



Amazon EMR on EKS 개발 안내서

# Amazon EMR



# Amazon EMR: Amazon EMR on EKS 개발 안내서

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 트레이드 드레스는 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 Amazon 계열사, 관련 업체 또는 Amazon의 지원 업체 여부에 상관없이 해당 소유자의 자산입니다.

# Table of Contents

|                                                                      |    |
|----------------------------------------------------------------------|----|
| Amazon EMR on EKS란 무엇인가요? .....                                      | 1  |
| Amazon EMR on EKS에 대한 아키텍처 .....                                     | 2  |
| Amazon EMR on EKS 개념 및 용어 이해 .....                                   | 3  |
| Kubernetes 네임스페이스 .....                                              | 3  |
| 가상 클러스터 .....                                                        | 3  |
| 작업 실행 .....                                                          | 4  |
| Amazon EMR 컨테이너 .....                                                | 4  |
| Amazon EMR on EKS 가상 클러스터에 작업을 제출하면 어떻게 되나요? .....                   | 5  |
| Amazon EMR on EKS 시작하기 .....                                         | 6  |
| Spark 애플리케이션 실행 .....                                                | 7  |
| 모범 사례 .....                                                          | 12 |
| 보안 .....                                                             | 12 |
| Pyspark 작업 제출 .....                                                  | 12 |
| 스토리지 .....                                                           | 12 |
| 메타스토어 통합 .....                                                       | 13 |
| 디버깅 .....                                                            | 13 |
| Amazon EMR on EKS 문제 해결 .....                                        | 13 |
| 노드 배치 .....                                                          | 13 |
| 성능 .....                                                             | 13 |
| 비용 최적화 .....                                                         | 14 |
| 사용 AWS Outposts .....                                                | 14 |
| 도커 이미지 사용자 지정 .....                                                  | 15 |
| 도커 이미지를 사용자 지정하는 방법 .....                                            | 15 |
| 사전 조건 .....                                                          | 16 |
| 1단계: Amazon Elastic Container Registry(Amazon ECR)에서 기본 이미지 검색 ..... | 16 |
| 2단계: 기본 이미지 사용자 지정 .....                                             | 17 |
| 3단계: (선택적 권장 사항) 사용자 이미지 검증 .....                                    | 18 |
| 4단계: 사용자 지정 이미지 게시 .....                                             | 19 |
| 5단계: 사용자 지정 이미지를 사용하여 Amazon EMR에서 Spark 워크로드 제출 .....               | 20 |
| 대화형 엔드포인트에 대한 도커 이미지 사용자 지정 .....                                    | 22 |
| 다중 아키텍처 이미지 작업 .....                                                 | 24 |
| 기본 이미지 URI 선택에 관한 세부 정보 .....                                        | 26 |
| Amazon ECR 레지스트리 계정 .....                                            | 27 |
| 이미지 사용자 지정에 대한 고려 사항 .....                                           | 29 |

|                                                      |     |
|------------------------------------------------------|-----|
| Flink 작업 실행 .....                                    | 30  |
| Flink Kubernetes 운영자 .....                           | 30  |
| 설정 .....                                             | 31  |
| Flink Kubernetes 연산자 설치 .....                        | 32  |
| Flink 애플리케이션 실행 .....                                | 34  |
| Flink 애플리케이션을 실행하기 위한 보안 역할 권한 .....                 | 38  |
| 운영자 제거 .....                                         | 40  |
| Flink 네이티브 Kubernetes .....                          | 41  |
| 설정 .....                                             | 41  |
| 시작하기 .....                                           | 42  |
| 보안 요구 사항 .....                                       | 44  |
| Flink 및 FluentD에 대한 Docker 이미지 사용자 지정 .....          | 45  |
| 사전 조건 .....                                          | 45  |
| Amazon Elastic Container Registry에서 기본 이미지 검색 .....  | 46  |
| 기본 이미지 사용자 지정 .....                                  | 46  |
| 사용자 지정 이미지 게시 .....                                  | 47  |
| Flink 워크로드 제출 .....                                  | 48  |
| 모니터링 .....                                           | 48  |
| Amazon Managed Service for Prometheus 사용 .....       | 49  |
| Flink UI 사용 .....                                    | 50  |
| 모니터링 구성 사용 .....                                     | 51  |
| Flink에서고가용성 및 작업 복원력을 지원하는 방법 .....                  | 56  |
| 고가용성 사용 .....                                        | 56  |
| 재시작 시간 최적화 .....                                     | 62  |
| 정상적인 서비스 해제 .....                                    | 68  |
| Autoscaler 사용 .....                                  | 71  |
| 오토스케일러 파라미터 자동 조정 .....                              | 73  |
| Amazon EMR on EKS에서 Flink 작업에 대한 유지 관리 및 문제 해결 ..... | 81  |
| Flink 애플리케이션 유지 관리 .....                             | 82  |
| 문제 해결 .....                                          | 83  |
| 지원되는 릴리스 .....                                       | 87  |
| Spark 작업 실행 .....                                    | 88  |
| StartJobRun .....                                    | 88  |
| 설정 .....                                             | 89  |
| StartJobRun을 사용하여 작업 실행 제출 .....                     | 116 |
| 작업 제출자 분류 사용 .....                                   | 119 |

|                                      |     |
|--------------------------------------|-----|
| Amazon EMR 컨테이너 기본 분류 사용 .....       | 125 |
| Spark 운영자 .....                      | 127 |
| 설정 .....                             | 128 |
| 시작하기 .....                           | 129 |
| 수직적 자동 크기 조정 .....                   | 133 |
| 제거 .....                             | 137 |
| 모니터링 구성을 사용한 Spark 모니터링 .....        | 138 |
| 보안 .....                             | 145 |
| spark-submit .....                   | 156 |
| 설정 .....                             | 156 |
| 시작하기 .....                           | 157 |
| 보안 .....                             | 158 |
| Apache Livy .....                    | 164 |
| 설정 .....                             | 164 |
| 시작하기 .....                           | 165 |
| Spark 애플리케이션 실행 .....                | 170 |
| 설치 제거 .....                          | 172 |
| 보안 .....                             | 172 |
| 설치 속성 .....                          | 182 |
| 일반적인 환경 변수 형식 오류 문제 해결 .....         | 187 |
| 작업 실행 관리 .....                       | 188 |
| CLI를 사용하여 관리 .....                   | 188 |
| Spark SQL 스크립트 실행 .....              | 194 |
| 작업 실행 상태 .....                       | 196 |
| 콘솔에서 작업 보기 .....                     | 197 |
| 일반적인 작업 실행 오류 .....                  | 197 |
| 작업 템플릿 사용 .....                      | 203 |
| 작업 템플릿을 생성 및 사용하여 작업 실행 시작 .....     | 204 |
| 작업 템플릿 파라미터 정의 .....                 | 205 |
| 작업 템플릿에 대한 액세스 제어 .....              | 207 |
| 포드 템플릿 사용 .....                      | 209 |
| 일반적인 시나리오 .....                      | 209 |
| Amazon EMR on EKS에서 포드 템플릿 활성화 ..... | 211 |
| 포드 템플릿 필드 .....                      | 213 |
| sidecar 컨테이너 관련 고려 사항 .....          | 216 |
| 재시도 정책 사용 .....                      | 218 |

|                                      |     |
|--------------------------------------|-----|
| 재시도 정책 설정 .....                      | 218 |
| 정책 상태 검색 .....                       | 220 |
| 작업 모니터링 .....                        | 221 |
| 드라이버 로그 찾기 .....                     | 222 |
| Spark 이벤트 로그 로테이션 사용 .....           | 222 |
| Spark 컨테이너 로그 로테이션 사용 .....          | 223 |
| 수직 자동 조정 사용 .....                    | 225 |
| 설정 .....                             | 225 |
| 시작하기 .....                           | 228 |
| 구성 .....                             | 230 |
| 권장 사항 모니터링 .....                     | 235 |
| 설치 제거 .....                          | 236 |
| 대화형 워크로드 실행 .....                    | 237 |
| 대화형 엔드포인트 개요 .....                   | 237 |
| 대화형 엔드포인트 필수 조건 .....                | 239 |
| AWS CLI .....                        | 239 |
| eksctl .....                         | 239 |
| Amazon EKS 클러스터 .....                | 240 |
| 클러스터에 액세스 권한 부여 .....                | 240 |
| 서비스 계정에 대한 IAM 역할 활성화 .....          | 240 |
| IAM 작업 실행 역할 생성 .....                | 240 |
| 사용자에게 액세스 권한 부여 .....                | 241 |
| Amazon EMR에 Amazon EKS 클러스터 등록 ..... | 241 |
| 로드 밸런서 컨트롤러 .....                    | 241 |
| 인터페이스 엔드포인트 생성 .....                 | 242 |
| 대화형 엔드포인트 생성 .....                   | 242 |
| 사용자 지정 파라미터 지정 .....                 | 243 |
| .....                                | 244 |
| 대화형 엔드포인트 파라미터 .....                 | 244 |
| 대화형 엔드포인트에 대한 설정 구성 .....            | 245 |
| Spark 작업 모니터링 .....                  | 246 |
| 사용자 지정 포드 템플릿 .....                  | 247 |
| JEG 포드를 노드 그룹에 배포 .....              | 248 |
| JEG 구성 옵션 .....                      | 252 |
| PySpark 파라미터 수정 .....                | 252 |
| 사용자 지정 커널 이미지 .....                  | 253 |

|                                                                        |     |
|------------------------------------------------------------------------|-----|
| 대화형 엔드포인트 모니터링 .....                                                   | 254 |
| 예제 .....                                                               | 256 |
| 자체 호스팅 Jupyter Notebook 사용 .....                                       | 257 |
| 보안 그룹 생성 .....                                                         | 257 |
| 대화형 엔드포인트 생성 .....                                                     | 258 |
| 게이트웨이 서버 URL 가져오기 .....                                                | 258 |
| 인증 토큰 가져오기 .....                                                       | 259 |
| 노트북 배포 .....                                                           | 260 |
| 정리 .....                                                               | 265 |
| CLI 명령을 사용하여 대화형 엔드포인트에 대한 정보 가져오기 .....                               | 266 |
| .....                                                                  | 266 |
| 대화형 엔드포인트 나열 .....                                                     | 267 |
| 대화형 엔드포인트 삭제 .....                                                     | 269 |
| 데이터 업로드 .....                                                          | 270 |
| 사전 조건 .....                                                            | 270 |
| 시작하기 .....                                                             | 270 |
| 작업 모니터링 .....                                                          | 273 |
| Amazon CloudWatch Events를 사용하여 작업 모니터링 .....                           | 273 |
| CloudWatch Events를 통해 Amazon EMR on EKS 자동화 .....                      | 274 |
| 예제: Lambda를 간접 호출하는 규칙 설정 .....                                        | 275 |
| Amazon CloudWatch Events를 사용하여 재시도 정책을 포함하는 작업의 드라이버 포드 모니터<br>링 ..... | 276 |
| 가상 클러스터 관리 .....                                                       | 277 |
| 가상 클러스터 생성 .....                                                       | 277 |
| 가상 클러스터 나열 .....                                                       | 278 |
| 가상 클러스터 설명 .....                                                       | 279 |
| 가상 클러스터 삭제 .....                                                       | 279 |
| 가상 클러스터 상태 .....                                                       | 279 |
| 자습서 .....                                                              | 280 |
| Delta Lake 사용 .....                                                    | 280 |
| Iceberg 사용 .....                                                       | 281 |
| 카탈로그 통합을 위한 Spark 세션 구성 .....                                          | 282 |
| PyFlink 사용 .....                                                       | 283 |
| Flink에서 AWS Glue 사용 .....                                              | 284 |
| Apache Hudi 사용 .....                                                   | 287 |
| Apache Hudi 작업 제출 .....                                                | 287 |

