubernetes 컨트롤 플레인 인스턴스에 대해 선택하는 인스턴스 유형으로 구성된 호스트가 3개 이상 있어야 합니다. 랙 수준 스프레드를 사용하는 스프레드 배치 그룹은 모든 Kubernetes 컨트롤 플레인 인스턴스가 Outpost 배포의 개별 랙에서 실행되도록 합니다. 이전 옵션에서 설명한 대로 호스트 수준 분산 배치 그룹을 사용할 수도 있습니다.

원하는 배치 그룹을 생성하는 것은 본인 책임입니다. CreateCluster API를 호출할 때 배치 그룹을 지정합니다. 배치 그룹과 생성 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 [배치 그룹](#)을 참조하세요.

- 배치 그룹을 지정할 때 로컬 Amazon EKS 클러스터를 생성하려면 Outpost에 사용 가능한 배정된 용량이 있어야 합니다. 용량은 호스트 유형을 사용하는지 아니면 랙 분산 유형을 사용하는지에 따라 달라집니다. 용량이 부족하면 클러스터가 Creating 상태로 유지됩니다. [DescribeCluster](#) API 응답의 상태 필드에서 Insufficient Capacity Error를 확인할 수 있습니다. 생성 프로세스를 진행하려면 용량을 확보해야 합니다.
- Amazon EKS 로컬 클러스터 플랫폼 및 버전 업데이트 중에는 클러스터의 Kubernetes 컨트롤 플레인 인스턴스가 롤링 업데이트 전략을 통해 새 인스턴스로 바뀝니다. 이 대체 프로세스 중에 각 컨트롤 플레인 인스턴스가 종료되어 해당 슬롯이 비워집니다. 업데이트한 새 인스턴스가 대신에 프로비저닝됩니다. 업데이트한 인스턴스는 릴리스된 슬롯에 배치될 수도 있습니다. 관련 없는 다른 인스턴스에서 슬롯을 사용하고 필요한 분산 토폴로지 요구 사항을 충족하는 용량이 더는 남아 있지 않으면 클러스터가 Updating 상태로 유지됩니다. [DescribeCluster](#) API 응답의 상태 필드에서 각 Insufficient Capacity Error를 확인할 수 있습니다. 업데이트 프로세스를 진행하고 이전의 고가용성 수준을 다시 설정할 수 있도록 용량을 확보해야 합니다.
- 각 AWS 리전에서 계정당 최대 500개의 배치 그룹을 생성할 수 있습니다. 자세한 내용은 Amazon EC2 사용 설명서의 [일반 규칙 및 제한 사항](#)을 참조하세요.

AWS Outposts의 로컬 Amazon EKS 클러스터 문제 해결

이 주제에서는 로컬 클러스터를 사용하는 동안 발생할 수 있는 몇 가지 일반적 오류와 문제를 해결하는 방법을 설명합니다. 로컬 클러스터는 클라우드의 Amazon EKS 클러스터와 유사하지만, Amazon EKS 서비스에서 관리하는 방식에는 몇 가지 차이점이 있습니다.

Important

AWS Support에서 명시적으로 지시하지 않는 한 Outpost에서 실행되는 관리형 EKS 로컬 클러스터 Kubernetes 컨트롤 플레인 인스턴스를 종료하지 마십시오. 이러한 인스턴스를 종료하면 여러 인스턴스가 동시에 종료되는 경우 로컬 클러스터가 손실되는 등 로컬 클러스터 서비스 가용성이 위협해질 수 있습니다. EKS 로컬 클러스터 Kubernetes 컨트롤 플레인 인스턴스는 EC2 인스턴스 콘솔에서 `eks-local:controlplane-name` 태그로 식별됩니다.

API 동작

로컬 클러스터는 Amazon EKS API를 통해 생성되지만, 비동기 방식으로 실행됩니다. 즉, Amazon EKS API에 대한 요청이 로컬 클러스터에 즉시 반환됩니다. 그러나 이러한 요청은 성공하거나, 입력 검증 오류로 인해 빠르게 실패하거나, 실패하고 설명이 포함된 검증 오류가 있을 수 있습니다. 이 동작은 Kubernetes API와 유사합니다.

로컬 클러스터는 FAILED 상태로 전환되지 않습니다. Amazon EKS에서는 클러스터 상태를 사용자가 요청한 원하는 상태와 지속적으로 조정하려고 시도합니다. 따라서 로컬 클러스터가 근본적인 문제가 해결될 때까지 장기간 CREATING 상태로 유지될 수도 있습니다.

클러스터 상태 필드 설명

로컬 클러스터 문제는 [describe-cluster](#) Amazon EKS AWS CLI 명령을 사용하여 검색할 수 있습니다. 로컬 클러스터 문제는 `describe-cluster` 명령 응답의 `cluster.health` 필드에 표시됩니다. 이 필드에 포함된 메시지에는 오류 코드, 설명이 포함된 메시지 및 관련 리소스 ID가 포함됩니다. 이 정보는 Amazon EKS API와 AWS CLI를 통해서만 제공됩니다. 다음 예제에서 *my-cluster*를 로컬 클러스터의 이름으로 바꿉니다.

```
aws eks describe-cluster --name my-cluster --query 'cluster.health'
```

예제 출력은 다음과 같습니다.

```
{
  "issues": [
```

```
{
  "code": "ConfigurationConflict",
  "message": "The instance type 'm5.large' is not supported in Outpost 'my-outpost-arn'.",
  "resourceIds": [
    "my-cluster-arn"
  ]
}
```

문제를 복구할 수 없는 경우 로컬 클러스터를 삭제하고 새 클러스터를 생성해야 할 수 있습니다. 예를 들어, Outpost에서 사용할 수 없는 인스턴스 유형으로 클러스터를 프로비저닝하려고 합니다. 다음 표에는 일반적인 상태 관련 오류가 포함되어 있습니다.

오류 시나리오	코드	메시지	ResourceIds
제공된 서브넷을 찾을 수 없습니다.	ResourceNotFound	The subnet ID <i>subnet-id</i> does not exist	제공된 모든 서브넷 ID
제공된 서브넷이 동일한 VPC에 속하지 않습니다.	ConfigurationConflict	Subnets specified must belong to the same VPC	제공된 모든 서브넷 ID
제공된 일부 서브넷이 지정된 Outpost에 속하지 않습니다.	ConfigurationConflict	Subnet <i>subnet-id</i> expected to be in <i>outpost-arn</i> , but is in <i>other-outpost-arn</i>	문제가 있는 서브넷 ID
제공된 일부 서브넷이 어느 Outpost에도 속하지 않습니다.	ConfigurationConflict	Subnet <i>subnet-id</i> is not part of any Outpost	문제가 있는 서브넷 ID
제공된 일부 서브넷에 컨트롤 플레인 인스턴스용 탄력적 네트워크	ResourceLimitExceeded	The specified subnet does not have enough	문제가 있는 서브넷 ID

오류 시나리오	코드	메시지	ResourceIds
인터페이스를 생성하기에 충분한 여유 주소가 없습니다.		free addresses to satisfy the request.	
지정된 컨트롤 플레인 인스턴스 유형이 Outpost에서 지원되지 않습니다.	ConfigurationConflict	The instance type <i>type</i> is not supported in Outpost <i>outpost-arn</i>	클러스터 ARN
컨트롤 플레인 Amazon EC2 인스턴스를 종료했거나 run-instance에 성공했지만, 관찰된 상태가 Terminated로 변경됩니다. 이는 Outpost가 다시 연결되고 Amazon EBS 내부 오류로 인해 Amazon EC2 내부 워크플로가 실패한 후 일정 기간 동안 발생할 수 있습니다.	InternalFailure	EC2 instance state "Terminated" is unexpected	클러스터 ARN
Outpost의 용량이 부족합니다. 이는 Outpost가 AWS 리전에서 연결 해제된 경우 클러스터를 생성 중일 때 발생할 수도 있습니다.	ResourceLimitExceeded	There is not enough capacity on the Outpost to launch or start the instance.	클러스터 ARN
계정에서 보안 그룹 할당량을 초과했습니다.	ResourceLimitExceeded	Amazon EC2 API에서 반환된 오류 메시지	대상 VPC ID

오류 시나리오	코드	메시지	ResourceIds
계정에서 탄력적 네트워크 인터페이스 할당량을 초과했습니다.	ResourceLimitExceeded	Amazon EC2 API에서 반환된 오류 메시지	대상 서브넷 ID
컨트롤 플레인 인스턴스는 AWS 시스템 관리자를 통해 연결할 수 없습니다. 해결 방법은 the section called “컨트롤 플레인 인스턴스는 AWS 시스템 관리자를 통해 연결할 수 없습니다.” 섹션을 참조하세요.	ClusterUnreachable	Amazon EKS control plane instances are not reachable through SSM. Please verify your SSM and network configuration, and reference the EKS on Outposts troubleshooting documentation.(Amazon EKS 컨트롤 플레인 인스턴스는 SSM을 통해 연결할 수 없습니다. SSM 및 네트워크 구성을 확인하고 Outposts의 EKS 문제 해결 문서를 참조하세요.)	Amazon EC2 인스턴스 ID
관리형 보안 그룹 또는 탄력적 네트워크 인터페이스에 대한 세부 정보를 가져오는 동안 오류가 발생했습니다.	Amazon EC2 클라이언트 오류 코드 기준	Amazon EC2 API에서 반환된 오류 메시지	모든 관리형 보안 그룹 ID

오류 시나리오	코드	메시지	ResourceIds
보안 그룹 수신 규칙을 승인하거나 취소하는 동안 오류가 발생했습니다. 이는 클러스터 및 컨트롤 플레인 보안 그룹 모두에 적용됩니다.	Amazon EC2 클라이언트 오류 코드 기준	Amazon EC2 API에서 반환된 오류 메시지	문제가 있는 보안 그룹 ID
컨트롤 플레인 인스턴스에 대한 탄력적 네트워크 인터페이스를 삭제하는 동안 오류가 발생했습니다.	Amazon EC2 클라이언트 오류 코드 기준	Amazon EC2 API에서 반환된 오류 메시지	문제가 있는 탄력적 네트워크 인터페이스 ID

다음 표에는 describe-cluster 응답의 상태 필드에 표시되는 다른 AWS 서비스의 오류가 나열되어 있습니다.

Amazon EC2 오류 코드	클러스터 상태 문제 코드	설명
AuthFailure	AccessDenied	이 오류는 여러 가지 이유로 발생할 수 있습니다. 가장 일반적인 이유는 서비스 연결 역할 정책의 범위를 좁히기 위해 서비스에서 사용하는 태그를 컨트롤 플레인에서 실수로 제거했기 때문입니다. 이렇게 되면 Amazon EKS에서 더는 이러한 AWS 리소스를 관리하고 모니터링할 수 없습니다.
UnauthorizedOperation	AccessDenied	이 오류는 여러 가지 이유로 발생할 수 있습니다. 가장 일반적인 이유는 서비스 연결 역할 정책의 범위를 좁히기 위해 서비스에서 사용하는 태그를 컨

Amazon EC2 오류 코드	클러스터 상태 문제 코드	설명
		트를 플레인에서 실수로 제거했기 때문입니다. 이렇게 되면 Amazon EKS에서 더는 이러한 AWS 리소스를 관리하고 모니터링할 수 없습니다.
InvalidSubnetID.NotFound	ResourceNotFound	이 오류는 보안 그룹의 수신 규칙에 대한 서브넷 ID를 찾을 수 없을 때 발생합니다.
InvalidPermission.NotFound	ResourceNotFound	이 오류는 보안 그룹의 수신 규칙에 대한 권한이 올바르지 않을 때 발생합니다.
InvalidGroup.NotFound	ResourceNotFound	이 오류는 보안 그룹의 수신 규칙 그룹을 찾을 수 없을 때 발생합니다.
InvalidNetworkInterfaceID.NotFound	ResourceNotFound	이 오류는 보안 그룹의 수신 규칙에 대한 네트워크 인터페이스 ID를 찾을 수 없을 때 발생합니다.
InsufficientFreeAddressesInSubnet	ResourceLimitExceeded	이 오류는 서브넷 리소스 할당량을 초과했을 때 발생합니다.
InsufficientCapacityOnOutpost	ResourceLimitExceeded	이 오류는 출력 용량 할당량을 초과했을 때 발생합니다.
NetworkInterfaceLimitExceeded	ResourceLimitExceeded	이 오류는 탄력적 네트워크 인터페이스 할당량을 초과했을 때 발생합니다.
SecurityGroupLimitExceeded	ResourceLimitExceeded	이 오류는 보안 그룹 할당량을 초과했을 때 발생합니다.

Amazon EC2 오류 코드	클러스터 상태 문제 코드	설명
VcpuLimitExceeded	ResourceLimitExceeded	이는 새 계정에서 Amazon EC2 인스턴스를 생성할 때 관찰됩니다. 오류는 다음과 비슷할 수 있습니다. "You have requested more vCPU capacity than your current vCPU limit of 32 allows for the instance bucket that the specified instance type belongs to. Please visit http://aws.amazon.com/contact-us/ec2-request to request an adjustment to this limit."
InvalidParameterValue	ConfigurationConflict	지정된 인스턴스 유형이 Outpost에서 지원되지 않는 경우 Amazon EC2가 이 오류 코드를 반환합니다.
기타 모든 오류	InternalFailure	없음

클러스터를 생성하거나 수정할 수 없음

클라우드에서 호스팅되는 Amazon EKS 클러스터와 다른 권한과 정책이 로컬 클러스터에 필요합니다. 클러스터가 생성되지 않고 InvalidPermissions 오류가 발생하면 사용 중인 클러스터 역할에 [AmazonEKSLocalOutpostClusterPolicy](#) 관리형 정책이 연결되어 있는지 다시 확인합니다. 다른 모든 API 직접 호출에는 클라우드의 Amazon EKS 클러스터와 동일한 권한 집합이 필요합니다.

클러스터가 **CREATING** 상태에서 멈춤

로컬 클러스터 생성에 걸리는 시간은 여러 가지 요인에 따라 다릅니다. 네트워크 구성, Outpost 구성 및 클러스터의 구성이 이러한 요인에 포함됩니다. 일반적으로 15~20분 이내에 로컬 클러스터가 생성되고

ACTIVE 상태로 변경됩니다. 로컬 클러스터가 CREATING 상태로 유지되는 경우 `cluster.health` 출력 필드에서 원인에 대한 정보의 `describe-cluster`를 직접 호출할 수 있습니다.

가장 일반적인 문제는 다음과 같습니다.

- 클러스터에서 Systems Manager가 있는 AWS 리전의 컨트롤 플레인 인스턴스에 연결할 수 없습니다. 리전 내 Bastion Host에서 `aws ssm start-session --target instance-id` 를 직접 호출하여 이를 확인할 수 있습니다. 명령이 작동하지 않으면 Systems Manager가 컨트롤 플레인 인스턴스에서 실행 중인지 확인합니다. 클러스터를 삭제한 다음에 다시 생성하여 해결하는 방법도 있습니다.
- EBS 볼륨에 대한 KMS 키 권한으로 인해 컨트롤 플레인 인스턴스가 생성되지 않습니다. 암호화된 EBS 볼륨에 사용자 관리형 KMS 키를 사용하면 키에 액세스할 수 없는 경우 컨트롤 플레인 인스턴스가 종료됩니다. 인스턴스가 종료되면 AWS 관리형 KMS 키로 전환하거나 사용자 관리형 키 정책이 클러스터 역할에 필요한 권한을 부여하는지 확인하세요.
- Systems Manager 컨트롤 플레인 인스턴스에 인터넷 액세스 권한이 없을 수도 있습니다. 클러스터를 생성할 때 제공한 서브넷에 NAT 게이트웨이와 VPC(인터넷 게이트웨이 포함)가 있는지 확인합니다. VPC 연결성 분석기를 사용하여 컨트롤 플레인 인스턴스가 인터넷 게이트웨이에 연결할 수 있는지 확인합니다. 자세한 내용을 알아보려면 [Getting started with VPC Reachability Analyzer](#)(VPC Reachability Analyzer 시작하기)를 참조하세요.
- 제공한 역할 ARN에 정책이 없습니다. [AWS 관리형 정책: AmazonEKSLocalOutpostClusterPolicy](#)가 역할에서 제거되었는지 확인합니다. AWS CloudFormation 스택이 잘못 구성된 경우에도 이 문제가 발생할 수 있습니다.
- 제공된 모든 서브넷이 동일한 Outpost와 연결되어야 하며 서로 연결되어야 합니다. 클러스터가 생성될 때 여러 서브넷이 지정되면 Amazon EKS에서는 컨트롤 플레인 인스턴스를 여러 서브넷에 분산하려고 시도합니다.
- Amazon EKS 관리형 보안 그룹은 탄력적 네트워크 인터페이스에 적용됩니다. 그러나 NACL 방화벽 규칙과 같은 기타 구성 요소가 탄력적 네트워크 인터페이스의 규칙과 충돌할 수도 있습니다.

VPC 및 서브넷 DNS 구성이 잘못 구성되었거나 누락됨

[AWS Outposts에서 Amazon EKS 클러스터를 위한 VPC 및 서브넷 생성](#) 검토

클러스터가 **UPDATING** 상태에서 멈춤

Amazon EKS에서 기존의 모든 로컬 클러스터를 해당하는 Kubernetes 마이너 버전에 대한 최신 플랫폼 버전으로 자동으로 업데이트합니다. 플랫폼 버전에 대한 자세한 내용은 [the section called “EKS 플랫폼 버전”](#) 섹션을 참조하세요.

자동 플랫폼 버전 롤아웃 중에 클러스터 상태가 UPDATING으로 변경됩니다. 업데이트 프로세스는 모든 Kubernetes 컨트롤 플레인 인스턴스를 해당 Kubernetes 마이너 버전에 대해 릴리스된 최신 보안 경로 및 버그 수정이 포함된 새 인스턴스로 대체하는 것으로 구성됩니다. 일반적으로 로컬 클러스터 플랫폼 업데이트 프로세스는 30분 내에 완료되며 클러스터는 ACTIVE 상태로 다시 변경됩니다. 로컬 클러스터가 장기가 UPDATING 상태로 유지되는 경우 `describe-cluster`를 직접적으로 직접 호출하여 `cluster.health` 출력 필드에서 원인에 대한 정보를 확인할 수 있습니다.

Amazon EKS는 로컬 클러스터 가용성을 유지하고 서비스 중단을 방지하기 위해 Kubernetes 컨트롤 플레인 인스턴스 3개 중 최소 2개가 정상이고 작동하는 클러스터 노드인지 확인합니다. 로컬 클러스터가 UPDATING 상태에서 멈춘 경우 이는 대개 프로세스가 계속 진행되더라도 두 인스턴스의 최소 가용성을 보장할 수 없는 인프라 구성 문제가 있기 때문입니다. 따라서 로컬 클러스터 서비스 중단을 방지하기 위해 업데이트 프로세스 진행이 중지됩니다.

UPDATING 상태에 멈춘 로컬 클러스터의 문제를 해결하고 근본 원인을 해결하여 업데이트 프로세스를 완료하고 3개의 Kubernetes 컨트롤 플레인 인스턴스의 고가용성을 통해 로컬 클러스터를 다시 ACTIVE로 복원하는 것이 중요합니다.

AWS Support에서 명시적으로 지시하지 않는 한 Outposts에서 관리형 EKS 로컬 클러스터 Kubernetes 인스턴스를 종료하지 마세요. 이는 특히 UPDATING 상태에 멈춰 있는 로컬 클러스터의 경우 특히 중요합니다. 다른 컨트롤 플레인 노드가 완전 정상이 아닐 가능성이 높고 잘못된 인스턴스를 종료하면 서비스가 중단되고 로컬 클러스터 데이터가 손실될 위험이 있기 때문입니다.

가장 일반적인 문제는 다음과 같습니다.

- 로컬 클러스터가 처음 생성된 이후 네트워킹 구성이 변경되어 하나 이상의 컨트롤 플레인 인스턴스가 System Manager에 연결할 수 없습니다. 리전 내 Bastion Host에서 `aws ssm start-session --target instance-id`를 직접 호출하여 이를 확인할 수 있습니다. 명령이 작동하지 않으면 Systems Manager가 컨트롤 플레인 인스턴스에서 실행 중인지 확인합니다.
- EBS 볼륨에 대한 KMS 키 권한으로 인해 새 컨트롤 플레인 인스턴스를 생성하지 못했습니다. 암호화된 EBS 볼륨에 사용자 관리형 KMS 키를 사용하면 키에 액세스할 수 없는 경우 컨트롤 플레인 인스턴스가 종료됩니다. 인스턴스가 종료되면 AWS 관리형 KMS 키로 전환하거나 사용자 관리형 키 정책이 클러스터 역할에 필요한 권한을 부여하는지 확인합니다.
- Systems Manager 컨트롤 플레인 인스턴스가 인터넷 액세스 권한을 잃었을 수 있습니다. 클러스터를 생성할 때 제공된 서브넷에 NAT 게이트웨이와 VPC(인터넷 게이트웨이 포함)가 있는지 확인합니다. VPC 연결성 분석기를 사용하여 컨트롤 플레인 인스턴스가 인터넷 게이트웨이에 연결할 수 있는지 확인합니다. 자세한 내용을 알아보려면 [Getting started with VPC Reachability Analyzer](#)(VPC Reachability Analyzer 시작하기)를 참조하세요. 프라이빗 네트워크에 아웃바운드 인터넷 연결이 없

는 경우 클러스터의 리전 서브넷에 필요한 모든 VPC 엔드포인트와 게이트웨이 엔드포인트가 여전히 존재하는지 확인합니다([the section called “AWS 서비스에 대한 서브넷 액세스”](#) 참조).

- 제공한 역할 ARN에 정책이 없습니다. [AWS 관리형 정책: AmazonEKSLocalOutpostClusterPolicy](#)가 역할에서 제거되지 않았는지 확인합니다.
- 새 Kubernetes 컨트롤 플레인 인스턴스 중 하나에서 예기치 않은 부트스트래핑 실패가 발생했을 수 있습니다. 이 예외적인 경우 문제 해결 및 로그 수집에 대한 추가 지침은 [AWS Support 센터](#)에 티켓을 제출하세요.

클러스터에 노드를 조인할 수 없음

- AMI 문제:
 - 지원되지 않는 AMI를 사용하고 있습니다. 최적화된 [Amazon Linux AMIs Amazon EKS 최적화 Amazon Linux가 있는 노드 생성](#)에는 [v20220620](#) 이상을 사용해야 합니다.
 - AWS CloudFormation 템플릿을 사용하여 노드를 생성한 경우 지원되지 않는 AMI를 사용하고 있지 않은지 확인합니다.
- AWS IAM Authenticator ConfigMap 누락 – 누락된 경우 생성해야 합니다. 자세한 정보는 [the section called “클러스터에 aws-authConfigMap 적용”](#)을 참조하세요.
- 잘못된 보안 그룹 사용 - 워커 노드의 보안 그룹에 `eks-cluster-sg-cluster-name-uniqueid`를 사용해야 합니다. 선택한 보안 그룹은 스택이 사용될 때마다 새 보안 그룹을 허용하도록 AWS CloudFormation에 의해 변경됩니다.
- 예상치 못한 프라이빗 링크 VPC 단계 수행 – 잘못된 CA 데이터(`--b64-cluster-ca`) 또는 API 엔드포인트(`--apiserver-endpoint`)가 전달됩니다.

로그 수집

Outpost가 연결된 AWS 리전에서 연결 해제되면 Kubernetes 클러스터는 정상적으로 계속 작동할 수 있습니다. 그러나 클러스터가 제대로 작동하지 않는 경우 [네트워크 연결 해제를 위해 AWS Outposts에서 로컬 Amazon EKS 클러스터 준비](#)의 문제 해결 단계를 따릅니다. 다른 문제가 발생하면 AWS Support에 문의하세요. AWS 지원은 로그 수집 도구를 다운로드하고 실행하는 방법을 안내할 수 있습니다. 그렇게 하면 Kubernetes 클러스터 컨트롤 플레인 인스턴스에서 로그를 수집하여 추가 조사를 위해 AWS Support 지원에 보낼 수 있습니다.

컨트롤 플레인 인스턴스는 AWS 시스템 관리자를 통해 연결할 수 없습니다.

AWS 시스템 관리자(Systems Manager)를 통해 Amazon EKS 컨트롤 플레인 인스턴스에 연결할 수 없는 경우, Amazon EKS는 클러스터에 대해 다음 오류를 표시합니다.

Amazon EKS control plane instances are not reachable through SSM. Please verify your SSM and network configuration, and reference the EKS on Outposts troubleshooting documentation.

이 문제를 해결하려면 VPC 및 서브넷이 [AWS Outposts에서 Amazon EKS 클러스터를 위한 VPC 및 서브넷 생성](#)의 요구 사항을 충족하는지 확인하고 AWS 시스템 관리자 사용 설명서의 [세션 관리자 설정](#)의 단계를 완료했는지 확인하세요.

AWS Outposts에 Amazon Linux 노드 생성

이 주제에서는 Amazon EKS 클러스터에 등록하는 Outpost에서 Amazon Linux 노드의 Auto Scaling을 시작하는 방법을 설명합니다. 클러스터는 AWS 클라우드 또는 Outpost에 있을 수 있습니다.

- 기존 Outpost. 자세한 내용은 [AWS Outposts이란 무엇입니까?](#) 섹션을 참조하세요.
- 기존 Amazon EKS 클러스터. AWS 클라우드에 클러스터를 배포하려면 [the section called “클러스터 생성”](#) 부분을 참조하세요. Outpost에 클러스터를 배포하려면 [the section called “로컬 클러스터 실행”](#) 부분을 참조하세요.
- AWS 클라우드의 클러스터에서 노드를 생성 중이며 활성화된 AWS Outposts, AWS Wavelength 또는 AWS 로컬 영역이 있는 AWS 리전에 서브넷이 있다고 가정합니다. 그러면 클러스터를 생성할 때 해당 서브넷이 전달되지 않았어야 합니다. Outpost의 클러스터에 노드를 생성하는 경우 클러스터 생성 시 Outpost 서브넷을 전달해야 합니다.
- (AWS 클라우드의 클러스터에 권장됨) 필요한 IAM 정책이 연결된 자체 IAM 역할로 구성된 Kubernetes용 Amazon VPC CNI 플러그인 추가 기능입니다. 자세한 내용은 [the section called “IRSA에 대한 구성”](#) 섹션을 참조하세요. 로컬 클러스터는 서비스 계정에 대한 IAM 역할을 지원하지 않습니다.

eksctl 또는 AWS Management Console(AWS CloudFormation 템플릿 사용)을 사용하여 자체 관리형 Amazon Linux 노드 그룹을 생성할 수 있습니다. Terraform도 사용할 수 있습니다.

이 페이지에 설명된 다음 도구를 사용하여 로컬 클러스터의 자체 관리형 노드 그룹을 생성할 수 있습니다.

- [the section called “eksctl”](#)
- [the section called “AWS Management Console”](#)

⚠ Important

- 자체 관리형 노드 그룹은 사용자 계정에 Amazon EC2 인스턴스를 포함합니다. 이러한 인스턴스는 사용자 또는 Amazon EKS가 컨트롤 플레인 버전을 업데이트할 때 자동으로 업그레이드되지 않습니다. 자체 관리형 노드 그룹에는 콘솔에 업데이트가 필요하다는 표시가 없습니다. 클러스터의 개요 탭에 있는 노드 목록에서 노드를 선택하여 노드에 설치된 kubelet 버전을 확인하면 업데이트가 필요한 노드를 확인할 수 있습니다. 노드를 수동으로 업데이트해야 합니다. 자세한 내용은 [the section called “업데이트 방법”](#) 섹션을 참조하세요.
- 자체 관리형 노드에서 kubelet이 사용하는 인증서는 1년 만료로 발급됩니다. 기본적으로 인증서 교체는 비활성화되어 있습니다(<https://kubernetes.io/docs/reference/config-api/kubelet-config.v1beta1/#kubelet-config-k8s-io-v1beta1-KubeletConfiguration> 참조). 따라서 1년 이상 실행 중인 자체 관리형 노드가 있는 경우 이제 Kubernetes API에 인증할 수 없게 됩니다.
- 모범 사례에 따라 고객은 자체 관리형 노드 그룹을 정기적으로 업데이트하여 최신 Amazon EKS 최적화 AMI에서 CVE 및 보안 패치를 받는 것이 좋습니다. 자체 관리형 노드 그룹에서 사용되는 AMI를 업데이트하면 노드가 다시 생성되고 만료된 kubelet 인증서로 인해 문제가 발생하지 않습니다.
- 또는 자체 관리형 노드 그룹을 생성할 때 클라이언트 인증서 교체(<https://kubernetes.io/docs/tasks/tls/certificate-rotation/> 참조)를 활성화하여 현재 인증서의 만료가 다가오면 kubelet 인증서가 갱신되도록 할 수도 있습니다.

eksctl

eksctl을 사용하여 자체 관리형 Linux 노드 시작

1. 장치에 설치된 eksctl 명령줄 도구의 버전 0.210.0 이상 또는 AWS CloudShell이 필요합니다. eksctl을 설치 또는 업그레이드하려면 eksctl 설명서에서 [설치](#)를 참조하세요.
2. 클러스터가 AWS 클라우드에 있고 AmazonEKS_CNI_Policy 관리형 IAM 정책이 [Amazon EKS 노드 IAM 역할](#)에 연결된 경우 대신 Kubernetes aws-node 서비스 계정에 연결하는 IAM 역할에 할당하는 것이 좋습니다. 자세한 내용은 [the section called “IRSA에 대한 구성”](#) 섹션을 참조하세요. 클러스터가 Outpost에 있는 경우 정책이 노드 역할에 연결되어야 합니다.
3. 다음 명령은 기존 클러스터의 노드 그룹을 생성합니다. eksctl을 사용하여 클러스터가 생성되어야 합니다. **al-nodes**를 노드 그룹의 이름으로 바꿉니다. 노드 그룹 이름은 63자를 초과할 수 없습니다. 문자나 숫자로 시작하되, 나머지 문자의 경우 하이픈과 밑줄을 포함할 수 있습니다. **my-cluster**를 해당 클러스터의 이름으로 바꿉니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는

AWS 리전과 AWS 계정 내에서 고유해야 합니다. 클러스터가 Outpost에 있는 경우 *id*를 Outpost 서브넷의 ID로 바꿉니다. 클러스터가 AWS 클라우드에 있는 경우 *id*를 클러스터를 생성할 때 지정하지 않은 서브넷의 ID로 바꿉니다. *instance-type*을 Outpost에서 지원하는 인스턴스 유형으로 바꿉니다. 나머지 *## #*을 자신의 값으로 바꿉니다. 노드는 기본적으로 제어 영역과 동일한 Kubernetes 버전으로 작성됩니다.

*instance-type*을 Outpost에서 사용 가능한 인스턴스 유형으로 바꿉니다.

*my-key*를 Amazon EC2 키 페어 또는 퍼블릭 키 이름으로 바꿉니다. 이 키는 노드를 시작한 후 SSH로 연결하는 데 사용됩니다. Amazon EC2 키 페어가 아직 없는 경우 AWS Management Console에서 새로 생성할 수 있습니다. 자세한 내용을 알아보려면 Amazon EC2 key pairs의 [Amazon EC2 키 페어](#)를 참조하세요.