|                                                      |     |
|------------------------------------------------------|-----|
| Spark RAPIDS 사용 .....                                | 291 |
| Redshift에서 Spark 사용 .....                            | 295 |
| Spark 애플리케이션 시작 .....                                | 295 |
| Amazon Redshift에 대한 인증 .....                         | 296 |
| Amazon Redshift에 대한 읽고 쓰기 .....                      | 299 |
| 고려 사항 .....                                          | 300 |
| Volcano 사용 .....                                     | 301 |
| 개요 .....                                             | 301 |
| 설치 .....                                             | 302 |
| 제출: Spark 운영자 .....                                  | 303 |
| 제출: spark-submit .....                               | 305 |
| YuniKorn 사용 .....                                    | 306 |
| 개요 .....                                             | 306 |
| 클러스터 생성 .....                                        | 306 |
| YuniKorn 설치 .....                                    | 308 |
| 제출: Spark 운영자 .....                                  | 309 |
| 제출: spark-submit .....                               | 311 |
| 보안 .....                                             | 12  |
| 모범 사례 .....                                          | 315 |
| 최소 권한의 원칙 적용 .....                                   | 315 |
| 엔드포인트에 대한 액세스 제어 목록 .....                            | 315 |
| 사용자 지정 이미지에 대한 최신 보안 업데이트 받기 .....                   | 315 |
| 포트 보안 인증 액세스 제한 .....                                | 316 |
| 신뢰할 수 없는 애플리케이션 코드 격리 .....                          | 316 |
| 역할 기반 액세스 제어(RBAC) 권한 .....                          | 316 |
| 노드 그룹 IAM 역할 또는 인스턴스 프로파일 보안 인증에 대한 액세스를 제한합니다. .... | 317 |
| 데이터 보호 .....                                         | 317 |
| 저장 시 암호화 .....                                       | 318 |
| 전송 중 암호화 .....                                       | 320 |
| 자격 증명 및 액세스 관리 .....                                 | 321 |
| 대상 .....                                             | 321 |
| ID를 통한 인증 .....                                      | 322 |
| 정책을 사용하여 액세스 관리 .....                                | 323 |
| Amazon EMR on EKS에서 IAM을 사용하는 방법 .....               | 324 |
| 서비스 연결 역할 사용 .....                                   | 329 |
| Amazon EMR on EKS 관리형 정책 .....                       | 333 |



|                                                      |     |
|------------------------------------------------------|-----|
| Amazon EMR on EKS에서 작업 실행 역할 사용 .....                | 334 |
| ID 기반 정책 예시 .....                                    | 336 |
| 태그 기반 액세스 제어를 위한 정책 .....                            | 339 |
| 문제 해결 .....                                          | 343 |
| AWS Lake Formation에서 Amazon EMR on EKS 사용 .....      | 344 |
| Amazon EMR on EKS가 AWS Lake Formation과 작동하는 방식 ..... | 345 |
| Amazon EMR on EKS를 사용한 Lake Formation 활성화 .....      | 346 |
| 고려 사항 및 제한 사항 .....                                  | 354 |
| 문제 해결 .....                                          | 357 |
| 로그 및 모니터링 .....                                      | 359 |
| 로그 암호화 .....                                         | 359 |
| CloudTrail 로그 .....                                  | 363 |
| S3 Access Grants .....                               | 365 |
| 개요 .....                                             | 365 |
| 클러스터 시작 .....                                        | 366 |
| 고려 사항 .....                                          | 367 |
| 규정 준수 검증 .....                                       | 367 |
| 복원력 .....                                            | 368 |
| 인프라 보안 .....                                         | 368 |
| 구성 및 취약성 분석 .....                                    | 368 |
| 인터페이스 VPC 엔드포인트 .....                                | 369 |
| Amazon EMR on EKS에 대한 VPC 엔드포인트 정책 생성 .....          | 369 |
| 크로스 계정 액세스 .....                                     | 372 |
| 사전 조건 .....                                          | 372 |
| 크로스 계정 Amazon S3 버킷 또는 DynamoDB 테이블에 액세스하는 방법 .....  | 373 |
| 리소스에 태그 지정 .....                                     | 378 |
| 태그 기본 사항 .....                                       | 378 |
| 리소스에 태그 지정 .....                                     | 379 |
| 태그 제한 .....                                          | 379 |
| AWS CLI 및 Amazon EMR on EKS API를 사용하여 태그 작업 .....    | 380 |
| 문제 해결 .....                                          | 13  |
| PVC 작업 실패 .....                                      | 382 |
| Verification(확인) .....                               | 382 |
| 패치 .....                                             | 383 |
| 수동 패치 .....                                          | 386 |
| 수직 자동 조정 실패 .....                                    | 389 |

|                                 |     |
|---------------------------------|-----|
| 403 금지됨 오류 .....                | 389 |
| 네임스페이스를 찾을 수 없음 .....           | 389 |
| Docker 보안 인증 오류 .....           | 389 |
| Spark 운영자 실패 .....              | 390 |
| 차트 Helm 설치 실패 .....             | 390 |
| 지원되지 않는 파일 시스템 예외 .....         | 390 |
| 서비스 엔드포인트 및 할당량 .....           | 392 |
| 서비스 엔드포인트 .....                 | 392 |
| Service Quotas .....            | 394 |
| 할당량 보기 및 할당량 증가 요청 .....        | 396 |
| 릴리스 버전 .....                    | 397 |
| 7.13.0 릴리스 .....                | 398 |
| 출시 .....                        | 398 |
| 릴리스 노트 .....                    | 400 |
| 변경 사항 및 기능 .....                | 402 |
| emr-7.13.0-최신 .....             | 402 |
| emr-7.13.0-20260410 .....       | 402 |
| emr-7.13.0-flink-latest .....   | 402 |
| emr-7.13.0-flink-20260410 ..... | 402 |
| 7.12.0 릴리스 .....                | 403 |
| 출시 .....                        | 403 |
| 릴리스 노트 .....                    | 405 |
| 변경 사항 및 기능 .....                | 406 |
| emr-7.12.0-최신 .....             | 406 |
| emr-7.12.0-20251111 .....       | 407 |
| emr-7.12.0-flink-latest .....   | 407 |
| emr-7.12.0-flink-20251111 ..... | 407 |
| 7.11.0 릴리스 .....                | 407 |
| 출시 .....                        | 407 |
| 릴리스 노트 .....                    | 409 |
| 변경 사항 및 기능 .....                | 411 |
| emr-7.11.0-최신 .....             | 411 |
| emr-7.11.0-20251020 .....       | 411 |
| emr-7.11.0-flink-latest .....   | 411 |
| emr-7.11.0-flink-20251020 ..... | 412 |
| 7.10.0 릴리스 .....                | 412 |

|                                 |     |
|---------------------------------|-----|
| 출시 .....                        | 412 |
| 릴리스 노트 .....                    | 414 |
| 변경 사항 및 기능 .....                | 415 |
| emr-7.10.0-최신 .....             | 415 |
| emr-7.10.0-20250801 .....       | 415 |
| emr-7.10.0-flink-latest .....   | 416 |
| emr-7.10.0-flink-20250801 ..... | 416 |
| 7.9.0 릴리스 .....                 | 416 |
| 출시 .....                        | 416 |
| 릴리스 노트 .....                    | 418 |
| 변경 사항 .....                     | 419 |
| emr-7.9.0-최신 .....              | 420 |
| emr-7.9.0-20250425 .....        | 420 |
| emr-7.9.0-flink-latest .....    | 420 |
| emr-7.9.0-flink-20250425 .....  | 420 |
| 7.8.0 릴리스 .....                 | 421 |
| 출시 .....                        | 421 |
| 릴리스 노트 .....                    | 422 |
| 변경 사항 .....                     | 424 |
| emr-7.8.0-최신 .....              | 424 |
| emr-7.8.0-20250228 .....        | 424 |
| emr-7.8.0-flink-latest .....    | 425 |
| emr-7.8.0-flink-20250228 .....  | 425 |
| 7.7.0 릴리스 .....                 | 425 |
| 출시 .....                        | 425 |
| 릴리스 노트 .....                    | 427 |
| 변경 사항 .....                     | 428 |
| emr-7.7.0-최신 .....              | 429 |
| emr-7.7.0-20250131 .....        | 429 |
| emr-7.7.0-flink-latest .....    | 429 |
| emr-7.7.0-flink-20250131 .....  | 429 |
| 7.6.0 릴리스 .....                 | 430 |
| 출시 .....                        | 430 |
| 릴리스 노트 .....                    | 431 |
| 특성 .....                        | 433 |
| 변경 사항 .....                     | 433 |

|                                |     |
|--------------------------------|-----|
| emr-7.6.0-최신 .....             | 433 |
| emr-7.6.0-20241213 .....       | 434 |
| emr-7.6.0-flink-latest .....   | 434 |
| emr-7.6.0-flink-20241213 ..... | 434 |
| 7.5.0 릴리스 .....                | 434 |
| 출시 .....                       | 435 |
| 릴리스 노트 .....                   | 435 |
| 7.4.0 릴리스 .....                | 435 |
| 출시 .....                       | 435 |
| 릴리스 노트 .....                   | 435 |
| 7.3.0 릴리스 .....                | 436 |
| 출시 .....                       | 436 |
| 릴리스 노트 .....                   | 437 |
| 특성 .....                       | 439 |
| 변경 사항 .....                    | 439 |
| emr-7.3.0-latest .....         | 440 |
| emr-7.3.0-20240920 .....       | 440 |
| emr-7.3.0-flink-latest .....   | 440 |
| emr-7.3.0-flink-29240920 ..... | 440 |
| 7.2.0 릴리스 .....                | 441 |
| 출시 .....                       | 441 |
| 릴리스 노트 .....                   | 442 |
| 특성 .....                       | 444 |
| emr-7.2.0-latest .....         | 445 |
| emr-7.2.0-20240610 .....       | 445 |
| emr-7.2.0-flink-latest .....   | 445 |
| emr-7.2.0-flink-20240610 ..... | 445 |
| 7.1.0 릴리스 .....                | 446 |
| 출시 .....                       | 446 |
| 릴리스 노트 .....                   | 447 |
| 특성 .....                       | 449 |
| emr-7.1.0-latest .....         | 449 |
| emr-7.1.0-20240321 .....       | 449 |
| emr-7.1.0-flink-latest .....   | 450 |
| emr-7.1.0-flink-20240321 ..... | 450 |
| 7.0.0 릴리스 .....                | 450 |

|                                 |     |
|---------------------------------|-----|
| 출시 .....                        | 450 |
| 릴리스 노트 .....                    | 452 |
| 특성 .....                        | 453 |
| 변경 사항 .....                     | 453 |
| emr-7.0.0-latest .....          | 454 |
| emr-7.0.0-2024321 .....         | 454 |
| emr-7.0.0-20231211 .....        | 454 |
| emr-7.0.0-flink-latest .....    | 454 |
| emr-7.0.0-flink-2024321 .....   | 455 |
| emr-7.0.0-flink-20231211 .....  | 455 |
| 6.15.0 릴리스 .....                | 455 |
| 출시 .....                        | 455 |
| 릴리스 노트 .....                    | 457 |
| 특성 .....                        | 458 |
| emr-6.15.0-latest .....         | 459 |
| emr-6.15.0-20240105 .....       | 459 |
| emr-6.15.0-20231109 .....       | 459 |
| emr-6.15.0-flink-latest .....   | 459 |
| emr-6.15.0-flink-20240105 ..... | 460 |
| emr-6.15.0-flink-20231109 ..... | 460 |
| 6.14.0 릴리스 .....                | 460 |
| 출시 .....                        | 460 |
| 릴리스 노트 .....                    | 462 |
| 특성 .....                        | 463 |
| emr-6.14.0-latest .....         | 463 |
| emr-6.14.0-20231005 .....       | 463 |
| 6.13.0 릴리스 .....                | 464 |
| 출시 .....                        | 464 |
| 릴리스 노트 .....                    | 465 |
| 특성 .....                        | 467 |
| emr-6.13.0-latest .....         | 467 |
| emr-6.13.0-20230814 .....       | 467 |
| 6.12.0 릴리스 .....                | 468 |
| 출시 .....                        | 468 |
| 릴리스 노트 .....                    | 468 |
| 특성 .....                        | 470 |

|                           |     |
|---------------------------|-----|
| emr-6.12.0-latest .....   | 470 |
| emr-6.12.0-20240321 ..... | 470 |
| emr-6.12.0-20230701 ..... | 471 |
| 6.11.0 releases .....     | 471 |
| 출시 .....                  | 471 |
| 릴리스 노트 .....              | 472 |
| 특성 .....                  | 473 |
| emr-6.11.0-latest .....   | 474 |
| emr-6.11.0-20230905 ..... | 474 |
| emr-6.11.0-20230509 ..... | 474 |
| 6.10.0 릴리스 .....          | 474 |
| emr-6.10.0-latest .....   | 477 |
| emr-6.10.0-20230905 ..... | 477 |
| emr-6.10.0-20230624 ..... | 477 |
| emr-6.10.0-20230421 ..... | 478 |
| emr-6.10.0-20230403 ..... | 478 |
| emr-6.10.0-20230220 ..... | 478 |
| 6.9.0 릴리스 .....           | 478 |
| emr-6.9.0-latest .....    | 481 |
| emr-6.9.0-20230905 .....  | 481 |
| emr-6.9.0-20230624 .....  | 482 |
| emr-6.9.0-20221108 .....  | 482 |
| 6.8.0 릴리스 .....           | 482 |
| emr-6.8.0-latest .....    | 486 |
| emr-6.8.0-20230905 .....  | 486 |
| emr-6.8.0-20230624 .....  | 486 |
| emr-6.8.0-20221219 .....  | 486 |
| emr-6.8.0-20220802 .....  | 486 |
| 6.7.0 릴리스 .....           | 487 |
| emr-6.7.0-latest .....    | 488 |
| emr-6.7.0-20240321 .....  | 489 |
| emr-6.7.0-20230624 .....  | 489 |
| emr-6.7.0-20221219 .....  | 489 |
| emr-6.7.0-20220630 .....  | 489 |
| 6.6.0 릴리스 .....           | 490 |
| emr-6.6.0-latest .....    | 491 |

|                           |     |
|---------------------------|-----|
| emr-6.6.0-20240321 .....  | 491 |
| emr-6.6.0-20230624 .....  | 492 |
| emr-6.6.0-20221219 .....  | 492 |
| emr-6.6.0-20220411 .....  | 492 |
| 6.5.0 릴리스 .....           | 492 |
| emr-6.5.0-latest .....    | 494 |
| emr-6.5.0-20240321 .....  | 494 |
| emr-6.5.0-20221219 .....  | 494 |
| emr-6.5.0-20220802 .....  | 494 |
| emr-6.5.0-20211119 .....  | 495 |
| 6.4.0 릴리스 .....           | 495 |
| emr-6.4.0-latest .....    | 496 |
| emr-6.4.0-20240321 .....  | 496 |
| emr-6.4.0-20221219 .....  | 497 |
| emr-6.4.0-20210830 .....  | 497 |
| 6.3.0 릴리스 .....           | 497 |
| emr-6.3.0-latest .....    | 498 |
| emr-6.3.0-20240321 .....  | 499 |
| emr-6.3.0-20220802 .....  | 499 |
| emr-6.3.0-20211008 .....  | 499 |
| emr-6.3.0-20210802 .....  | 499 |
| emr-6.3.0-20210429 .....  | 500 |
| 6.2.0 릴리스 .....           | 500 |
| emr-6.2.0-latest .....    | 501 |
| emr-6.2.0-20240321 .....  | 501 |
| emr-6.2.0-20220802 .....  | 502 |
| emr-6.2.0-20211008 .....  | 502 |
| emr-6.2.0-20210802 .....  | 502 |
| emr-6.2.0-20210615 .....  | 502 |
| emr-6.2.0-20210129 .....  | 503 |
| emr-6.2.0-20201218 .....  | 503 |
| emr-6.2.0-20201201 .....  | 503 |
| 5.36.0 릴리스 .....          | 503 |
| emr-5.36.0-latest .....   | 505 |
| emr-5.36.0-20240321 ..... | 505 |
| emr-5.36.0-20221219 ..... | 505 |

|                           |      |
|---------------------------|------|
| emr-5.36.0-20220620 ..... | 505  |
| emr-5.36.0-20220525 ..... | 506  |
| 5.35.0 릴리스 .....          | 506  |
| emr-5.35.0-latest .....   | 507  |
| emr-5.35.0-20240321 ..... | 507  |
| emr-5.35.0-20221219 ..... | 508  |
| emr-5.35.0-20220802 ..... | 508  |
| emr-5.35.0-20220307 ..... | 508  |
| 5.34 릴리스 .....            | 508  |
| emr-5.34.0-latest .....   | 509  |
| emr-5.34.0-20240321 ..... | 510  |
| emr-5.34.0-20220802 ..... | 510  |
| emr-5.34.0-20211208 ..... | 510  |
| 5.33.0 릴리스 .....          | 510  |
| emr-5.33.0-latest .....   | 512  |
| emr-5.33.0-20240321 ..... | 512  |
| emr-5.33.0-20221219 ..... | 512  |
| emr-5.33.0-20220802 ..... | 512  |
| emr-5.33.0-20211008 ..... | 513  |
| emr-5.33.0-20210802 ..... | 513  |
| emr-5.33.0-20210615 ..... | 513  |
| emr-5.33.0-20210323 ..... | 513  |
| 5.32.0 릴리스 .....          | 514  |
| emr-5.32.0-latest .....   | 515  |
| emr-5.32.0-20240321 ..... | 515  |
| emr-5.32.0-20220802 ..... | 516  |
| emr-5.32.0-20211008 ..... | 516  |
| emr-5.32.0-20210802 ..... | 516  |
| emr-5.32.0-20210615 ..... | 516  |
| emr-5.32.0-20210129 ..... | 517  |
| emr-5.32.0-20201218 ..... | 517  |
| emr-5.32.0-20201201 ..... | 517  |
| 문서 이력 .....               | 518  |
| .....                     | dxxi |



# Amazon EMR on EKS란 무엇인가요?

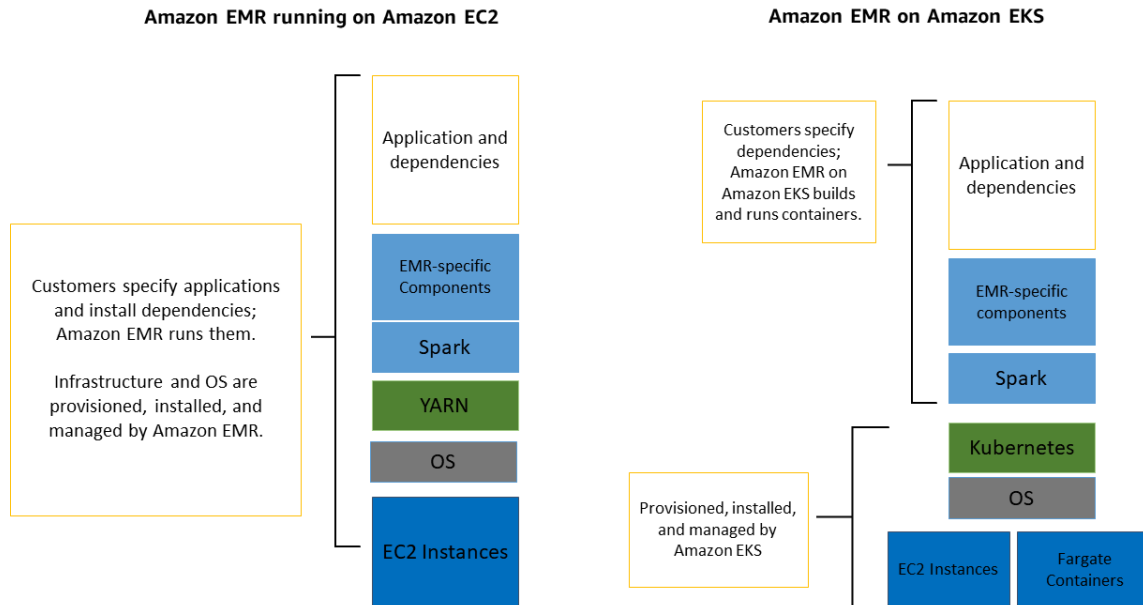
Amazon EMR on EKS는 Amazon Elastic Kubernetes Service(Amazon EKS)에서 오픈 소스 빅 데이터 프레임워크를 실행할 수 있는 Amazon EMR용 배포 옵션을 제공합니다. 이 배포 옵션을 사용하면 Amazon EMR on EKS에서 오픈 소스 애플리케이션용 컨테이너를 구축, 구성 및 관리하는 동안 분석 워크로드 실행에 집중할 수 있습니다.

이미 Amazon EMR을 사용하고 있다면 이제 동일한 Amazon EKS 클러스터에서 다른 유형의 애플리케이션과 함께 Amazon EMR 기반 애플리케이션을 실행할 수 있습니다. 또한 이 배포 옵션을 사용하면 리소스 사용률이 향상되고 여러 가용 영역 전반의 인프라 관리가 간소화됩니다. 이미 Amazon EKS에서 빅 데이터 프레임워크를 실행하고 있다면 이제 Amazon EMR을 사용하여 프로비저닝 및 관리를 자동화하고 Apache Spark를 더 빠르게 실행할 수 있습니다.

Amazon EMR on EKS를 사용하면 팀이 더 효율적으로 협업하고 대량의 데이터를 더 쉽고 비용 효율적으로 처리할 수 있습니다.

- 인프라를 프로비저닝할 필요 없이 공통 리소스 풀에서 애플리케이션을 실행할 수 있습니다. [Amazon EMR Studio](#) 및 AWS SDK 또는를 사용하여 EKS 클러스터에서 실행되는 분석 애플리케이션을 개발, 제출 및 진단 AWS CLI 할 수 있습니다. 자체 관리형 Apache Airflow 또는 Amazon Managed Workflows for Apache Airflow(MWAA)를 사용하여 Amazon EMR on EKS에서 예약된 작업을 실행할 수 있습니다.
- 인프라 팀은 공통 컴퓨팅 플랫폼을 중앙에서 관리하여 Amazon EMR 워크로드를 다른 컨테이너 기반 애플리케이션과 통합할 수 있습니다. 일반적인 Amazon EKS 도구를 사용하여 인프라 관리를 간소화하고 다양한 버전의 오픈 소스 프레임워크가 필요한 워크로드에 대해 공유 클러스터를 활용할 수 있습니다. 또한 자동화된 Kubernetes 클러스터 관리 및 OS 패칭을 통해 운영 오버헤드를 줄일 수 있습니다. Amazon EC2 및를 사용하면 성능 AWS Fargate, 운영 또는 재무 요구 사항을 충족하기 위해 여러 컴퓨팅 리소스를 활성화할 수 있습니다.

다음 다이어그램은 Amazon EMR의 두 가지 배포 모델을 보여줍니다.



## 주제

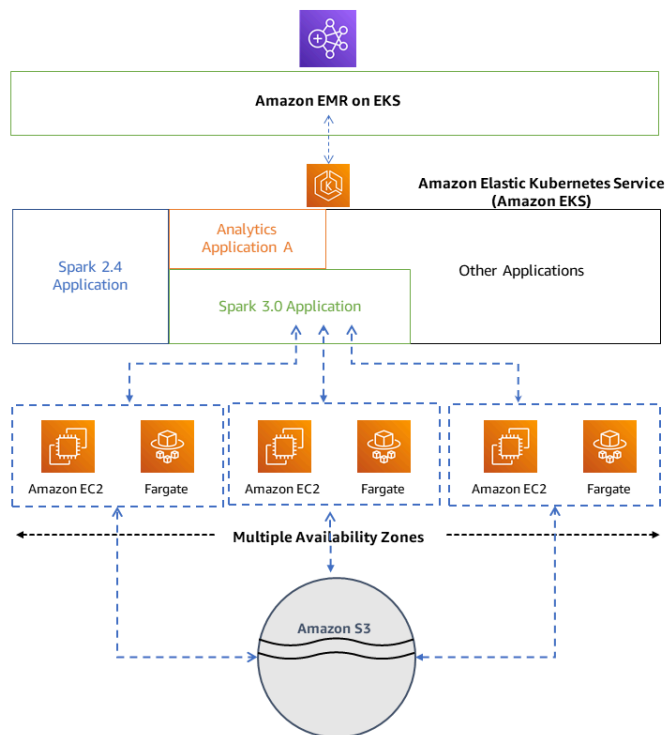
- [Amazon EMR on EKS에 대한 아키텍처](#)
- [Amazon EMR on EKS 개념 및 용어 이해](#)
- [Amazon EMR on EKS 가상 클러스터에 작업을 제출하면 어떻게 되나요?](#)

## Amazon EMR on EKS에 대한 아키텍처

Amazon EMR on EKS는 애플리케이션 및 이 애플리케이션이 실행되는 인프라 사이에서 느슨한 결합을 지원합니다. 각 인프라 계층은 후속 계층을 위한 오케스트레이션을 제공합니다. Amazon EMR에 작업을 제출하면 작업 정의에 애플리케이션별 파라미터가 모두 포함됩니다. Amazon EMR은 이러한 파라미터를 사용하여 Amazon EKS에 배포할 포드와 컨테이너를 지시합니다. 그런 다음 Amazon EKS는 Amazon EC2에서 작업을 실행하는 데 AWS Fargate 필요한 컴퓨팅 리소스를 온라인으로 가져옵니다.