다음 명령을 사용하여 노드 그룹을 생성합니다.

```
eksctl create nodegroup --cluster my-cluster --name al-nodes --node-type instance-type \
    --nodes 3 --nodes-min 1 --nodes-max 4 --managed=false --node-volume-type gp2 --
    subnet-ids subnet-id
```

AWS 클라우드에 클러스터가 배포된 경우:

- 배포하는 노드 그룹에서 인스턴스의 블록과 다른 CIDR 블록의 포드에 IPv4 주소를 할당할 수 있습니다. 자세한 내용은 [the section called “사용자 지정 네트워킹”](#) 섹션을 참조하세요.
- 배포하는 노드 그룹에는 아웃바운드 인터넷 액세스 권한이 필요하지 않습니다. 자세한 내용은 [the section called “프라이빗 클러스터”](#) 섹션을 참조하세요.

사용 가능한 모든 옵션과 기본값의 전체 목록은 eksctl 설명서의 [AWS Outposts 지원](#)을 참조하세요.

- 노드가 클러스터에 참여하지 못하면 [Amazon EKS 클러스터 및 노드 문제 해결](#)의 [the section called “노드가 클러스터 조인에 실패”](#)와 [AWS 전초 기지에서 로컬 Amazon EKS 클러스터 문제 해결](#)의 [the section called “클러스터에 노드를 조인할 수 없음”](#)를 참조하세요.
- 예제 출력은 다음과 같습니다. 노드가 생성되는 동안 여러 줄이 출력됩니다. 출력의 마지막 줄 중 하나는 다음 예제 줄과 유사합니다.

```
[#] created 1 nodegroup(s) in cluster "my-cluster"
```

4. (선택 사항) [샘플 애플리케이션](#)을 배포하여 클러스터 및 Linux 노드를 테스트합니다.

AWS Management Console

1단계: AWS Management Console을 사용하여 자체 관리형 Linux 노드 시작

1. AWS CloudFormation 템플릿의 최신 버전 다운로드

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2022-12-23/amazon-eks-nodegroup.yaml
```

2. [AWS CloudFormation 콘솔](#)을 엽니다.

3. 스택 생성을 선택한 다음 새 리소스 사용(표준)을 선택합니다.

4. 템플릿 지정에서 템플릿 파일 업로드를 선택한 다음 파일 선택을 선택합니다. 이전 단계에서 다운로드한 amazon-eks-nodegroup.yaml 파일을 선택한 후 다음을 선택합니다.

5. 스택 세부 정보 지정 페이지에서 이에 따라 다음 파라미터를 입력한 후 다음을 선택합니다.

- 스택 이름: AWS CloudFormation 스택에 대한 스택 이름을 선택합니다. 예를 들어, **al-nodes**라고 할 수 있습니다. 이름에는 영숫자(대소문자 구분)와 하이픈만 사용할 수 있습니다. 영숫자로 시작해야 하며 100자 이하여야 합니다. 이름은 클러스터를 생성하는 AWS 리전과 AWS 계정 내에서 고유해야 합니다.
- ClusterName: 클러스터의 이름을 입력합니다. 이 이름이 클러스터 이름과 일치하지 않으면 노드가 클러스터에 조인할 수 없습니다.
- ClusterControlPlaneSecurityGroup: [VPC](#)를 생성할 때 생성한 AWS CloudFormation 출력에서 SecurityGroups값을 선택합니다.

다음 단계에서는 해당 그룹을 검색하는 한 가지 작업을 보여줍니다.

i. [Amazon EKS 콘솔](#)을 엽니다.

ii. 클러스터의 이름을 선택합니다.

iii. 네트워킹 탭을 선택합니다.

iv. ClusterControlPlaneSecurityGroup 드롭다운 목록에서 선택할 때 추가 보안 그룹 값을 참조로 사용하세요.

- NodeGroupName: 노드 그룹의 이름을 입력합니다. 이 이름은 나중에 노드에 대해 생성된 Auto Scaling 노드 그룹을 식별하는 데 사용할 수 있습니다.
- NodeAutoScalingGroupMinSize: Auto Scaling 그룹이 축소할 수 있는 노드의 최소 노드 수를 입력합니다.
- NodeAutoScalingGroupDesiredCapacity: 스택을 생성할 때 조정할 원하는 노드 수를 입력합니다.

- `NodeAutoScalingGroupMaxSize`: Auto Scaling 그룹이 확장할 수 있는 노드의 최대 노드 수를 입력합니다.
- `NodeInstanceType`: 노드에 대한 인스턴스 유형을 선택합니다. 클러스터가 AWS 클라우드에서 실행 중인 경우 자세한 내용을 알아보려면 [the section called “Amazon EC2 인스턴스 유형”](#) 부분을 참조하세요. 클러스터가 Outpost에서 실행 중인 경우 Outpost에서 사용할 수 있는 인스턴스 유형만 선택할 수 있습니다.
- `NodeImageIdSSMParam`: 가변 Kubernetes 버전용 최신 Amazon EKS 최적화 AMI의 Amazon EC2 Systems Manager 파라미터로 미리 채워집니다. Amazon EKS에서 지원되는 다른 Kubernetes 마이너 버전을 사용하려면 **1.XX**를 다른 [지원되는 버전](#)으로 바꿉니다. 클러스터와 동일한 Kubernetes 버전을 지정하는 것이 좋습니다.

Amazon EKS 최적화 가속 AMI를 사용하려면 **amazon-linux-2**를 `amazon-linux-2-gpu`로 바꿉니다. Amazon EKS 최적화 Arm AMI를 사용하려면 **amazon-linux-2**를 `amazon-linux-2-arm64`로 바꿉니다.

Note

Amazon EKS 노드 AMI는 Amazon Linux를 기반으로 합니다. 원하는 버전에 대한 탭을 선택하여 [Amazon Linux 보안 센터](#)에서 Amazon Linux의 보안 또는 개인정보 보호 이벤트를 추적할 수 있습니다. 해당 RSS 피드를 구독할 수도 있습니다. 보안 및 프라이버시 이벤트에는 문제의 개요, 영향을 받는 패키지 및 인스턴스를 업데이트하여 문제를 해결하는 방법이 포함됩니다.

- `NodeImageId`: (선택사항) Amazon EKS 최적화 AMI 대신 사용자 정의 AMI를 사용 중인 경우, 해당 AWS 리전에 노드 AMI ID를 입력합니다. 여기에서 값을 지정하면 `NodeImageIdSSMParam` 필드에 있는 모든 값이 재정의됩니다.
- `NodeVolumeSize`: 노드에 대해 루트 볼륨 크기를 GiB 단위로 지정합니다.
- `NodeVolumeType`: 노드의 루트 볼륨 유형을 지정합니다.
- `KeyName`: 시작 이후 SSH를 사용하여 노드에 연결하는 데 사용할 수 있는 Amazon EC2 SSH 키 페어 이름을 입력합니다. Amazon EC2 키 페어가 아직 없는 경우 AWS Management Console에서 새로 생성할 수 있습니다. 자세한 내용을 알아보려면 Amazon EC2 key pairs의 [Amazon EC2 키 페어](#)를 참조하세요.

Note

여기에 키 페어를 입력하지 않으면 AWS CloudFormation 스택이 생성되지 않습니다.

- **BootstrapArguments:** 노드에 전달할 수 있는 몇 가지 선택적 인수가 있습니다. 자세한 내용은 GitHub의 [부트스트랩 스크립트 사용 정보](#)를 참조하십시오. AWS Outposts에서 Kubernetes 컨트롤 플레인 인스턴스가 실행되는 경우 AWS Outposts의 Amazon EKS 로컬 클러스터에 노드를 추가하고 있으며 클러스터(프라이빗 클러스터라고도 함)에 수신 및 송신 인터넷 연결이 없으면 다음과 같은 부트스트랩 인수를 한 줄로 제공해야 합니다.

```
--b64-cluster-ca ${CLUSTER_CA} --apiserver-endpoint https://${APISERVER_ENDPOINT}
--enable-local-outpost true --cluster-id ${CLUSTER_ID}
```

Amazon EKS 로컬 클러스터의 CLUSTER_CA, APISERVER_ENDPOINT, CLUSTER_ID 값을 검색하려면 다음 AWS CLI 명령을 실행합니다. cluster-name을 클러스터 이름으로 바꾸고 리전(예: us-east-1)을 클러스터의 AWS 리전으로 바꿉니다.

```
echo "CLUSTER_CA=$(aws eks describe-cluster --name cluster-name --region region --
query cluster.certificateAuthority.data --output text)"

echo "APISERVER_ENDPOINT=$(aws eks describe-cluster --name cluster-name --region
region --query cluster.endpoint --output text)"

echo "CLUSTER_ID=$(aws eks describe-cluster --name cluster-name --region region --
query cluster.id --output text)"
```

- **DisableIMDSv1:** 기본적으로 각 노드는 인스턴스 메타데이터 서비스 버전 1(IMDSv1) 및 IMDSv2를 지원합니다. IMDSv1을 사용 중지할 수 있습니다. 향후 노드 그룹의 노드와 포드가 IMDSv1을 사용하지 못하게 하려면 DisableIMDSv1을 true로 설정합니다. IMDS에 대한 자세한 내용은 [인스턴스 메타데이터 서비스 구성](#)을 참조하세요. 노드의 IMDS 액세스 제한에 대한 자세한 내용은 [작업자 노드에 할당된 인스턴스 프로필에 대한 액세스 제한](#)을 참조하세요.
 - **VpcId:** 생성한 [VPC](#)에 대한 ID를 입력합니다. VPC를 선택하기 전에 [VPC 요구 사항과 고려 사항](#)을 검토하세요.
 - **서브넷:** 클러스터가 Outpost에 있는 경우 VPC에서 프라이빗 서브넷을 하나 이상 선택합니다. 서브넷을 선택하기 전에 [을 검토하세요](#). 클러스터의 Networking에서 각 서브넷 링크를 열면 어떤 서브넷이 프라이빗인지 확인할 수 있습니다.
6. 스택 옵션 구성 페이지에서 원하는 선택 항목을 선택한 후 다음을 선택합니다.
 7. AWS CloudFormation에서 IAM 리소스를 생성할 수 있음을 인정합니다의 왼쪽에 있는 확인란을 선택한 다음 스택 생성을 선택합니다.
 8. 스택이 생성된 후 콘솔에서 이를 선택하고 출력을 선택합니다.

9. 생성된 노드 그룹에 대해 NodeInstanceRole을 기록합니다. Amazon EKS 노드를 구성할 때 필요합니다.

2단계: 노드가 클러스터에 조인하도록 하려면

1. aws-auth ConfigMap이 이미 있는지 확인합니다.

```
kubectl describe configmap -n kube-system aws-auth
```

2. aws-auth ConfigMap이 표시되면 필요에 따라 이를 업데이트합니다.

- a. 편집을 위해 ConfigMap을 엽니다.

```
kubectl edit -n kube-system configmap/aws-auth
```

- b. 필요에 따라 새 mapRoles 항목을 추가합니다. rolearn 값을 이전 절차에서 기록한 NodeInstanceRole 값으로 설정합니다.

```
[...]
data:
  mapRoles: |
    - rolearn: <ARN of instance role (not instance profile)>
      username: system:node:{{EC2PrivateDNSName}}
      groups:
        - system:bootstrappers
        - system:nodes
[...]
```

- c. 파일을 저장하고 텍스트 편집기를 종료합니다.

3. "Error from server (NotFound): configmaps "aws-auth" not found라는 오류 메시지가 표시되면 스텝 ConfigMap을 적용합니다.

- a. 구성 맵을 다운로드합니다.

```
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/cloudformation/2020-10-29/
aws-auth-cm.yaml
```

- b. aws-auth-cm.yaml 파일에서 rolearn을 이전 절차에서 기록한 NodeInstanceRole 값으로 설정합니다. 이 작업은 텍스트 편집기를 사용하거나 *my-node-instance-role*을 대체하고 다음 명령을 실행하여 수행할 수 있습니다.

```
sed -i.bak -e 's|<ARN of instance role (not instance profile)>|my-node-instance-role|' aws-auth-cm.yaml
```

- c. 구성을 적용합니다. 이 명령을 완료하는 데 몇 분이 걸릴 수 있습니다.

```
kubectl apply -f aws-auth-cm.yaml
```

4. 노드의 상태를 확인하고 Ready 상태가 될 때까지 대기합니다.

```
kubectl get nodes --watch
```

Ctrl+C를 입력하여 셸 프롬프트로 돌아갑니다.

Note

권한 부여 또는 리소스 유형 오류가 표시되는 경우 문제 해결 주제의 [the section called “권한이 없거나 액세스가 거부됨\(kubectl\)”](#) 부분을 참조하세요.

노드가 클러스터에 참여하지 못하면 [Amazon EKS 클러스터 및 노드 문제 해결](#)의 [the section called “노드가 클러스터 조인에 실패”](#)와 [AWS 전초 기지에서 로컬 Amazon EKS 클러스터 문제 해결](#)의 [the section called “클러스터에 노드를 조인할 수 없음”](#)를 참조하세요.

5. Amazon EBS CSI 드라이버를 설치합니다. 자세한 내용을 알아보려면 GitHub에서 [설치](#)를 참조하세요. 드라이버 권한 설정 섹션에서 IAM 인스턴스 프로파일 사용 옵션에 대한 지침을 따릅니다. gp2 스토리지 클래스를 사용해야 합니다. gp3 스토리지 클래스는 지원되지 않습니다.

클러스터에 gp2 스토리지 클래스를 생성하려면 다음 단계를 수행합니다.

- a. 다음 명령을 실행해 gp2-storage-class.yaml 파일을 생성합니다.

```
cat >gp2-storage-class.yaml <<EOF
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  annotations:
    storageclass.kubernetes.io/is-default-class: "true"
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
parameters:
```

```
type: gp2
encrypted: "true"
allowVolumeExpansion: true
EOF
```

b. 매니페스트를 클러스터에 적용합니다.

```
kubectl apply -f gp2-storage-class.yaml
```

6. (GPU 노드만 해당) GPU 인스턴스 유형과 Amazon EKS 최적화 가속 AMI를 선택한 경우 클러스터에 [Kubernetes용 NVIDIA 디바이스 플러그인](#)을 DaemonSet으로 적용해야 합니다. 다음 명령을 실행하기 전에 **vX.X.X**를 원하는 [NVIDIA/k8s-device-plugin](#) 버전으로 바꿉니다.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/
deployments/static/nvidia-device-plugin.yml
```

3단계: 추가 작업

1. (선택 사항) [샘플 애플리케이션](#)을 배포하여 클러스터 및 Linux 노드를 테스트합니다.
2. 클러스터가 Outpost에 배포된 경우 이 단계를 건너뛰십시오. 클러스터가 AWS 클라우드에 배포된 경우 다음 정보는 선택 사항입니다. AmazonEKS_CNI_Policy 관리형 IAM 정책이 [Amazon EKS 노드 IAM 역할](#)에 연결되어 있는 경우 대신 Kubernetes aws-node 서비스 계정에 연결한 IAM 역할에 할당하는 것이 좋습니다. 자세한 내용은 [the section called "IRSA에 대한 구성"](#) 섹션을 참조하세요.

Amazon EKS에서 기계 학습 개요

Amazon Elastic Kubernetes Service(EKS)는 조직이 탁월한 유연성과 제어력을 바탕으로 AI 및 기계 학습(ML) 워크로드를 배포, 관리 및 확장할 수 있도록 지원하는 관리형 Kubernetes 플랫폼입니다. 오픈 소스 Kubernetes 에코시스템을 기반으로 구축된 EKS를 사용하면 기존 Kubernetes 전문 지식을 활용 하면서 오픈 소스 도구 및 AWS 서비스와 원활하게 통합할 수 있습니다.

대규모 모델 훈련, 실시간 온라인 추론 실행, 생성형 AI 애플리케이션 배포 등 어떤 작업을 수행하든 EKS는 AI/ML 프로젝트에 필요한 성능, 확장성, 비용 효율성을 제공합니다.

AI/ML용 EKS를 선택해야 하는 이유

EKS는 복잡한 AI/ML 워크로드를 배포하고 관리하는 데 도움이 되는 관리형 Kubernetes 플랫폼입니다. 오픈 소스 Kubernetes 에코시스템을 기반으로 구축되었으며 AWS 서비스와 통합되어 고급 프로젝트에 필요한 제어 및 확장성을 제공합니다. AI/ML 배포를 처음 접하는 팀의 경우 기존 Kubernetes 기술이 바로 이전되므로 여러 워크로드를 효율적으로 오케스트레이션할 수 있습니다.

EKS는 운영 체제 사용자 지정부터 컴퓨팅 스케일링까지 모든 것을 지원하며, 오픈 소스 기반은 기술적 유연성을 촉진하여 향후 인프라 결정을 위한 선택의 폭을 넓혀줍니다. 이 플랫폼은 다음과 같은 기능을 지원하여 AI/ML 워크로드에 필요한 성능 및 튜닝 옵션을 제공합니다.

- 숨겨진 추상화 없이 비용과 구성을 미세 조정할 수 있는 전체 클러스터 제어
- 프로덕션 환경에서 실시간 추론 워크로드에 대한 1초 미만의 지연 시간
- 멀티 인스턴스 GPU, 멀티 클라우드 전략, OS 수준 튜닝과 같은 고급 사용자 지정
- AI/ML 파이프라인 전반에서 통합 오케스트레이터로 EKS를 사용하여 워크로드를 중앙 집중화하는 기능

주요 사용 사례

Amazon EKS는 다양한 기술과 배포 패턴을 지원하여 광범위한 AI/ML 워크로드를 위한 강력한 플랫폼을 제공합니다.

- 실시간(온라인) 추론: EKS는 Amazon EC2 [Inf1](#) 및 [Inf2](#) 인스턴스에서 [TorchServe](#), [Triton Inference Server](#), [KServe](#)와 같은 도구를 사용하여 1초 미만의 지연 시간으로 사기 탐지와 같은 수신 데이터에 대한 즉각적인 예측을 수행합니다. 이러한 워크로드는 [Karpenter](#) 및 [KEDA](#)를 통한 동적 스케일링의 이점을 누리는 동시에 포드 간 모델 샤딩을 위해 [Amazon EFS](#)를 활용합니다. [Amazon ECR 풀스루](#)

[캐시\(PTC\)](#)는 모델 업데이트를 가속화하고, [Amazon EBS](#) 최적화 볼륨을 갖춘 [Bottlerocket](#) 데이터 볼륨은 빠른 데이터 액세스를 보장합니다.

- 일반 모델 훈련: 조직은 EKS를 활용하여 [Amazon EC2 P4d](#) 및 [Amazon EC2 Trn1](#) 인스턴스에서 [Kubeflow 훈련 운영자\(KRO\)](#), [Ray Serve](#) 및 [Torch Distributed Elastic](#)을 사용해 장기간에 걸쳐 대규모 데이터 세트에서 복잡한 모델을 훈련합니다. 이러한 워크로드는 [Volcano](#), [Yunikorn](#), [Kueue](#)와 같은 도구를 사용한 배치 예약에서 지원됩니다. [Amazon EFS](#)를 사용하면 모델 체크포인트를 공유할 수 있으며 [Amazon S3](#)는 버전 관리를 위한 수명 주기 정책에 따라 모델 가져오기/내보내기를 처리합니다.
- 검색 증강 생성(RAG) 파이프라인: EKS는 검색 및 생성 프로세스를 통합하여 고객 지원 챗봇 및 이와 유사한 애플리케이션을 관리합니다. 이러한 워크로드는 오케스트레이션을 위한 [Argo Workflows](#), [Kubeflow](#) 등의 도구, [Pinecone](#), [Weaviate](#) 등의 벡터 데이터베이스 또는 [Amazon OpenSearch](#)를 사용하며 [Application Load Balancer Controller\(LBC\)](#)를 통해 애플리케이션을 사용자에게 노출하는 경우가 많습니다. [NVIDIA NIM](#)은 GPU 활용도를 최적화하고, [Prometheus](#)와 [Grafana](#)는 리소스 사용량을 모니터링합니다.
- 생성형 AI 모델 배포: 기업은 Amazon [EC2 G5](#) 및 [Inferentia](#) 액셀러레이터에서 [Ray Serve](#), [vLLM](#) 및 [Triton Inference Server](#)를 사용하여 텍스트나 이미지 생성과 같은 실시간 콘텐츠 생성 서비스를 EKS에 배포합니다. 이러한 배포는 대규모 모델의 성능과 메모리 사용량을 최적화합니다. [JupyterHub](#)는 반복 개발을 가능하게 하고, [Gradio](#)는 간단한 웹 인터페이스를 제공하며, [S3 Mountpoint CSI 드라이버](#)는 대용량 모델 파일에 액세스하기 위한 파일 시스템으로 S3 버킷을 마운트할 수 있게 합니다.
- 배치(오프라인) 추론: 조직은 [AWS Batch](#) 또는 [Volcano](#)를 사용하여 예약된 작업을 통해 대규모 데이터 세트를 효율적으로 처리합니다. 이러한 워크로드는 AWS [Inferentia](#) 칩용 [Inf1](#) 및 [Inf2](#) 인스턴스, NVIDIA T4 GPU용 Amazon EC2 [G4dn](#) 인스턴스 또는 [c5](#) 및 [c6i](#) CPU 인스턴스를 사용하여 분석 작업의 피크 외 시간 동안 리소스 사용량을 극대화하는 경우가 많습니다. [AWS Neuron SDK](#) 및 NVIDIA GPU 드라이버는 성능을 최적화하고, MIG/TS는 GPU 공유를 가능하게 합니다. 스토리지 솔루션에는 다양한 스토리지 클래스를 위한 CSI 드라이버와 함께 [Amazon S3](#), Amazon [EFS](#) 및 [FSx for Lustre](#)가 포함됩니다. 모델 관리는 [Kubeflow Pipelines](#), [Argo Workflows](#), [Ray Cluster](#)와 같은 도구를 활용하고, 모니터링은 [Prometheus](#), [Grafana](#) 및 사용자 지정 모델 모니터링 도구로 처리됩니다.

사례 연구

고객은 다음 사례 연구와 같이 GPU 사용량 최적화 또는 1초 미만의 지연 시간으로 실시간 추론 워크로드 실행 등의 다양한 이유로 Amazon EKS를 선택합니다. Amazon EKS에 대한 모든 사례 연구 목록은 [AWS 고객 성공 사례](#)를 참조하세요.

- [Unitary](#)는 높은 처리량과 지연 시간이 짧은 추론이 필요한 콘텐츠 조정에 AI를 사용하여 매일 2,600만 개의 동영상을 처리하며, 컨테이너 부팅 시간을 80% 단축하여 트래픽 변동에 따른 스케일링 이벤트에 빠르게 대응할 수 있습니다.
- 전 세계 7,000만 명의 사용자를 지원하는 시각적 협업 플랫폼인 [Miro](#)는 이전의 자체 관리형 Kubernetes 클러스터에 비해 컴퓨팅 비용이 80% 절감되었다고 보고했습니다.
- 고객이 텍스트 프롬프트에서 사실적인 동영상을 제작할 수 있도록 생성형 AI 동영상 제작 서비스를 제공하는 [Synthesia](#)는 ML 모델 훈련 처리량을 30배 향상시켰습니다.
- 호텔 업계에서 HR 기술을 제공하는 [Harri](#)는 [AWS Graviton 프로세서](#)로 마이그레이션하여 수요 급증에 대응하여 스케일링 속도를 90% 높이고 컴퓨팅 비용을 30% 절감했습니다.
- AI 기반 고객 서비스 자동화 회사인 [Ada Support](#)는 컴퓨팅 비용을 15% 절감하고 컴퓨팅 효율성을 30% 향상시켰습니다.
- 기업이 파운데이션 모델과 대규모 언어 모델을 구축하고 조정할 수 있도록 지원하는 [Snorkel AI](#)는 GPU 리소스에 대한 지능형 스케일링 메커니즘을 구현하여 40% 이상의 비용 절감을 달성했습니다.

EKS에서 기계 학습 사용 시작

AWS 클라우드의 EKS에서 기계 학습 플랫폼 및 워크로드에 대한 계획 수립 및 사용을 시작하려면 [the section called “ML 시작하기”](#) 섹션으로 이동합니다.

ML 시작하기

EKS의 기계 학습으로 이동하려면 먼저 이러한 권장 패턴 중에서 선택하여 ML 워크로드 실행을 시작할 수 있도록 EKS 클러스터와 ML 소프트웨어 및 하드웨어를 빠르게 가져옵니다. 이러한 패턴의 대부분은 [Amazon EKS의 데이터](#) 사이트에서 사용할 수 있는 Terraform 블루프린트를 기반으로 합니다. 시작하기 전에 유의해야 할 몇 가지 사항은 다음과 같습니다.

- 이러한 절차를 실행하려면 GPU 또는 Neuron 인스턴스가 필요합니다. 리소스의 가용성이 부족하면 클러스터 생성 또는 노드 자동 크기 조정 중에 이러한 절차가 실패할 수 있습니다.
- Neuron SDK(Tranium 및 Inferentia 기반 인스턴스)는 비용을 절감할 수 있으며 NVIDIA GPU보다 가용성이 높습니다. 따라서 워크로드가 허용하는 경우 기계 학습 워크로드에 Neuron을 사용하는 것이 좋습니다([AWS Neuron 시작](#) 참조).
- 여기에서 제공되는 일부 시작 경험은 자신의 [Hugging Face](#) 계정을 통해 데이터를 얻어야 합니다.

시작하려면 기계 학습 워크로드를 실행하기 위한 인프라 설정을 시작하도록 설계된 다음 패턴 중에서 선택합니다.

- [EKS의 JupyterHub](#) : Time Slicing 및 MIG 기능과 프로파일을 사용하는 다중 테넌트 구성을 보여주는 [JupyterHub 블루프린트](#)를 살펴봅니다. 이는 EKS에 대규모 JupyterHub 플랫폼을 구축하는 데 적합합니다.
- [AWS Neuron 및 RayServe의 대규모 언어 모델](#) : [AWS Neuron](#)을 사용하여 Amazon EKS, AWS Trainium, AWS Inferentia 가속기에서 대규모 언어 모델(LLM)을 실행합니다. 다음과 같은 구성 요소를 사용하여 추론 요청을 수행하기 위한 플랫폼을 설정하는 방법에 대한 지침은 [AWS Neuron에서 RayServe 및 vLLM을 사용하여 LLM 지원](#)을 참조하세요.
 - 딥 러닝용 AWS Neuron SDK 툴킷
 - AWS Inferentia 및 Trainium 가속기
 - vLLM-가변 길이 언어 모델([vLLM](#) 설명서 사이트 참조)
 - RayServe 확장 가능 모델 서비스 라이브러리([Ray Serve: Scalable and Programmable Serving](#) 사이트 참조)
 - 자신의 [Hugging Face](#) 계정을 사용하는 Llama-3 언어 모델.
 - AWS CloudWatch 및 Neuron Monitor를 통한 관찰성
 - WebUI 열기
- [NVIDIA 및 Triton의 대규모 언어 모델](#) : Amazon EKS 및 NVIDIA GPU에 여러 대규모 언어 모델(LLM)을 배포합니다. 다음과 같은 구성 요소를 사용하여 추론 요청을 수행하기 위한 플랫폼을 설정하는 방법에 대한 지침은 [NVIDIA Triton Server 및 vLLM을 사용하여 여러 대규모 언어 모델 배포](#)를 참조하세요.
 - NVIDIA Triton 추론 서버([Triton 추론 서버](#) GitHub 사이트 참조)
 - vLLM-가변 길이 언어 모델([vLLM](#) 설명서 사이트 참조)
 - 자신의 [Hugging Face](#) 계정을 사용하는 mistralai/Mistral-7B-Instruct-v0.2 및 meta-llama/Llama-2-7b-chat-hf의 두 가지 언어 모델.

EKS에서 ML 계속

이 페이지에서 설명한 블루프린트 중에서 선택하는 것 외에도 원하는 경우 EKS에서 ML 사용 설명서를 진행하는 다른 방법이 있습니다. 예를 들어, 다음을 수행할 수 있습니다.

- EKS에서 ML 사용 자습서 - EKS에서 자체 기계 학습 모델을 구축하고 실행하기 위한 다른 엔드 투 엔드 자습서를 실행합니다. [the section called “EKS에서 ML 사용을 위한 자습서 사용”](#) 섹션을 참조하세요.

EKS에서 ML 사용 작업을 개선하려면 다음을 참조하세요.

- ML 준비-사용자 지정 AMI 및 GPU 예약과 같은 기능을 사용하여 ML에서 EKS 사용을 준비하는 방법을 알아봅니다. [the section called “ML 준비”](#)을(를) 참조하십시오.

ML 클러스터 준비

EKS에서의 기계 학습 경험을 개선할 방법이 있습니다. 이 섹션의 다음 페이지는 다음과 같은 효과를 얻는 데 도움이 됩니다.

- EKS에서 ML 사용을 위한 선택 사항 이해
- EKS 및 ML 환경 준비

특히 다음과 같은 이점이 있습니다.

- AMI 선택: AWS는 EKS에서 ML 워크로드를 실행하기 위한 여러 사용자 지정 AMI를 제공합니다. [the section called “Linux GPU AMI 실행”](#) 및 [the section called “Windows GPU AMI 실행”](#) 섹션을 참조하세요.
- AMI 사용자 지정: AWS의 사용자 지정 AMI 추가로 수정하여 특정 사용 사례에 필요한 다른 소프트웨어 및 드라이버를 추가할 수 있습니다. [the section called “SMN용 GPU 예약”](#) 섹션을 참조하세요.
- GPU 예약: GPU에 대한 수요로 인해 필요한 GPU를 필요할 때 사용할 수 있도록 필요한 GPU를 미리 예약할 수 있습니다. [the section called “테인트 GPU 노드”](#) 섹션을 참조하세요.
- EFA 추가: 노드 간 클러스터 통신을 위한 네트워크 성능을 개선하려면 Elastic Fabric Adapter를 추가합니다. [the section called “EFA를 사용하여 클러스터 훈련”](#) 섹션을 참조하세요.
- AWSInferentia 워크로드 사용: Amazon EC2 Inf1 인스턴스를 사용하여 EKS 클러스터를 생성합니다. [the section called “Inferentia 클러스터 준비”](#)을(를) 참조하십시오.

주제

- [GPU 가속 컨테이너 실행\(EC2의 Linux\)](#)
- [GPU 가속 컨테이너 실행\(Windows on EC2 G-Series\)](#)
- [ML용 용량 블록을 사용하여 관리형 노드 그룹 생성](#)
- [ML용 용량 블록을 사용하여 자체 관리형 노드 생성](#)
- [특정 노드에서 포드가 예약되지 않도록 방지](#)
- [Elastic Fabric Adapter를 사용한 Amazon EKS에서 기계 학습 훈련 실행](#)
- [기계 학습을 위해 Amazon EKS와 함께 AWS Inferentia 인스턴스 사용](#)

GPU 가속 컨테이너 실행(EC2의 Linux)

Amazon EKS 최적화 가속 Amazon Linux AMI는 표준 Amazon EKS 최적화 Amazon Linux AMI를 기반으로 빌드됩니다. AMI에 대한 자세한 내용은 [the section called “Amazon EKS 최적화 가속 Amazon Linux AMI”](#) 섹션을 참조하세요. 다음 텍스트에서는 AWS Neuron 기반 워크로드를 활성화하는 방법을 설명합니다.

AWS Neuron(ML 가속기) 기반 워크로드를 활성화하려면

Amazon EKS에서 Neuron을 사용하는 교육 및 추론 워크로드에 대한 자세한 내용은 다음 참조를 참조하세요.

- AWS Neuron Documentation의 [Containers - Kubernetes - Getting Started](#)
- GitHub의 AWS Neuron EKS Samples의 [Training](#)
- [Amazon EKS에서 AWSInferentia로 ML 추론 워크로드 배포하기](#)

다음 절차에서는 Amazon EKS 최적화 가속 AMI를 사용하여 GPU 기반 인스턴스에서 워크로드를 실행하는 방법을 설명합니다.

1. GPU 노드가 클러스터에 조인하면 클러스터에서 [Kubernetes용 NVIDIA 디바이스 플러그인](#)을 DaemonSet으로 적용해야 합니다. 다음 명령을 실행하기 전에 **vX.X.X**를 원하는 [NVIDIA/k8s-device-plugin](#) 버전으로 바꿉니다.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/vX.X.X/deployments/static/nvidia-device-plugin.yml
```

2. 다음 명령으로 노드에 할당 가능한 GPU가 있는지 확인할 수 있습니다.

```
kubectl get nodes "-o=custom-columns=NAME:.metadata.name,GPU:.status.allocatable.nvidia\.com/gpu"
```

3. 다음 콘텐츠를 가진 `nvidia-smi.yaml`이라는 파일을 생성합니다: **##**를 [nvidia/cuda](#)에 대해 원하는 태그로 바꿉니다. 이 매니페스트는 노드에서 `nvidia-smi`를 실행하는 [NVIDIA CUDA](#) 컨테이너를 실행합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: nvidia-smi
```

```
spec:
  restartPolicy: OnFailure
  containers:
  - name: nvidia-smi
    image: nvidia/cuda:tag
    args:
    - "nvidia-smi"
    resources:
      limits:
        nvidia.com/gpu: 1
```

4. 다음 명령으로 매니페스트를 적용합니다.

```
kubectl apply -f nvidia-smi.yaml
```

5. 포드 실행이 끝난 후, 다음 명령을 사용하여 로그를 확인합니다.

```
kubectl logs nvidia-smi
```

예제 출력은 다음과 같습니다.

```
Mon Aug  6 20:23:31 20XX
+-----+
| NVIDIA-SMI  XXX.XX                Driver Version:  XXX.XX                |
+-----+-----+-----+-----+-----+-----+
| GPU   Name           Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
+-----+-----+-----+-----+-----+-----+
|    0   Tesla V100-SXM2...    On   | 00000000:00:1C.0 Off  |           0         |
| N/A   46C    P0     47W / 300W |  0MiB / 16160MiB |      0%    Default   |
+-----+-----+-----+-----+-----+-----+
+-----+
| Processes:                                                       GPU Memory |
|  GPU       PID    Type    Process name                     Usage    |
+-----+-----+-----+-----+-----+-----+
| No running processes found                                         |
+-----+
```

GPU 가속 컨테이너 실행(Windows on EC2 G-Series)

Important

TensorWorks의 [DirectX용 Kubernetes 디바이스 플러그인](#)은 AWS에서 보증, 지원 또는 유지 관리하지 않는 타사 도구입니다. AWS는 이 플러그인의 보안, 안정성 또는 성능에 대해 책임을 지지 않습니다.

TensorWorks의 DirectX용 Kubernetes 디바이스 플러그인과 함께 NVIDIA GPU를 사용하여 Amazon EKS(Elastic Kubernetes Service)에서 GPU 가속 Windows 컨테이너 워크로드를 실행하는 방법을 알아봅니다. 자세한 내용은 [Kubernetes Device Plugin for DirectX](#)을 참조하세요.

Windows 컨테이너에 대한 GPU 가속화를 설정하는 두 가지 주요 접근 방식이 있습니다.

- 옵션 1: 필요한 GPU 드라이버가 사전 설치된 [사용자 지정 EKS Windows 최적화 AMI를 빌드](#)합니다.
 - GPU 가속 Windows 컨테이너를 실행할 준비가 된 일관되고 사전 구성된 환경이 필요하고, 사용자 지정 AMI를 구축 및 유지 관리하는 데 추가 노력을 투자할 수 있는 경우 이 접근 방식을 사용합니다.
- 옵션 2: 인스턴스를 시작한 후 EKS 워커 노드에 필요한 GPU 드라이버를 설치합니다.
 - 더 간단한 설정 프로세스를 원하고 새로운 각 워커 노드에 GPU 드라이버를 설치하는 것이 괜찮다면 이 접근 방식을 사용합니다. GPU 가속 워크로드를 평가하거나 프로토타이핑하는 경우 개발 환경에 더 적합합니다.

이 가이드에 설명된 단계를 사용하여 두 접근 방식을 모두 활용할 수 있습니다.

고려 사항

이 가이드에서는 NVIDIA GPU, NVIDIA GRID 드라이버, TensorWorks의 [DirectX용 Kubernetes 디바이스 플러그인](#)을 사용하여 Windows 컨테이너에 GPU 가속화를 설치하고 설정하는 단계를 제공합니다. 해당 단계는 Amazon EKS에서 Windows 컨테이너 워크로드에 GPU 가속화를 제공하기 위해 테스트 및 확인되었습니다. 호환되는 드라이버 및 디바이스 플러그인에 대한 자세한 내용은 [the section called “알려진 제한 사항”](#) 섹션을 참조하세요. 계속하기 전에 다음 사항에 유의하세요.

- [NVIDIA GRID 드라이버](#)가 있는 G-패밀리 인스턴스 유형만 이 가이드와 함께 작동하도록 테스트 및 확인되었습니다. 다른 인스턴스 유형 및 드라이버 조합도 GPU 가속 Windows 컨테이너를 실행할 수 있지만 이 가이드에서 다루지 않은 추가 구성 단계가 필요할 수 있습니다.

- DirectX 기반 워크로드만 이 가이드와 함께 작동하도록 테스트 및 확인되었습니다. OpenGL, Vulkan, OpenCL과 같은 다른 GPU API는 GPU 가속 Windows 컨테이너 실행에서 호환될 수 있지만 이 가이드에서 다루지 않은 추가 구성 단계가 필요할 수 있습니다.
- GPU 가속 Windows 컨테이너를 실행하기 전에 알아야 할 몇 가지 알려진 제한 사항이 있습니다. 자세한 내용은 [the section called “알려진 제한 사항”](#) 섹션을 참조하세요.

사전 조건

Amazon EKS에서 Windows 컨테이너에 GPU 가속화를 활성화하려면 진행하기 전에 다음 요구 사항을 준비해야 합니다.

- Kubernetes v1.27 이상을 사용하여 Amazon EKS 클러스터를 시작합니다.
- Windows Server 2022 이상으로 Windows 노드를 프로비저닝합니다.
- [G4](#) 또는 [G5](#)와 같은 인스턴스 유형의 G 패밀리에 Windows 노드를 프로비저닝합니다.
- Containerd 1.7.x 또는 2.x.x를 사용하여 컨테이너 런타임으로 Windows 노드를 프로비저닝합니다. (Amazon EKS 최적화 AMI의 Containerd 버전을 확인하려면 [the section called “버전 정보 받기”](#) 섹션을 참조하세요.)

각 Windows 노드에 GPU 드라이버 설치

EKS 워커 노드에 NVIDIA GRID 드라이버를 설치하려면 [Amazon EC2 인스턴스의 NVIDIA 드라이버](#)에 설명된 단계를 따릅니다. [설치 옵션-옵션 3: GRID 드라이버](#)로 이동하여 설치 단계를 따릅니다.

Windows Server Core용 설치

데스크톱 경험이 없는 Windows Server Core의 경우 다음 명령을 사용하여 NVIDIA GRID 드라이버를 자동으로 설치합니다.

```
$nvidiaInstallerFilePath = nvidia-driver-installer.exe # Replace with path to installer
$installerArguments = "-s -clean -noreboot -noeula"
Start-Process -FilePath $nvidiaInstallerFilePath -ArgumentList $installerArguments -
Wait -NoNewWindow -PassThru
```

설치 확인

다음 PowerShell 명령을 실행하여 인스턴스의 GPU에 대한 진단 정보를 표시합니다.

```
nvidia-smi
```

이 명령은 NVIDIA 드라이버 버전과 GPU 하드웨어 관련 정보를 표시합니다. 이 명령의 출력이 설치될 것으로 예상되는 NVIDIA GRID 드라이버 버전과 일치하는지 확인합니다.

각 노드에 GPU 디바이스 플러그인 배포

Windows 노드의 컨테이너에 대한 GPU 리소스의 검색 및 노출을 활성화하려면 디바이스 플러그인이 필요합니다. EKS 클러스터에서 Tensorworks의 [DirectX 디바이스 플러그인](#)을 DaemonSet로 실행하여 각 워커 노드에 배포합니다. 다음 단계를 수반하는 [README.md](#)에 지정된 설치 안내를 따릅니다. 다음을 수행하는 것이 좋습니다.

- kube-system 네임스페이스에 디바이스 플러그인을 배포합니다.
- DaemonSet에 대한 적절한 리소스 제한을 설정하여 노드에서 과도한 리소스를 소비하지 않도록 합니다.

Note

디바이스 플러그인 DaemonSet는 권한이 높은 호스트 프로세스 컨테이너로 모든 노드에서 실행됩니다. 권한 있는 사용자만 권한 있는 명령을 실행할 수 있도록 RBAC 제어를 구현하여 이 DaemonSet에 대한 액세스를 제한하는 것이 좋습니다.

GPU 가속 컨테이너를 실행할 때 디바이스 플러그인은 두 가지 모드를 지원합니다.

- 단일 테넌시 모드: 이 모드는 인스턴스의 단일 컨테이너에 모든 GPU 리소스를 전용합니다. 다음 명령을 사용하여 단일 테넌시 지원으로 디바이스 플러그인을 설치합니다. 자세한 내용은 README.md를 참조하세요.

```
kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-plugins/main/deployments/default-daemonsets.yml"
```

- 다중 테넌시 모드: 이 모드에서는 인스턴스의 여러 컨테이너 간에 GPU 리소스를 공유할 수 있습니다. 다음 명령을 사용하여 다중 테넌시 지원으로 디바이스 플러그인을 설치합니다. 자세한 내용은 README.md를 참조하세요.

```
kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-plugins/main/deployments/multitenancy-inline.yml"
```

또는 ConfigMap을 사용하여 다중 테넌시를 지정합니다.

```
kubectl apply -f "https://raw.githubusercontent.com/TensorWorks/directx-device-plugins/main/deployments/multitenancy-configmap.yml"
```

디바이스 플러그인 배포 확인

디바이스 플러그인을 배포한 후 <namespace>를 바꾸고 다음 명령을 실행하여 DirectX 디바이스 플러그인이 모든 Windows 노드에서 올바르게 실행되고 있는지 확인합니다.

```
kubectl get ds device-plugin-wddm -n <namespace>
```

컨테이너 배포 준비가 되었는지 확인

디바이스 플러그인 DaemonSet가 GPU 기반 Windows 워커 노드에서 실행되면 다음 명령을 사용하여 각 노드에 할당 가능한 GPU가 있는지 확인합니다. 해당 숫자는 각 노드의 DirectX 디바이스 수와 일치해야 합니다.

```
kubectl get nodes "-o=custom-columns=NAME:.metadata.name,DirectX:.status.allocatable.directx\.microsoft\.com/display"
```

GPU 가속화를 사용하여 Windows 컨테이너 실행

포드를 시작하기 전에 `.spec.containers[].resources`에서 `directx.microsoft.com/display` 리소스 이름을 지정합니다. 이는 컨테이너에 GPU 지원 기능이 필요하고 kube-scheduler가 사용 가능한 GPU 리소스가 있는 사전 구성된 Windows 노드에 포드를 배치하려고 시도함을 나타냅니다.

예를 들어, Monte Carlo 시뮬레이션을 실행하여 pi 값을 추정하기 위해 Job을 시작하는 아래의 샘플 명령을 참조하세요. 이 예제는 [DirectX용 Kubernetes 디바이스 플러그인](#) GitHub 리포지토리의 예제로, Windows 노드 GPU 기능을 테스트하기 위해 실행할 수 있는 [여러 예제](#) 중에서 선택할 수 있습니다.

```
cat <<EOF | kubectl apply -f -
apiVersion: batch/v1
kind: Job
metadata:
  name: example-cuda-montecarlo-wddm
spec:
```

```

template:
  spec:
    containers:
      - name: example-cuda-montecarlo-wddm
        image: "index.docker.io/tensorworks/example-cuda-montecarlo:0.0.1"
        resources:
          limits:
            directx.microsoft.com/display: 1
        nodeSelector:
          "kubernetes.io/os": windows
        restartPolicy: Never
    backoffLimit: 0
EOF

```

알려진 제한 사항

모든 GPU 사용 가능

지정된 컨테이너에 대해 특정 수의 GPU를 요청하더라도 인스턴스의 모든 GPU는 호스트의 각 실행 컨테이너에서 사용할 수 있습니다. 또한 기본 동작은 노드에 사용 가능한 GPU가 여러 개 있더라도 호스트에서 실행되는 모든 컨테이너가 인덱스 0과 함께 GPU를 사용합니다. 따라서 다중 GPU 작업이 올바르게 작동하려면 애플리케이션의 코드 내에서 사용할 특정 GPU 디바이스를 명시적으로 지정해야 합니다.

애플리케이션에 사용할 디바이스를 할당하는 정확한 구현은 사용 중인 프로그래밍 언어 또는 프레임 워크에 따라 달라집니다. 예를 들어, CUDA 프로그래밍을 사용하는 경우 특정 GPU를 선택하려면 함수 [cudaSetDevice\(\)](#)를 사용하여 애플리케이션 코드에 사용할 디바이스를 명시적으로 지정할 수 있습니다.

디바이스를 명시적으로 지정해야 하는 이유는 Windows 컨테이너에 영향을 미치는 알려진 문제 때문입니다. [microsoft/Windows-Containers 문제 #333](#)에서 이 문제 해결의 진행 상황을 추적할 수 있습니다. 다음 표는 이 GPU 할당 동작의 시각적 표현과 실제 예제를 나타냅니다.

4개의 GPU와 함께 제공되는 EC2 인스턴스 유형 g4dn.12xlarge의 단일 Windows 노드가 있는 시나리오를 생각해 보세요. 이 인스턴스에서 포트 3개가 시작되는 시나리오를 생각해 보세요. 이 표는 각 컨테이너에서 요청한 GPU 수와 관계없이 포트 3개 모두 인스턴스의 GPU 4개 모두에 액세스할 수 있으며 기본적으로 디바이스 인덱스가 0인 GPU를 활용합니다.

Pod	요청된 GPU	실제 GPU 액세스	기본 GPU 사용량	사용 가능한 GPU 인덱스	인스턴스 GPU 합계
포드 1	GPU 1개	GPU 4개 모두	인덱스가 0인 GPU	0, 1, 2, 3	4
포드 2	GPU 2개	GPU 4개 모두	인덱스가 0인 GPU	0, 1, 2, 3	4
포드 3	GPU 1개	GPU 4개 모두	인덱스가 0인 GPU	0, 1, 2, 3	4

Kubernetes 디바이스 플러그인 지원

NVIDIA의 [Kubernetes 디바이스 플러그인](#) 공식 구현은 Windows를 지원하지 않습니다. [NVIDIA/k8s-device-plugin 문제 #419](#)에서 공식 Windows 지원 추가 진행 상황을 추적할 수 있습니다.

GPU 컴퓨팅 인스턴스 제한 사항

AWS 계정 구성에 따라 시작할 수 있는 Amazon EC2 GPU 컴퓨팅 인스턴스의 수와 유형에 대한 서비스 제한이 있을 수 있습니다. 추가 용량이 필요한 경우 [할당량 증가 요청](#)을 할 수 있습니다.

Windows GPU 최적화 AMI를 빌드해야 함

Amazon EKS에서 제공하는 EKS Windows GPU 최적화 AMI 또는 EC2 Image Builder 관리형 구성 요소는 없습니다. 이 가이드의 단계에 따라 필요한 GPU 드라이버가 사전 설치된 사용자 지정 EKS Windows 최적화 AMI를 빌드하거나 인스턴스를 시작한 후 EKS 워커 노드에 필요한 GPU 드라이버를 설치해야 합니다.

Inferentia 및 Trainium 지원 안 됨

AWS [Inferentia](#) 및 AWS [Trainium](#) 기반 워크로드는 Windows에서 지원되지 않습니다.

ML용 용량 블록을 사용하여 관리형 노드 그룹 생성

기계 학습용 용량 블록(ML)을 사용하면 향후 날짜에 GPU 인스턴스를 예약하여 단기간의 ML 워크로드를 지원할 수 있습니다. 자세한 내용을 알아보려면 Amazon EC2 - Linux 인스턴스용 사용 설명서의 [ML용 용량 블록](#)을 참조하세요.

고려 사항

Important

- 용량 블록은 특정 Amazon EC2 인스턴스 유형 및 AWS 리전에서만 사용할 수 있습니다. 호환성 정보는 Amazon EC2 - Linux 인스턴스용 사용 설명서의 [용량 블록 작업 사전 조건](#)을 참조하세요.
- 자세한 내용은 Amazon EC2 Auto Scaling 사용 설명서의 [Use Capacity Blocks for machine learning workloads](#)를 참조하세요.
- 용량 블록이 있는 관리형 노드 그룹은 사용자 지정 시작 템플릿을 통해서만 생성할 수 있습니다.
- 용량 블록으로 관리되는 노드 그룹을 업그레이드할 때는 원하는 노드 그룹 크기가 0으로 설정되어 있는지 확인하세요.

Amazon EC2 용량 블록을 사용하여 관리형 노드 그룹 생성

GPU 가속화 워커 노드를 프로비저닝하고 규모를 조정하기 위해 Amazon EKS 관리형 노드 그룹과 함께 용량 블록을 사용할 수 있습니다. AWS CloudFormation 템플릿 예는 프로덕션 클러스터에 필요한 모든 측면을 다루지는 않습니다. 일반적으로 노드를 클러스터에 조인하고 Amazon EKS 가속 AMI를 지정하기 위한 부트스트래핑 스크립트도 필요합니다. 자세한 내용은 [the section called “생성”](#) 단원을 참조하십시오.

1. 워크로드에 적합하고 Amazon EKS 관리형 노드 그룹과 작동하는 시작 템플릿을 생성합니다. 자세한 내용은 [the section called “시작 템플릿”](#) 섹션을 참조하세요.

위 절차의 요구 사항 외에도 LaunchTemplateData에 다음 사항이 포함되어 있는지 확인합니다.

- MarketType를 "capacity-block"로 설정한 InstanceMarketOptions
- CapacityReservationId가 용량 블록으로 설정된
CapacityReservationSpecification: CapacityReservationTarget(예: `cr-02168da1478b509e0`)
- InstanceType이 용량 블록을 지원하는 인스턴스 유형으로 설정(예: `p5.48xlarge`)

다음은 용량 블록을 대상으로 하는 시작 템플릿을 생성하는 CloudFormation 템플릿의 발췌문입니다. 사용자 지정 AMI 관리형 노드 그룹을 생성하려면 ImageId 및 UserData 파라미터를 추가할 수 있습니다.

```

NodeLaunchTemplate:
  Type: "AWS::EC2::LaunchTemplate"
  Properties:
    LaunchTemplateData:
      InstanceMarketOptions:
        MarketType: "capacity-block"
      CapacityReservationSpecification:
        CapacityReservationTarget:
          CapacityReservationId: "cr-02168da1478b509e0"
      InstanceType: p5.48xlarge

```

2. 시작 템플릿을 사용하여 관리형 노드 그룹을 생성합니다.

다음은 용량 블록에 대한 노드 그룹 생성 명령의 예입니다. *example-values*를 클러스터에 해당하는 것으로 변경합니다.

용량 블록 관리형 노드 그룹을 생성하는 경우 다음을 수행합니다.

- `capacity-type`을 "CAPACITY_BLOCK"로 설정합니다. 용량 유형이 "CAPACITY_BLOCK"으로 설정되지 않았거나 위의 필수 시작 템플릿 값 중 하나라도 누락된 경우 생성 요청이 거부됩니다.
- 생성 요청에서 subnets를 지정할 때는 용량 예약과 동일한 가용 영역의 서브넷만 지정해야 합니다.
- 생성 요청에 0이 아닌 `desiredSize`를 지정하면 Amazon EKS는 Auto Scaling 그룹(ASG)을 생성할 때 이를 따릅니다. 그러나 용량 예약이 활성화되기 전에 생성 요청이 이루어지면 ASG는 활성화될 때까지 Amazon EC2 인스턴스를 시작할 수 없습니다. 결과적으로 ASG 규모 조정 활동에서 시작 오류가 발생합니다. 예약이 활성화될 때마다 인스턴스 실행이 성공하고 ASG가 생성 시 언급된 `desiredSize`까지 스케일 업합니다.

```

aws eks create-nodegroup \
  --cluster-name my-cluster \
  --nodegroup-name my-mng \
  --node-role node-role-arn \
  --region region-code \
  --subnets subnet-id \
  --scaling-config minSize=node-group-min-size,maxSize=node-group-max-size,desiredSize=node-group-desired-size \
  --ami-type "AL2023_x86_64_NVIDIA" \
  --capacity-type "CAPACITY_BLOCK" \
  --launch-template id="lt-id",version=1

```

3. 스케일 업 후 노드가 조인되었는지 확인합니다. 용량 블록이 있는 관리형 노드 그룹을 사용하는 Amazon EKS 클러스터는 시작된 인스턴스가 실제로 클러스터에 조인하고 등록되는지 확인하는 검증을 수행하지 않습니다.
4. 생성 시 `desiredSize`를 0으로 설정한 경우 용량 예약이 활성화되면 노드 그룹을 스케일 업할 수 있는 다양한 옵션이 있습니다.
 - 용량 블록 예약 시작 시간에 맞춰 ASG에 대한 예약된 규모 조정 정책을 생성합니다. 자세한 정보는 Amazon EC2 Auto Scaling 사용 설명서의 [Amazon EC2 Auto Scaling 예정 스케일링](#)을 참조하세요.
 - Amazon EKS 콘솔 또는 `eks update-nodegroup-config`를 사용하여 규모 조정 구성을 업데이트하고 원하는 노드 그룹 크기를 설정합니다.
 - Kubernetes Cluster Autoscaler 사용 자세한 내용은 [AWS의 Cluster Autoscaler](#)를 참조하세요.
5. 이제 노드 그룹이 워크로드와 포드를 예약할 준비가 되었습니다.
6. 예약이 종료되기 전에 포드가 원활하게 드레이닝될 수 있도록 Amazon EKS는 예약된 규모 조정 정책을 사용하여 노드 그룹 크기를 0으로 스케일 다운합니다. 이 예약된 규모 조정은 Amazon EKS Node Group Capacity Scaledown Before Reservation End라는 이름으로 설정됩니다. 이 작업을 편집하거나 삭제하지 않는 것이 좋습니다.

Amazon EC2는 예약 종료 시간 30분 전에 인스턴스를 종료하기 시작합니다. 결과적으로 Amazon EKS는 포드를 안전하고 원활하게 제거하기 위해 예약 종료 40분 전에 노드 그룹에 예약된 스케일 다운을 설정합니다.

ML용 용량 블록을 사용하여 자체 관리형 노드 생성

기계 학습용 용량 블록(ML)을 사용하면 향후 날짜에 GPU 인스턴스를 예약하여 단기간의 ML 워크로드를 지원할 수 있습니다. 자세한 내용을 알아보려면 Amazon EC2 - Linux 인스턴스용 사용 설명서의 [ML용 용량 블록](#)을 참조하세요.

고려 사항

Important

- 용량 블록은 특정 Amazon EC2 인스턴스 유형 및 AWS 리전에서만 사용할 수 있습니다. 호환성 정보는 Amazon EC2 - Linux 인스턴스용 사용 설명서의 [용량 블록 작업 사전 조건](#)을 참조하세요.
- 현재 용량 블록은 Karpenter에서 사용할 수 없습니다.

- 용량 예약이 활성화되기 전에 자체 관리형 노드 그룹을 생성하는 경우 원하는 용량을 0으로 설정합니다.
- 노드를 부드럽게 드레인하는 데 충분한 시간을 할애하려면 용량 블록 예약 종료 시각 30분 전까지 0으로 규모를 조정하도록 스케일링을 예약하는 것이 좋습니다.
- 포드가 정상적으로 드레이닝되도록 하려면 예제 단계에 설명된 대로 AWS 노드 종료 처리기를 설정하는 것이 좋습니다.

자체 관리형 노드가 있는 용량 블록 사용

Amazon EKS와 함께 용량 블록을 사용하여 자체 관리형 노드를 프로비저닝하고 규모를 조정할 수 있습니다. 다음 단계는 일반적인 예제 개요를 제공합니다. AWS CloudFormation 템플릿 예제에서 프로덕션 워크로드에 필요한 모든 측면을 다루지는 않습니다. 일반적으로 부트스트래핑 스크립트를 사용하여 노드를 클러스터에 조인하고, Amazon EKS 가속 AMI를 지정하며, 클러스터에 조인하기 위한 적절한 인스턴스 프로필을 지정해야 합니다. 자세한 내용은 [the section called “Amazon Linux”](#) 단원을 참조하십시오.

1. 워크로드에 적용할 수 있는 시작 템플릿을 생성합니다. 자세한 내용은 Amazon EC2 Auto Scaling 사용 설명서의 [Use Capacity Blocks for machine learning workloads](#)를 참조하세요.

LaunchTemplateData에 다음이 포함되어 있는지 확인합니다.

- MarketType를 "capacity-block"로 설정한 InstanceMarketOptions
- CapacityReservationId가 용량 블록으로 설정된 CapacityReservationSpecification: CapacityReservationTarget(예: `cr-02168da1478b509e0`)
- 해당 `iam-instance-profile-arn`으로 설정된 Arn이 있는 IamInstanceProfile
- ImageId 해당 `image-id`로 설정
- InstanceType이 용량 블록을 지원하는 인스턴스 유형으로 설정(예: `p5.48xlarge`)
- SecurityGroupIds는 해당 ID로 설정합니다(예: `sg-05b1d815d1EXAMPLE`).
- 자체 관리형 노드 그룹을 위해 UserData가 해당 `user-data`로 설정

다음은 용량 블록을 대상으로 하는 시작 템플릿을 생성하는 CloudFormation 템플릿의 발췌문입니다.

```
NodeLaunchTemplate:
  Type: "aws::EC2::LaunchTemplate"
```

```

Properties:
  LaunchTemplateData:
    InstanceMarketOptions:
      MarketType: "capacity-block"
    CapacityReservationSpecification:
      CapacityReservationTarget:
        CapacityReservationId: "cr-02168da1478b509e0"
    IamInstanceProfile:
      Arn: iam-instance-profile-arn
    ImageId: image-id
    InstanceType: p5.48xlarge
    KeyName: key-name
    SecurityGroupIds:
      - sg-05b1d815d1EXAMPLE
    UserData: user-data

```

용량 블록은 영역 단위이므로 예약이 이루어진 가용 영역의 서브넷을 전달해야 합니다.

2. 시작 템플릿을 사용하여 자체 관리형 노드 그룹을 생성합니다. 용량 예약이 활성화되기 전에 이 작업을 수행하는 경우 원하는 용량을 0으로 설정합니다. 노드 그룹을 생성할 때는 용량이 예약된 가용 영역에 해당하는 서브넷만 지정하도록 해야 합니다.

다음은 워크로드에 적용할 수 있는 템플릿을 생성할 때 참조할 수 있는 샘플 CloudFormation 템플릿입니다. 이 예제에서는 이전 단계에 표시된 `AWS::Amazon::EC2::LaunchTemplate` 리소스 중 `LaunchTemplateId` 및 `Version`을 가져옵니다. 또한 동일한 템플릿의 다른 곳에 선언된 `DesiredCapacity`, `MaxSize`, `MinSize` 및 `VPCZoneIdentifier`의 값도 가져옵니다.

```

NodeGroup:
  Type: "AWS::AutoScaling::AutoScalingGroup"
  Properties:
    DesiredCapacity: !Ref NodeAutoScalingGroupDesiredCapacity
    LaunchTemplate:
      LaunchTemplateId: !Ref NodeLaunchTemplate
      Version: !GetAtt NodeLaunchTemplate.LatestVersionNumber
    MaxSize: !Ref NodeAutoScalingGroupMaxSize
    MinSize: !Ref NodeAutoScalingGroupMinSize
    VPCZoneIdentifier: !Ref Subnets
    Tags:
      - Key: Name
        PropagateAtLaunch: true
        Value: !Sub ${ClusterName}-${NodeGroupName}-Node
      - Key: !Sub kubernetes.io/cluster/${ClusterName}
        PropagateAtLaunch: true

```

Value: owned

3. 노드 그룹이 성공적으로 생성되면 생성된 노드 그룹의 NodeInstanceRole을(를) 기록해야 합니다. 노드 그룹이 규모를 조정할 때 새 노드가 클러스터에 가입하고 Kubernetes가 노드를 인식할 수 있도록 하려면 이 정보가 필요합니다. 자세한 내용은 [자체 관리형 Amazon Linux 노드 생성](#)의 AWS Management Console 지침을 참조하세요.
4. 용량 블록 예약 시간에 맞춰 Auto Scaling 그룹에 대해 예약된 규모 조정 정책을 생성하는 것이 좋습니다. 자세한 정보는 Amazon EC2 Auto Scaling 사용 설명서의 [Amazon EC2 Auto Scaling 예정 스케일링](#)을 참조하세요.

용량 블록 종료 시간 30분 전까지 예약한 모든 인스턴스를 사용할 수 있습니다. 해당 시점에 계속 실행 중인 인스턴스는 종료되기 시작합니다. 노드를 부드럽게 드레인하는 데 충분한 시간을 할애하려면 용량 블록 예약 종료 시각 30분 전까지 0으로 규모를 조정하도록 스케일링을 예약하는 것이 좋습니다.

용량 예약이 Active이(가)될 때마다 수동으로 규모를 조정하려면 용량 블록 예약을 시작할 때 Auto Scaling 그룹의 원하는 용량을 업데이트해야 합니다. 또한 용량 블록 예약 종료 시각 30분 전까지 수동으로 스케일 다운해야 합니다.

5. 이제 노드 그룹이 워크로드와 포드를 예약할 준비가 되었습니다.
6. 포드를 부드럽게 드레이닝하려면 AWS 노드 종료 핸들러를 설정하는 것이 좋습니다. 이 핸들러는 EventBridge를 사용하여 Amazon EC2 Auto Scaling에서 'ASG 스케일 인' 수명 주기 이벤트를 감시하고 인스턴스가 사용할 수 없게 되기 전에 Kubernetes 컨트롤 플레인인 필요한 조치를 취하도록 할 수 있습니다. 그렇지 않으면 포드 및 Kubernetes 객체가 보류 상태에서 멈추게 됩니다. 자세한 내용은 GitHub의 [AWS 노드 종료 핸들러](#)를 참조하세요.

노드 종료 핸들러를 설정하지 않은 경우 30분 전에 수동으로 포드의 드레인을 시작하여 부드럽게 드레이닝할 수 있는 충분한 시간을 확보하는 것이 좋습니다.

특정 노드에서 포드가 예약되지 않도록 방지

GPU와 같은 특수 프로세서가 있는 노드는 더 많은 표준 시스템에서 실행되는 노드보다 실행 비용이 더 많이 들 수 있습니다. 따라서 특수 하드웨어가 필요하지 않은 워크로드가 해당 노드에 배포되지 않도록 해당 노드를 보호하는 것이 좋습니다. 이를 위한 한 가지 방법은 테인트를 사용하는 것입니다.