이러한 느슨한 서비스 결합을 통해 안전하게 격리된 여러 작업을 동시에 실행할 수 있습니다. 또한 동일한 작업을 서로 다른 컴퓨팅 백엔드로 벤치마킹하거나 작업을 여러 가용 영역에 분산하여 가용성을 높일 수 있습니다.

다음 다이어그램은 Amazon EMR on EKS가 다른 AWS 서비스와 작동하는 방식을 보여줍니다.



## Amazon EMR on EKS 개념 및 용어 이해

Amazon EMR on EKS는 Amazon Elastic Kubernetes Service(Amazon EKS)에서 오픈 소스 빅 데이터 프레임워크를 실행할 수 있는 Amazon EMR용 배포 옵션을 제공합니다. 이 주제에서는 처리를 위해 제출하는 작업 단위인 네임스페이스, 가상 클러스터 및 작업 실행을 포함하여 몇 가지 일반적인 용어에 대한 컨텍스트를 제공합니다.

### Kubernetes 네임스페이스

Amazon EKS는 Kubernetes 네임스페이스를 사용하여 클러스터 리소스를 여러 사용자와 애플리케이션 사이에서 분할합니다. 이러한 네임스페이스는 멀티테넌트 환경의 기반입니다. Kubernetes 네임스페이스는 Amazon EC2 또는 컴퓨팅 공급자 AWS Fargate 로 가질 수 있습니다. 이러한 유연성을 통해 작업을 실행할 수 있는 다양한 성능 및 비용 옵션을 제공합니다.

### 가상 클러스터

가상 클러스터는 Amazon EMR이 등록된 Kubernetes 네임스페이스입니다. Amazon EMR은 가상 클러스터를 사용하여 작업을 실행하고 엔드포인트를 호스팅합니다. 동일한 물리적 클러스터가 여러 가상 클러스터를 지원할 수 있습니다. 하지만 각 가상 클러스터는 EKS 클러스터의 네임스페이스 하나에 매

평됩니다. 가상 클러스터는 청구서에 기여하거나 서비스 외부에서 수명 주기 관리가 필요한 활성 리소스를 생성하지 않습니다.

## 작업 실행

작업 실행은 Spark jar, PySpark 스크립트 또는 SparkSQL 쿼리와 같이 Amazon EMR on EKS에 제출하는 작업 단위입니다. 한 작업에 여러 작업 실행이 포함될 수 있습니다. 작업 실행을 제출할 때는 다음 정보를 포함해야 합니다.

- 작업을 실행해야 하는 가상 클러스터.
- 작업을 식별하기 위한 작업 이름.
- 실행 역할 - 작업을 실행하는 범위가 지정된 IAM 역할로, 이를 통해 작업에서 액세스할 수 있는 리소스를 지정할 수 있습니다.
- 사용할 오픈 소스 애플리케이션 버전을 지정하는 Amazon EMR 릴리스 레이블.
- 작업을 제출할 때 사용할 아티팩트(예: spark-submit 파라미터).

기본적으로 로그는 Spark 기록 서버에 업로드되며 AWS Management Console에서 액세스할 수 있습니다. 또한 이벤트 로그, 실행 로그 및 지표를 Amazon S3 및 Amazon CloudWatch에 푸시할 수 있습니다.

## Amazon EMR 컨테이너

Amazon EMR 컨테이너는 [Amazon EMR on EKS의 API 이름](#)입니다. emr-containers 접두사는 다음 시나리오에서 사용됩니다.

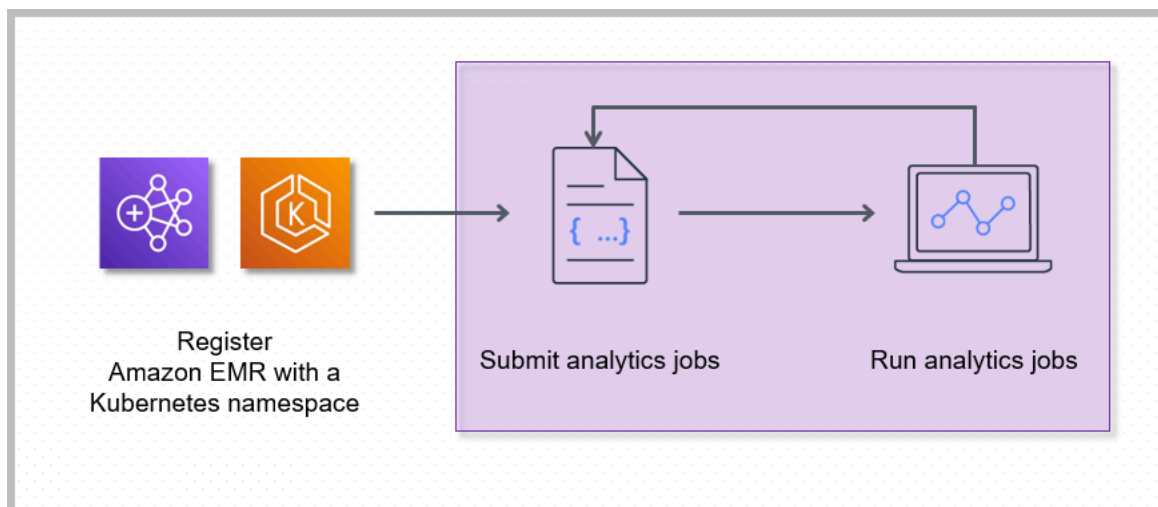
- Amazon EMR on EKS에 대한 CLI 명령의 접두사입니다. 예를 들어 `aws emr-containers start-job-run`입니다.
- Amazon EMR on EKS에 대한 IAM 정책 작업 앞에 붙는 접두사입니다. 예를 들어 "Action": [ "emr-containers:StartJobRun" ]입니다. 자세한 내용은 [Amazon EMR on EKS에 대한 정책 작업](#)을 참조하세요.
- Amazon EMR on EKS 서비스 엔드포인트에 사용되는 접두사입니다. 예를 들어 `emr-containers.us-east-1.amazonaws.com`입니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

# Amazon EMR on EKS 가상 클러스터에 작업을 제출하면 어떻게 되나요?

Amazon EMR을 Amazon EKS의 Kubernetes 네임스페이스에 등록하면 가상 클러스터가 생성됩니다. 그러면 Amazon EMR이 해당 네임스페이스에서 분석 워크로드를 실행할 수 있습니다. Amazon EMR on EKS를 사용하여 Spark 작업을 가상 클러스터에 제출하는 경우 Amazon EMR on EKS는 Amazon EKS의 Kubernetes 스케줄러에 포드 예약을 요청합니다.

다음 단계 및 다이어그램은 Amazon EMR on EKS 워크플로를 보여줍니다.

- 기존 Amazon EKS 클러스터를 사용하거나 [eksctl](#) 명령줄 유틸리티 또는 Amazon EKS 콘솔을 사용하여 클러스터를 생성합니다.
- Amazon EMR을 EKS 클러스터의 네임스페이스에 등록하여 가상 클러스터를 생성합니다.
- AWS CLI 또는 SDK를 사용하여 가상 클러스터에 작업을 제출합니다.



Amazon EMR on EKS는 실행하는 각 작업에 대해 Amazon Linux 2 기본 이미지, Apache Spark 및 관련 종속 항목을 포함하는 컨테이너를 생성합니다. 각 작업은 컨테이너를 다운로드하고 실행을 시작하는 포드에서 실행됩니다. 작업이 종료된 후 포드가 종료됩니다. 컨테이너의 이미지가 이전에 노드에 배포된 경우 캐싱된 이미지가 사용되며 다운로드를 무시됩니다. 로그 또는 지표 전달자와 같은 Sidecar 컨테이너를 포드에 배포할 수 있습니다. 작업이 종료된 후에도 Amazon EMR 콘솔에서 Spark 애플리케이션 UI를 사용하여 디버깅할 수 있습니다.

# Amazon EMR on EKS 시작하기

이 주제는 가상 클러스터에 Spark 애플리케이션을 배포하여 Amazon EMR on EKS 사용을 시작하는 데 도움이 됩니다. 여기에는 올바른 권한을 설정하고 작업을 시작하는 단계가 포함되어 있습니다. 시작하기 전에 먼저 [Amazon EMR on EKS 설정](#)의 단계를 완료해야 합니다. 이렇게 하면 가상 클러스터를 생성하기 전에 AWS CLI 설정과 같은 도구를 얻을 수 있습니다. 시작하는 데 도움이 되는 다른 템플릿은 GitHub의 [EMR Containers Best Practices Guide](#)를 참조하세요.

설정 단계에서 다음 정보가 필요합니다.

- Amazon EMR에 등록된 Amazon EKS 클러스터 및 Kubernetes 네임스페이스의 가상 클러스터 ID

## Important

EKS 클러스터를 생성할 때 m5.xlarge를 인스턴스 유형으로 사용하거나 CPU와 메모리가 더 큰 다른 인스턴스 유형으로 사용해야 합니다. m5.xlarge보다 CPU 또는 메모리가 더 적은 인스턴스 유형을 사용하면 클러스터에서 사용 가능한 리소스가 부족하여 작업이 실패할 수 있습니다.

- 작업 실행에 사용되는 IAM 역할의 이름
- Amazon EMR 릴리스의 릴리스 레이블(예: emr-6.4.0-latest)
- 로깅 및 모니터링을 위한 대상:
  - Amazon CloudWatch 로그 그룹 이름 및 로그 스트림 접두사
  - 이벤트 및 컨테이너 로그를 저장하는 Amazon S3 위치

## Important

Amazon EMR on EKS 작업은 모니터링 및 로깅을 위한 대상으로 Amazon CloudWatch 및 Amazon S3를 사용합니다. 이러한 대상으로 전송된 작업 로그를 확인하여 작업 진행 상황을 모니터링하고 실패 문제를 해결할 수 있습니다. 로깅을 활성화하려면 작업 실행을 위해 IAM 역할과 연결된 IAM 정책에 대상 리소스에 액세스하는 데 필요한 권한이 있어야 합니다. IAM 정책에 필요한 권한이 없는 경우 이 샘플 작업을 실행하기 전에 [작업 실행 역할의 신뢰 정책 업데이트](#), [Amazon S3 로그를 사용하도록 작업 실행 구성](#) 및 [CloudWatch Logs를 사용하도록 작업 실행 구성](#)에 설명된 단계를 따라야 합니다.

## Spark 애플리케이션 실행

Amazon EMR on EKS에서 간단한 Spark 애플리케이션을 실행하려면 다음 단계를 수행합니다. Spark Python 애플리케이션의 애플리케이션 entryPoint 파일은 `s3://REGION.elasticmapreduce/emr-containers/samples/wordcount/scripts/wordcount.py`에 있습니다. *REGION*은 Amazon EMR on EKS 가상 클러스터가 상주하는 리전(예: *us-east-1*)입니다.

1. 다음 정책 명령문에서 볼 수 있듯이 작업 실행 역할에 대한 IAM 정책을 필요한 권한으로 업데이트합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadFromLoggingAndInputScriptBuckets",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::*.elasticmapreduce",
        "arn:aws:s3:::*.elasticmapreduce/*",
        "arn:aws:s3:::amzn-s3-demo-bucket",
        "arn:aws:s3:::amzn-s3-demo-bucket/*",
        "arn:aws:s3:::amzn-s3-demo-bucket-b",
        "arn:aws:s3:::amzn-s3-demo-bucket-b/*"
      ]
    },
    {
      "Sid": "WriteToLoggingAndOutputDataBuckets",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/*",
        "arn:aws:s3:::amzn-s3-demo-bucket-b/*"
      ]
    }
  ]
}
```

```

    ]
  },
  {
    "Sid": "DescribeAndCreateCloudwatchLogStream",
    "Effect": "Allow",
    "Action": [
      "logs:CreateLogStream",
      "logs:DescribeLogGroups",
      "logs:DescribeLogStreams"
    ],
    "Resource": [
      "arn:aws:logs:*:*:*"
    ]
  },
  {
    "Sid": "WriteToCloudwatchLogs",
    "Effect": "Allow",
    "Action": [
      "logs:PutLogEvents"
    ],
    "Resource": [
      "arn:aws:logs:*:*:log-group:my_log_group_name:log-
stream:my_log_stream_prefix/*"
    ]
  }
]
}

```

- 이 정책의 첫 번째 ReadFromLoggingAndInputScriptBuckets 문에서는 다음 Amazon S3 버킷에 ListBucket 및 GetObjects 액세스 권한을 부여합니다.
  - *REGION*.elasticmapreduce - 애플리케이션 entryPoint 파일이 있는 버킷.
  - *amzn-s3-demo-destination-bucket* - 출력 데이터에 대해 정의하는 버킷.
  - *amzn-s3-demo-logging-bucket* - 로깅 데이터에 대해 정의하는 버킷.
- 이 정책의 두 번째 WriteToLoggingAndOutputDataBuckets 문에서는 출력 및 로깅 버킷에 각각 데이터를 쓸 수 있는 작업 권한을 부여합니다.
- 세 번째 DescribeAndCreateCloudwatchLogStream 문에서는 Amazon CloudWatch Logs를 설명하고 생성할 수 있는 작업 권한을 부여합니다.



- 네 번째 WriteToCloudwatchLogs 문에서는 `my_log_stream_prefix` 로그 스트림 아래의 Amazon CloudWatch `my_log_group_name` 로그 그룹에 로그를 작성할 수 있는 권한을 부여합니다.
2. Spark Python 애플리케이션을 실행하려면 다음 명령을 사용합니다. 모든 교체 가능한 `###` `####` 값을 적절한 값으로 바꿉니다. `REGION`은 Amazon EMR on EKS 가상 클러스터가 상주하는 리전 (예: `us-east-1`)입니다.