Amazon EKS는 관리형 노드 그룹을 통해 Kubernetes 테인트 구성을 지원합니다. 테인트 및 허용 오차는 함께 작동하여 포드가 부적절한 노드에 예약되지 않도록 합니다. 하나 이상의 테인트를 노드에 적용할 수 있습니다. 이것은 노드가 테인트를 용납하지 않는 포드를 허용해서는 안 된다는 것을 나타냅니다.

다. 허용 오차는 포드에 적용되며 일치하는 테인트가 있는 노드에 포드를 예약할 수는 있지만 필요하지 않습니다. 자세한 내용은 Kubernetes 설명서의 [테인트와 허용 오차](#)를 참조하세요.

AWS Management Console을 사용하거나 Amazon EKS API를 통해 Kubernetes 노드 테인트를 신규 및 기존 관리형 노드 그룹에 적용할 수 있습니다.

- AWS Management Console을 사용하여 테인트가 있는 노드 그룹을 생성하는 방법에 대한 자세한 내용은 [the section called “생성”](#)를 참조하세요.
- 다음은 AWS CLI를 사용하여 테인트가 있는 노드 그룹을 생성하는 예제입니다.

```
aws eks create-nodegroup \
  --cli-input-json '
{
  "clusterName": "my-cluster",
  "nodegroupName": "node-taints-example",
  "subnets": [
    "subnet-1234567890abcdef0",
    "subnet-abcdef01234567890",
    "subnet-021345abcdef67890"
  ],
  "nodeRole": "arn:aws:iam::111122223333:role/AmazonEKSNodeRole",
  "taints": [
    {
      "key": "dedicated",
      "value": "gpuGroup",
      "effect": "NO_SCHEDULE"
    }
  ]
}'
```

자세한 내용 및 사용 예는 Kubernetes 참조 설명서의 [taint](#)를 참조하세요.

Note

- UpdateNodegroupConfig API를 사용하여 노드 그룹을 생성한 후 테인트를 업데이트할 수 있습니다.
- 테인트 키는 문자 또는 숫자로 시작해야 합니다. 문자, 숫자, 하이픈(-), 마침표(.) 또는 밑줄(_)을 포함할 수 있습니다. 최대 63자까지 가능합니다.

- 경우에 따라, 테인트 키는 DNS 하위 도메인 접두사와 단일 /로 시작할 수 있습니다. DNS 하위 도메인 접두사로 시작하는 경우 253자가 될 수 있습니다.
- 이 값은 선택 사항이며 문자 또는 숫자로 시작해야 합니다. 문자, 숫자, 하이픈(-), 마침표(.) 또는 밑줄(_)을 포함할 수 있습니다. 최대 63자까지 가능합니다.
- Kubernetes를 직접 사용하거나 AWS Management Console을 사용하는 경우 테인트 효과는 NoSchedule, PreferNoSchedule 또는 NoExecute여야 합니다. AWS CLI나 API를 직접 사용하는 경우 오염 효과는NO_SCHEDULEPREFER_NO_SCHEDULE, 또는 NO_EXECUTE 이어야 합니다.
- 한 노드 그룹에 대해 최대 50개의 테인트가 허용됩니다.
- 관리형 노드 그룹을 사용하여 생성한 테인트를 노드에서 수동으로 제거하는 경우 Amazon EKS는 해당 테인트를 노드에 다시 추가하지 않습니다. 이는 관리형 대상 노드 그룹 구성에 테인트가 지정된 경우에도 마찬가지입니다.

[aws eks update-nodegroup-config](#) AWS CLI 명령을 사용하여 관리형 노드 그룹의 테인트를 추가, 제거 또는 교체할 수 있습니다.

Elastic Fabric Adapter를 사용한 Amazon EKS에서 기계 학습 훈련 실행

이 주제에서는 Amazon EKS 클러스터에 배포된 포드와 Elastic Fabric Adapter(EFA)를 통합하는 방법에 대해 설명합니다. Elastic Fabric Adapter(EFA)는 Amazon EC2 인스턴스용 네트워크 인터페이스로, AWS에서 대규모로 높은 수준의 노드 간 통신을 필요로 하는 애플리케이션을 실행할 수 있도록 지원합니다. 맞춤형으로 구축된 운영 체제 바이패스 하드웨어 인터페이스는 인스턴스 간 통신의 성능을 향상시켜 애플리케이션을 확장하는 데 매우 중요합니다. EFA를 사용하면 메시지 전달 인터페이스(MPI)를 사용하는 고성능 컴퓨팅(HPC) 애플리케이션과 NVIDIA Collective Communications Library(NCCL)를 사용하는 기계 학습(ML) 애플리케이션을 수천 개의 CPU 또는 GPU로 확장할 수 있습니다. 따라서 온프레미스 HPC 클러스터의 애플리케이션 성능을 AWS 클라우드의 온디맨드 탄력성과 유연성을 통해 실현할 수 있습니다. Amazon EKS 클러스터에서 실행되는 애플리케이션과 EFA를 통합하면 클러스터에 인스턴스를 추가하지 않고도 대규모 분산 훈련 워크로드를 완료하는 데 걸리는 시간을 줄일 수 있습니다. EFA에 대한 자세한 내용은 [Elastic Fabric Adapter\(EFA\)](#) 섹션을 참조하세요.

EFA를 사용한 인스턴스 유형

AWS EFA Kubernetes 디바이스 플러그인은 EFA가 있는 모든 Amazon EC2 인스턴스 유형을 지원합니다. EFA가 있는 모든 인스턴스 유형 목록을 보려면 Amazon EC2 사용 설명서의 [지원되는 인스턴스 유형](#)을 참조하세요. 그러나 ML 애플리케이션을 빠르게 실행하려면 인스턴스에 EFA 외에 nVidia GPU, [AWS Inferentia](#) 칩 또는 [AWS Trainium](#) 칩과 같은 하드웨어 가속 칩이 있는 것이 좋습니다. 하드웨어

가속 칩과 EFA가 있는 인스턴스 유형 목록을 보려면 Amazon EC2 사용 설명서의 [가속 컴퓨팅](#)을 참조하세요.

인스턴스 유형을 비교하여 선택할 때는 해당 인스턴스 유형에 사용할 수 있는 EFA 네트워크 카드 수뿐만 아니라 가속기 카드 수, CPU 사용량, 메모리 크기도 고려하세요. 네트워크 카드당 최대 1개의 EFA를 할당할 수 있습니다. EFA는 네트워크 인터페이스 수 계산에 포함됩니다. EFA가 있는 각 인스턴스 유형에 사용할 수 있는 EFA 수를 확인하려면 Amazon EC2 사용 설명서의 [네트워크 카드](#) 목록을 참조하세요.

EFA 및 EFA-only 인터페이스

Elastic Fabric Adapter(EFA)는 AWS Scalable Reliable Datagram(SRD) 프로토콜로 구동되는 Elastic Network Adapter(ENA)와 OS 바이패스 인터페이스의 기능을 결합하는 네트워크 인터페이스입니다. EFA 기능을 사용하면 애플리케이션이 지연 시간이 짧은 전송을 위해 하드웨어와 직접 통신할 수 있습니다. EFA-only 인터페이스를 사용하여 EFA 기능에만 액세스하도록 선택하여 동일한 가용 영역 내의 인터페이스로 통신을 제한할 수 있습니다.

EFA 전용 인터페이스를 가질 수 있는 노드를 생성하려면 사용자 지정 EC2 시작 템플릿을 사용하고 `InterfaceType`을 `efa-only`로 설정해야 합니다. 사용자 지정 시작 템플릿에서는 네트워크 카드 0를 EFA 전용 인터페이스로 설정할 수 없습니다. EC2 인스턴스의 기본 네트워크 카드 및 네트워크 인터페이스이기 때문입니다. EFA 전용 인터페이스의 경우 VPC CNI 버전 1.18.5 이상을 사용해야 합니다. Amazon Linux 2를 사용하는 경우 EFA 전용 인터페이스의 ami 버전은 v20240928 이상이어야 합니다.

다음 절차는 nVidia GPU 및 EFA 인터페이스가 있는 노드가 있는 `eksctl`로 EKS 클러스터를 생성하는 방법을 안내합니다. `eksctl`를 사용하여 EFA-only 인터페이스를 사용하는 노드 및 노드 그룹을 생성할 수 없습니다.

사전 조건

- 기존 Amazon EKS 클러스터. 기존 클러스터가 없는 경우 [시작](#)을 사용하여 클러스터를 생성합니다. 노드를 배포할 수 있는 충분한 IP 주소가 있는 프라이빗 서브넷이 하나 이상 있는 VPC에 클러스터를 배포해야 합니다. 프라이빗 서브넷에는 NAT 게이트웨이와 같은 외부 디바이스에서 제공하는 아웃바운드 인터넷 액세스가 있어야 합니다.

`eksctl`을 사용하여 노드 그룹을 생성하려는 경우 `eksctl`이 클러스터를 생성할 수도 있습니다.

- 장치에 설치 및 구성된 AWS 명령줄 인터페이스(AWS CLI)의 버전 2.12.3 이상 또는 버전 1.27.160 이상 또는 AWS CloudShell. 현재 버전을 확인하려면 `aws --version | cut -d / -f2 | cut -d ' ' -f1`을 사용합니다. yum, apt-get 또는 macOS용 Homebrew와 같은 패키지

지 관리자는 최신 버전의 AWS CLI 이전에 나온 버전이 몇 가지 있을 때도 있습니다. 최신 버전을 설치하려면 AWS 명령줄 인터페이스 사용 설명서에서 [설치](#) 및 [aws config를 사용하여 빠른 구성](#)을 참조하세요. AWS CloudShell에 설치된 AWS CLI 버전도 최신 버전보다 여러 버전 이전일 수도 있습니다. 업데이트하려면 AWS CloudShell 사용 설명서의 [홈 디렉터리에 AWS CLI 설치하기](#)를 참조하세요.

- kubectl 명령줄 도구는 장치 또는 AWS CloudShell에 설치됩니다. 버전은 클러스터의 Kubernetes 버전과 동일하거나 최대 하나 이전 또는 이후의 마이너 버전일 수 있습니다. 예를 들어, 클러스터 버전이 1.29인 경우 kubectl 버전 1.28, 1.29 또는 1.30를 함께 사용할 수 있습니다. kubectl을 설치하거나 업그레이드하려면 [the section called “kubectl 및 eksctl 설정”](#) 부분을 참조하세요.
- p4d 또는 p5와 같이 여러 Elastic Fabric Adapter를 지원하는 워커 노드를 시작하기 전에 Kubernetes용 Amazon VPC CNI 플러그인 버전 1.7.10 이상이 설치되어 있어야 합니다. Kubernetes용 Amazon VPC CNI 플러그인 버전 업데이트에 대한 자세한 내용은 [the section called “Amazon VPC CNI”](#) 섹션을 참조하세요.

Important

Kubernetes에 EFA를 도입하는 데 필요한 중요한 고려 사항은 Huge Page를 클러스터에서 리소스로 구성하고 관리하는 것입니다. 자세한 내용은 Kubernetes 문서의 [대용량 페이지 관리 \(Manage Huge Pages\)](#)를 참조하세요. EFA 드라이버가 설치된 Amazon EC2 인스턴스는 5128 2MiB Huge Page를 미리 할당하며 이를 작업 사양에 따라 사용할 리소스로 요청할 수 있습니다.

노드 그룹 생성

다음 절차를 따르면 EFA 인터페이스 및 GPUDirect RDMA를 사용하여 p4d.24xlarge 지원 노드 그룹으로 노드 그룹을 생성하고, EFA를 사용하여 다중 노드 NCCL 성능에 대한 예시 NVIDIA Collective Communications Library(NCCL) 테스트를 실행하는 데 도움이 됩니다. 이 예제는 EFA를 사용하는 Amazon EKS에 대한 분산 딥 러닝 훈련용 템플릿으로 사용할 수 있습니다.

1. 노드를 배포하려는 AWS 리전에서 사용할 수 있는 EFA를 지원하는 Amazon EC2 인스턴스 유형을 결정합니다. **region-code**를 노드 그룹을 배포할 AWS 리전으로 바꿉니다.

```
aws ec2 describe-instance-types --region region-code \
  --filters Name=network-info.efa-supported,Values=true \
  --query "InstanceTypes[*].[InstanceType]" --output text
```

노드를 배포할 때 배포하려는 인스턴스 유형은 클러스터가 속한 AWS 리전에서 사용할 수 있어야 합니다.

2. 배포하려는 인스턴스 유형이 사용 가능한 가용 영역을 결정합니다. 이 자습서에서는 p5.48xlarge 인스턴스 유형이 사용되며 이전 단계에서 지정한 AWS 리전에 대한 출력으로 반환되어야 합니다. 프로덕션 클러스터에 노드를 배포할 때 **p5.48xlarge**를 이전 단계에서 반환된 인스턴스 유형으로 바꿉니다.

```
aws ec2 describe-instance-type-offerings --region region-code \
  --location-type availability-zone --filters Name=instance-
type,Values=p4d.24xlarge,p5.48xlarge \
  --query 'InstanceTypeOfferings[*].Location' --output text
```

예제 출력은 다음과 같습니다.

```
us-west-2a    us-west-2c    us-west-2b
```

이후 단계에서 사용하기 위해 반환된 가용 영역을 기록해 둡니다. 클러스터에 노드를 배포할 때 VPC에는 출력에 반환된 가용 영역 중 하나에 사용 가능한 IP 주소가 있는 서브넷이 있어야 합니다.

3. eksctl을 사용하여 노드 그룹을 생성합니다. 장치에 설치된 eksctl 명령줄 도구의 버전 0.210.0 이상 또는 AWS CloudShell이 필요합니다. eksctl을 설치 또는 업그레이드하려면 eksctl 설명서에서 [설치](#)를 참조하세요.
 - a. 다음 콘텐츠를 **efa-cluster.yaml**라는 파일에 복사합니다. **## #**을 사용자의 값으로 바꿉니다. **p5.48xlarge**를 다른 인스턴스로 바꿀 수 있지만, 그렇게 할 경우 availabilityZones의 값이 1단계에서 인스턴스 유형에 대해 반환된 가용 영역이어야 합니다.

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-efa-cluster
  region: region-code
  version: "1.XX"

iam:
  withOIDC: true

availabilityZones: ["us-west-2a", "us-west-2c"]
```

```
managedNodeGroups:
  - name: my-efa-ng
    instanceType: p5.48xlarge
    minSize: 1
    desiredCapacity: 2
    maxSize: 3
    availabilityZones: ["us-west-2a"]
    volumeSize: 300
    privateNetworking: true
    efaEnabled: true
```

b. 기존 클러스터에 관리형 노드 그룹을 생성합니다.

```
eksctl create nodegroup -f efa-cluster.yaml
```

기존 클러스터가 없는 경우 다음 명령을 실행하여 클러스터와 노드 그룹을 생성할 수 있습니다.

```
eksctl create cluster -f efa-cluster.yaml
```

Note

이 예제에 사용된 인스턴스 유형에는 GPU가 있으므로 Amazon Linux 2를 사용할 때 eksctl은 각 인스턴스에 NVIDIA Kubernetes 디바이스 플러그인을 자동으로 설치합니다. NVIDIA 디바이스 플러그인이 Bottlerocket의 EKS NVIDIA 버전에 내장되어 있으므로 Bottlerocket의 경우 이 작업이 필요하지 않습니다. 노드 그룹 구성에서 efaEnabled가 true로 설정된 경우 eksctl은 노드에 EFA 디바이스 플러그인을 자동으로 배포합니다.

EFA와 함께 Bottlerocket 사용

Bottlerocket AMI 버전 1.28.0 이상에는 EFA에 대한 공식 지원이 포함되어 있습니다. EFA 지원 노드에 Bottlerocket을 사용하려면 구성에 amiFamily: Bottlerocket을 지정합니다. 사용자 지정 AMI ID를 사용해야 하는 경우 nodeGroups 대신 표준 managedNodeGroups를 사용해야 합니다.

다음은 구성의 예입니다.

```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
```

```

name: my-efa-bottlerocket-cluster
region: region-code
version: "1.XX"

iam:
  withOIDC: true

availabilityZones: ["us-west-2a", "us-west-2c"]

managedNodeGroups:
  - name: my-efa-bottlerocket-ng
    instanceType: p5.48xlarge
    minSize: 1
    desiredCapacity: 2
    maxSize: 3
    availabilityZones: ["us-west-2a"]
    volumeSize: 300
    privateNetworking: true
    efaEnabled: true
    amiFamily: Bottlerocket
    bottlerocket:
      enableAdminContainer: true
      settings:
        kernel:
          sysctl:
            "vm.nr_hugepages": "3000" # Configures 3000 * 2Mi = 6000Mi hugepages

```

위의 `vm.nr_hugepages` `sysctl` 설정은 2Mi HugePages 수를 구성합니다. 이 예제에서 3000은 $3000 * 2\text{Mi} = 6000\text{Mi}$ HugePages를 의미합니다.

EFA 디바이스 플러그인 설치 확인

`efaEnabled: true`로 노드 그룹을 생성하면 `eksctl`이 자동으로 EFA Kubernetes 디바이스 플러그인을 배포합니다. 디바이스 플러그인이 설치되어 제대로 작동하는지 확인할 수 있습니다.

1. DaemonSet 상태를 확인합니다.

```
kubectl get daemonsets -n kube-system
```

샘플 출력:

NAME	DESIRED	CURRENT	READY	UP-TO-DATE
AVAILABLE NODE SELECTOR AGE				
aws-efa-k8s-device-plugin-daemonset	2	2	2	2
<none> 6m16s				
...				

여기에서는 2개의 노드에서 EFA 디바이스 플러그인 DaemonSet가 실행되고 있습니다. 둘 다 READY 및 AVAILABLE입니다.

2. 다음으로 DaemonSet에서 생성된 포드를 확인합니다.

```
kubectl get pods -n kube-system -l name=aws-efa-k8s-device-plugin
```

샘플 출력:

NAME	READY	STATUS	RESTARTS	AGE
aws-efa-k8s-device-plugin-daemonset-d68bs	1/1	Running	0	6m16s
aws-efa-k8s-device-plugin-daemonset-w4l8t	1/1	Running	0	6m16s

EFA 디바이스 플러그인 포드가 실행 중 상태이므로 플러그인이 성공적으로 배포되고 작동하고 있는 것입니다.

3. 리소스 등록을 확인합니다.

노드를 설명하여 `vpc.amazonaws.com/efa` 리소스가 kubelet에 등록되었는지 확인할 수 있습니다.

```
kubectl describe nodes
```

EFA 리소스가 제대로 등록되면 노드의 용량 및 할당 가능한 리소스 아래에 해당 리소스가 나열됩니다. 예시:

```
Capacity:
...
vpc.amazonaws.com/efa: 4
Allocatable:
...
vpc.amazonaws.com/efa: 4
```

이 출력은 노드가 EFA 리소스를 인식하여 이를 요청하는 포드에서 사용할 수 있음을 확인합니다.

(선택사항) EFA의 성능 테스트

EFA 설정을 테스트하는 것이 좋습니다. GitHub의 [aws-samples/awesome-distributed-training](#) 리포지토리에서 [NCCL Tests](#)를 사용할 수 있습니다. [NCCL Tests](#)는 NVIDIA Collective Communication Library를 사용하여 네트워크 성능을 평가합니다. 다음 단계는 Amazon EKS에서 NCCL 테스트를 제출합니다.

1. Kubeflow MPI 연산자 배포

NCCL 테스트를 위해 Kubeflow MPI 연산자를 적용할 수 있습니다. MPI 연산자를 사용하면 Kubernetes에서 Allreduce 스타일의 분산 훈련을 쉽게 실행할 수 있습니다. 자세한 내용은 GitHub에서 [MPI 연산자](#)를 참조하세요.

2. GPUDirectRDMA/EFA 검증을 위한 다중 노드 NCCL 성능 테스트 실행

EFA를 통한 GPUDirectRDMA를 사용하여 NCCL 성능을 검증하려면 표준 NCCL 성능 테스트를 실행합니다. 자세한 내용은 GitHub에서 공식 [NCCL-Tests](#) 리포지토리를 참조하세요.

2개 노드 NCCL 성능 테스트를 실행하려면 다음 단계를 완료합니다. 예제 NCCL 테스트 작업에서 각 작업자는 8개의 GPU, 5210Mi의 hugepages-2Mi, 4개의 EFA, 8000Mi 메모리를 요청합니다. 이는 실질적으로 각 작업자가 p5.48xlarge 인스턴스의 리소스를 모두 사용한다는 것을 의미합니다.

a. MPIJob 매니페스트 생성:

다음은 `nccl-tests.yaml`이라는 파일에 복사합니다.

```
apiVersion: kubeflow.org/v2beta1
kind: MPIJob
metadata:
  name: nccl-tests
spec:
  runPolicy:
    cleanPodPolicy: Running
    backoffLimit: 20
  slotsPerWorker: 8
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
```



```

spec:
  restartPolicy: OnFailure
  containers:
  - image: public.ecr.aws/hpc-cloud/nccl-tests:latest
    imagePullPolicy: IfNotPresent
    name: test-nccl-launcher
    env:
      - name: PATH
        value: $PATH:/opt/amazon/efa/bin:/usr/bin
      - name: LD_LIBRARY_PATH
        value: /opt/amazon/openmpi/lib:/opt/nccl/build/lib:/opt/amazon/efa/
lib:/opt/aws-ofi-nccl/install/lib:/usr/local/nvidia/lib:$LD_LIBRARY_PATH
      - name: NCCL_DEBUG
        value: INFO
      - name: NCCL_BUFFSIZE
        value: '8388608'
      - name: NCCL_P2P_NET_CHUNKSIZE
        value: '524288'
      - name: NCCL_TUNER_PLUGIN
        value: /opt/aws-ofi-nccl/install/lib/libnccl-ofi-tuner.so
    command:
      - /opt/amazon/openmpi/bin/mpirun
      - --allow-run-as-root
      - --tag-output
      - -np
      - "16"
      - -N
      - "8"
      - --bind-to
      - none
      - -x
      - PATH
      - -x
      - LD_LIBRARY_PATH
      - -x
      - NCCL_DEBUG=INFO
      - -x
      - NCCL_BUFFSIZE
      - -x
      - NCCL_P2P_NET_CHUNKSIZE
      - -x
      - NCCL_TUNER_PLUGIN
      - --mca
      - pml

```

```

- ^cm,ucx
- --mca
- btl
- tcp,self
- --mca
- btl_tcp_if_exclude
- lo,docker0,veth_def_agent
- /opt/nccl-tests/build/all_reduce_perf
- -b
- "8"
- -e
- "16G"
- -f
- "2"
- -g
- "1"
- -c
- "1"
- -n
- "100"

```

Worker:

replicas: 2

template:

spec:

nodeSelector:

node.kubernetes.io/instance-type: "p5.48xlarge"

containers:

- image: public.ecr.aws/hpc-cloud/nccl-tests:latest

imagePullPolicy: IfNotPresent

name: nccl-tests-worker

volumeMounts:

- name: shm

mountPath: /dev/shm

resources:

limits:

nvidia.com/gpu: 8

hugepages-2Mi: 5120Mi

vpc.amazonaws.com/efa: 32

memory: 32000Mi

requests:

nvidia.com/gpu: 8

hugepages-2Mi: 5120Mi

vpc.amazonaws.com/efa: 32

memory: 32000Mi

```
volumes:
- name: shm
  hostPath:
    path: /dev/shm
```

b. NCCL-test MPIJob을 적용합니다:

매니페스트를 적용하여 MPIJob을 제출합니다. 그러면 두 p5.48xlarge Amazon EC2 인스턴스가 생성됩니다.

```
kubectl apply -f nccl-tests.yaml
```

예제 출력은 다음과 같습니다.

```
mpijob.kubeflow.org/nccl-tests created
```

c. 작업이 포드를 시작했는지 확인

실행 중인 포드를 봅니다.

```
kubectl get pods
```

예제 출력은 다음과 같습니다.

NAME	READY	STATUS	RESTARTS	AGE
nccl-tests-launcher-nbql9	0/1	Init:0/1	0	2m49s
nccl-tests-worker-0	1/1	Running	0	2m49s
nccl-tests-worker-1	1/1	Running	0	2m49s

MPI 연산자는 런처 포드와 작업자 포드 2개(각 노드에 하나씩)를 생성합니다.

d. 로그를 사용하여 작업이 성공적으로 실행되고 있는지 확인

nccl-tests-launcher 포드에 대한 로그를 확인합니다. *nbql9*를 출력된 값과 바꿉니다.

```
kubectl logs -f nccl-tests-launcher-nbql9
```

테스트가 성공적으로 완료되면 Nvidia Collective Communication Library를 사용하는 애플리케이션을 배포할 수 있습니다.

기계 학습을 위해 Amazon EKS와 함께 AWS Inferentia 인스턴스 사용

이 주제에서는 [Amazon EC2 Inf1](#) 인스턴스를 실행하는 노드가 있는 Amazon EKS 클러스터를 생성하고 샘플 애플리케이션을 배포(선택사항)하는 방법을 설명합니다. Amazon EC2 Inf1 인스턴스는 [AWS Inferentia](#) 칩으로 구동됩니다. 이 칩은 클라우드에서 고성능의 가장 저렴한 추론을 제공하기 위해 AWS에서 사용자 지정으로 구축됩니다. 기계 학습 모델은 Inferentia 칩의 기계 학습 추론 성능을 최적화하는 컴파일러, 런타임 및 프로파일링 도구로 구성된 특수 소프트웨어 개발 키트(SDK)인 [AWS Neuron](#)을 사용하여 컨테이너에 배포됩니다. AWS Neuron은 TensorFlow, PyTorch, MXNet과 같은 인기 있는 기계 학습 프레임워크를 지원합니다.

Note

Neuron 디바이스 논리적 ID는 연속적이어야 합니다. Neuron 디바이스를 여러 개 요청하는 포드가 `inf1.6xlarge` 또는 `inf1.24xlarge` 인스턴스 유형(둘 이상의 Neuron 디바이스가 있음)에서 예약된 경우 Kubernetes 스케줄러가 불연속적인 디바이스 ID를 선택하면 해당 포드가 시작되지 않습니다. 자세한 내용은 GitHub의 [Device logical IDs must be contiguous](#)를 참조하십시오.

사전 조건

- `eksctl`이 컴퓨터에 설치되어 있어야 합니다. 설치되지 않은 경우 `eksctl` 설명서에서 [설치](#)를 참조하세요.
- `kubectl`이 컴퓨터에 설치되어 있어야 합니다. 자세한 내용은 [the section called “kubectl 및 eksctl 설정”](#) 섹션을 참조하세요.
- (선택사항) `python3`이 컴퓨터에 설치되어 있어야 합니다. 설치되어 있지 않은 경우 [Python downloads](#)를 참조하여 설치 지침을 확인하세요.

클러스터 생성

1. Inf1 Amazon EC2 인스턴스 노드를 사용하여 클러스터를 생성합니다. `inf1.2xlarge`를 모든 [Inf1 인스턴스 유형](#)으로 바꿀 수 있습니다. `eksctl` 유틸리티는 Inf1 인스턴스 유형을 사용하여 노드 그룹을 시작한다는 것을 감지하고 Amazon EKS 최적화 가속 Amazon Linux AMI 중 하나를 사용하여 노드를 시작합니다.

Note

TensorFlow Serving에서는 [서비스 계정에 IAM 역할을](#) 사용할 수 없습니다.

```
eksctl create cluster \
  --name inferentia \
  --region region-code \
  --nodegroup-name ng-inf1 \
  --node-type inf1.2xlarge \
  --nodes 2 \
  --nodes-min 1 \
  --nodes-max 4 \
  --ssh-access \
  --ssh-public-key your-key \
  --with-oidc
```

Note

출력의 다음 행의 값을 적어 둡니다. 이후 (선택사항) 단계에서 이 값을 사용합니다.

```
[9] adding identity "arn:aws:iam::111122223333:role/eksctl-inferentia-nodegroup-ng-
in-NodeInstanceRole-FI7HIYS3BS09" to auth ConfigMap
```

Inf1 인스턴스로 노드 그룹을 시작하면 eksctl은 AWS Neuron Kubernetes 디바이스 플러그인을 자동으로 설치합니다. 이 플러그인은 Neuron 디바이스를 Kubernetes 스케줄러에 시스템 리소스로 알립니다. 이 스케줄러는 컨테이너에서 요청할 수 있습니다. 기본 Amazon EKS 노드 IAM 정책 외에도 Amazon S3 읽기 전용 액세스 정책이 추가되어 이후 단계에서 다루는 샘플 애플리케이션이 Amazon S3에서 훈련된 모델을 로드할 수 있습니다.

2. 모든 포드가 올바르게 시작되었는지 확인합니다.

```
kubectl get pods -n kube-system
```

간략한 출력:

NAME	READY	STATUS	RESTARTS	AGE

[...]

neuron-device-plugin-daemonset-6djhp	1/1	Running	0	5m
neuron-device-plugin-daemonset-hwjsj	1/1	Running	0	5m

(선택사항) TensorFlow Serving 애플리케이션 이미지 배포

훈련된 모델은 Inferentia 인스턴스에서 배포하기 전에 Inferentia 대상에 컴파일되어야 합니다. 계속하려면, Amazon S3 저장된 [Neuron 최적화 TensorFlow](#) 모델이 필요합니다. 아직 SavedModel이 없는 경우 [Neuron 호환 ResNet50 모델 생성](#)에 대한 자습서를 따르고 그 결과로 생성된 SavedModel을 S3에 업로드합니다. ResNet-50은 이미지 인식 작업에 사용되는 인기 있는 기계 학습 모델입니다. Neuron 모델 컴파일에 대한 자세한 내용은 AWS 딥러닝 AMI 개발자 가이드에서 [DLAMI를 사용한 AWS Inferentia](#) [칩](#) 섹션을 참조하세요.

샘플 배포 매니페스트는 AWS Deep Learning Containers에서 제공되는 사전 구축된 TensorFlow용 추론 Serving 컨테이너를 관리합니다. 이 컨테이너 내부에는 AWS Neuron 런타임 및 TensorFlow Serving 애플리케이션이 있습니다. Neuron에 최적화된 사전 구축된 Deep Learning Containers의 전체 목록은 GitHub의 [사용 가능한 이미지](#)에서 관리됩니다. 시작 시 DLC가 Amazon S3에서 모델을 가져오고, 저장된 모델로 Neuron TensorFlow Serving을 시작하고, 예측 요청을 기다립니다.

Serving 애플리케이션에 할당된 Neuron 디바이스의 수는 배포 yaml에서 `aws.amazon.com/neuron` 리소스를 변경하여 조정할 수 있습니다. TensorFlow Serving과 Neuron 런타임 간의 통신은 GRPC를 통해 발생하므로 IPC_LOCK 기능을 컨테이너에 전달해야 합니다.

1. [클러스터 생성](#)의 1단계에서 생성한 노드 인스턴스 역할에 AmazonS3ReadOnlyAccess IAM 정책을 추가합니다. 이는 샘플 애플리케이션이 Amazon S3에서 훈련된 모델을 로드하기 위해 필요합니다.

```
aws iam attach-role-policy \
  --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess \
  --role-name eksctl-inferentia-nodegroup-ng-in-NodeInstanceRole-FI7HIYS3BS09
```

2. 다음 콘텐츠를 가진 `rn50_deployment.yaml`이라는 파일을 생성합니다: 원하는 설정과 일치하도록 리전 코드 및 모델 경로를 업데이트합니다. 모델 이름은 클라이언트가 TensorFlow 서버에 요청할 때 식별을 위한 것입니다. 이 예에서는 모델 이름을 사용하여 예측 요청을 보내는 이후 단계에서 사용할 샘플 ResNet50 클라이언트 스크립트와 일치시킵니다.