```
aws emr-containers start-job-run \
--virtual-cluster-id cluster_id \
--name sample-job-name \
--execution-role-arn execution-role-arn \
--release-label emr-6.4.0-latest \
--job-driver '{
  "sparkSubmitJobDriver": {
    "entryPoint": "s3://REGION.elasticmapreduce/emr-containers/samples/wordcount/
scripts/wordcount.py",
    "entryPointArguments": ["s3://amzn-s3-demo-destination-bucket/
wordcount_output"],
    "sparkSubmitParameters": "--conf spark.executor.instances=2 --
conf spark.executor.memory=2G --conf spark.executor.cores=2 --conf
spark.driver.cores=1"
  }
}' \
--configuration-overrides '{
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "my_log_group_name",
      "logStreamNamePrefix": "my_log_stream_prefix"
    },
    "s3MonitoringConfiguration": {
      "logUri": "s3://amzn-s3-demo-logging-bucket"
    }
  }
}'
```

이 작업의 출력 데이터는 `s3://amzn-s3-demo-destination-bucket/wordcount_output`에서 확인할 수 있습니다.

작업 실행을 위해 지정된 파라미터를 사용하여 JSON 파일을 생성할 수도 있습니다. 그런 다음 JSON 파일 경로와 함께 `start-job-run` 명령을 실행합니다. 자세한 내용은 [StartJobRun](#)을

[사용하여 작업 실행 제출](#) 단원을 참조하십시오. 작업 실행 파라미터 구성에 대한 자세한 내용은 [작업 실행 구성 옵션](#) 섹션을 참조하세요.

3. Spark SQL 애플리케이션을 실행하려면 다음 명령을 사용합니다. 모든 **###** **####** 값을 적절한 값으로 바꿉니다. **REGION**은 Amazon EMR on EKS 가상 클러스터가 상주하는 리전(예: **us-east-1**)입니다.

```
aws emr-containers start-job-run \
--virtual-cluster-id cluster_id \
--name sample-job-name \
--execution-role-arn execution-role-arn \
--release-label emr-6.7.0-latest \
--job-driver '{
  "sparkSqlJobDriver": {
    "entryPoint": "s3://query-file.sql",
    "sparkSqlParameters": "--conf spark.executor.instances=2 --
conf spark.executor.memory=2G --conf spark.executor.cores=2 --conf
spark.driver.cores=1"
  }
}' \
--configuration-overrides '{
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "my_log_group_name",
      "logStreamNamePrefix": "my_log_stream_prefix"
    },
    "s3MonitoringConfiguration": {
      "logUri": "s3://amzn-s3-demo-logging-bucket"
    }
  }
}'
```

샘플 SQL 쿼리 파일은 다음과 같습니다. 테이블의 데이터가 저장되는 외부 파일 스토어(예: S3)가 있어야 합니다.

```
CREATE DATABASE demo;
CREATE EXTERNAL TABLE IF NOT EXISTS demo.amazonreview( marketplace string,
customer_id string, review_id string, product_id string, product_parent string,
product_title string, star_rating integer, helpful_votes integer, total_votes
integer, vine string, verified_purchase string, review_headline string,
review_body string, review_date date, year integer) STORED AS PARQUET LOCATION
's3://URI to parquet files';
```

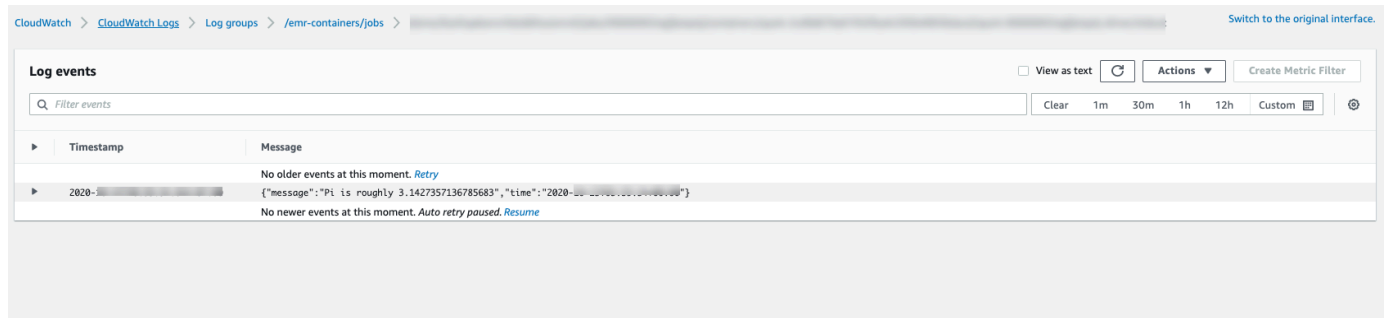
```
SELECT count(*) FROM demo.amazonreview;
SELECT count(*) FROM demo.amazonreview WHERE star_rating = 3;
```

이 작업에 대한 출력은 구성된 `monitoringConfiguration`에 따라 S3 또는 CloudWatch의 드라이버 stdout 로그에서 확인할 수 있습니다.

4. 작업 실행을 위해 지정된 파라미터를 사용하여 JSON 파일을 생성할 수도 있습니다. 그런 다음 JSON 파일 경로와 함께 `start-job-run` 명령을 실행합니다. 자세한 내용은 작업 실행 제출을 참조하세요. 작업 실행 파라미터 구성에 대한 자세한 내용은 작업 실행 구성 옵션을 참조하세요.

작업 진행 상황을 모니터링하거나 실패를 디버깅하기 위해 Amazon S3, CloudWatch Logs 또는 둘 다에 업로드된 로그를 검사할 수 있습니다. [S3 로그를 사용하도록 작업 실행 구성](#)에서 Amazon S3의 로그 경로를 참조하고, [CloudWatch Logs를 사용하도록 작업 실행 구성](#)에서 CloudWatch 로그의 로그 경로를 참조하세요. CloudWatch Logs에서 로그를 보려면 아래 지침을 따릅니다.

- <https://console.aws.amazon.com/cloudwatch/>에서 CloudWatch 콘솔을 엽니다.
- 탐색 창에서 로그를 선택합니다. 그리고 로그 그룹을 선택합니다.
- Amazon EMR on EKS에 대한 로그 그룹을 선택하고 업로드된 로그 이벤트를 확인합니다.



### ⚠ Important

작업에는 [기본적으로 구성된 재시도 정책](#)이 있습니다. 구성을 수정하거나 비활성화하는 방법에 대한 자세한 내용은 [작업 재시도 정책 사용](#)을 참조하세요.

# GitHub의 Amazon EMR on EKS 모범 사례 안내서에 대한 링크

오픈 소스 커뮤니티 협업을 통해 [Amazon EMR on EKS 모범 사례 안내서](#)를 제작했습니다. 이를 통해 가상 클러스터의 생성 및 실행을 빠르게 반복하고 이와 관련된 권장 사항을 제공할 수 있습니다. 섹션별로 [Amazon EMR on EKS 모범 사례 안내서](#)를 사용하는 것이 좋습니다. 각 섹션의 링크를 선택하여 GitHub 사이트로 이동합니다.

## 보안

### Note

Amazon EMR on EKS에서의 보안에 대한 자세한 내용은 [Amazon EMR on EKS 보안 모범 사례](#) 섹션을 참조하세요.

[Encryption best practices](#): 저장 데이터 및 전송 중인 데이터에 대해 암호화를 사용하는 방법.

[Managing network security](#)에서는 Amazon RDS 및 Amazon Redshift와 같이 AWS 서비스에 호스팅되는 데이터 소스에 연결하는 동안 Amazon EMR on EKS 포드에 대한 보안 그룹을 구성하는 방법을 설명합니다.

[AWS 보안 암호 관리자를 사용하여 보안 암호를 저장합니다.](#)

## Pyspark 작업 제출

[Pyspark job submission](#): zip, egg, wheel, pex 등의 패키징 형식을 사용하여 pySpark 애플리케이션에 대한 다양한 유형의 패키징을 지정합니다.

## 스토리지

[Using EBS volumes](#): EBS 볼륨이 필요한 작업에 대해 정적 및 동적 프로비저닝을 사용하는 방법.

[Using Amazon FSx for Lustre volumes](#): Amazon FSx for Lustre 볼륨이 필요한 작업에 대해 정적 및 동적 프로비저닝을 사용하는 방법.

[Using Instance store volumes](#): 작업 처리를 위해 인스턴스 스토어 볼륨을 사용하는 방법.

## 메타스토어 통합

[Using Hive metastore:](#) Hive 메타스토어를 사용하는 다양한 방법을 제공합니다.

[Glue 사용: AWS](#)는 AWS Glue 카탈로그를 구성하는 다양한 방법을 제공합니다.

## 디버깅

[Using Spark debugging:](#) 로그 수준 변경 방법.

[Connecting to Spark UI on the driver pod.](#)

[How to use self-hosted Spark history server with Amazon EMR on EKS.](#)

## Amazon EMR on EKS 문제 해결

[Troubleshooting.](#)

## 노드 배치

[Using Kubernetes node selectors:](#) single-az 및 기타 사용 사례에 대해 모범 사례.

[Using Fargate node placement.](#)

## 성능

[Using Dynamic Resource Allocation \(DRA\).](#)

기본적으로 `spark.dynamicAllocation.preallocateExecutors`는 Amazon EMR Spark에서 활성화됩니다. `spark.dynamicAllocation.initialExecutors` 및 `spark.dynamicAllocation.minExecutors`가 설정되지 않은 경우 Spark는 소규모 워크로드의 경우에도 예상 작업 수를 기반으로 시작 시 많은 수의 실행기를 요청할 수 있습니다. 과도한 컨테이너 이탈을 방지하려면 다음 방법 중 하나를 사용합니다.

- `spark.dynamicAllocation.initialExecutors` 또는 `spark.dynamicAllocation.minExecutors`를 워크로드 크기에 적합한 값으로 설정합니다.
- 시작 시 요청 `spark.dynamicAllocation.preallocateExecutors.maxEstimatedTasks`된 실행기 수를 제한하려면 더 낮은 값으로 설정합니다.

- 실행기 사전 할당을 완전히 비활성화false하려면  
spark.dynamicAllocation.preallocateExecutors로 설정합니다.

[EKS best practices](#): Amazon VPC 컨테이너 네트워크 인터페이스 플러그인(CNI), Cluster Autoscaler, 코어 DNS에 대한 모범 사례.

## 비용 최적화

[Using spot instances](#): Amazon EC2 스팟 인스턴스 모범 사례 및 Spark 노드 서비스 해제 기능 사용 방법.

## 사용 AWS Outposts

[를 사용하여 Amazon EMR on EKS 실행 AWS Outposts](#)

# Amazon EMR on EKS에 대한 도커 이미지 사용자 지정

Amazon EMR on EKS에서 사용자 지정된 도커 이미지를 사용할 수 있습니다. Amazon EMR on EKS 런타임 이미지를 사용자 지정하면 다음과 같은 이점이 있습니다.

- Package 애플리케이션 종속 항목과 런타임 환경을 변경 불가 단일 컨테이너로 패키징하여 이식성을 높이고 각 워크로드의 종속 항목 관리를 간소화합니다.
- 워크로드에 최적화된 패키지를 설치하고 구성합니다. Amazon EMR 런타임의 퍼블릭 배포에 이 패키지가 널리 제공되지 않을 수 있습니다.
- 로컬 개발 및 테스트를 포함하여 조직 내에 현재 수립된 빌드, 테스트 및 배포 프로세스와 Amazon EMR on EKS를 통합합니다.
- 조직 내 규정 준수 및 거버넌스 요구 사항을 충족하는 확립된 보안 프로세스(예: 이미지 스캔)를 적용합니다.

## 주제

- [도커 이미지를 사용자 지정하는 방법](#)
- [기본 이미지 URI 선택에 관한 세부 정보](#)
- [이미지 사용자 지정에 대한 고려 사항](#)

## 도커 이미지를 사용자 지정하는 방법

다음 단계를 수행하여 Amazon EMR on EKS에 대한 Docker 이미지를 사용자 지정합니다. 이 단계에서는 기본 이미지를 가져오고, 사용자 지정 및 게시하며, 이미지를 사용하여 워크로드를 제출하는 방법을 보여줍니다.

- [사전 조건](#)
- [1단계: Amazon Elastic Container Registry\(Amazon ECR\)에서 기본 이미지 검색](#)
- [2단계: 기본 이미지 사용자 지정](#)
- [3단계: \(선택적 권장 사항\) 사용자 이미지 검증](#)
- [4단계: 사용자 지정 이미지 게시](#)
- [5단계: 사용자 지정 이미지를 사용하여 Amazon EMR에서 Spark 워크로드 제출](#)

**Note**

Docker 이미지를 사용자 지정할 때 고려할 몇 가지 옵션으로, 대화형 엔드포인트에 맞게 사용자 지정하는 방법(이를 통해 필요한 종속 항목을 확보함) 또는 다중 아키텍처 컨테이너 이미지를 사용하는 방법이 있습니다.

- [대화형 엔드포인트에 대한 도커 이미지 사용자 지정](#)
- [다중 아키텍처 이미지 작업](#)

## 사전 조건

- Amazon EMR on EKS에 대한 [Amazon EMR on EKS 설정](#) 단계를 수행합니다.
- 환경에 Docker를 설치합니다. 자세한 내용은 [Get Docker](#)를 참조하세요.

## 1단계: Amazon Elastic Container Registry(Amazon ECR)에서 기본 이미지 검색

기본 이미지에는 다른 AWS 서비스에 액세스하는 데 사용되는 Amazon EMR 런타임 및 커넥터가 포함되어 있습니다. Amazon EMR 6.9.0 이상에서는 Amazon ECR 퍼블릭 갤러리에서 기본 이미지를 가져올 수 있습니다. 갤러리를 탐색하여 이미지 링크를 찾은 다음, 이미지를 로컬 Workspace로 가져옵니다. 예를 들어 Amazon EMR 7.13.0 릴리스의 경우 다음 `docker pull` 명령은 가장 최근의 표준 기본 이미지를 가져옵니다. `emr-7.13.0:latest`를 `emr-7.13.0-spark-rapids:latest`로 바꾸어 Nvidia RAPIDS 액셀러레이터가 있는 이미지를 검색할 수 있습니다. `emr-7.13.0:latest`를 `emr-7.13.0-java11:latest`로 대체하여 Java 11 런타임에서 이미지를 검색할 수도 있습니다.