```
aws ecr list-images --repository-name neuron-rtd --registry-id 790709498068 --region us-west-2
```

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: eks-neuron-test
  labels:
    app: eks-neuron-test
    role: master
spec:
  replicas: 2
  selector:
    matchLabels:
      app: eks-neuron-test
      role: master
  template:
    metadata:
      labels:
        app: eks-neuron-test
        role: master
    spec:
      containers:
        - name: eks-neuron-test
          image: 763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-
neuron:1.15.4-neuron-py37-ubuntu18.04
          command:
            - /usr/local/bin/entrypoint.sh
          args:
            - --port=8500
            - --rest_api_port=9000
            - --model_name=resnet50_neuron
            - --model_base_path=s3://${your-bucket-of-models}/resnet50_neuron/
          ports:
            - containerPort: 8500
            - containerPort: 9000
          imagePullPolicy: IfNotPresent
          env:
            - name: AWS_REGION
              value: "us-east-1"
            - name: S3_USE_HTTPS
              value: "1"
            - name: S3_VERIFY_SSL
              value: "0"
            - name: S3_ENDPOINT
              value: s3.us-east-1.amazonaws.com
```

```

      - name: AWS_LOG_LEVEL
        value: "3"
    resources:
      limits:
        cpu: 4
        memory: 4Gi
        aws.amazon.com/neuron: 1
      requests:
        cpu: "1"
        memory: 1Gi
    securityContext:
      capabilities:
        add:
          - IPC_LOCK

```

3. 모델을 배포합니다.

```
kubectl apply -f rn50_deployment.yaml
```

4. 다음 콘텐츠를 가진 rn50_service.yaml이라는 파일을 생성합니다: HTTP 및 gRPC 포트가 예측 요청을 수락하기 위해 열립니다.

```

kind: Service
apiVersion: v1
metadata:
  name: eks-neuron-test
  labels:
    app: eks-neuron-test
spec:
  type: ClusterIP
  ports:
    - name: http-tf-serving
      port: 8500
      targetPort: 8500
    - name: grpc-tf-serving
      port: 9000
      targetPort: 9000
  selector:
    app: eks-neuron-test
    role: master

```

5. TensorFlow 모델 Serving 애플리케이션을 위한 Kubernetes 서비스를 만듭니다.


```
kubectl apply -f rn50_service.yaml
```

(선택사항) TensorFlow Serving 서비스에 대한 예측

1. 로컬로 테스트하려면 gRPC 포트를 eks-neuron-test 서비스로 전달합니다.

```
kubectl port-forward service/eks-neuron-test 8500:8500 &
```

2. 다음 콘텐츠를 통해 tensorflow-model-server-infer.py라는 Python 스크립트를 생성합니다. 이 스크립트는 서비스 프레임워크인 gRPC를 통해 추론을 실행합니다.

```
import numpy as np
import grpc
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input
from tensorflow_serving.apis import predict_pb2
from tensorflow_serving.apis import prediction_service_pb2_grpc
from tensorflow.keras.applications.resnet50 import decode_predictions

if __name__ == '__main__':
    channel = grpc.insecure_channel('localhost:8500')
    stub = prediction_service_pb2_grpc.PredictionServiceStub(channel)
    img_file = tf.keras.utils.get_file(
        "./kitten_small.jpg",
        "https://raw.githubusercontent.com/aws-labs/mxnet-model-server/master/docs/
images/kitten_small.jpg")
    img = image.load_img(img_file, target_size=(224, 224))
    img_array = preprocess_input(image.img_to_array(img)[None, ...])
    request = predict_pb2.PredictRequest()
    request.model_spec.name = 'resnet50_inf1'
    request.inputs['input'].CopyFrom(
        tf.make_tensor_proto(img_array, shape=img_array.shape))
    result = stub.Predict(request)
    prediction = tf.make_ndarray(result.outputs['output'])
    print(decode_predictions(prediction))
```

3. 스크립트를 실행하여 서비스에 예측을 제출합니다.

```
python3 tensorflow-model-server-infer.py
```

예제 출력은 다음과 같습니다.

```
[[('n02123045', 'tabby', 0.68817204), ('n02127052', 'lynx', 0.12701613),
 ('n02123159', 'tiger_cat', 0.08736559), ('n02124075', 'Egyptian_cat',
 0.063844085), ('n02128757', 'snow_leopard', 0.009240591)]]
```

EKS에서 기계 학습 워크로드를 배포하려는 경우 자습서 참조

EKS에서 기계 학습 플랫폼 및 프레임워크를 설정하는 데 관심이 있다면 이 페이지에서 설명한 자습서를 살펴보세요. 이 자습서에서는 GPU 프로세서를 최대한 활용하기 위한 패턴부터 모델링 도구 선택, 특수 산업을 위한 프레임워크 구축에 이르기까지 모든 것을 다룹니다.

EKS에서 생성형 AI 플랫폼 구축

- [Amazon EKS에 생성형 AI 모델 배포](#)
- [Amazon EKS에서 다중 테넌트 JupyterHub 플랫폼 구축](#)

EKS에서 특수 생성형 AI 프레임워크 실행

- [Amazon EKS의 NVIDIA NeMo 프레임워크로 생성형 AI 분산 훈련 워크로드 가속화](#)
- [Amazon Elastic Kubernetes Service에서 TorchServe 실행](#)

EKS에서 ML용 NVIDIA GPU 성능 극대화

- GPU 공유를 구현하여 EKS 클러스터에 NVIDIA GPU를 효율적으로 사용합니다.

[NVIDIA 시분할 및 가속화된 EC2 인스턴스를 사용하여 Amazon EKS에서 GPU 공유](#)

- 다중 인스턴스 GPU(MIG) 및 NIM 마이크로서비스를 사용하여 EKS 클러스터에서 GPU당 더 많은 포드를 실행합니다.

[Amazon EKS에서 NVIDIA의 다중 인스턴스 GPU\(MIG\)를 사용하여 GPU 사용률 극대화: 성능 향상을 위해 GPU당 더 많은 포드 실행](#)

- [AWS에서 Kubeflow를 사용하여 Kubernetes에 확장 가능한 기계 학습 시스템 구축 및 배포](#)

EKS에서 비디오 인코딩 워크로드 실행

- [Amazon EKS의 컨테이너에서 단편적인 GPU를 사용하여 비디오 콘텐츠 제공](#)

추론 워크로드의 이미지 로드 가속화

- [H2O.ai가 Karpenter 및 Bottlerocket을 사용하여 AI/ML 인프라를 최적화하고 보호하는 방법](#)

ML 워크로드 모니터링

- [AWS 관리형 오픈 소스 서비스를 사용하여 Amazon EKS에서 GPU 워크로드 모니터링](#)
- [Amazon CloudWatch에서 포드 기반 GPU 지표 활성화](#)

EKS에서 ML 사용 관련 공지

- [NVIDIA GPU에 대한 Bottlerocket 지원](#)
- [Amazon Elastic Kubernetes Service의 Amazon EC2 P4d 인스턴스에서 NVIDIA 다중 인스턴스 GPU\(MIG\) 사용](#)
- [NVIDIA GPUDirectRDMA를 사용하여 Amazon Elastic Kubernetes Service에 관리형 P4d 인스턴스 배포](#)

오픈 소스 프로젝트를 사용한 Amazon EKS 기능 확장

이러한 오픈 소스 프로젝트는 Amazon EKS에서 관리하는 클러스터를 비롯하여 AWS 외부에서 실행되는 Kubernetes 클러스터의 기능을 확장합니다.

EKS에 배포된 소프트웨어 지원

Amazon EKS 문서를 검토할 때 절차 및 예제 전체에서 다양한 오픈 소스 도구와 소프트웨어에 대한 참조를 볼 수 있습니다. [Kubernetes 지표 서버](#)와 [Cert Manager](#)가 이러한 도구에 포함됩니다.

배포하기로 선택한 타사 또는 오픈 소스 소프트웨어는 AWS Support 계약의 범위를 벗어난다는 점에 유의하시기 바랍니다. Kubernetes 사용의 이점은 활발한 오픈 소스 커뮤니티입니다. 관련 오픈 소스 커뮤니티 및 프로젝트 유지 관리 담당자와 직접 협력하여 이러한 구성 요소에 대한 적절한 지원 채널을 설정하는 것이 좋습니다. 자세한 내용은 Cloud Native Computing Foundation(CNCF)의 [Graduated and Incubating Projects](#)를 참조하세요.

Kubernetes 에코시스템에는 다양한 수준의 커뮤니티 지원, 응답 시간 및 의도된 사용 사례와 함께 제공되는 수많은 프로젝트와 구성 요소가 포함되어 있습니다. EKS와 함께 이러한 기술을 구현할 때는 각 구성 요소에 대한 지원 매트릭스를 이해해야 합니다.

AWS는 EKS 컨트롤 플레인에 통합되는 오픈 소스 구성 요소를 유지 관리합니다. 여기에는 배포하는 모든 컨테이너 이미지 및 바이너리에 대한 빌드 확인, 취약성 스캔, 검증 테스트 및 패치 관리를 다루는 포괄적인 보안 파이프라인이 포함됩니다. 예를 들어 AWS는 [Kubernetes API 서버](#)를 담당합니다. Kubernetes API 서버는 [Amazon EKS 서비스 수준 계약](#)의 적용을 받습니다. [Amazon Web Services 지원 플랜](#)을 사용하여 Kubernetes API 서버의 문제를 해결하거나 일반 지침을 얻을 수 있습니다.

다양한 Amazon EKS 추가 기능에 대해 제공되는 지원을 주의 깊게 검토해야 합니다. AWS 추가 기능은 AWS에서 완전히 지원하는 유일한 유형의 Amazon EKS 추가 기능입니다. AWS Marketplace 추가 기능은 주로 AWS 파트너가 지원합니다. 커뮤니티 추가 기능은 AWS로부터 기본 수명 주기 지원을 받습니다. 자세한 내용은 [추가 기능 지원](#)을 참조하세요.

유형에 관계없이 모든 EKS 추가 기능은 Marketplace 추가 기능을 포함하여 EKS로부터 기본 수명 주기 지원을 받습니다. 기본 수명 주기 지원에는 추가 기능의 설치 및 제거가 포함됩니다. 사용 가능한 Amazon EKS 추가 기능의 유형 및 관련 지원 수준에 대한 자세한 내용은 [Amazon EKS 추가 기능에 대한 지원 범위](#)를 참조하세요. AWS에서 완전히 지원되는 추가 기능을 보려면 [Amazon Web Services 추가 기능](#)을 참조하세요.

- 보안 사례와 지원 경계에 대한 자세한 내용은 [Amazon EKS의 보안](#)을 참조하세요.

- Amazon EKS 추가 기능을 통해 사용할 수 있는 커뮤니티 및 AWS 마켓플레이스 추가 기능에 대한 자세한 내용은 [EKS 추가 기능 지원](#)을 참조하세요.

관리 도구

Amazon EKS 및 Kubernetes 클러스터용 관련 관리 도구입니다.

eksctl

eksctl은 Amazon EKS에 클러스터를 생성하기 위한 간단한 CLI 도구입니다.

- [프로젝트 URL](#)
- [프로젝트 설명서](#)
- AWS 오픈 소스 블로그: [eksctl: Amazon EKS cluster with one command](#)

Kubernetes용 AWS 컨트롤러

Kubernetes용 AWS 컨트롤러를 사용하여 AWS 리소스를 Kubernetes 클러스터에서 직접 생성하고 관리할 수 있습니다.

- [프로젝트 URL](#)
- AWS 오픈 소스 블로그: [AWS service operator for Kubernetes now available](#)

Flux CD

Flux는 Git을 사용하여 클러스터 구성을 관리하는 데 사용할 수 있는 도구입니다. 클러스터의 운영자를 사용하여 Kubernetes 내부에서 배포를 트리거합니다. 연산자에 대한 자세한 내용은 GitHub의 [OperatorHub.io](#)를 참조하세요.

- [프로젝트 URL](#)
- [프로젝트 설명서](#)

Kubernetes용 CDK

Kubernetes용 CDK(cdk8s)를 사용하면 익숙한 프로그래밍 언어로 Kubernetes 앱과 구성 요소를 정의할 수 있습니다. cdk8s 앱은 표준 Kubernetes 매니페스트로 합성되며 모든 Kubernetes 클러스터에 적용할 수 있습니다.

- [프로젝트 URL](#)
- [프로젝트 설명서](#)
- AWS 컨테이너 블로그: [cdk8s+ 소개: Kubernetes 객체를 위한 인텐트 기반 API](#)

네트워킹

Amazon EKS 및 Kubernetes 클러스터와 관련된 네트워킹 프로젝트입니다.

Kubernetes용 Amazon VPC CNI 플러그 인

Amazon EKS는 Kubernetes용 Amazon VPC CNI 플러그 인을 통한 기본 VPC 네트워킹을 지원합니다. 이 플러그인은 VPC에서 각 포드에 IP 주소를 할당합니다.

- [프로젝트 URL](#)
- [프로젝트 설명서](#)

Kubernetes용 AWS 로드 밸런서 컨트롤러

AWS 로드 밸런서 컨트롤러는 Kubernetes 클러스터의 AWS Elastic Load Balancer를 관리하는 데 유용합니다. AWS Application Load Balancer를 프로비저닝하여 Kubernetes 수신 리소스를 충족합니다. AWS Network Load Balancer를 프로비저닝하여 Kubernetes 서비스 리소스를 충족합니다.

- [프로젝트 URL](#)
- [프로젝트 설명서](#)

ExternalDNS

ExternalDNS는 노출된 Kubernetes 서비스 및 수신을 Amazon Route 53 및 AWS Service Discovery를 포함한 DNS 공급자와 동기화합니다.

- [프로젝트 URL](#)
- [프로젝트 설명서](#)

기계 학습

Amazon EKS 및 Kubernetes 클러스터와 관련된 기계 학습 프로젝트입니다.

Kubeflow

Kubernetes용 Machine Learning 도구 키트입니다.

- [프로젝트 URL](#)
- [프로젝트 설명서](#)
- AWS 오픈 소스 블로그: [Kubeflow on Amazon EKS](#)

오토 스케일링

Amazon EKS 및 Kubernetes 클러스터와 관련된 Auto Scaling 프로젝트입니다.

Cluster autoscaler

Cluster Autoscaler는 CPU 및 메모리 압력에 따라 Kubernetes 클러스터의 크기를 자동으로 조정하는 도구입니다.

- [프로젝트 URL](#)
- [프로젝트 설명서](#)
- Amazon EKS 워크숍: [Cluster Autoscaler](#)

Karpenter

Karpenter는 유연성, 성능, 단순성을 위해 빌드된 Kubernetes Node Autoscaler입니다.

- [프로젝트 URL](#)
- [프로젝트 설명서](#)
- Amazon EKS 워크숍: [Karpenter](#)

Escalator

Escalator는 Kubernetes용 배치 또는 작업에 최적화된 수평 자동 조정기입니다.

- [프로젝트 URL](#)
- [프로젝트 설명서](#)

모니터링

Amazon EKS 및 Kubernetes 클러스터와 관련된 모니터링 프로젝트입니다.

Prometheus

Prometheus는 오픈 소스 시스템 모니터링 및 알림 도구 키트입니다.

- [프로젝트 URL](#)
- [프로젝트 설명서](#)
- Amazon EKS 워크숍: https://eksworkshop.com/intermediate/240_monitoring/

연속 통합/연속 배포

Amazon EKS 및 Kubernetes 클러스터와 관련된 CI/CD 프로젝트입니다.

Jenkins X

Amazon EKS 및 Kubernetes 클러스터에 실행되는 현대적 클라우드 애플리케이션의 CI/CD 솔루션입니다.

- [프로젝트 URL](#)
- [프로젝트 설명서](#)

Amazon EKS 새로운 기능 및 로드맵에 대해 자세히 알아보기

새로운 Amazon EKS 기능에 대해 알아보려면 [AWS의 새로운 기능](#) 페이지로 스크롤합니다. 뿐만 아니라 GitHub에서 [로드맵](#)을 검토할 수도 있습니다. 향후 Amazon EKS를 사용할 방법을 계획할 수 있도록 예정된 기능 및 우선 순위에 대해 알 수 있습니다. 로드맵 우선 순위에 대해 직접 피드백을 제공할 수 있습니다.

문서 기록

다음 표에서 Amazon EKS 사용 설명서의 몇 가지 중요한 업데이트와 새로운 기능이 나와 있습니다. 이 테이블이 새 항목을 가져올 때 알림을 받으려면 RSS 리더를 사용하여 다음 URL을 구독하면 됩니다.

<https://docs.aws.amazon.com/eks/latest/userguide/doc-history.rss>

변경 사항	설명	날짜
Amazon EKS Auto Mode를 NodeClass 로 업데이트	자동 모드 노드용 NodeClass 템플릿에 별도의 포드 서브넷에 대한 구성이 추가되었습니다. 포드의 서브넷과 보안 그룹을 설정하기 위해 선택적 키 `podSubnetSelectorTerms` 와 podSecurityGroupSelectorTerms 가 추가되었습니다.	2025년 6월 13일
EKS Pod Identity를 사용하여 보조 및 교차 계정 역할 대상으로 지정	Amazon EKS는 자동 역할 체이닝을 위해 대상 IAM 역할을 EKS Pod Identity에 추가합니다. 이를 사용하여 다른 계정에 서 역할을 자동으로 수입할 수 있으며 EKS Pod Identity는 임시 자격 증명을 교체합니다. 각 Pod Identity 연결에는 먼저 수입할 동일한 계정에 IAM 역할이 있어야 합니다. 그런 다음 해당 역할을 사용하여 대상 역할을 수입합니다.	2025년 6월 11일
Amazon EKS 플랫폼 버전 업데이트	이 업데이트는 보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. 이러한 플랫폼	2025년 6월 11일

버전에는 새로운 Kubernetes 패치 버전이 없습니다.

[Amazon EKS AWS 리전 확장](#)

이제 아시아 태평양(타이페이) (ap-east-2) AWS 리전에서 Amazon EKS를 사용할 수 있습니다.

2025년 6월 6일

[새 IPv6 클러스터의 듀얼 스택 퍼블릭 엔드포인트에 대한 IPv6 액세스 제어](#)

2025년 6월 5일

[Amazon EKS 플랫폼 버전 업데이트](#)

이 업데이트는 보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. 여기에는 Kubernetes의 새로운 패치 버전인 1.32.5, 1.31.9 및 1.30.13이 포함됩니다.

2025년 5월 30일

[EKS Hybrid Nodes에 대한 새 클러스터 인사이트](#)

Amazon EKS는 하이브리드 노드의 구성을 확인하는 새로운 클러스터 인사이트를 추가합니다. 이러한 인사이트 검사는 온프레미스 노드 및 포드 문제와 클러스터의 원격 네트워크 구성에 대해 경고합니다.

2025년 5월 29일

[Kubernetes 버전 1.33](#)

새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.33 지원이 추가되었습니다.

2025년 5월 29일

[Amazon FSx CSI 드라이버에 대한 추가 기능 지원](#)

이제 AWS Management Console, AWS CLI 및 API를 사용하여 Amazon FSx CSI 드라이버를 관리할 수 있습니다.

2025년 5월 23일

콘솔에서 Prometheus 스크레이퍼 편집	이제 Amazon EKS 콘솔에서 Amazon Managed Service for Prometheus 스크레이퍼를 편집할 수 있습니다.	2025년 5월 22일
관리형 정책 업데이트	AmazonEKSDashboard ServiceRolePolicy 정책이 추가되었습니다.	2025년 5월 21일
Amazon EKS 플랫폼 버전 업데이트	이 업데이트는 보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. 이러한 플랫폼 버전에는 새로운 Kubernetes 패치 버전이 없습니다.	2025년 5월 16일
Amazon FSx for Lustre 성능에 대한 새로운 페이지	Amazon FSx for Lustre 성능 최적화에 대한 세부 정보가 포함된 새 주제가 추가되었습니다.	2025년 5월 2일
Amazon EKS Auto Mode를 NodeClass 로 업데이트	Auto Mode 노드용 NodeClass 템플릿에 순방향 네트워크 프록시에 대한 구성이 추가되었습니다. 이로 인해 HTTPS 프록시를 설정하는 선택적 키 advancedNetworking 이 추가됩니다.	2025년 4월 30일
하이브리드 노드용 Bottlerocket	이제 EKS Hybrid Nodes에서 Bottlerocket을 사용할 수 있습니다.	2025년 4월 29일
Amazon EKS 플랫폼 버전 업데이트	이 업데이트는 보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. 이러한 플랫폼 버전에는 새로운 Kubernetes 패치 버전이 없습니다.	2025년 4월 29일

[하이브리드 네트워킹에 대한 새로운 개념 페이지](#)

EKS Hybrid Nodes 개념에 대한 페이지가 추가되었습니다. 여기에서는 다이어그램을 사용하여 온프레미스 및 클라우드 네트워킹을 자세히 설명합니다.

2025년 4월 18일

[AWS 관리형 정책 업데이트](#)

VPC CNI에서 생성된 탄력적 네트워크 인터페이스에 Amazon EKS를 허용하도록 AmazonEKSClusterPolicy에 권한이 추가되었습니다. 이는 VPC CNI가 예기치 않게 종료되는 경우 EKS가 남아 있는 탄력적 네트워크 인터페이스를 정리할 수 있도록 하는데 필요합니다.

2025년 4월 16일

[AWS 관리형 정책 업데이트](#)

EKS AI/ML 고객이 기본 EKS 클러스터 보안 그룹에 송신 규칙을 추가할 수 있도록 AmazonEKSServiceRolePolicy에 권한이 추가되었습니다.

2025년 4월 14일

[EKS Hybrid Nodes의 노드 상태](#)

버전 1.2.0-eksbuild.1 부터 하이브리드 노드에서 eks-node-monitoring-agent를 사용할 수 있습니다. eks-node-monitoring-agent를 Amazon EKS 추가 기능으로 실행하여 상태 문제를 감지하고 표시할 수 있습니다.

2025년 3월 31일

[기존 클러스터용 EKS Hybrid Nodes](#)

이제 기존 클러스터의 하이브리드 노드 구성을 추가, 변경 또는 제거할 수 있습니다. 이전에는 새 클러스터를 생성할 때만 하이브리드 노드 구성을 추가할 수 있었습니다. Amazon EKS Hybrid Nodes를 사용하면 온프레미스 및 엣지 인프라를 Amazon EKS 클러스터의 노드로 사용할 수 있습니다. AWS는 Amazon EKS 클러스터의 AWS 호스팅 Kubernetes 컨트롤 플레인을 관리하고 온프레미스 또는 엣지 환경에서 실행되는 하이브리드 노드를 관리합니다.

2025년 3월 31일

[롤백: 클러스터 인사이트로 실수로 인한 업그레이드 방지](#)

Amazon EKS는 특정 클러스터 인사이트 문제가 있을 때 `--force` 플래그를 사용하여 클러스터를 업그레이드해야 하는 기능을 일시적으로 롤백했습니다. 자세한 내용은 GitHub의 [Temporary rollback of enforcing upgrade insights on update cluster version](#)을 참조하세요.

2025년 3월 28일

[Bottlerocket FIPS AMI](#)

이제 표준 관리형 노드 그룹에서 Bottlerocket FIPS AMI를 사용할 수 있습니다.

2025년 3월 27일

[AWS 관리형 정책 업데이트](#)

Amazon EKS가 Auto Mode에서 생성된 EC2 인스턴스를 종료할 수 있는 권한을 AmazonEKSServiceRolePolicy에 추가했습니다. Amazon EKS가 Auto Mode에서 생성된 EC2 인스턴스를 종료할 수 있는 권한을 AmazonEKSServiceRolePolicy에 추가했습니다.

2025년 2월 28일

[관리형 노드 그룹에 대한 전략 업데이트](#)

이제 업데이트 전략을 사용하여 관리형 노드 그룹에 대한 버전 업데이트 프로세스를 구성할 수 있습니다. 이렇게 하면 새 노드를 만들기 전에 노드를 종료하는 최소 업데이트 전략이 도입되어 용량 제약 환경에서 유용합니다. 기본 업데이트 전략은 기존 동작을 계속합니다.

2025년 1월 27일

[Kubernetes 버전 1.32](#)

새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.32 지원이 추가되었습니다.

2025년 1월 23일

[Amazon EKS AWS 리전 확장](#)

이제 Amazon EKS는 아시아 태평양(태국) 리전(ap-southeast-7) 및 멕시코(중부)(mx-central-1) AWS 리전에서 사용할 수 있습니다. EKS API에 대한 EKS 자율 모드 및 VPC 엔드포인트는 두 리전 모두에서 사용할 수 없습니다.

2025년 1월 14일

[AWS 관리형 정책 업데이트](#)

Amazon EBS CSI 드라이버가 모든 스냅샷을 복원하고 EBS 볼륨에서 빠른 스냅샷 복원(FSR)을 활성화하며 볼륨에서 태그를 수정할 수 있도록 AmazonEBSCSIDriver Policy 에 여러 권한을 추가했습니다.

2025년 1월 13일

[AWS 관리형 정책 업데이트](#)

AmazonEKSLoadBalancingPolicy 에 권한이 추가되었습니다.

2024년 12월 26일

[업데이트된 클러스터 인사이트](#)

Amazon EKS 업그레이드 인사이트는 이제 더 많은 클러스터 상태 및 버전 호환성 문제에 대해 경고합니다. 서로 다른 Kubernetes와 kubelet, kube-proxy , Amazon EKS 추가 기능과 같은 Amazon EKS 구성 요소 간의 문제를 탐지할 수 있습니다.

2024년 12월 20일

[노드 모니터링 에이전트 및 자동 복구](#)

새로운 eks-node-monitoring-agent 를 Amazon EKS 추가 기능으로 사용하여 상태 문제를 감지하고 표시할 수 있습니다. 문제가 감지되면 노드 자동 복구를 활성화하여 노드를 자동으로 교체할 수도 있습니다.

2024년 12월 16일

[Amazon EKS Hybrid Nodes](#)

이제 Amazon EKS 클러스터에 연결된 온프레미스 노드를 실행할 수 있습니다. Amazon EKS Hybrid Nodes를 사용하면 온프레미스 및 엣지 인프라를 Amazon EKS 클러스터의 노드로 사용할 수 있습니다. AWS는 Amazon EKS 클러스터의 AWS 호스팅 Kubernetes 컨트롤 플레인을 관리하고 온프레미스 또는 엣지 환경에서 실행되는 하이브리드 노드를 관리합니다.

2024년 12월 1일

[Amazon EKS Auto Mode](#)

Amazon EKS Auto Mode는 AWS의 컴퓨팅, 스토리지, 네트워킹에 대한 Kubernetes 클러스터 인프라 관리를 완전히 자동화합니다. 인프라를 자동으로 프로비저닝하고, 최적의 컴퓨팅 인스턴스를 선택하고, 리소스를 동적으로 확장하고, 비용을 지속적으로 최적화하고, 운영 체제를 패치하고, AWS 보안 서비스와 통합하여 Kubernetes 관리를 간소화합니다.

2024년 12월 1일

[Amazon EKS 플랫폼 버전 업데이트](#)

이 업데이트는 보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. 여기에는 Kubernetes 1.31.2, 1.30.6, 1.29.10, 1.28.15의 새 패치 버전이 포함됩니다.

2024년 11월 22일

[AWS 관리형 정책 업데이트](#)

중국 리전과의 호환성을 위해 AWSServiceRoleForAmazonEKSNodegroup 이 업데이트되었습니다.

2024년 11월 22일

[이제 Kubernetes 버전 1.30을 AWS Outposts의 로컬 클러스터에서 사용할 수 있습니다.](#)

이제 Kubernetes 버전 1.30을 사용하여 AWS Outposts에서 Amazon EKS 로컬 클러스터를 생성할 수 있습니다.

2024년 11월 21일

[AWS 관리형 정책 업데이트](#)

EKS가 AWS 관리형 정책 AmazonEKSLocalOutpostClusterPolicy 를 업데이트했습니다. 클러스터 컨트롤 플레인의 AWS Cloud Controller Manager가 각 노드가 있는 가용 영역을 식별할 수 있도록 ec2:DescribeAvailabilityZones 권한이 추가되었습니다.

2024년 11월 21일

[FIPS 140-3을 사용하는 Bottlerocket AMI](#)

FIPS 140-3 검증 암호화 모듈을 사용하도록 사전 구성된 Bottlerocket AMI를 사용할 수 있습니다. 여기에는 Amazon Linux 2023 Kernel Crypto API 암호화 모듈과 AWS-LC 암호화 모듈이 포함됩니다.

2024년 11월 20일

[AWS 관리형 정책 업데이트](#)

Amazon EKS 관리형 노드 그룹에서 생성한 인스턴스에서 `ec2:RebootInstances`를 허용하도록 `AWSServiceRoleForAmazonEKSNodegroup` 정책이 업데이트되었습니다. Amazon EC2 리소스에 대한 `ec2:CreateTags` 권한이 제한되었습니다.

2024년 11월 20일

[관찰성 대시보드](#)

관찰성 대시보드를 사용하면 문제를 신속하게 감지 및 해결할 수 있습니다. AWS/EKS 네임스페이스에 사용할 수 있는 새로운 [CloudWatch 변환 지표](#)도 있습니다.

2024년 11월 18일

[AWS 관리형 정책 업데이트](#)

EKS가 AWS 관리형 정책 `AmazonEKSServiceRolePolicy`를 업데이트했습니다. EKS 액세스 정책, 로드 밸런서 관리, 자동 클러스터 리소스 정리에 대한 권한이 추가되었습니다.

2024년 11월 16일

[콘솔에서 EKS Pod Identity를 지원하는 추가 기능을 위한 새 역할 생성](#)

콘솔을 사용하여 EKS Pod Identity를 지원하는 추가 기능을 생성하거나 업데이트할 때 적절한 이름, 역할 정책 및 추가 기능에 대한 신뢰 정책으로 IAM 역할을 자동으로 생성할 수 있는 새로운 단계가 있습니다.

2024년 11월 15일

[AWS 로컬 영역의 관리형 노드 그룹](#)

이제 AWS 로컬 영역에서 관리형 노드 그룹을 생성할 수 있습니다.

2024년 11월 15일

[새로운 지표 사용 가능](#)

API 그룹 `metrics.eks.amazonaws.com` 에서 사용할 수 있는 새로운 지표가 있습니다.

2024년 11월 11일

[AWS 관리형 정책 업데이트](#)

EKS가 AWS 관리형 정책 `AmazonEKSComputePolicy` 를 업데이트했습니다. `iam:AddRoleToInstanceProfile` 작업에 대한 리소스 권한이 업데이트되었습니다.

2024년 11월 7일

[AWS 관리형 정책 업데이트](#)

EKS는 새로운 AWS 관리 정책 `AmazonEKSComputePolicy` 을 추가했습니다.

2024년 11월 1일

[AWS 관리형 정책 업데이트](#)

`AmazonEKSClusterPolicy` 에 권한이 추가되었습니다. Amazon EKS가 노드에 토폴로지 정보를 레이블로 연결할 수 있는 `ec2:DescribeInstanceTopology` 권한이 추가되었습니다.

2024년 11월 1일

[AWS 관리형 정책 업데이트](#)

EKS는 새로운 AWS 관리 정책 `AmazonEKSBlockStoragePolicy` 을 추가했습니다.

2024년 10월 30일

[AWS 관리형 정책 업데이트](#)

EKS는 새로운 AWS 관리 정책 `AmazonEKSLoadBalancingPolicy` 을 추가했습니다.