```
docker pull public.ecr.aws/emr-on-eks/spark/emr-7.13.0:latest
```

Amazon EMR 6.9.0 이하 릴리스의 기본 이미지를 검색하거나 각 리전의 Amazon ECR 레지스트리 계정에서 검색하려는 경우 다음 단계를 사용합니다.

1. 기본 이미지 URI를 선택합니다. 다음 예제에서 볼 수 있듯이 이미지 URI는 이 `ECR-registry-account.dkr.ecr.Region.amazonaws.com/spark/container-image-tag` 형식을 따릅니다.

```
895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest
```



리전의 기본 이미지를 선택하려면 [기본 이미지 URI 선택에 관한 세부 정보](#) 섹션을 참조하세요.

2. 기본 이미지가 저장된 Amazon ECR 리포지토리에 로그인합니다. **895885662937** 및 **us-west-2**를 Amazon ECR 레지스트리 계정 및 선택한 AWS 리전으로 바꿉니다.

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin 895885662937.dkr.ecr.us-west-2.amazonaws.com
```

3. 기본 이미지를 로컬 Workspace로 가져옵니다. **emr-6.6.0:latest**를 선택한 컨테이너 이미지 태그로 바꿉니다.

```
docker pull 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest
```

## 2단계: 기본 이미지 사용자 지정

다음 단계를 수행하여 Amazon ECR에서 가져온 기본 이미지를 사용자 지정합니다.

1. 로컬 Workspace에 새 Dockerfile을 생성합니다.
2. 방금 만든 Dockerfile을 수정하고 다음 내용을 추가합니다. 이 Dockerfile에서는 **895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest**에서 가져온 컨테이너 이미지를 사용합니다.

```
FROM 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest
USER root
### Add customization commands here ###
USER hadoop:hadoop
```

3. Dockerfile에 명령을 추가하여 기본 이미지를 사용자 지정합니다. 예를 들어, 다음 Dockerfile에서와 같이 Python 라이브러리를 설치하는 명령을 추가합니다.

```
FROM 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.6.0:latest
USER root
RUN pip3 install --upgrade boto3 pandas numpy // For python 3
USER hadoop:hadoop
```

4. Dockerfile이 생성된 동일한 디렉터리에서 다음 명령을 실행하여 도커 이미지를 빌드합니다. 도커 이미지의 이름(예: **emr6.6\_custom**)을 입력합니다.

```
docker build -t emr6.6_custom .
```

### 3단계: (선택적 권장 사항) 사용자 이미지 검증

사용자 지정 이미지를 게시하기 전에 사용자 지정 이미지의 호환성을 테스트하는 것이 좋습니다.

[Amazon EMR on EKS 사용자 지정 이미지 CLI](#)를 사용하여 Amazon EMR on EKS에서 실행하는 데 필요한 파일 구조와 올바른 구성이 이미지에 있는지 확인할 수 있습니다.

#### Note

Amazon EMR on EKS 사용자 지정 이미지 CLI에서는 이미지에 오류가 없는지 확인할 수 없습니다. 기본 이미지에서 종속성을 제거할 때 주의합니다.

다음 명령을 실행하여 사용자 지정 이미지를 검증합니다.

1. Amazon EMR on EKS 사용자 지정 이미지 CLI를 다운로드하고 설치합니다. 자세한 내용은 [Amazon EMR on EKS custom image CLI Installation Guide](#)를 참조하세요.
2. 다음 명령을 실행하여 설치를 테스트합니다.

```
emr-on-eks-custom-image --version
```

다음은 출력 예제입니다.

```
Amazon EMR on EKS Custom Image CLI  
Version: x.xx
```

3. 다음 명령을 실행하여 사용자 지정 이미지를 검증합니다.

```
emr-on-eks-custom-image validate-image -i image_name -r release_version [-  
t image_type]
```

- -i에서 검증해야 하는 로컬 이미지 URI를 지정합니다. 이는 이미지 URI, 이미지에 정의한 이름 또는 태그일 수 있습니다.
- -r에서 기본 이미지의 정확한 릴리스 버전(예: `emr-6.6.0-latest`)을 지정합니다.

- -t에서 이미지 유형을 지정합니다. Spark 이미지인 경우 spark를 입력합니다. 기본값은 spark입니다. 현재 Amazon EMR on EKS 사용자 지정 이미지 CLI 버전은 Spark 런타임 이미지만 지원합니다.

명령을 성공적으로 실행하고 사용자 지정 이미지가 모든 필수 구성 및 파일 구조를 충족하면 다음 예제에서 볼 수 있듯이 반환된 출력에 모든 테스트 결과가 표시됩니다.

```
Amazon EMR on EKS Custom Image Test
Version: x.xx
... Checking if docker cli is installed
... Checking Image Manifest
[INFO] Image ID: xxx
[INFO] Created On: 2021-05-17T20:50:07.986662904Z
[INFO] Default User Set to hadoop:hadoop : PASS
[INFO] Working Directory Set to /home/hadoop : PASS
[INFO] Entrypoint Set to /usr/bin/entrypoint.sh : PASS
[INFO] SPARK_HOME is set with value: /usr/lib/spark : PASS
[INFO] JAVA_HOME is set with value: /etc/alternatives/jre : PASS
[INFO] File Structure Test for spark-jars in /usr/lib/spark/jars: PASS
[INFO] File Structure Test for hadoop-files in /usr/lib/hadoop: PASS
[INFO] File Structure Test for hadoop-jars in /usr/lib/hadoop/lib: PASS
[INFO] File Structure Test for bin-files in /usr/bin: PASS
... Start Running Sample Spark Job
[INFO] Sample Spark Job Test with local:///usr/lib/spark/examples/jars/spark-examples.jar : PASS
-----
Overall Custom Image Validation Succeeded.
-----
```

사용자 지정 이미지가 필수 구성 또는 파일 구조를 충족하지 않는 경우 오류 메시지가 나타납니다. 반환된 출력은 잘못된 구성 또는 파일 구조에 대한 정보를 제공합니다.

## 4단계: 사용자 지정 이미지 게시

Amazon ECR 레지스트리에 새 도커 이미지를 게시합니다.

1. 다음 명령을 실행하여 도커 이미지를 저장할 Amazon ECR 리포지토리를 생성합니다. 리포지토리의 이름을 입력합니다 (예: *emr6.6\_custom\_repo*(. *us-west-2*를 해당 리전으로 바꿉니다).

```
aws ecr create-repository \
```

```
--repository-name emr6.6_custom_repo \  
--image-scanning-configuration scanOnPush=true \  
--region us-west-2
```

자세한 내용은 Amazon ECR 사용 설명서에서 [리포지토리 생성](#)을 참조하세요.

2. 다음 명령을 실행하여 기본 레지스트리에 인증합니다.

```
aws ecr get-login-password --region us-west-2 | docker login --username AWS --  
password-stdin aws_account_id.dkr.ecr.us-west-2.amazonaws.com
```

자세한 내용은 Amazon ECR 사용 설명서에서 [기본 레지스트리에 대해 인증](#)을 참조하세요.

3. 이미지에 태그를 지정하고 생성한 Amazon ECR 리포지토리에 이미지를 업로드합니다.

이미지에 태그를 지정합니다.

```
docker tag emr6.6_custom aws_account_id.dkr.ecr.us-  
west-2.amazonaws.com/emr6.6_custom_repo
```

이미지를 푸시합니다.

```
docker push aws_account_id.dkr.ecr.us-west-2.amazonaws.com/emr6.6_custom_repo
```

자세한 내용은 Amazon ECR 사용 설명서에서 [Amazon ECR에 이미지 게시](#)를 참조하세요.

## 5단계: 사용자 지정 이미지를 사용하여 Amazon EMR에서 Spark 워크로드 제출

사용자 지정 이미지를 만들고 게시한 후 사용자 지정 이미지를 사용하여 Amazon EMR on EKS 작업을 제출할 수 있습니다.

먼저, 다음 예제 JSON 파일에서 볼 수 있듯이 start-job-run-request.json 파일을 생성하고 사용자 지정 이미지를 참조할 spark.kubernetes.container.image 파라미터를 지정합니다.

### Note

아래 JSON 스니펫의 entryPoint 인수와 함께 표시된 것처럼 scheme을 사용하여 사용자 지정 이미지에서 사용 가능한 파일을 참조하도록 local:// 스키마를 사용할 수 있습니다. local:// 스키마를 사용하여 애플리케이션 종속성을 참조할 수도 있습니다. local:// 스키

마를 사용하여 참조되는 모든 파일 및 종속성은 사용자 지정 이미지의 지정된 경로에 이미 있어야 합니다.

```
{
  "name": "spark-custom-image",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.6.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "local:///usr/lib/spark/examples/jars/spark-examples.jar",
      "entryPointArguments": [
        "10"
      ],
      "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf spark.kubernetes.container.image=123456789012.dkr.ecr.us-west-2.amazonaws.com/emr6.6_custom_repo"
    }
  }
}
```

다음 예제에서 볼 수 있듯이 applicationConfiguration 속성을 사용하여 사용자 지정 이미지를 참조할 수도 있습니다.

```
{
  "name": "spark-custom-image",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.6.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "local:///usr/lib/spark/examples/jars/spark-examples.jar",
      "entryPointArguments": [
        "10"
      ],
      "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi"
    }
  },
}
```

```

    "configurationOverrides": {
      "applicationConfiguration": [
        {
          "classification": "spark-defaults",
          "properties": {
            "spark.kubernetes.container.image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/emr6.6_custom_repo"
          }
        }
      ]
    }
  }
}

```

그런 다음 `start-job-run` 명령을 실행하여 작업을 제출합니다.

```
aws emr-containers start-job-run --cli-input-json file:///./start-job-run-request.json
```

위의 JSON 예제에서 `emr-6.6.0-latest`를 Amazon EMR 릴리스 버전으로 바꿉니다. 선택한 버전에 최신 보안 업데이트가 포함되도록 `-latest` 릴리스 버전을 사용하는 것이 좋습니다. Amazon EMR 릴리스 버전 및 이미지 태그에 대한 자세한 내용은 [기본 이미지 URI 선택에 관한 세부 정보](#) 섹션을 참조하세요.