2024년 10월 30일

[AWS 관리형 정책 업데이트](#)

Amazon EKS가 Amazon CloudWatch에 지표를 게시할 수 있는 `cloudwatch:PutMetricData` 권한이 `AmazonEKSServiceRolePolicy`에 추가되었습니다.

2024년 10월 29일

[AWS 관리형 정책 업데이트](#)

EKS는 새로운 AWS 관리 정책 `AmazonEKSNetworkingPolicy`을 추가했습니다.

2024년 10월 28일

[새 IPv6 클러스터를 위한 듀얼 스택 엔드포인트](#)

듀얼 스택 `eks-cluster. region.api.aws` 엔드포인트를 사용하여 새 IPv6 클러스터에 연결합니다. 이 엔드포인트는 이러한 클러스터를 설명할 때 반환된다. IPv4, IPv6 또는 듀얼 스택 환경의 `kubectl` 및 기타 쿠버네티스 API 클라이언트는 퍼블릭 또는 프라이빗 클러스터에 대해 이러한 엔드포인트를 확인하고 연결할 수 있다.

2024년 10월 21일

[AWS 관리형 정책 업데이트](#)

중국 지역의 `AWSServiceRoleForAmazonEKSNodegroup`에 `autoscaling:ResumeProcesses`, `autoscaling:SuspendProcesses` 및 관련 권한을 추가하여 EKS용 Amazon 애플리케이션 복구 컨트롤러와 통합합니다. 다른 리전에는 변경 사항이 없습니다.

2024년 10월 21일

AL2023 가속 AMIs	이제 AL2023 기반 AMIs에 가속 NVIDIA 및 AWS Neuron 인스턴스를 사용할 수 있습니다.	2024년 10월 11일
새 소스 형식	레이아웃을 일부 변경하여 새로운 소스 형식으로 전환했습니다. 일시적으로 사소한 서식 지정 문제가 발생하여 이를 해결하고 있습니다.	2024년 10월 10일
AWS 관리형 정책 업데이트	AmazonEKSServicePolicy 및 AmazonEKSServiceRolePolicy에 권한을 추가했습니다. EKS가 보안 그룹 정보를 읽고 관련 태그를 업데이트할 수 있도록 ec2:GetSecurityGroupsForVpc 및 연결된 태그 권한이 추가되었습니다.	2024년 10월 10일
AWS 관리형 정책 업데이트-새 정책	EKS는 새로운 AWS 관리형 정책을 추가했습니다.	2024년 10월 3일
Kubernetes 버전 1.31	새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.31 지원이 추가되었습니다.	2024년 9월 24일
AWS 관리형 정책 업데이트-기존 정책에 대한 업데이트	Amazon EKS에서는 기존 AWS 관리형 정책을 업데이트했습니다.	2024년 8월 21일
이제 Kubernetes 버전 1.29를 AWS Outposts의 로컬 클러스터에서 사용할 수 있습니다.	이제 Kubernetes 버전 1.29를 사용하여 AWS Outposts에서 Amazon EKS 로컬 클러스터를 생성할 수 있습니다.	2024년 8월 20일

[AWS GovCloud \(미국\)의 EKS Pod Identity](#)

Amazon EKS Pod Identity는 IAM 역할을 Kubernetes 서비스 계정에 연결합니다. 이 기능을 사용하면 노드 IAM 역할에 확장된 권한을 더 이상 제공할 필요가 없습니다. 이렇게 하면 해당 노드의 포드가 AWS API를 직접적으로 직접 호출할 수 있습니다. 서비스 계정에 대한 IAM 역할과 달리 EKS Pod Identity는 완전히 EKS 내에 있으므로 OIDC ID 제공업체가 필요하지 않습니다.

2024년 8월 14일

[시나리오 기반 콘텐츠 업데이트](#)

전체 안내서에서 좀 더 시나리오 중심으로 주제의 이름을 변경하고 업데이트했습니다.

2024년 8월 9일

[Amazon EKS용 이중 스택 VPC 인터페이스 엔드포인트](#)

이제 IPv4 및 IPv6 IP 주소와 DNS 이름을 모두 사용하여 Amazon EKS용 이중 스택 VPC 인터페이스 엔드포인트를 생성할 수 있습니다.

2024년 8월 7일

[IPv6 주소가 있는 Amazon EKS API를 위한 새로운 이중 스택 엔드포인트](#)

클러스터 생성 및 관리를 위한 EKS API와 클러스터용 OIDC 발급자 URL에는 새로운 이중 스택 엔드포인트가 있습니다. Amazon EKS API의 새 DNS 이름은 `eks.region.api.aws`이며 이는 IPv4 주소와 IPv6 주소로 확인됩니다. 새 클러스터에는 새로운 이중 스택 OIDC 발급자 URL(`oidc-eks.region.api.aws`)이 있습니다.

2024년 8월 1일

관리형 노드 그룹의 용량 블록	이제 관리형 노드 그룹에 용량 블록을 사용할 수 있습니다.	2024년 7월 1일
Auto Scaling 그룹 지표 수집은 기본적으로 활성화되어 있음	Amazon EKS 관리형 노드 그룹에는 이제 Amazon EC2 Auto Scaling 그룹 지표가 추가 비용 없이 기본적으로 활성화되어 있습니다. 이전에는 이 기능을 활성화하려면 여러 단계를 수행해야 했습니다.	2024년 6월 28일
AWS 관리형 정책 업데이트-기존 정책에 대한 업데이트	Amazon EKS에서는 기존 AWS 관리형 정책을 업데이트했습니다.	2024년 6월 27일
Kubernetes 버전 1.26	현재 Kubernetes 버전 1.26이 확장 지원됩니다.	2024년 6월 12일
AMI 정보 참조 개선	특히 Bottlerocket에 대한 AMI 정보 참조를 개선했습니다.	2024년 6월 12일
Kubernetes 버전 1.30	새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.30 지원이 추가되었습니다.	2024년 5월 23일
CoreDNS 자동 크기 조정	CoreDNS 오토스케일러는 노드 및 CPU 코어 수에 따라 EKS 클러스터에서 CoreDNS 배포의 복제본 수를 동적으로 조정합니다. 이 기능은 CoreDNS v1.9와 EKS 릴리스 버전 1.25 이상의 최신 플랫폼 버전에서 작동합니다.	2024년 5월 14일

Amazon EKS 플랫폼 버전 업데이트	이 업데이트는 보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. 여기에는 Kubernetes의 새로운 패치 버전인 1.29.4, 1.28.9 및 1.27.13이 포함됩니다.	2024년 5월 14일
Windows에 대한 CloudWatch Container Insights 지원	이제 Amazon CloudWatch Observability Operator 추가 기능을 통해 클러스터의 Windows 워커 노드에서도 Container Insights를 허용합니다.	2024년 4월 10일
Kubernetes 개념	새 Kubernetes 개념 주제를 추가했습니다.	2024년 4월 5일
액세스 및 IAM 콘텐츠 재구성	인증 구성 맵, 액세스 항목, Pod Identity, IRSA 등 액세스 및 IAM 주제와 관련된 기존 페이지를 새 섹션으로 이동합니다. 개요 콘텐츠를 수정합니다.	2024년 4월 2일
Amazon S3 CSI 드라이버에 대한 Bottlerocket OS 지원	Mountpoint for Amazon S3 CSI 드라이버는 현재 Bottlerocket과 호환됩니다.	2024년 3월 13일
AWS 관리형 정책 업데이트-기존 정책에 대한 업데이트	Amazon EKS에서는 기존 AWS 관리형 정책을 업데이트했습니다.	2024년 3월 4일
Amazon Linux 2023	Amazon Linux 2023(AL2023)은 클라우드 애플리케이션에 안전하고 안정적이면서 고성능의 환경을 제공하도록 설계된 새로운 Linux 기반 운영 체제입니다.	2024년 2월 29일

[Kubernetes 1.29의 EKS Pod Identity와 IRSA 지원 사이드카](#)

Kubernetes 1.29에서는 Amazon EKS 클러스터에서 사이드카 컨테이너를 사용할 수 있습니다. 서비스 계정 또는 EKS Pod Identity에 대한 IAM 역할에서는 사이드카 컨테이너가 지원됩니다. 사이드카에 대한 자세한 내용은 Kubernetes Documentation의 [Sidecar Containers](#)를 참조하세요.

2024년 2월 26일

[Kubernetes 버전 1.29](#)

새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.29 지원이 추가되었습니다.

2024년 1월 23일

[전체 릴리스: Kubernetes 버전에 대한 Amazon EKS 확장 지원](#)

확장 Kubernetes 버전 지원을 통해 특정 Kubernetes 버전을 14개월 이상 유지할 수 있습니다.

2024년 1월 16일

[AWS 클라우드에서 Amazon EKS 클러스터 상태 감지](#)

Amazon EKS는 Amazon EKS 클러스터 및 클러스터 사전 조건 인프라에서 문제를 감지하여 클러스터 상태에 저장합니다. EKS API에서 클러스터의 health 및 AWS Management Console에서 EKS 클러스터 문제를 확인할 수 있습니다. 콘솔에서 감지되고 콘솔에 표시되는 문제 외에도 이 문제를 확인할 수 있습니다. 이전에는 AWS Outposts의 로컬 클러스터에서만 클러스터 상태를 사용할 수 있었습니다.

2023년 12월 28일

[클러스터 인사이트](#)

이제 반복 검사를 기반으로 클러스터에 대한 권장 사항을 얻을 수 있습니다.

2023년 12월 20일

[Amazon EKS AWS 리전 확장](#)

이제 Amazon EKS를 캐나다 서부(캘거리)(ca-west-1) AWS 리전에서 사용할 수 있습니다.

2023년 12월 20일

[이제 액세스 항목을 사용하여 IAM 역할 및 사용자에게 클러스터에 대한 액세스 권한을 부여할 수 있습니다.](#)

액세스 항목을 도입하기 전에는 aws-auth ConfigMap 에 항목을 추가하여 IAM 역할 및 사용자에게 클러스터에 대한 액세스 권한을 부여했습니다. 이제 각 클러스터에는 액세스 모드가 있으며, 일정에 따라 액세스 항목을 사용하도록 전환할 수 있습니다. 모드를 전환한 후에는 AWS CLI, AWS CloudFormation 및 AWS SDK 에서 액세스 항목을 추가하여 사용자를 추가할 수 있습니다.

2023년 12월 18일

[Amazon EKS 플랫폼 버전 업데이트](#)

이 업데이트는 보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. 여기에는 Kubernetes 1.28.4, 1.27.8, 1.26.11, 1.25.16의 새 패치 버전이 포함됩니다.

2023년 12월 12일

[Mountpoint for Amazon S3 CSI 드라이버](#)

이제 Amazon EKS 클러스터에 Mountpoint for Amazon S3 CSI 드라이버를 설치할 수 있습니다.

2023년 11월 27일

[클러스터를 생성할 때 Prometheus 지표 켜기](#)

이제 AWS Management Console에서 클러스터를 생성할 때 Prometheus 지표를 켤 수 있습니다. 또한 관찰성 탭에서 Prometheus 스크레이퍼 세부 정보를 볼 수 있습니다.

2023년 11월 26일

[Amazon EKS Pod Identity](#)

Amazon EKS Pod Identity는 IAM 역할을 Kubernetes 서비스 계정에 연결합니다. 이 기능을 사용하면 노드 IAM 역할에 확장된 권한을 더 이상 제공할 필요가 없습니다. 이렇게 하면 해당 노드의 포드가 AWS API를 직접적으로 직접 호출할 수 있습니다. 서비스 계정에 대한 IAM 역할과 달리 EKS Pod Identity는 완전히 EKS 내에 있으므로 OIDC ID 제공업체가 필요하지 않습니다.

2023년 11월 26일

[AWS 관리형 정책 업데이트-기존 정책에 대한 업데이트](#)

Amazon EKS에서는 기존 AWS 관리형 정책을 업데이트했습니다.

2023년 11월 26일

[CSI 스냅샷 컨트롤러](#)

이제 Amazon EBS CSI 드라이버와 같은 호환 CSI 드라이버에 사용할 CSI 스냅샷 컨트롤러를 설치할 수 있습니다.

2023년 11월 17일

ADOT Operator 주제 재작성	ADOT Operator에 대한 Amazon EKS 추가 기능 지원 섹션이 AWS Distro for OpenTelemetry 설명서와 중복되었습니다. 오래되고 일관적이지 않은 정보를 줄이기 위해 나머지 필수 정보를 해당 리소스로 마이그레이션했습니다.	2023년 11월 14일
Prometheus 지표에 대한 CoreDNS EKS 추가 기능 지원	CoreDNS에 대한 EKS 추가 기능의 v1.10.1-eksbuild.5 , v1.9.3-eksbuild.9 , v1.8.7-eksbuild.8 버전은 kube-dns 서비스에서 CoreDNS가 지표를 게시하는 포트를 노출합니다. 이를 통해 모니터링 시스템에서 CoreDNS 지표를 더욱 쉽게 포함할 수 있습니다.	2023년 11월 10일
Amazon EKS CloudWatch Observability Operator 추가 기능	Amazon EKS CloudWatch Observability Operator 페이지가 추가되었습니다.	2023년 11월 6일
미국 동부(오하이오)의 자체 관리형 P5 인스턴스용 용량 블록	미국 동부(오하이오)에서는 이제 자체 관리형 P5 인스턴스에 용량 블록을 사용할 수 있습니다.	2023년 10월 31일

[클러스터는 서브넷 및 보안 그룹을 수정하도록 지원합니다](#)

클러스터를 업데이트하여 클러스터에서 사용하는 서브넷과 보안 그룹을 변경할 수 있습니다. AWS Management Console, 최신 버전의 AWS CLI, AWS CloudFormation 및 eksctl 버전 v0.164.0-rc.0 이상에서 업데이트할 수 있습니다. 서브넷에 사용 가능한 IP 주소를 더 많이 제공하여 클러스터 버전을 성공적으로 업그레이드하려면 이 작업을 수행해야 할 수도 있습니다.

2023년 10월 24일

[클러스터 역할 및 관리형 노드 그룹 역할은 고객 관리형 AWS ID 및 액세스 관리 정책을 지원합니다.](#)

클러스터 역할에 [AmazonEKS ClusterPolicy](#) AWS 관리형 정책 대신 사용자 지정 IAM 정책을 사용할 수 있습니다. 또한 관리형 노드 그룹의 노드 역할에 대해 [AmazonEKSWorkerNodePolicy](#) AWS 관리형 정책 대신 사용자 지정 IAM 정책을 사용할 수 있습니다. 엄격한 규정 준수 요구 사항을 충족할 수 있는 최소 권한이 있는 정책을 생성하려면 이렇게 하세요.

2023년 10월 23일

[eksctl 설치 링크 수정](#)

페이지가 이동된 후 eksctl의 설치 링크를 수정합니다.

2023년 10월 6일

[평가판 릴리스: Kubernetes 버전 1.28에 대한 Amazon EKS 확장 지원](#)

확장 Kubernetes 버전 지원을 통해 특정 Kubernetes 버전을 14개월 이상 유지할 수 있습니다.

2023년 10월 4일

AWS 앱 메시 통합에 대한 참조 제거	App Mesh의 기존 고객에 한해 Amazon EKS와 AWS App Mesh의 통합이 유지됩니다.	2023년 9월 29일
Kubernetes 버전 1.28	새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.28 지원이 추가되었습니다.	2023년 9월 26일
기존 클러스터는 Kubernetes용 Amazon VPC CNI 플러그인에 서 Kubernetes 네트워크 정책을 적용을 지원합니다.	타사 솔루션을 필요로 하는 대신 Kubernetes용 Amazon VPC CNI 플러그인을 사용하는 기존 클러스터에서 Kubernetes 네트워크 정책을 사용할 수 있습니다. 타사 솔루션을 필요로 하는 대신 Kubernetes용 Amazon VPC CNI 플러그인을 사용하는 기존 클러스터에서 Kubernetes 네트워크 정책을 사용할 수 있습니다.	2023년 9월 15일
CoreDNS Amazon EKS 추가 기능에서 PDB 수정 지원	버전 v1.9.3-eksbuild.7 이상 및 v1.10.1-eksbuild.4 이상에서 CoreDNS에 대한 EKS 추가 기능의 PodDisruptionBudget 을 수정할 수 있습니다.	2023년 9월 15일
공유 서브넷에 대한 Amazon EKS 지원	공유 서브넷에서 Amazon EKS 클러스터를 만드는 것에 대한 새로운 공유 서브넷 요구 사항 및 고려 사항 .	2023년 9월 7일
Amazon EKS란 무엇입니까? 업데이트	새로운 일반 사용 사례 및 아키텍처 주제 추가. 기타 주제 갱신.	2023년 9월 6일

[Kubernetes용 Amazon VPC CNI 플러그인의 Kubernetes 네트워크 정책 적용](#)

타사 솔루션을 필요로 하는 대신 Kubernetes용 Amazon VPC CNI 플러그인을 사용하는 기존 클러스터에서 Kubernetes 네트워크 정책을 사용할 수 있습니다. 타사 솔루션을 필요로 하는 대신 Kubernetes용 Amazon VPC CNI 플러그인을 사용하는 기존 클러스터에서 Kubernetes 네트워크 정책을 사용할 수 있습니다.

2023년 8월 29일

[Amazon EKS AWS 리전 확장](#)

Amazon EKS는 이제 이스라엘 (텔아비브)(il-central-1) AWS 리전에서 사용할 수 있습니다.

2023년 8월 1일

[구성 가능한 Fargate용 임시 스토리지](#)

Amazon EKS Fargate에서 각 포드 실행용 임시 스토리지 총량을 늘릴 수 있습니다.

2023년 7월 31일

[Amazon EFS CSI 드라이버에 대한 추가 기능 지원](#)

이제 AWS Management Console, AWS CLI 및 API를 사용하여 Amazon EFS CSI 드라이버를 관리할 수 있습니다.

2023년 7월 26일

[AWS 관리형 정책 업데이트 - 새 정책](#)

Amazon EKS에서는 새 AWS 관리형 정책을 추가했습니다.

2023년 7월 26일

[1.27, 1.26, 1.25 및 1.24의 Kubernetes 버전 업데이트가 이제 AWS Outposts의 로컬 클러스터에 제공됩니다.](#)

이제 AWS Outposts의 로컬 클러스터에서 1.27.3, 1.26.6, 1.25.11 및 1.24.15용 Kubernetes 버전 업데이트를 사용할 수 있습니다.

2023년 7월 20일

<u>Windows 노드에 대한 IP 접두사 지원</u>	노드에 IP 접두사를 할당하면 노드에 개별 보조 IP 주소를 할당할 때보다 훨씬 더 많은 수의 포드를 노드에서 호스팅할 수 있습니다.	2023년 7월 6일
<u>Amazon FSx for OpenZFS CSI 드라이버</u>	이제 Amazon EKS 클러스터에 Amazon FSx for OpenZFS CSI 드라이버를 설치할 수 있습니다.	2023년 6월 30일
<u>IPv4 클러스터의 Linux 노드에 있는 포드가 이제 IPv6 엔드포인트와 통신할 수 있습니다.</u>	노드에 IPv6 주소를 할당하면 포드의 IPv4 주소는 노드가 실행 중인 노드의 IPv6 주소로 변환된 네트워크 주소입니다.	2023년 6월 19일
<u>AWS GovCloud(미국) 리전의 Windows 관리형 노드 그룹</u>	이제 AWS GovCloud(미국) 리전에서 Amazon EKS 관리형 노드 그룹은 Windows 컨테이너를 실행할 수 있습니다.	2023년 5월 30일
<u>Kubernetes 버전 1.27</u>	새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.27 지원이 추가되었습니다.	2023년 5월 24일
<u>Kubernetes 버전 1.26</u>	새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.26 지원이 추가되었습니다.	2023년 4월 11일
<u>도메인 없는 gMSA</u>	이제 Windows 포드에서 도메인 없는 gMSA를 사용할 수 있습니다.	2023년 3월 27일
<u>Amazon EKS AWS 리전 확장</u>	이제 아시아 태평양(멜버른)(ap-southeast-4) AWS 리전에서 Amazon EKS를 사용할 수 있습니다.	2023년 3월 10일

Amazon File Cache CSI 드라이버	이제 Amazon EKS 클러스터에 Amazon 파일 캐시 CSI 드라이버를 설치할 수 있습니다.	2023년 3월 3일
이제 Kubernetes 버전 1.25를 AWS Outposts의 로컬 클러스터에서 사용할 수 있습니다.	이제 Outpost에서 Kubernetes 버전 1.22~1.25를 사용하여 Amazon EKS 로컬 클러스터를 생성할 수 있습니다.	2023년 3월 1일
Kubernetes 버전 1.25	새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.25 지원이 추가되었습니다.	2023년 2월 22일
AWS 관리형 정책 업데이트-기존 정책에 대한 업데이트	Amazon EKS에서는 기존 AWS 관리형 정책을 업데이트했습니다.	2023년 2월 7일
Amazon EKS AWS 리전 확장	Amazon EKS는 이제 아시아 태평양(하이데라바드)(ap-south-2), 유럽(취리히)(eu-central-2) 및 유럽(스페인)(eu-south-2) AWS 리전에서 사용할 수 있습니다.	2023년 2월 6일
Kubernetes 버전 1.21 – 1.24는 이제 AWS Outposts의 로컬 클러스터에 사용할 수 있습니다.	이제 Outpost에서 Kubernetes 버전 1.21~1.24를 사용하여 Amazon EKS 로컬 클러스터를 생성할 수 있습니다. 이전에는 1.21 버전만 사용할 수 있었습니다.	2023년 1월 17일
Amazon EKS는 이제 AWS PrivateLink를 지원합니다.	AWS 프라이빗 링크를 사용하여 VPC와 Amazon EKS 사이에 프라이빗 연결을 생성할 수 있습니다.	2022년 12월 16일

관리형 노드 그룹 Windows 지원	이제 Amazon EKS 관리형 노드 그룹에 Windows를 사용할 수 있습니다.	2022년 12월 15일
이제 독립 소프트웨어 공급업체의 Amazon EKS 애드온을 AWS 마켓플레이스에서 사용할 수 있습니다.	이제 AWS 마켓플레이스를 통해 독립 소프트웨어 공급업체의 Amazon EKS 추가 기능을 검색하고 구독할 수 있습니다.	2022년 11월 28일
AWS 관리형 정책 업데이트-기존 정책에 대한 업데이트	Amazon EKS에서는 기존 AWS 관리형 정책을 업데이트했습니다.	2022년 11월 17일
Kubernetes 버전 1.24	새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.24 지원이 추가되었습니다.	2022년 11월 15일
Amazon EKS AWS 리전 확장	이제 중동(UAE)(me-central-1) AWS 리전에서 Amazon EKS를 사용할 수 있습니다.	2022년 11월 3일
AWS 관리형 정책 업데이트-기존 정책에 대한 업데이트	Amazon EKS에서는 기존 AWS 관리형 정책을 업데이트했습니다.	2022년 10월 24일
AWS 관리형 정책 업데이트-기존 정책에 대한 업데이트	Amazon EKS에서는 기존 AWS 관리형 정책을 업데이트했습니다.	2022년 10월 20일
이제 AWS Outposts에서 로컬 클러스터를 사용할 수 있습니다.	이제 Outpost에서 Amazon EKS 로컬 클러스터를 생성할 수 있습니다.	2022년 9월 19일
Fargate vCPU 기반 할당량	Fargate는 포드 기반 할당량에서 vCPU 기반 할당량으로 전환하고 있습니다.	2022년 9월 8일

AWS 관리형 정책 업데이트-기 존 정책에 대한 업데이트	Amazon EKS에서는 기존 AWS 관리형 정책을 업데이트했습니다.	2022년 8월 31일
비용 모니터링	Amazon EKS는 이제 포드, 노드, 네임스페이스 및 레이블을 포함한 Kubernetes 리소스별로 분류된 비용을 모니터링하는 데 사용할 수 있는 Kubecost를 지원합니다.	2022년 8월 24일
AWS 관리형 정책 업데이트 - 새 정책	Amazon EKS에서는 새 AWS 관리형 정책을 추가했습니다.	2022년 8월 24일
AWS 관리형 정책 업데이트 - 새 정책	Amazon EKS에서는 새 AWS 관리형 정책을 추가했습니다.	2022년 8월 23일
리소스에 결제용 태깅	모든 클러스터에 대한 <code>aws:eks:cluster-name</code> 생성 비용 할당 태그 지원이 추가되었습니다.	2022년 8월 16일
Fargate 프로필 와일드카드	네임스페이스, 레이블 키 및 레이블 값에 대한 선택기 기준에서 Fargate 프로필 와일드카드에 대한 지원이 추가되었습니다.	2022년 8월 16일
Kubernetes 버전 1.23	새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.23 지원이 추가되었습니다.	2022년 8월 11일
AWS Management Console에 서 Kubernetes 리소스 보기	이제 AWS Management Console을 사용하여 클러스터에 배포된 Kubernetes 리소스에 대한 정보를 볼 수 있습니다.	2022년 5월 3일

Amazon EKS AWS 리전 확장	이제 아시아 태평양(자카르타) (ap-southeast-3) AWS 리전에서 Amazon EKS를 사용할 수 있습니다.	2022년 5월 2일
Observability 페이지 및 ADOT 추가 기능 지원	관찰 가능성 페이지 및 AWS Distro for OpenTelemetry(ADOT)가 추가되었습니다.	2022년 4월 21일
Kubernetes 버전 1.22	새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.22 지원이 추가되었습니다.	2022년 4월 4일
AWS 관리형 정책 업데이트 - 새 정책	Amazon EKS에서는 새 AWS 관리형 정책을 추가했습니다.	2022년 4월 4일
Fargate 포드 패치 세부 정보 추가	Fargate 포드를 업그레이드하면 Amazon EKS는 먼저 포드 중단 예산에 따라 포드를 제거하려고 시도합니다. 실패한 제거에 대응하는 이벤트 규칙을 생성한 후에 포드를 삭제할 수 있습니다.	2022년 4월 1일
정식 릴리스: Amazon EBS CSI 드라이버에 대한 추가 기능 지원	이제 AWS Management Console, AWS CLI 및 API를 사용하여 Amazon EBS CSI 드라이버를 관리할 수 있습니다.	2022년 3월 31일
AWS Outposts 콘텐츠 업데이트	AWS Outposts에서 Amazon EKS 클러스터를 배포하는 방법.	2022년 3월 22일
AWS 관리형 정책 업데이트-기존 정책에 대한 업데이트	Amazon EKS에서는 기존 AWS 관리형 정책을 업데이트했습니다.	2022년 3월 21일

Windows containerd 지원	이제 Windows 노드용 containerd 런타임을 선택할 수 있습니다.	2022년 3월 14일
Amazon EKS Connector 고려 사항이 보안 문서에 추가	연결된 클러스터와 관련된 공동 책임 모델을 설명합니다.	2022년 2월 25일
포드 및 서비스에 IPv6 주소 할당	이제 포드와 서비스에 IPv6 주소를 할당하는 1.21 이상 클러스터를 생성할 수 있습니다.	2022년 1월 6일
AWS 관리형 정책 업데이트-기존 정책에 대한 업데이트	Amazon EKS에서는 기존 AWS 관리형 정책을 업데이트했습니다.	2021년 12월 13일
미리 보기 릴리스: Amazon EBS CSI 드라이버에 대한 추가 기능 지원	이제 AWS Management Console, AWS CLI 및 API를 통해 미리 보기하여 Amazon EBS CSI 드라이버를 관리할 수 있습니다.	2021년 12월 9일
Karpenter Autoscaler 지원	이제 Karpenter 오픈 소스 프로젝트를 사용하여 노드를 자동 확장할 수 있습니다.	2021년 11월 29일
Fargate 로깅에서 Fluent Bit Kubernetes 필터 지원	이제 Fargate 로깅과 함께 Fluent Bit Kubernetes 필터를 사용할 수 있습니다.	2021년 11월 10일
제어 영역에서 Windows 지원 사용 가능	이제 제어 영역에서 Windows 지원을 사용할 수 있습니다. 더 이상 데이터 영역에서 Windows 지원을 사용 설정할 필요가 없습니다.	2021년 11월 9일

[관리형 노드 그룹에 대한 AMI
유형으로 Bottlerocket 추가](#)

이전에는 Bottlerocket을 자체 관리형 노드 옵션으로만 사용할 수 있었습니다. 이제 관리형 노드 그룹으로 구성할 수 있어 노드 규정 준수 요구 사항을 충족하는 데 필요한 노력을 줄일 수 있습니다.

2021년 10월 28일

[DL1 드라이버 지원](#)

사용자 정의 Amazon Linux AMI는 이제 Amazon Linux 2용 딥 러닝 워크로드를 지원합니다. 이를 통해 일반 온프레미스 또는 클라우드 기준 구성이 허용됩니다.

2021년 10월 25일

[VT1 비디오 지원](#)

사용자 지정 Amazon Linux AMI는 이제 일부 배포에 대해 VT1을 지원합니다. 이 활성화를 통해 Amazon EKS 클러스터에 Xilinx U30 디바이스를 알립니다.

2021년 9월 13일

[이제 Amazon EKS Connector
를 사용할 수 있습니다.](#)

Amazon EKS Connector를 사용하여 적합한 Kubernetes 클러스터를 AWS에 등록하고 연결할 수 있으며 Amazon EKS 콘솔에서 시각화할 수 있습니다.

2021년 9월 8일

[이제 Amazon EKS Anywhere
를 사용할 수 있습니다.](#)

Amazon EKS Anywhere는 온프레미스에서 Kubernetes 클러스터를 생성하고 운영하기 위해 사용할 수 있는 Amazon EKS에 대한 새로운 배포 옵션입니다.

2021년 9월 8일

[Amazon FSx for NetApp
ONTAP CSI 드라이버](#)

Amazon FSx for NetApp
ONTAP CSI 드라이버에 대해
간략히 설명하고 다른 참조에
대한 링크를 제공하는 주제가
추가되었습니다.

2021년 9월 2일

[이제 관리형 노드 그룹이 노드
에 대해 Amazon EKS 권장 최
대 포드를 자동 계산합니다.](#)

이제 관리형 노드 그룹은 시작
템플릿 없이 배포하거나 AMI
ID를 지정하지 않은 시작 템플
릿으로 배포하는 노드에 대한
Amazon EKS 최대 포드 수를
자동으로 계산합니다.

2021년 8월 30일

[Amazon EKS 추가 기능 소프
트웨어를 제거하지 않고 추가
기능 설정의 Amazon EKS 관
리 제거](#)

이제 클러스터에서 추가 기능
소프트웨어를 제거하지 않고도
Amazon EKS 추가 기능을 제
거할 수 있습니다.

2021년 8월 20일

[Multus를 사용하여 멀티홈 파
드 생성](#)

이제 Multus를 사용하여 여러
개의 네트워크 인터페이스를
포드에 추가할 수 있습니다.

2021년 8월 2일

[Linux Amazon EC2 노드에 더
많은 IP 주소 추가](#)

Linux Amazon EC2 노드에 훨
씬 더 많은 IP 주소를 추가할 수
있습니다. 즉, 각 노드에서 더
높은 밀도로 포드를 실행할 수
있습니다. Linux Amazon EC2
노드에 훨씬 더 많은 IP 주소를
추가할 수 있습니다. 즉, 각 노
드에서 더 높은 밀도로 포드를
실행할 수 있습니다.