#### Note

`spark.kubernetes.driver.container.image` 및 `spark.kubernetes.executor.container.image`를 사용하여 드라이버 및 실행기 포트에 대해 다른 이미지를 지정할 수 있습니다.

## 대화형 엔드포인트에 대한 도커 이미지 사용자 지정

또한 대화형 엔드포인트에 맞게 도커 이미지를 사용자 지정하여 사용자 지정된 기본 커널 이미지를 실행할 수 있습니다. 이를 통해 EMR Studio에서 대화형 워크로드를 실행할 때 필요한 종속성을 보유할 수 있습니다.

- 위에 설명된 [1~4단계](#)에 따라 도커 이미지를 사용자 지정합니다. Amazon EMR 6.9.0 릴리스 이상의 경우 Amazon ECR 퍼블릭 갤러리에서 기본 이미지 URI를 가져올 수 있습니다. Amazon EMR 6.9.0 이전 릴리스의 경우 각 AWS 리전의 Amazon ECR 레지스트리 계정에서 이미지를 가져올 수

있으며 유일한 차이점은 Dockerfile의 기본 이미지 URI입니다. 기본 이미지 URI는 다음 형식을 따릅니다.

```
ECR-registry-account.dkr.ecr.Region.amazonaws.com/notebook-spark/container-image-tag
```

spark 대신 기본 이미지 URI에서 notebook-spark를 사용해야 합니다. 기본 이미지에는 Spark 런타임 및 이와 함께 실행되는 노트북 커널이 포함되어 있습니다. 리전 및 컨테이너 이미지 태그 선택에 대한 자세한 내용은 [기본 이미지 URI 선택에 관한 세부 정보](#) 섹션을 참조하세요.

#### Note

현재는 기본 이미지의 재정의만 지원되며 기본 이미지에서 AWS 제공하는 것이 아닌 다른 유형의 완전히 새로운 커널을 도입하는 것은 지원되지 않습니다.

2. 사용자 지정 이미지와 함께 사용할 수 있는 대화형 엔드포인트를 생성합니다.

다음 콘텐츠를 포함하여 custom-image-managed-endpoint.json JSON 파일을 생성합니다.

```
{
  "name": "endpoint-name",
  "virtualClusterId": "virtual-cluster-id",
  "type": "JUPYTER_ENTERPRISE_GATEWAY",
  "releaseLabel": "emr-6.6.0-latest",
  "executionRoleArn": "execution-role-arn",
  "certificateArn": "certificate-arn",
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "jupyter-kernel-overrides",
        "configurations": [
          {
            "classification": "python3",
            "properties": {
              "container-image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/custom-notebook-python:latest"
            }
          },
          {
            "classification": "spark-python-kubernetes",
```

```

        "properties": {
            "container-image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/custom-notebook-spark:latest"
        }
    }
}

```

다음으로, 다음 예제와 같이 JSON 파일에 지정된 구성을 사용하여 대화형 엔드포인트를 생성합니다.

```
aws emr-containers create-managed-endpoint --cli-input-json custom-image-managed-
endpoint.json
```

자세한 내용은 [가상 클러스터의 대화형 엔드포인트 생성](#)을 참조하세요.

3. EMR Studio를 통해 대화형 엔드포인트에 연결합니다. 자세한 내용은 [Studio에서 연결](#)을 참조하세요.

## 다중 아키텍처 이미지 작업

Amazon EMR on EKS는 Amazon Elastic Container Registry(Amazon ECR)에 대한 다중 아키텍처 컨테이너 이미지를 지원합니다. 자세한 내용은 [Introducing multi-architecture container images for Amazon ECR](#)을 참조하세요.

Amazon EMR on EKS 사용자 지정 이미지는 AWS Graviton 기반 EC2 인스턴스와 non-Graviton-based EC2 인스턴스를 모두 지원합니다. Graviton 기반 이미지는 Graviton 기반 이미지가 아닌 이미지와 동일한 Amazon ECR의 이미지 리포지토리에 저장됩니다.

예를 들어 Docker 매니페스트 목록에서 6.6.0 이미지를 검사하려면 다음 명령을 실행합니다.

```
docker manifest inspect 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/
emr-6.6.0:latest
```

다음은 출력입니다. arm64 아키텍처는 Graviton 인스턴스용입니다. amd64는 Graviton 외 인스턴스용입니다.



```
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.docker.distribution.manifest.list.v2+json",
  "manifests": [
    {
      "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
      "size": 1805,
      "digest":
"xxx123:6b971cb47d11011ab3d45fff925e9442914b4977ae0f9fbcdcf5cfa99a7593f0",
      "platform": {
        "architecture": "arm64",
        "os": "linux"
      }
    },
    {
      "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
      "size": 1805,
      "digest":
"xxx123:6f2375582c9c57fa9838c1d3a626f1b4fc281e287d2963a72dfe0bd81117e52f",
      "platform": {
        "architecture": "amd64",
        "os": "linux"
      }
    }
  ]
}
```

다음 단계를 수행하여 다중 아키텍처 이미지를 생성합니다.

1. arm64 이미지를 가져올 수 있도록 다음 콘텐츠를 포함하는 Dockerfile을 생성합니다.

```
FROM --platform=arm64 895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/
emr-6.6.0:latest
USER root

RUN pip3 install boto3 // install customizations here
USER hadoop:hadoop
```

2. 다중 아키텍처 이미지를 빌드하려면 [Introducing multi-architecture container images for Amazon ECR](#)의 지침을 따릅니다.

**Note**

arm64 인스턴스에서 arm64 이미지를 생성해야 합니다. 마찬가지로 amd64 인스턴스에서 amd64 이미지를 빌드해야 합니다.

또한 Docker buildx 명령을 사용하여 각 특정 인스턴스 유형에 빌드하지 않고도 다중 아키텍처 이미지를 빌드할 수 있습니다. 자세한 내용은 [Leverage multi-CPU architecture support](#)를 참조하세요.

- 다중 아키텍처 이미지를 빌드한 후 동일한 `spark.kubernetes.container.image` 파라미터를 사용하여 작업을 제출하고 해당 이미지를 가리킬 수 있습니다. AWS Graviton 기반 및 non-Graviton-based EC2 인스턴스가 모두 있는 이기종 클러스터에서 인스턴스는 이미지를 가져오는 인스턴스 아키텍처를 기반으로 올바른 아키텍처 이미지를 결정합니다.

## 기본 이미지 URI 선택에 관한 세부 정보

**Note**

Amazon EMR 6.9.0 릴리스 이상의 경우 Amazon ECR 퍼블릭 갤러리에서 기본 이미지를 검색할 수 있으므로 이 페이지의 지침에 따라 기본 이미지 URI를 구성할 필요가 없습니다. 기본 이미지의 컨테이너 이미지 태그를 찾으려면 Amazon EMR on EKS의 해당 릴리스에 대한 [릴리스 정보 페이지](#)를 참조하세요.

사용자가 선택할 수 있는 기본 도커 이미지는 Amazon Elastic Container Registry(Amazon ECR)에 저장됩니다. 다음 예제에서 볼 수 있듯이 이미지 URI는 이 `ECR-registry-account.dkr.ecr.Region.amazonaws.com/spark/container-image-tag` 형식을 따릅니다.

```
895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-7.13.0:latest
```

대화형 엔드포인트의 이미지 URI는 다음 예제에서와 같이 `ECR-registry-account.dkr.ecr.Region.amazonaws.com/notebook-spark/container-image-tag` 형식을 따릅니다. spark 대신 기본 이미지 URI에서 notebook-spark를 사용해야 합니다.

```
895885662937.dkr.ecr.us-west-2.amazonaws.com/notebook-spark/emr-7.13.0:latest
```

마찬가지로 대화형 엔드포인트에 대한 Spark 이외의 python3 이미지의 경우 이미지 URI는 `ECR-registry-account.dkr.ecr.Region.amazonaws.com/notebook-python/container-image-tag`입니다. 다음 예제 URI는 올바른 형식으로 지정되었습니다.

```
895885662937.dkr.ecr.us-west-2.amazonaws.com/notebook-python/emr-7.13.0:latest
```

기본 이미지의 컨테이너 이미지 태그를 찾으려면 Amazon EMR on EKS의 해당 릴리스에 대한 [릴리스 정보 페이지](#)를 참조하세요.

## 리전별 Amazon ECR 레지스트리 계정

네트워크 지연 시간이 길어지지 않도록 가장 가까운에서 기본 이미지를 가져옵니다 AWS 리전. 다음 테이블을 기반으로 이미지를 가져오는 리전에 대응하는 Amazon ECR 레지스트리 계정을 선택합니다.

| 리전             | Amazon ECR 레지스트리 계정 |
|----------------|---------------------|
| ap-east-1      | 736135916053        |
| ap-northeast-1 | 059004520145        |
| ap-northeast-2 | 996579266876        |
| ap-northeast-3 | 705689932349        |
| ap-southeast-3 | 946962994502        |
| ap-south-1     | 235914868574        |
| ap-south-2     | 691480105545        |
| ap-southeast-1 | 671219180197        |
| ap-southeast-2 | 038297999601        |
| ca-central-1   | 351826393999        |
| eu-central-1   | 107292555468        |

| 리전             | Amazon ECR 레지스트리 계정 |  |
|----------------|---------------------|--|
| eu-central-2   | 314408114945        |  |
| eu-north-1     | 830386416364        |  |
| eu-west-1      | 483788554619        |  |
| eu-west-2      | 118780647275        |  |
| eu-west-3      | 307523725174        |  |
| eu-south-1     | 238014973495        |  |
| eu-south-2     | 350796622945        |  |
| il-central-1   | 395734710648        |  |
| me-south-1     | 008085056818        |  |
| me-central-1   | 818935616732        |  |
| sa-east-1      | 052806832358        |  |
| us-gov-west-1  | 299385240661        |  |
| us-gov-east-1  | 299393998622        |  |
| us-east-1      | 755674844232        |  |
| us-east-2      | 711395599931        |  |
| us-west-1      | 608033475327        |  |
| us-west-2      | 895885662937        |  |
| af-south-1     | 358491847878        |  |
| cn-north-1     | 068337069695        |  |
| cn-northwest-1 | 068420816659        |  |

## 이미지 사용자 지정에 대한 고려 사항

도커 이미지를 사용자 지정할 때 작업의 정확한 런타임을 세부적인 수준에서 선택할 수 있습니다. 이 기능을 사용하는 경우 이 모범 사례를 따릅니다. 여기에는 보안, 구성 및 이미지 탑재에 대한 고려 사항이 포함됩니다.

- 보안은 AWS와 사용자 간의 공동 책임입니다. 이미지에 추가하는 바이너리의 보안 패치는 사용자의 책임입니다. [Amazon EMR on EKS 보안 모범 사례](#)(특히 [사용자 지정 이미지에 대한 최신 보안 업데이트 받기](#) 및 [최소 권한의 원칙 적용](#))의 지침을 따릅니다.
- 기본 이미지를 사용자 지정할 때는 작업이 루트 사용자로 실행되지 않도록 Docker 사용자를 `hadoop:hadoop`으로 변경해야 합니다.
- Amazon EMR on EKS는 런타임에 이미지 구성 위에 `spark-defaults.conf`와 같은 파일을 마운트합니다. 이러한 구성 파일을 재정의하려면 작업 제출 중에 `application0verrides` 파라미터를 사용하고 사용자 지정 이미지의 파일을 직접 수정하지 않는 것이 좋습니다.
- Amazon EMR on EKS는 런타임에 특정 폴더를 마운트합니다. 이러한 폴더에 대한 수정 사항은 이 컨테이너에서 사용할 수 없습니다. 사용자 지정 이미지에 대한 애플리케이션 또는 해당 종속성을 추가하려는 경우 다음과 같은 사전 정의된 경로에 속하지 않는 디렉터리를 선택하는 것이 좋습니다.
  - `/var/log/fluentd`
  - `/var/log/spark/user`
  - `/var/log/spark/apps`
  - `/mnt`
  - `/tmp`
  - `/home/hadoop`
- Amazon ECR, Docker Hub 또는 프라이빗 엔터프라이즈 리포지토리와 같은 Docker 호환 리포지토리에 사용자 지정 이미지를 업로드할 수 있습니다. 선택한 Docker 리포지토리로 Amazon EKS 클러스터 인증을 구성하는 방법에 대한 자세한 내용은 [Pull an Image from a Private Registry](#)를 참조하세요.