2021년 7월 27일

containerd 런타임 부트스트랩	Amazon EKS 최적화 가 속 Amazon Linux Amazon Machine Image(AMI)에는 이 제 부트스트랩 플래그가 포함 되어 있어 containerd 런타 임을 Amazon EKS 최적화 및 Bottlerocket AMI에서 사용 설 정할 수 있습니다. 이 플래그는 지원되는 모든 Kubernetes 버 전의 AMI에서 사용할 수 있습 니다.	2021년 7월 19일
Kubernetes 버전 1.21	추가된 Kubernetes 버전 1.21 지원	2021년 7월 19일
관리형 정책 주제 추가	2021년 6월 17일 이후 적용된 모든 Amazon EKS IAM 관리형 정책 및 변경 사항의 목록입니 다.	2021년 6월 17일
Fargate에서 포드에 보안 그룹 사용	이제 Fargate에서 포드에 보안 그룹을 사용할 수 있을 뿐 아니 라 Amazon EC2 노드와 함께 사용할 수도 있습니다.	2021년 6월 1일
CoreDNS 및 kube-proxy Amazon EKS 추가 기능 추가	Amazon EKS는 이제 CoreDNS 및 클러스터용 kube-proxy Amazon EKS 추가 기능을 제공합니다.	2021년 5월 19일
Kubernetes 버전 1.20	새 클러스터 및 버전 업그레이 드에 대한 Kubernetes 버전 1.20 지원이 추가되었습니다.	2021년 5월 18일

[AWS Load Balancer Controller](#) [2.2.0 릴리스](#)

이제 AWS 로드 밸런서 컨트롤러를 사용하여 인스턴스 또는 IP 대상을 사용하여 탄력적 로드 밸런서를 생성할 수 있습니다.

2021년 5월 14일

[관리형 노드 그룹의 노드 테인트](#)

Amazon EKS는 이제 관리 노드 그룹에 노드 테인트를 추가할 수 있도록 지원합니다.

2021년 5월 11일

[기존 클러스터에 대한 비밀 암호화](#)

Amazon EKS는 이제 기존 클러스터에 [비밀 암호화](#)를 추가할 수 있도록 지원합니다.

2021년 2월 26일

[Kubernetes 버전 1.19](#)

새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.19 지원이 추가되었습니다.

2021년 2월 16일

[Amazon EKS는 이제 버전 1.16 이상의 클러스터에서 사용자를 인증하는 방법으로 OpenID Connect\(OIDC\) 아이덴티티 제공업체를 지원합니다.](#)

OIDC ID 공급자는 AWS ID 및 액세스 관리(IAM)와 함께 또는 그 대안으로 사용할 수 있습니다.

2021년 2월 12일

[AWS Management Console에서 노드 및 워크로드 리소스 보기](#)

이제 관리형, 자체 관리형 및 Fargate 노드, 그리고 AWS Management Console에 배포된 Kubernetes 워크로드에 대한 세부 정보를 볼 수 있습니다.

2020년 12월 1일

[관리형 노드 그룹에 스팟 인스턴스 유형 배포](#)

이제 관리형 노드 그룹에 여러 스팟 또는 온디맨드 인스턴스 유형을 배포할 수 있습니다.

2020년 12월 1일

[이제 Amazon EKS가 클러스터의 특정 추가 기능을 관리할 수 있습니다.](#)

추가 기능을 직접 관리하거나 Amazon EKS가 Amazon EKS API를 통해 추가 기능의 시작 및 버전을 제어하도록 할 수 있습니다.

2020년 12월 1일

[여러 수신 간에 ALB 공유](#)

이제 AWS 여러 Kubernetes 수신 간에 Application Load Balancer(ALB)를 공유할 수 있습니다. 과거에는 각 수신마다 별도의 ALB를 배포해야 했습니다.

2020년 10월 23일

[NLB IP 대상 지원](#)

이제 IP 대상을 사용하여 Network Load Balancer를 배포할 수 있습니다. 즉, NLB를 사용하여 네트워크 트래픽을 Fargate 포드로, Amazon EC2 노드에서 실행되는 포드로 직접 로드 밸런싱할 수 있습니다.

2020년 10월 23일

[Kubernetes 버전 1.18](#)

새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.18 지원이 추가되었습니다.

2020년 10월 13일

[Kubernetes 서비스 IP 주소 할당에 대한 사용자 지정 CIDR 블록을 지정합니다.](#)

이제 Kubernetes가 서비스 IP 주소를 할당하는 사용자 지정 CIDR 블록을 지정할 수 있습니다.

2020년 9월 29일

[개별 포드에 보안 그룹 할당](#)

이제 여러 Amazon EC2 인스턴스 유형에서 실행되는 일부 개별 포드에 서로 다른 보안 그룹을 연결할 수 있습니다.

2020년 9월 9일

[노드에 Bottlerocket 배포](#)

이제 [Bottlerocket](#)을 실행하는 노드를 배포할 수 있습니다.

2020년 8월 31일

Arm 노드를 시작하는 기능 정식 지원	이제 관리형 노드 그룹과 자체 관리형 노드 그룹에서 Arm 노드를 시작할 수 있습니다.	2020년 8월 17일
관리 노드 그룹 시작 템플릿 및 사용자 지정 AMI	이제 Amazon EC2 시작 템플릿을 사용하는 관리형 노드 그룹을 배포할 수 있습니다. 원하는 경우 시작 템플릿에서 사용자 지정 AMI 지정할 수 있습니다.	2020년 8월 17일
AWS Fargate에 대한 EFS 지원	이제 AWS Fargate에 Amazon EFS를 사용할 수 있습니다.	2020년 8월 17일
Amazon EKS 플랫폼 버전 업데이트	이 업데이트는 보안 수정 및 개선 사항이 적용된 새로운 플랫폼 버전입니다. 여기에는 Kubernetes 버전 1.15 이상으로 Network Load Balancer를 사용할 때 LoadBalancer 유형의 서비스에 대한 UDP 지원이 포함됩니다. 자세한 내용을 알아보려면 GitHub에서 AWS Network Load Balancer에 대해 UDP 허용 문제를 참조하세요.	2020년 8월 12일
Amazon EKS AWS 리전 확장	Amazon EKS는 이제 아프리카(케이프타운)(af-south-1) 및 EU(밀라노)(eu-south-1) AWS 리전에서 사용할 수 있습니다.	2020년 8월 6일
Fargate 사용량 지표	AWS Fargate는 사용자 계정의 Fargate 온디맨드 리소스 사용량에 대한 가시성을 제공하는 CloudWatch 사용량 지표를 제공합니다.	2020년 8월 3일

Kubernetes 버전 1.17	새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.17 지원이 추가되었습니다.	2020년 7월 10일
Kubernetes용 App Mesh 컨트롤러를 사용하여 Kubernetes에서 App Mesh 리소스 생성 및 관리	Kubernetes에서 App Mesh 리소스를 생성하고 관리할 수 있습니다. 또한 컨트롤러는 사용자가 배포한 포드에 Envoy 프록시 및 init 컨테이너를 자동으로 삽입합니다.	2020년 6월 18일
Amazon EKS에서 Amazon EC2 Inf1 노드 지원	Amazon EC2 Inf1 노드를 클러스터에 추가할 수 있습니다.	2020년 6월 4일
Amazon EKS AWS 리전 확장	이제 AWS GovCloud (미국 동부)(us-gov-east-1) 및 AWS GovCloud (미국 서부)(us-gov-west-1) AWS 리전에서 Amazon EKS를 사용할 수 있습니다.	2020년 5월 13일
Amazon EKS에서 Kubernetes 1.12가 더 이상 지원되지 않습니다.	Amazon EKS에서 Kubernetes 버전 1.12가 더 이상 지원되지 않습니다. 서비스 중단이 발생하지 않도록 모든 1.12 클러스터를 버전 1.13 이상으로 업데이트하세요.	2020년 5월 12일
Kubernetes 버전 1.16	새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.16 지원이 추가되었습니다.	2020년 4월 30일
AWSServiceRoleForAmazonEKS 서비스 연결 역할 추가	AWSServiceRoleForAmazonEKS 서비스 연결 역할이 추가되었습니다.	2020년 4월 16일

Kubernetes 버전 1.15	새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.15 지원이 추가되었습니다.	2020년 3월 10일
Amazon EKS AWS 리전 확장	이제 베이징(cn-north-1) 및 Ningxia(cn-northwest-1) AWS 리전에서 Amazon EKS를 사용할 수 있습니다.	2020년 2월 26일
FSx for Lustre CSI 드라이버	Kubernetes 1.14 Amazon EKS 클러스터에 FSX for Lustre CSI 드라이버를 설치하기 위한 항목이 추가되었습니다.	2019년 12월 23일
클러스터의 퍼블릭 액세스 엔드포인트에 대한 네트워크 액세스 제한	이 업데이트를 통해 Amazon EKS를 사용하여 Kubernetes API 서버의 퍼블릭 액세스 엔드포인트와 통신할 수 있는 CIDR 범위를 제한할 수 있습니다.	2019년 12월 20일
VPC 외부에서 클러스터의 프라이빗 액세스 엔드포인트 주소 확인	이 업데이트를 통해 Amazon EKS를 사용하여 VPC 외부에서 Kubernetes API 서버의 프라이빗 액세스 엔드포인트를 확인할 수 있습니다.	2019년 12월 13일
(베타) Amazon EC2 A1 Amazon EC2 인스턴스 노드	Amazon EKS 클러스터에 등록하는 Amazon EC2 A1 Amazon EC2 인스턴스 노드를 시작합니다.	2019년 12월 4일
AWS Outpost에 클러스터 생성	이제 Amazon EKS는 AWS Outposts에서 클러스터 생성을 지원합니다.	2019년 12월 3일

Amazon EKS의 AWS Fargate	Amazon EKS Kubernetes 클러스터는 이제 Fargate에서 실행 중인 포드를 지원합니다.	2019년 12월 3일
Amazon EKS AWS 리전 확장	Amazon EKS는 이제 캐나다(중부)(ca-central-1) AWS 리전에서 사용할 수 있습니다.	2019년 11월 21일
관리형 노드 그룹	Amazon EKS 관리형 노드 그룹은 Amazon EKS Kubernetes 클러스터의 노드(Amazon EC2 인스턴스) 프로비저닝 및 수명 주기 관리를 자동화합니다.	2019년 11월 18일
Amazon EKS 플랫폼 버전 업데이트	CVE-2019-11253 을 해결하기 위한 새로운 플랫폼 버전	2019년 11월 6일
Amazon EKS에서 Kubernetes 1.11가 더 이상 지원되지 않습니다.	Amazon EKS에서 Kubernetes 버전 1.11이 더 이상 지원되지 않습니다. 서비스 중단이 발생하지 않도록 모든 1.11 클러스터를 버전 1.12 이상으로 업데이트하세요.	2019년 11월 4일
Amazon EKS AWS 리전 확장	이제 남아메리카(상파울루)(sa-east-1) AWS 리전에서 Amazon EKS를 사용할 수 있습니다.	2019년 10월 16일
Windows 지원	이제 Kubernetes 버전 1.14를 실행하는 Amazon EKS 클러스터에서 Windows 워크로드를 지원합니다.	2019년 10월 7일

AutoScaling	Amazon EKS 클러스터에서 지원되는 다양한 유형의 Kubernetes 자동 크기 조정 중 일부에 관해 설명하는 장을 추가했습니다.	2019년 9월 30일
Kubernetes 대시보드 업데이트	베타 2.0 버전을 사용하기 위해 Amazon EKS 클러스터에 Kubernetes 대시보드를 설치하는 방법에 관한 주제를 업데이트했습니다.	2019년 9월 28일
Amazon EFS CSI 드라이버	Kubernetes 1.14 Amazon EKS 클러스터에 Amazon EFS CSI 드라이버를 설치하는 방법에 관한 주제를 추가했습니다.	2019년 9월 19일
Amazon EKS 최적화 AMI ID에 대한 Amazon EC2 Systems Manager 파라미터	Amazon EC2 Systems Manager 파라미터를 사용해 Amazon EKS 최적화 AMI ID를 검색하는 방법에 관한 주제를 추가했습니다. 이 파라미터를 사용하면 AMI ID를 조회할 필요가 없습니다.	2019년 9월 18일
Amazon EKS 리소스 태깅	Amazon EKS 클러스터의 태깅을 관리할 수 있습니다.	2019년 9월 16일
Amazon EBS CSI 드라이버	Kubernetes 1.14 Amazon EKS 클러스터에 Amazon EBS CSI 드라이버를 설치하는 방법에 관한 주제를 추가했습니다.	2019년 9월 9일
CVE-2019-9512 및 CVE-2019-9514에 패치된 새 Amazon EKS 최적화 AMI	Amazon EKS는 CVE-2019-9512 및 CVE-2019-9514 를 해결하기 위해 Amazon EKS 최적화 AMI를 업데이트했습니다.	2019년 9월 6일

Amazon EKS에서 Kubernetes 1.11 사용 중단 발표	Amazon EKS는 2019년 11월 4일에 Kubernetes 버전 1.11에 대한 지원을 중단했습니다.	2019년 9월 4일
Kubernetes 버전 1.14	새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.14 지원이 추가되었습니다.	2019년 9월 3일
서비스 계정에 대한 IAM 역할	Amazon EKS 클러스터의 서비스 계정에 대한 IAM 역할과 함께 IAM 역할을 Kubernetes 서비스 계정에 연결할 수 있습니다. 이 기능을 사용하면 노드 IAM 역할에 확장된 권한을 더 이상 제공할 필요가 없습니다. 이렇게 하면 해당 노드의 포드가 AWS API를 직접적으로 직접 호출할 수 있습니다.	2019년 9월 3일
Amazon EKS AWS 리전 확장	이제 중동(바레인)(me-south-1) AWS 리전에서 Amazon EKS를 사용할 수 있습니다.	2019년 8월 29일
Amazon EKS 플랫폼 버전 업데이트	CVE-2019-9512 및 CVE-2019-9514 를 해결하기 위한 새로운 플랫폼 버전입니다.	2019년 8월 28일
Amazon EKS 플랫폼 버전 업데이트	CVE-2019-11247 및 CVE-2019-11249 를 해결하기 위한 새로운 플랫폼 버전입니다.	2019년 8월 5일
Amazon EKS 지역 확장	이제 아시아 태평양(홍콩)(ap-east-1) AWS 리전에서 Amazon EKS를 사용할 수 있습니다.	2019년 7월 31일

<u>Amazon EKS에서 더 이상 Kubernetes 1.10을 지원하지 않음</u>	Amazon EKS에서 Kubernetes 버전 1.10이 더 이상 지원되지 않습니다. 서비스 중단이 발생하지 않도록 모든 1.10 클러스터를 버전 1.11 이상으로 업데이트합니다.	2019년 7월 30일
<u>ALB 인그레스 컨트롤러에 대한 주제 추가</u>	Kubernetes에 대한 AWS ALB 수신 컨트롤러는 수신 리소스가 생성될 때 ALB가 생성되도록 하는 컨트롤러입니다.	2019년 7월 11일
<u>새로운 Amazon EKS 최적화 AMI</u>	AMI에서 불필요한 kubect1 바이너리 제거	2019년 7월 3일
<u>Kubernetes 버전 1.13</u>	새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.13 지원이 추가되었습니다.	2019년 6월 18일
<u>AWS-2019-005 에 패치된 새 Amazon EKS 최적화 AMI</u>	Amazon EKS는 링크:security/security-bulletins/AWS-2019-005/[AWS-2019-005,type="marketing"].에 설명된 취약점을 해결하기 위해 Amazon EKS에 최적화된 AMI를 업데이트했습니다.	2019년 6월 17일
<u>Amazon EKS에서 Kubernetes 1.10 지원 중단 발표</u>	Amazon EKS는 2019년 7월 22일에 Kubernetes 버전 1.10에 대한 지원을 중단했습니다.	2019년 5월 21일
<u>Amazon EKS 플랫폼 버전 업데이트</u>	kubelet 인증서에서 사용자 지정 DNS 이름을 지원하고 etcd 성능을 개선하기 위한 새로운 Kubernetes 1.11 및 1.10 클러스터용 플랫폼 버전입니다.	2019년 5월 21일

[eksctl 시작하기](#)

이 시작 가이드에서는 eksctl을 사용하여 Amazon EKS를 시작하는 데 필요한 모든 리소스를 설치할 수 있는 방법에 대해 설명합니다. 이 유틸리티는 Amazon EKS에서 Kubernetes 클러스터를 생성하고 관리하기 위한 간단한 명령줄 유틸리티입니다.

2019년 5월 10일

[AWS CLI get-token 명령](#)

aws eks get-token 명령이 AWS CLI에 추가되었습니다. 클러스터 API 서버 통신에 사용할 클라이언트 보안 토큰을 생성하기 위해 AWS IAM Authenticator for Kubernetes를 더 이상 설치할 필요가 없습니다. 이 새 기능을 사용하려면 AWS CLI 설치를 최신 버전으로 업그레이드하세요. 자세한 내용은 AWS 명령줄 인터페이스 사용 안내서에서 [AWS 명령줄 인터페이스 설치](#)를 참조하세요.

2019년 5월 10일

[Amazon EKS 플랫폼 버전 업데이트](#)

kubelet 인증서에서 사용자 지정 DNS 이름을 지원하고 etcd 성능을 개선하기 위한 새로운 Kubernetes 1.12 클러스터용 플랫폼 버전입니다. 이 버전에서 노드 kubelet 대몬이 몇 초마다 새로운 인증서를 요청하는 버그가 수정되었습니다.

2019년 5월 8일

Prometheus 자습서	Amazon EKS 클러스터에 Prometheus를 배포하는 방법에 대한 주제가 추가되었습니다.	2019년 4월 5일
Amazon EKS 컨트롤 영역 로깅	이 업데이트를 통해 Amazon EKS 제어판에서 감사 및 진단 로그를 직접 가져올 수 있습니다. 해당 계정의 이러한 CloudWatch 로그를 클러스터 보안 및 실행에 대한 참조로 사용할 수 있습니다.	2019년 4월 4일
Kubernetes 버전 1.12	새 클러스터 및 버전 업그레이드에 대한 Kubernetes 버전 1.12 지원이 추가되었습니다.	2019년 3월 28일
App Mesh 시작 가이드 추가	App Mesh 및 Kubernetes를 시작하기 위한 문서가 추가되었습니다.	2019년 3월 27일
Amazon EKS API 서버 엔드포인트 프라이빗 액세스	Amazon EKS 클러스터의 Kubernetes API 서버 엔드포인트에 대한 퍼블릭 액세스를 비활성화하기 위한 문서가 추가되었습니다.	2019년 3월 19일
Kubernetes 지표 서버를 설치하기 위한 주제가 추가됨	Kubernetes 지표 서버는 클러스터에서 리소스 사용량 데이터의 집계자입니다.	2019년 3월 18일
관련 오픈 소스 프로젝트의 목록 추가	이러한 오픈 소스 프로젝트는 Amazon EKS에서 관리하는 클러스터를 비롯하여 AWS에서 실행되는 Kubernetes 클러스터의 기능을 확장합니다.	2019년 3월 15일

[Helm을 로컬로 설치하기 위한 주제 추가](#)

Kubernetes용 helm 패키지 관리자는 Kubernetes 클러스터에서 애플리케이션을 설치하고 관리하는 데 도움이 됩니다. 이 주제에서는 helm 및 tiller 바이너리를 로컬에서 설치하고 실행하는 방법을 설명합니다. 이렇게 하면 로컬 시스템에서 Helm CLI를 사용하여 차트를 설치하고 관리할 수 있습니다.

2019년 3월 11일

[Amazon EKS 플랫폼 버전 업데이트](#)

[CVE-2019-1002100](#)을 해결하기 위해 Amazon EKS Kubernetes 1.11 클러스터를 패치 수준 1.11.8로 업데이트하는 새로운 플랫폼 버전입니다.

2019년 3월 8일

[클러스터 한도 향상](#)

Amazon EKS는 AWS 리전에서 생성할 수 있는 클러스터의 수를 3개에서 50개로 늘렸습니다.

2019년 2월 13일

[Amazon EKS AWS 리전 확장](#)

이제 EU(런던) (eu-west-2), EU(파리)(eu-west-3) 및 아시아 태평양(뭄바이)(ap-south-1) AWS 리전에서 Amazon EKS를 사용할 수 있습니다.

2019년 2월 13일

[ALAS-2019-1156 에 패치된 새 Amazon EKS 최적화 AMI](#)

Amazon EKS는 [ALAS-2019-1156](#)에서 설명하는 취약성을 해결하기 위해 Amazon EKS 최적화 AMI 업데이트했습니다.

2019년 2월 11일

ALAS2-2019-1141 에 패치된 새 Amazon EKS 최적화 AMI	Amazon EKS는 ALAS2-2019-1141 에서 참조하는 CVE를 해결하기 위해 Amazon EKS 최적화 AMI 업데이트했습니다.	2019년 1월 9일
Amazon EKS AWS 리전 확장	이제 아시아 태평양(서울)(ap-northeast-2) AWS 리전에서 Amazon EKS를 사용할 수 있습니다.	2019년 1월 9일
Amazon EKS 지역 확장	이제 EU(프랑크푸르트)(eu-central-1), 아시아 태평양(도쿄)(ap-northeast-1), 아시아 태평양(싱가포르)(ap-southeast-1) 및 아시아 태평양(시드니)(ap-southeast-2) 추가 AWS 리전에서 Amazon EKS를 사용할 수 있습니다.	2018년 12월 19일
Amazon EKS 클러스터 업데이트	Amazon EKS 클러스터 Kubernetes 버전 업데이트 및 노드 교체 에 대한 문서를 추가했습니다.	2018년 12월 12일
Amazon EKS AWS 리전 확장	이제 EU(스톡홀름)(eu-north-1) AWS 리전에서 Amazon EKS를 사용할 수 있습니다.	2018년 12월 11일
Amazon EKS 플랫폼 버전 업데이트	새 플랫폼 버전이 Kubernetes를 패치 수준 1.10.11로 업데이트하여 link:security/security-bulletins/AWS-2018-020/[CVE-2018-1002105,type="marketing"] 을 해결합니다.	2018년 12월 4일

<u>ALB 인그레스 컨트롤러에 대한 버전 1.0.0 지원 추가</u>	ALB 인그레스 컨트롤러는 AWS의 공식적인 지원과 함께 버전 1.0.0을 릴리스합니다.	2018년 11월 20일
<u>CNI 네트워크 구성에 대한 지원 추가</u>	이제 Kubernetes용 Amazon VPC CNI 플러그인 버전 1.2.1에서 보조 포트 네트워크 인터페이스에 대한 사용자 지정 네트워크 구성이 지원됩니다.	2018년 10월 16일
<u>MutatingAdmissionWebhook 및 ValidatingAdmissionWebhook에 대한 지원 추가</u>	이제 Amazon EKS 플랫폼 버전 1.10-eks.2 가 MutatingAdmissionWebhook 및 ValidatingAdmissionWebhook 승인 컨트롤러를 지원합니다.	2018년 10월 10일
<u>파트너 AMI 정보 추가</u>	Canonical은 Amazon EKS와 협력하여 클러스터에서 사용할 수 있는 노드 AMI를 만들었습니다.	2018년 10월 3일
<u>AWS CLI update-kubeconfig 명령에 대한 지침 추가</u>	Amazon EKS는 클러스터에 액세스하는 데 필요한 kubeconfig 파일을 만드는 과정을 단순화하기 위해 update-kubeconfig 를 AWS CLI에 추가했습니다.	2018년 9월 21일
<u>새로운 Amazon EKS 최적화 AMI</u>	Amazon EKS는 다양한 보안 수정 및 AMI 최적화를 제공하기 위해 Amazon EKS 최적화 AMI(GPU 지원 유무와 무관)를 업데이트했습니다.	2018년 9월 13일

Amazon EKS AWS 리전 확장	이제 EU(아일랜드)(eu-west-1) 리전에서 Amazon EKS를 사용할 수 있습니다.	2018년 9월 5일
Amazon EKS 플랫폼 버전 업데이트	Kubernetes 집계 계층 및 Horizontal Pod Autoscaler (HPA)를 지원하는 새로운 플랫폼 버전.	2018년 8월 31일
새로운 Amazon EKS 최적화 AMI 및 GPU 지원	Amazon EKS는 새로운 AWS CloudFormation 노드 템플릿 및 부트스트랩 스크립트 를 사용하도록 Amazon EKS 최적화 AMI를 업데이트했습니다. 또한 새로운 Amazon EKS 최적화 AMI(GPU 지원 포함) 도 이용할 수 있습니다.	2018년 8월 22일
ALAS2-2018-1058 에 패치된 새 Amazon EKS 최적화 AMI	Amazon EKS는 ALAS2-2018-1058 에서 참조하는 CVE를 해결하기 위해 Amazon EKS 최적화 AMI 업데이트했습니다.	2018년 8월 14일
Amazon EKS 최적화 AMI 빌드 스크립트	Amazon EKS는 Amazon EKS 최적화 AMI를 빌드하는 데 사용되는 빌드 스크립트를 오픈 소스로 제공합니다. 이러한 빌드 스크립트를 이제 GitHub에서 사용할 수 있습니다.	2018년 7월 10일
Amazon EKS 최초 릴리스	서비스 출시에 대한 최초 설명서	2018년 6월 5일

EKS 사용 설명서에 기여

AWS는 EKS 사용 설명서의 향상된 기여 경험을 출시했습니다.

이제 GitHub에서 직접 EKS 사용 설명서 소스를 편집할 수 있습니다.

이제 설명서는 마크다운과 유사한 강력한 작성 언어인 AsciiDoc를 사용합니다. AsciiDoc은 간단한 구문과 고급 형식 지정, 상호 참조, 보안 제어와 같은 엔터프라이즈 설명서의 기능을 결합합니다.

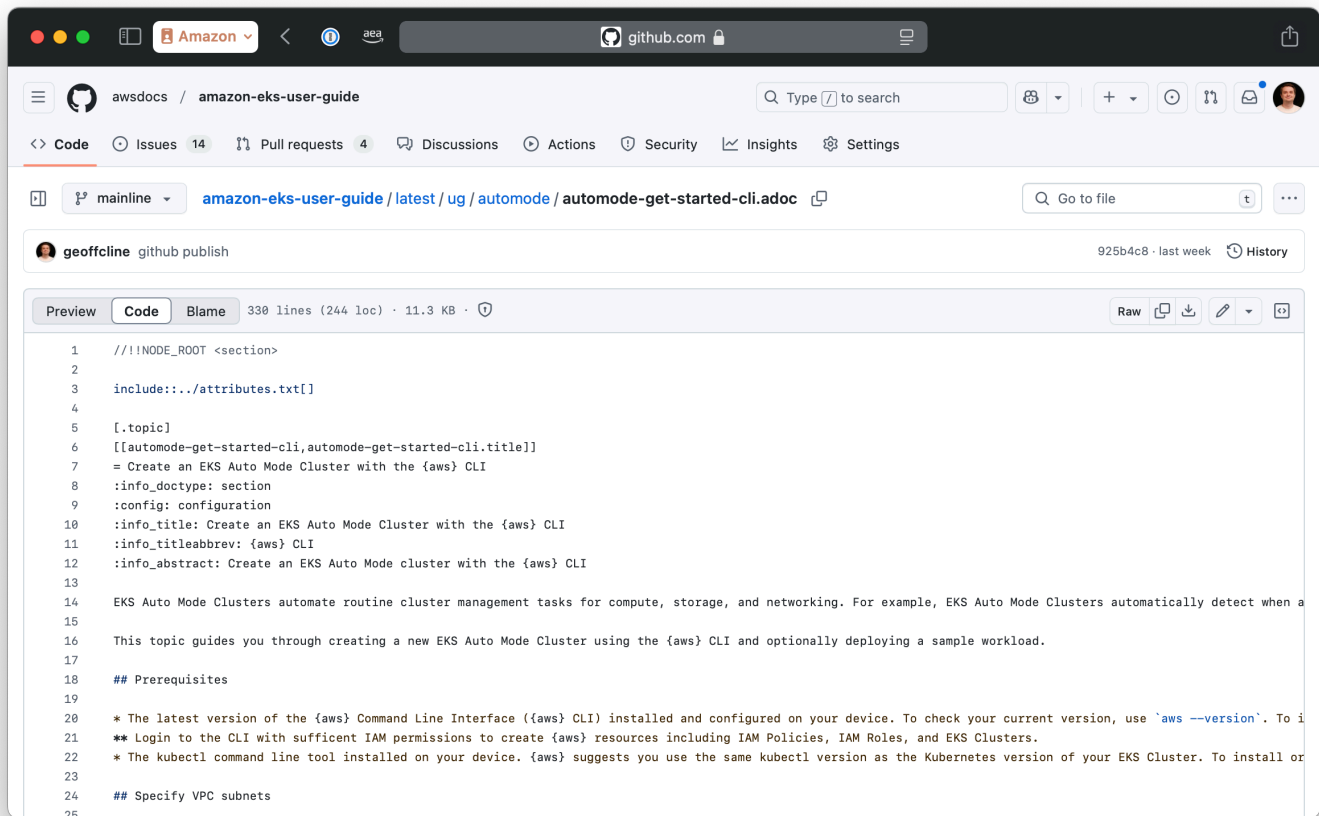
이제 GitHub에서 직접 EKS Docs를 편집할 수 있습니다. 간소화된 프로세스에는 다음이 포함됩니다.

- 더 빠른 풀 요청 처리
- 수동 단계 감소
- 자동화된 콘텐츠 품질 검사

사용자의 기여를 기대합니다.

웹 브라우저에서 단일 페이지 편집

웹 브라우저를 통해 직접 EKS 사용 설명서의 단일 페이지를 쉽게 편집할 수 있습니다.



웹 브라우저에서 여러 페이지를 편집하려면 [the section called “GitHub에서 파일 편집”](#) 섹션을 참조하세요.

사전 조건

- 웹 브라우저에서 변경할 설명서 페이지가 열려 있음
- GitHub에 로그인됨.

절차

1. Amazon EKS 사용 설명서에서 편집하려는 페이지로 이동합니다.
2. 오른쪽 창에서 GitHub에서 이 페이지 편집 링크를 선택합니다.
3. GitHub에서 다음 중 하나를 수행하여 편집기 열기:
 - 키보드에서 e 키 누름.
 - 연필 아이콘을 클릭하고 드롭다운 메뉴에서 제자리에서 편집을 선택합니다.

- 편집을 위한 옵션이 없는 경우 GitHub에 로그인해야 합니다. 변경 사항을 제안하는 데 GitHub 계정에 특별한 권한이 필요하지 않습니다. 하지만 내부 Amazon 기여자는 GitHub 프로파일을 연결해야 합니다.
4. GitHub 편집기에서 콘텐츠를 필요한 대로 변경합니다.
 - 편집기에서는 구문 강조 표시 및 미리 보기 기능을 제공합니다.
 - AsciiDoc 마크업을 사용하여 변경 사항의 형식을 지정할 수 있습니다.
 - `ctrl-f`를 사용하여 찾기/바꾸기 인터페이스를 열 수 있습니다.
 5. (선택 사항) 변경 사항을 미리 봅니다.
 - `preview` 탭을 사용하여 서식이 풍부한 변경 사항을 미리 봅니다.
 - `show diff` 옵션을 사용하여 변경된 섹션을 강조 표시합니다. 제거된 섹션에는 왼쪽 여백에 빨간색 표시기가 있습니다. 새 섹션에는 왼쪽 여백에 녹색 표시기가 있습니다.
 6. 편집을 마치면 편집기 맨 위에서 변경 사항 커밋... 버튼 클릭
 7. 커밋 대화 상자에서:
 - 이메일 주소가 올바른지 확인합니다.
 - 변경 사항을 설명하는 간단하지만 기술적인 커밋 메시지를 추가합니다.
 - 필요한 경우 선택적으로 더 긴 설명을 추가합니다.
 - 새 브랜치 및 풀 요청을 생성하도록 선택합니다.

제안된 변경 사항을 포함하여 풀 요청을 생성했습니다.

풀 요청 개요

PR을 생성하는 경우:

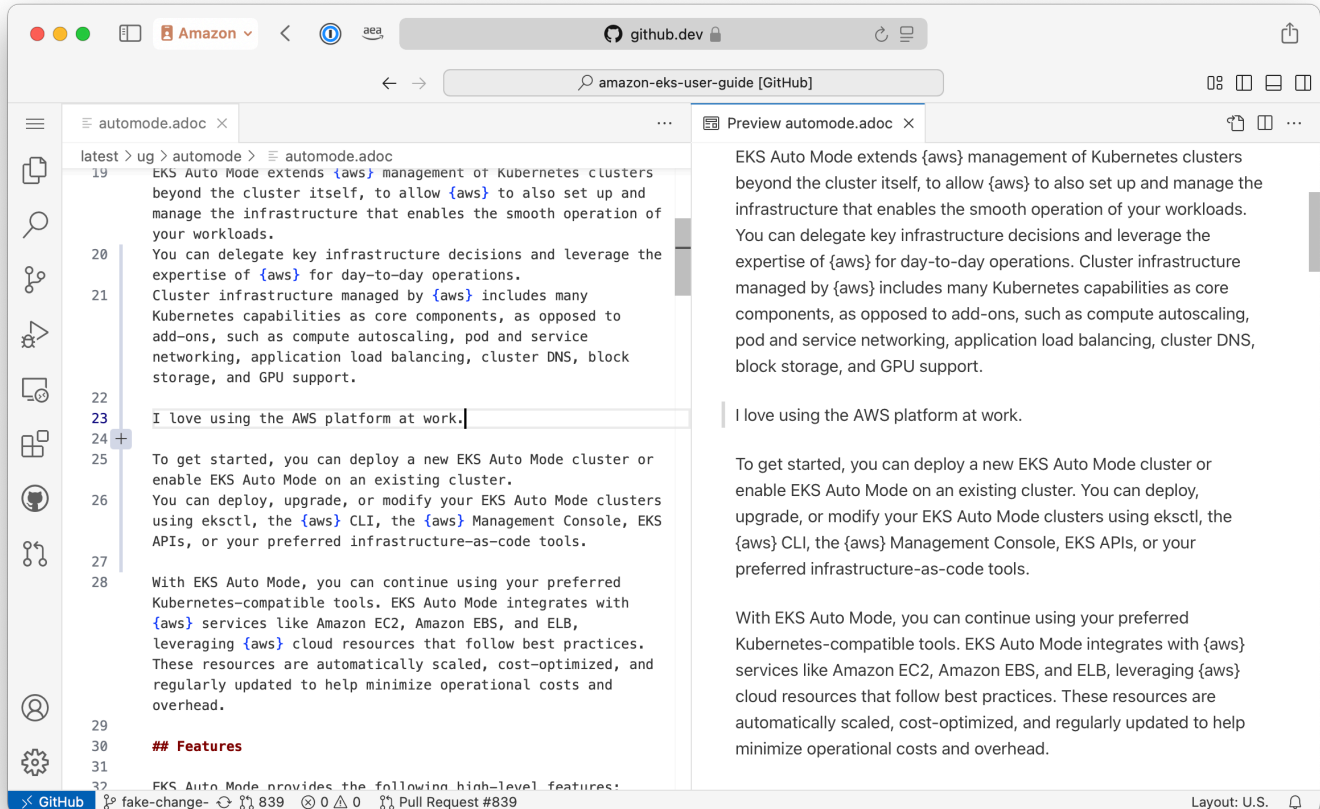
- 리포지토리 유지 관리자가 검토할 수 있도록 변경 사항이 제출됩니다.
- 검토자가 변경 사항에 대한 주석을 추가하고 수정을 요청할 수 있습니다.
- 자동화된 테스트를 실행하여 변경 사항을 검증할 수 있습니다.
- 승인되면 변경 사항을 기본 리포지토리에 병합할 수 있습니다.

풀 요청은 품질을 보장하고 변경 사항이 통합되기 전에 변경 사항에 대해 논의할 수 있는 방법을 제공합니다.

[GitHub Docs에서 풀 요청을 검토하고 승인하는 방법을 알아봅니다.](#)

GitHub 웹 편집기를 사용하여 웹 브라우저에서 여러 파일 편집

여러 페이지에 대한 변경 사항을 제안하거나 새 설명서 페이지를 생성하려면 GitHub.dev 웹 편집기를 사용합니다. 이 웹 편집기는 널리 사용되는 Visual Studio Code 텍스트 편집기에 기반합니다.



사전 조건

- GitHub에 로그인
- Visual Studio Code 편집기에 대한 사용 경험
- Git에 대한 사용 경험

절차

Note

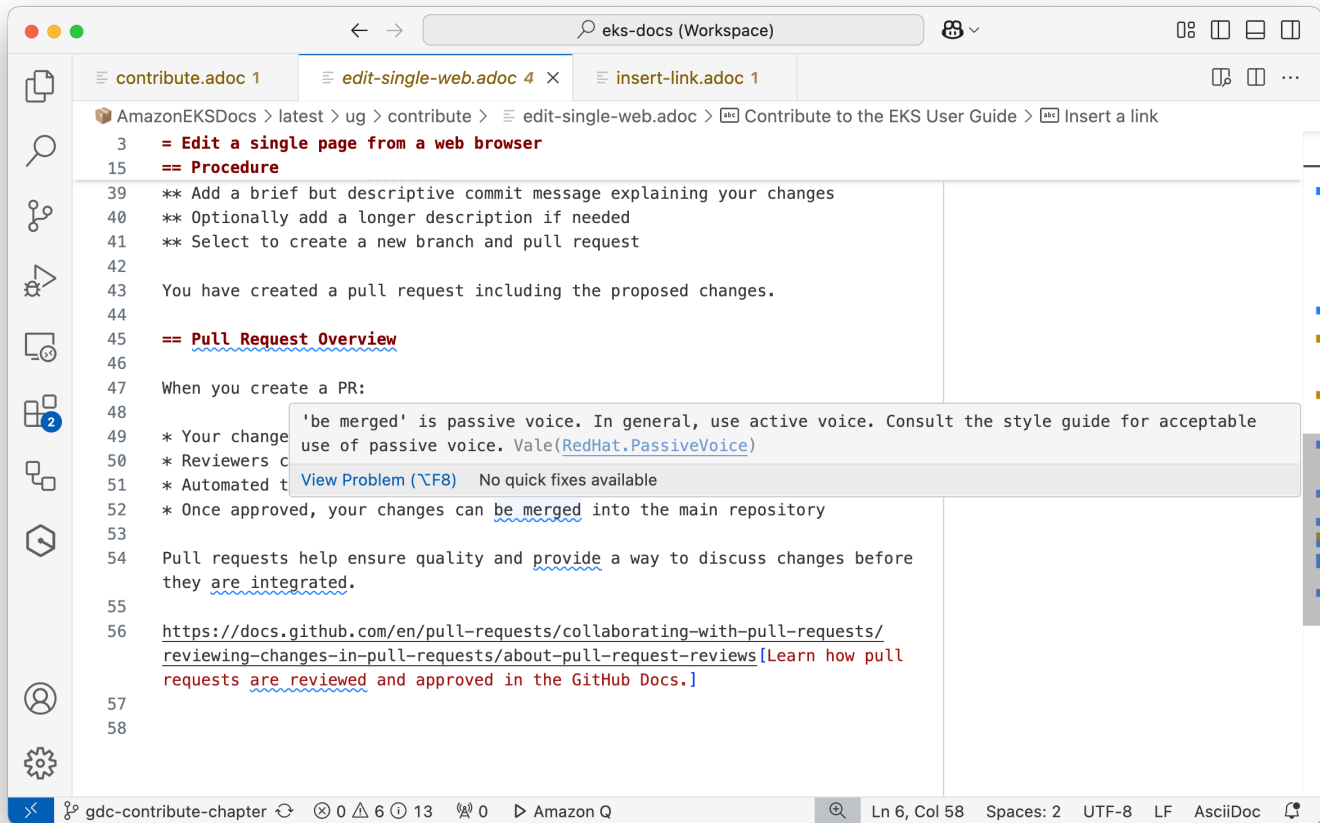
EKS Docs 팀은 텍스트 래핑 및 AsciiDoc 구문 강조 표시와 같은 편집기에 대해 제안된 구성이 포함된 작업 영역 파일을 생성했습니다. 이 작업 영역 파일을 로드하는 것이 좋습니다.

1. GitHub.dev에서 [작업 영역](#)을 여세요.
 - <https://github.dev/awsdocs/amazon-eks-user-guide/blob/mainline/eks-docs.code-workspace?workspace=true> URL을 북마크에 추가할 수 있습니다.
2. (처음 설정만 해당) 자체 GitHub 계정에서 리포지토리의 포크를 생성하라는 프롬프트가 표시될 수 있습니다. 이 프롬프트를 수락하세요. 자세한 내용은 GitHub Docs에서 [About forks](#)를 참조하세요.
3. (처음 설정만 해당) 오른쪽 하단의 프롬프트를 수락하여 AsciiDoc 확장 기능을 설치하세요.
4. latest/ug에 있는 설명서 콘텐츠로 이동합니다.
 - 설명서 파일은 최상위 섹션별로 구성됩니다. 예를 들어 '보안' 장의 페이지의 소스 파일은 'security/' 디렉터리 아래에 있습니다.
5. 설명서 페이지의 미리 보기를 보려면 오른쪽 상단의 사이드에 미리 보기 열기 버튼을 사용하세요. 아이콘에는 작은 돋보기가 포함되어 있습니다.
6. 왼쪽의 소스 제어 탭을 사용하여 변경 사항을 커밋하세요. 자세한 내용은 다음과 같은 Visual Studio Code 설명서를 참조하세요.
 - [Commit changes](#)
 - [풀 요청 생성](#)

풀 요청을 생성하면 Docs 팀이 요청을 검토합니다.

Vale을 로컬에 설치하여 입력할 때 스타일 피드백 보기

입력할 때 스타일 피드백을 볼 수 있습니다. 그러면 어색한 텍스트와 오타를 식별하는 데 도움이 됩니다.



개요:

- Vale CLI는 스타일 가이드를 로드하고 소스 파일에서 해당 가이드를 실행합니다.
- EKS Docs 리포지토리에는 스타일 가이드 및 로컬 규칙을 로드하는 Vale 구성 파일이 포함되어 있습니다.
- Visual Studio(VS) 코드의 Vale 확장 기능은 편집기 내에 Vale 피드백을 표시합니다.

Vale 설치

Vale CLI 설명서의 지침에 따라 [Package Manager를 사용하여 Vale을 설치](#)합니다.

VS Code Vale 확장 기능 설치

1. VS Code를 엽니다.
2. 활동 표시줄에서 확장 기능 아이콘을 클릭하거나 Ctrl+Shift+X를 누릅니다.
3. 'Vale'을 검색합니다.

4. Chris Chinchilla의 'Vale VSCode' 확장 기능에서 설치를 클릭합니다.
5. 프롬프트가 표시되면 VS Code를 다시 로드합니다.

값 동기화

Vale은 프로젝트 루트의 `.vale.ini` 구성 파일을 사용하여 적용할 스타일 규칙을 결정합니다.

1. VS Code를 엽니다.
2. 보기 > 터미널을 클릭하거나 또는 `Ctrl+``를 누릅니다.
3. 필요한 경우 프로젝트 루트 디렉터리로 이동합니다.
4. 명령을 실행합니다.

```
vale sync
```

5. Vale이 스타일 규칙 다운로드 및 동기화를 완료할 때까지 대기

VS Code에서 스타일 피드백 보기

1. VS Code에서 마크다운 또는 AsciiDoc 파일을 엽니다.
2. Vale 확장 기능은 스타일 규칙을 기준으로 텍스트를 자동으로 확인합니다.
3. 스타일 문제는 편집기에서 밑줄로 표시됩니다.
4. 밑줄이 그어진 텍스트 위로 마우스를 대고 특정 스타일 제안을 확인합니다.
5. 제안에 따르거나 스타일 설명서를 참조하여 문제를 해결합니다.

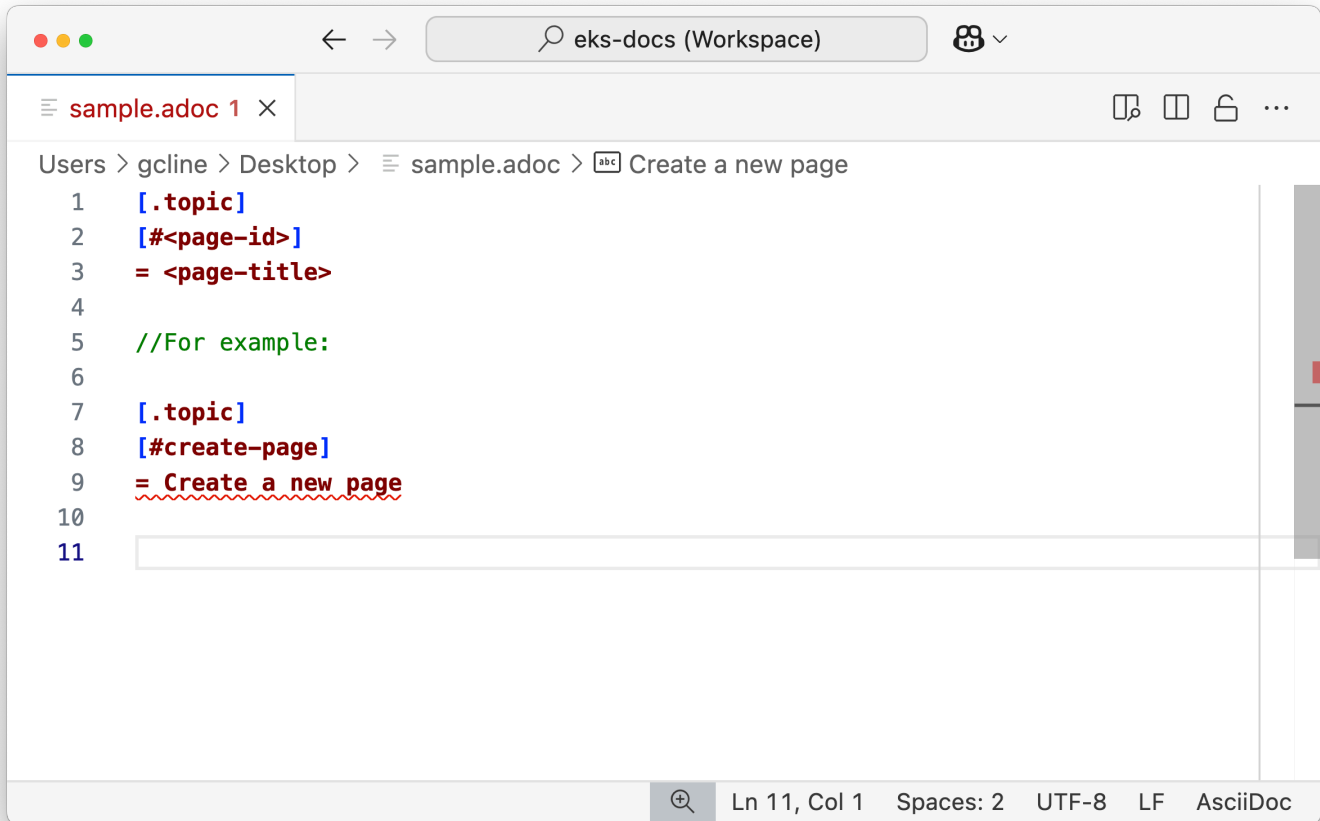
새 페이지 생성

새 문서 페이지 생성 방법을 알아봅니다. 이 주제에는 초기 페이지 메타데이터를 생성하고 설명서 목차에 페이지를 추가하는 지침이 포함되어 있습니다.

페이지 생성

1. 장 디렉터리로 이동하세요. 예를 들어 '보안' 섹션에서 새 페이지를 생성하려면 `latest/ug/security` 디렉터리로 이동하세요.
2. 페이지 ID를 결정하세요. 규칙상 페이지 ID는 모두 소문자이며 -로 분할됩니다. 이 페이지의 ID는 `create-page`입니다.

3. 페이지 ID 및 adoc 확장명을 사용하여 새 파일을 생성하세요. 예: `create-page.adoc`.
4. 이 템플릿을 사용하여 페이지 메타데이터를 삽입하세요.



탐색에 페이지 추가

1. 상위 페이지로 이동하세요. 최상위 섹션의 상위 페이지는 `book.adoc`입니다.
2. 상위 페이지 하단에 하위 페이지를 포함하세요.

```
include::${filename}[leveloffset=+1]
```

예:

```
include::create-page.adoc[leveloffset=+1]
```


링크 삽입

AsciiDoc는 여러 유형의 링크를 지원합니다. 링크가 다양한 환경에서 제대로 작동하려면 올바른 링크 유형을 사용해야 합니다.

EKS 사용 설명서에서 페이지 또는 섹션에 대한 링크

교차 참조(xref)를 사용하여 EKS 사용 설명서와 같은 동일한 설명서 사이트 내 페이지 또는 섹션을 연결합니다. 대상 섹션을 이동하거나 이름을 바꾸면 해당 섹션이 자동으로 업데이트됩니다.

사용자 지정 링크 텍스트 정의

```
xref :section-id[${link-text}]
```

페이지 제목을 링크 텍스트로 사용

각괄호로 묶은 섹션 ID를 사용합니다. 예를 들면 << page-id>>입니다.

AWS 설명서에서 다른 페이지에 대한 링크

1. AWS 설명서 페이지에 대한 링크 찾기
2. <https://docs.aws.amazon.com/> 접두사를 제거하고 경로만 유지하세요. 경로는 a~z 문자로 시작해야 합니다.
3. 아래와 같이 링크를 생성하세요.

```
link :AmazonS3/latest/userguide/create-bucket-overview.html["Create a bucket", type="documentation"]
```

인터넷에 대한 외부 링크

이 형식은 인터넷에 대한 표준 외부 링크를 생성합니다. GitHub의 콘텐츠 또는 Amazon 이외 콘텐츠에 사용합니다.

```
link :https://example.com[Visit Example Site]
```

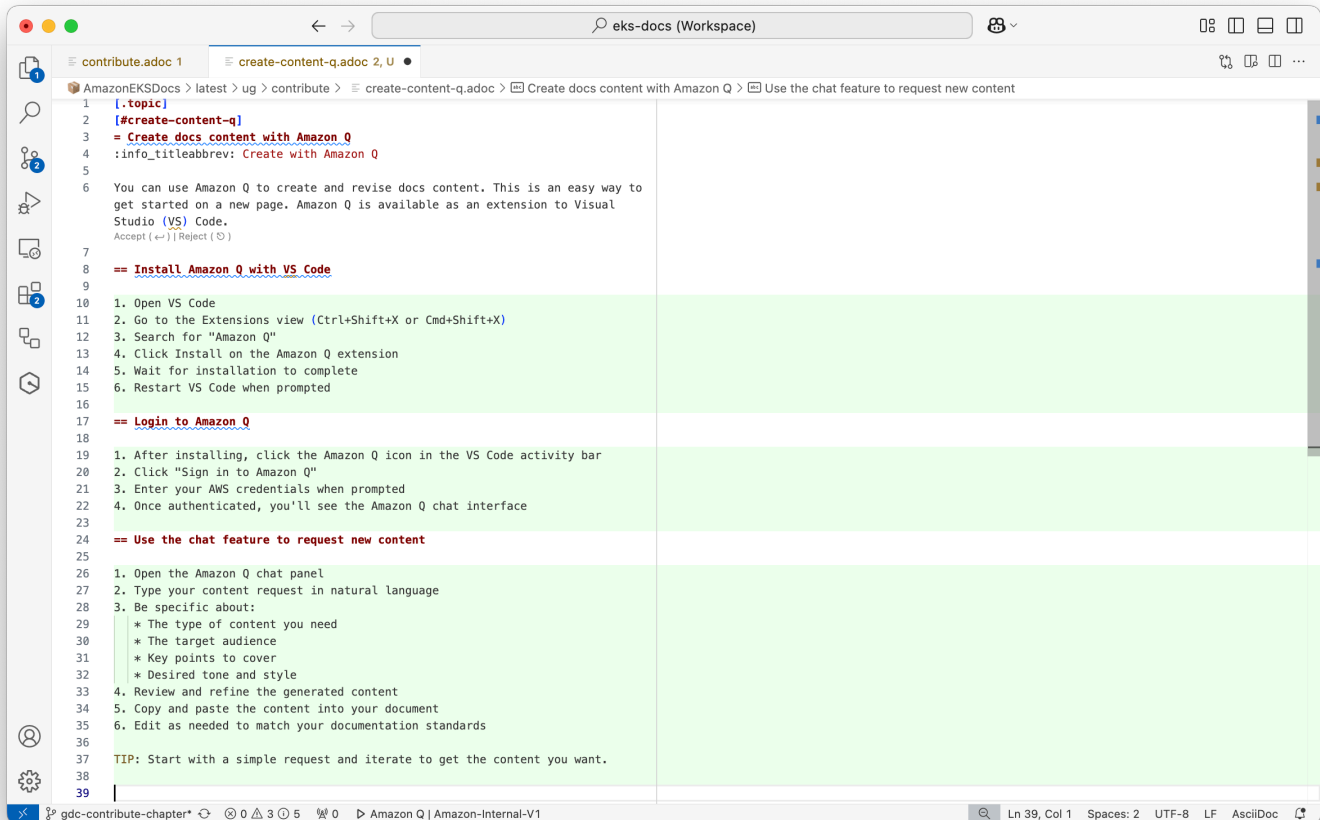
Note

외부 도메인을 허용 목록에 추가합니다. 허용 목록은 `vale/styles/EksDocs/ExternalDomains.yml`에 있습니다.

Amazon Q를 사용하여 설명서 콘텐츠 생성

Amazon Q를 사용하여 설명서 콘텐츠를 생성하고 수정할 수 있습니다. 새 페이지를 시작하는 쉬운 방법입니다. Amazon Q는 Visual Studio(VS) Code에 대한 확장 기능으로 사용할 수 있습니다.

다음 이미지에서 녹색으로 표시된 줄은 Amazon Q에서 생성된 내용입니다.



VS Code를 사용하여 Amazon Q 설치

1. VS Code 열기
2. 확장 기능 보기로 이동(Ctrl+Shift+X 또는 Cmd+Shift+X)
3. 'Amazon Q' 검색
4. Amazon Q 확장 기능에서 설치 클릭
5. 설치가 완료될 때까지 대기
6. 프롬프트가 표시되면 VS Code 다시 시작

Amazon Q에 로그인

1. 설치 후 VS Code 활동 표시줄에서 Amazon Q 아이콘을 선택합니다.
2. Amazon Q에 로그인을 선택합니다.
3. 프롬프트가 표시되면 AWS 자격 증명을 입력합니다.
4. 인증되면 Amazon Q 채팅 인터페이스가 표시됩니다.

Amazon Q를 사용하여 콘텐츠 생성

1. VS Code에서 편집할 파일을 엽니다.
2. 수정하려는 텍스트 또는 새 콘텐츠의 위치를 선택합니다.
3. Ctrl+I 또는 Cmd+I를 누릅니다.
4. 프롬프트에서 구체적으로 다음 항목 지정:
 - 필요한 콘텐츠 유형입니다.
 - 대상 독자입니다.
 - 다루어야 할 주요 사항입니다.
 - 원하는 어조 및 스타일입니다.
5. 인라인 미리 보기에서 생성된 콘텐츠를 검토합니다.
6. Enter를 사용하여 변경 사항 수락 또는 Esc를 사용하여 변경 사항 거부
7. 필요에 따라 추가로 편집합니다.

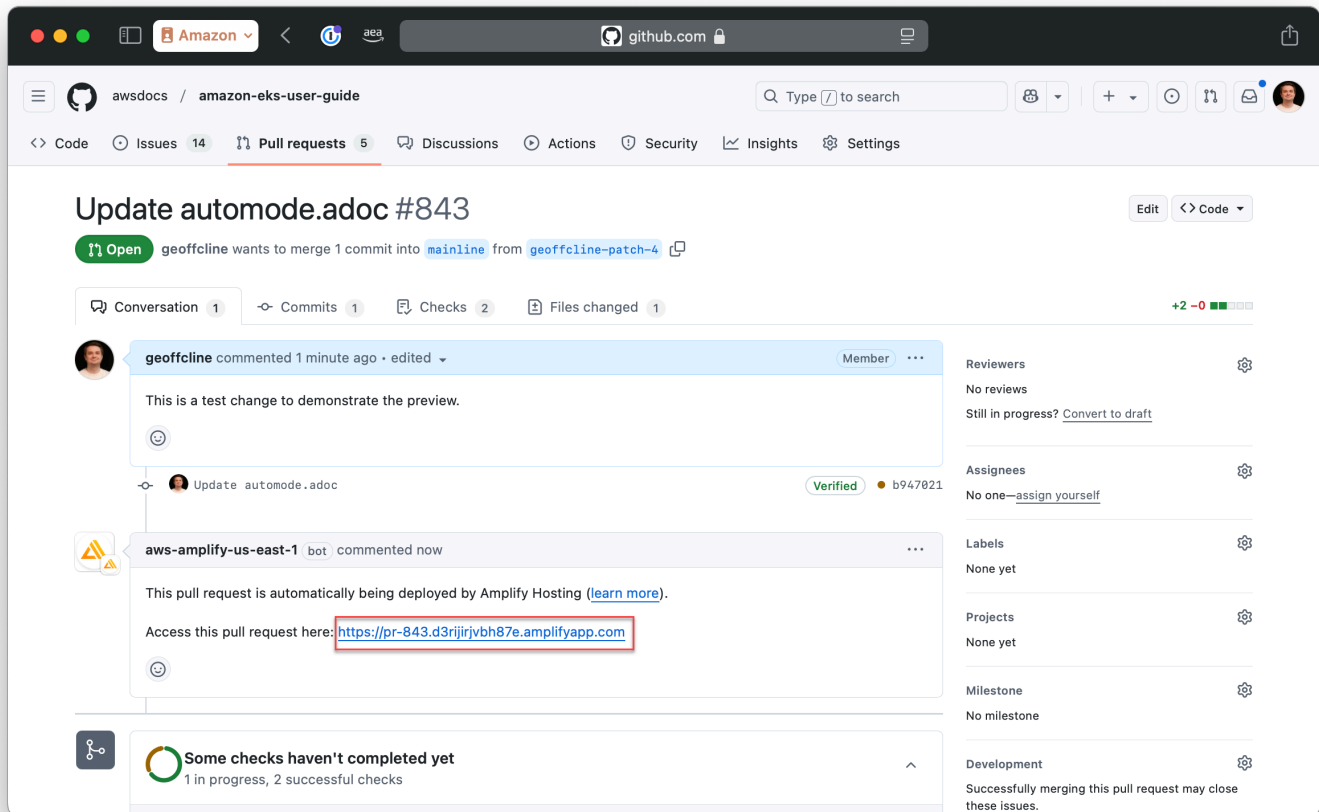
팁

- 간단한 요청으로 시작하고 이를 반복해 원하는 콘텐츠를 가져옵니다.
- 페이지 제목의 첫 번째 초안을 생성하고 해당 내용을 입력해줄 것을 Q에 요청합니다.
- Amazon Q는 마크다운을 출력할 수 있습니다. 그래도 괜찮습니다. AsciiDoc 도구는 대부분의 마크다운 구문을 이해할 수 있습니다.

Amazon Q Developer에 대한 자세한 내용은 [Using Amazon Q Developer in the IDE](#)를 참조하세요.

풀 요청 콘텐츠의 미리 보기 확인

Amazon EKS 사용 설명서 GitHub는 문서 사이트의 미리 보기를 빌드 및 생성하도록 구성되었습니다. 이 미리 보기에는 전체 AWS 테마가 없지만 여기에서 콘텐츠가 제대로 빌드되었고 링크가 작동하는지 확인합니다.



이 미리 보기는 AWS Amplify에서 임시 URL로 호스팅됩니다.

미리 보기 확인

풀 요청을 제출하는 경우 AWS Amplify는 콘텐츠의 미리 보기를 빌드 및 배포하려고 시도합니다.

빌드에 성공하면 aws-amplify-us-east-1에서 미리 보기 링크가 있는 풀 요청에 주석을 추가합니다. 스크린샷에서 빨간색 윤곽선으로 표시된 '여기에서 이 풀 요청에 액세스' 오른쪽의 링크를 선택합니다.

빌드에 실패하면 리포지토리 관리자가 로그를 보고 피드백을 제공할 수 있습니다.

Note

이전에 기여한 적이 없는 경우 프로젝트 유지 관리자가 빌드 실행을 승인해야 할 수 있습니다.

미리 보기 제한 사항

미리 보기는 하나의 큰 HTML 파일로 빌드됩니다. 게시되는 경우 여러 페이지로 표시됩니다.

작동되는 방식:

- 교차 참조(xref)
- 인터넷에 대한 링크
- 이미지
- samples/에서 호스팅된 콘텐츠

작동되지 않는 방식:

- type="documentation"을 사용하여 다른 AWS 콘텐츠에 연결합니다. 이 콘텐츠는 미리 보기 환경에 존재하지 않기 때문입니다.
- 속성 {aws}가 제대로 표시되지 않습니다. 이 값은 환경에 따라 달라집니다.

AsciiDoc 구문 참조

이 페이지는 AsciiDoc 구문에 대한 간략한 개요입니다.

AsciiDoc 구문을 사용하는 것이 좋습니다. AsciiDoc 도구는 제목 및 목록을 포함하여 마크다운 구문을 부분적으로 지원합니다.

고급 구문은 [AsciiDoc 웹 사이트](#)를 검토하세요.

AWS 문서에 대한 링크 구문을 올바르게 가져와야 합니다. [the section called “링크 삽입”](#) 방법을 검토하세요. 링크에 마크다운 구문을 사용할 수 없습니다.

기본 형식 지정

```
*bold text*  
_italic text_
```

```
`monospace text`
```

헤더

```
= Document Title (Header 1)
== Header 2
=== Header 3
==== Header 4
===== Header 5
===== Header 6
```

Lists

정렬되지 않은 목록:

```
- Item 1
- Item 2
-- Subitem 2.1
-- Subitem 2.2
- Item 3
```

정렬된 목록:

```
. First item
. Second item
.. Subitem 2.1
.. Subitem 2.2
. Third item
```

Links

```
External link: https://example.com[Link text]
Internal link: <<page-id>>
Internal link: <<page-id,Link text>>
```

이미지

```
image::image-file.jpg[Alt text]
```

코드 블록

```
[source,python]
----
def hello_world():
    print("Hello, World!")
----
```

표

[기본 테이블을 빌드하는 방법을 알아봅니다.](#)

```
[%header,cols="2"]
|===
|Cell in column 1, header row
|Cell in column 2, header row

|Cell in column 1, body row 1
|Cell in column 2, body row 1

|Cell in column 1, body row 2
|Cell in column 2, body row 2

|===
```

주의 문구

```
NOTE: This is a note admonition.

WARNING: This is a warning admonition.

TIP: This is a tip admonition.

IMPORTANT: This is an important admonition.

CAUTION: This is a caution admonition.
```

미리 보기:

 Note

참고에 관한 주의 문구입니다.

포함 내용

```
include::filename.adoc[]
```