# Amazon EMR on EKS를 사용하여 Flink 작업 실행

Amazon EMR 릴리스 6.13.0 이상은 Amazon EMR on EKS의 작업 제출 모델로 Apache Flink 또는 Flink Kubernetes 운영자를 사용하는 Amazon EMR on EKS를 지원합니다. Apache Flink가 포함된 Amazon EMR on EKS를 사용하면 자체 Amazon EKS 클러스터에서 Amazon EMR 릴리스 런타임으로 Flink 애플리케이션을 배포하고 관리할 수 있습니다. Amazon EKS 클러스터에 Flink Kubernetes 운영자를 배포한 후에는 운영자를 통해 Flink 애플리케이션을 직접 제출할 수 있습니다. 운영자는 Flink 애플리케이션의 수명 주기를 관리합니다.

## 주제

- [Flink Kubernetes 연산자 설정 및 사용](#)
- [Flink 네이티브 Kubernetes 사용](#)
- [Flink 및 FluentD에 대한 Docker 이미지 사용자 지정](#)
- [Flink Kubernetes 운영자 및 Flink 작업 모니터링](#)
- [Flink에서 고가용성 및 작업 복원력을 지원하는 방법](#)
- [Flink 애플리케이션에 대한 Autoscaler 사용](#)
- [Amazon EMR on EKS에서 Flink 작업에 대한 유지 관리 및 문제 해결](#)
- [Apache Flink가 포함된 Amazon EMR on EKS에 대해 지원되는 릴리스](#)

## Flink Kubernetes 연산자 설정 및 사용

다음 페이지에서는 Flink Kubernetes 운영자를 설정 및 사용하여, Amazon EMR on EKS에서 Flink 작업을 실행하는 방법을 설명합니다. 사용 가능한 주제에는 필수 사전 조건, 환경 설정 방법, Amazon EMR on EKS에서 Flink 애플리케이션 실행이 포함됩니다.

## 주제

- [Amazon EMR on EKS에 대한 Flink Kubernetes 운영자 설정](#)
- [Amazon EMR on EKS에 대한 Flink Kubernetes 연산자 설치](#)
- [Flink 애플리케이션 실행](#)
- [Flink 애플리케이션을 실행하기 위한 보안 역할 권한](#)
- [Amazon EMR on EKS에 대한 Flink Kubernetes 운영자 제거](#)

## Amazon EMR on EKS에 대한 Flink Kubernetes 운영자 설정

Amazon EKS에서 Flink Kubernetes 운영자를 설치하려면 먼저 설정을 위해 다음 작업을 수행합니다. Amazon Web Services(AWS)에 이미 가입했고 Amazon EKS를 사용하고 있는 경우 Amazon EMR on EKS를 사용할 준비를 거의 마친 상태입니다. Amazon EKS에서 Flink 운영자를 설정하려면 다음 작업을 수행합니다. 필수 조건 중 하나를 이미 완료한 경우 해당 조건을 건너뛰고 다음 조건으로 넘어갈 수 있습니다.

- [의 최신 버전 설치 또는 업데이트 AWS CLI](#) -를 이미 설치한 경우 최신 버전이 있는지 AWS CLI 확인합니다.
- [kubectl 및 eksctl 설치](#) - eksctl은 Amazon EKS와 통신하는 데 사용하는 명령줄 도구입니다.
- [Install Helm](#) - Kubernetes용 Helm 패키지 관리자는 Kubernetes 클러스터에서 애플리케이션을 설치하고 관리하는 데 도움이 됩니다.
- [Amazon EKS 시작하기 – eksctl](#) - 관련 단계를 수행하여 Amazon EKS에서 노드를 포함하는 새 Kubernetes 클러스터를 생성합니다.
- [Amazon EMR 릴리스 레이블 선택](#)(릴리스 6.13.0 이상) - Flink Kubernetes 연산자는 Amazon EMR 릴리스 6.13.0 이상에서 지원됩니다.
- [Amazon EKS 클러스터에서 서비스 계정에 대한 IAM 역할\(IRSA\)을 활성화](#)합니다.
- [작업 실행 역할을 생성](#)합니다.
- [작업 실행 역할의 신뢰 정책을 업데이트](#)합니다.
- 운영자 실행 역할을 생성합니다. 이 단계는 선택 사항입니다. Flink 작업과 운영자에 동일한 역할을 사용할 수 있습니다. 운영자에 대해 다른 IAM 역할을 사용하려는 경우 별도의 역할을 생성할 수 있습니다.
- 운영자 실행 역할의 신뢰 정책을 업데이트합니다. Amazon EMR Flink Kubernetes 운영자 서비스 계정에 사용할 역할에 대해 신뢰 정책 항목 하나를 명시적으로 추가해야 합니다. 다음 예제 형식을 따를 수 있습니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::AWS_ACCOUNT_ID:oidc-provider/OIDC_PROVIDER"
```

```

    },
    "Action": "sts:AssumeRoleWithWebIdentity",
    "Condition": {
      "StringEquals": {
        "OIDC_PROVIDER:aud": "sts.amazonaws.com",
        "OIDC_PROVIDER:sub": "system:serviceaccount:emr:emr-containers-sa-
flink-operator"
      }
    }
  }
]
}

```

## Amazon EMR on EKS에 대한 Flink Kubernetes 연산자 설치

이 주제는 Flink 배포를 준비하여 Amazon EKS에서 Flink Kubernetes 연산자 사용을 시작하는 데 도움을 줍니다.

### Kubernetes 연산자 설치

다음 단계를 사용하여 Apache Flink용 Kubernetes 운영자를 설치합니다.

1. 아직 실행하지 않았다면, [the section called “설정”](#)의 단계를 완료합니다.
2. **cert-manager**(Amazon EKS 클러스터당 하나)를 설치하여 웹훅 구성 요소 추가를 지원합니다.

```
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.12.0/cert-manager.yaml
```

3. 차트 Helm을 설치합니다.

```

export VERSION=7.13.0 # The Amazon EMR release version
export NAMESPACE=The Kubernetes namespace to deploy the operator

helm install flink-kubernetes-operator \
oci://public.ecr.aws/emr-on-eks/flink-kubernetes-operator \
--version $VERSION \
--namespace $NAMESPACE

```

출력 예시:



```
NAME: flink-kubernetes-operator
LAST DEPLOYED: Tue May 31 17:38:56 2022
NAMESPACE: $NAMESPACE
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

4. 배포가 완료될 때까지 기다린 후 차트 설치를 확인합니다.

```
kubectl wait deployment flink-kubernetes-operator --namespace $NAMESPACE --for
condition=Available=True --timeout=30s
```

5. 배포가 완료되면 다음 메시지가 표시됩니다.

```
deployment.apps/flink-kubernetes-operator condition met
```

6. 다음 명령을 사용하여 배치된 운영자를 확인합니다.

```
helm list --namespace $NAMESPACE
```

다음은 앱 버전 x.y.z-amzn-n이 Amazon EMR on EKS 릴리스의 Flink 운영자 버전과 일치하는 예제 출력을 보여줍니다. 자세한 내용은 [Apache Flink가 포함된 Amazon EMR on EKS에 대해 지원되는 릴리스](#) 단원을 참조하십시오.

| NAME                           | STATUS | CHART    | NAMESPACE   | REVISION | UPDATED                   | APP VERSION  |
|--------------------------------|--------|----------|-------------|----------|---------------------------|--------------|
| flink-kubernetes-operator-0500 | EST    | deployed | \$NAMESPACE | 1        | 2023-02-22 16:43:45.24148 | x.y.z-amzn-n |

## Kubernetes 연산자 업그레이드

Flink 연산자의 버전을 업그레이드하려면 다음 단계를 따르세요.

- 이전 flink-kubernetes-operator: `helm uninstall flink-kubernetes-operator -n <NAMESPACE>`를 제거합니다.
- `kubectl delete crd flinkdeployments.flink.apache.org`  
`flinksessionjobs.flink.apache.org` CRD를 삭제(helm은 이전 CRD를 자동으로 삭제하지 않으므로)합니다.

3. `flink-kubernetes-operator`를 최신 버전으로 다시 설치합니다.

## Flink 애플리케이션 실행

Amazon EMR 6.13.0 이상을 사용할 경우 Amazon EMR on EKS의 애플리케이션 모드에서 Flink Kubernetes 연산자로 Flink 애플리케이션을 실행할 수 있습니다. Amazon EMR 6.15.0 이상을 사용할 경우 세션 모드에서 Flink 애플리케이션을 실행할 수도 있습니다. 이 페이지는 Amazon EMR on EKS로 Flink 애플리케이션을 실행하기 위해 활용할 수 있는 두 가지 방법을 모두 설명합니다.

### Note

Flink 작업을 제출할 때고가용성 메타데이터를 저장할 Amazon S3 버킷을 생성해야 합니다. 이 기능을 사용하고 싶지 않은 경우 비활성화할 수 있습니다. 기본적으로 활성화됩니다.

전제 조건 – Flink Kubernetes 운영자로 Flink 애플리케이션을 실행하기 전에 [the section called “설정”](#) 및 [the section called “Kubernetes 연산자 설치”](#)의 단계를 완료합니다.

### Application mode

Amazon EMR 6.13.0 이상을 사용할 경우 Amazon EMR on EKS의 애플리케이션 모드에서 Flink Kubernetes 연산자로 Flink 애플리케이션을 실행할 수 있습니다.

1. 다음 예제에서 FlinkDeployment 정의 파일 `basic-example-app-cluster.yaml`을 생성합니다. [옵트인 AWS 리전](#) 중 하나를 활성화하고 사용하는 경우 구성의 주석 처리를 해제하고 구성해야 합니다 `fs.s3a.endpoint.region`.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example-app-cluster
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    #fs.s3a.endpoint.region: OPT_IN_AWS_REGION_NAME
    state.checkpoints.dir: CHECKPOINT_S3_STORAGE_PATH
    state.savepoints.dir: SAVEPOINT_S3_STORAGE_PATH
  flinkVersion: v1_17
  executionRoleArn: JOB_EXECUTION_ROLE_ARN
  emrReleaseLabel: "emr-6.13.0-flink-latest" # 6.13 or higher
```

```

jobManager:
  storageDir: HIGH_AVAILABILITY_STORAGE_PATH
  resource:
    memory: "2048m"
    cpu: 1
taskManager:
  resource:
    memory: "2048m"
    cpu: 1
job:
  # if you have your job jar in S3 bucket you can use that path as well
  jarURI: local:///opt/flink/examples/streaming/StateMachineExample.jar
  parallelism: 2
  upgradeMode: savepoint
  savepointTriggerNonce: 0
monitoringConfiguration:
  cloudWatchMonitoringConfiguration:
    logGroupName: LOG_GROUP_NAME

```

2. 다음 명령으로 Flink 배포를 제출합니다. 이렇게 하면 FlinkDeployment 객체(basic-example-app-cluster)도 생성됩니다.

```
kubectl create -f basic-example-app-cluster.yaml -n <NAMESPACE>
```

3. Flink UI에 액세스합니다.

```
kubectl port-forward deployments/basic-example-app-cluster 8081 -n NAMESPACE
```

4. localhost:8081을 열어 Flink 작업을 로컬에서 확인합니다.
5. 작업을 정리합니다. 이 작업을 위해 생성된 S3 아티팩트(예: 검사, 고가용성, 메타데이터 세이브포인트 및 CloudWatch 로그)를 정리해야 합니다.

Flink Kubernetes 운영자를 통해 Flink에 애플리케이션을 제출하는 방법에 대한 자세한 내용은 GitHub의 [apache/flink-kubernetes-operator](#) 폴더에 있는 [Flink Kubernetes operator examples](#)를 참조하세요.

## Session mode

Amazon EMR 6.15.0 이상을 사용할 경우 Amazon EMR on EKS의 세션 모드에서 Flink Kubernetes 연산자로 Flink 애플리케이션을 실행할 수 있습니다.

1. 다음 예제에서 이름이 `basic-example-app-cluster.yaml`인 `FlinkDeployment` 정의 파일을 생성합니다. [옵트인 AWS 리전](#) 중 하나를 활성화하고 사용하는 경우 구성의 주석 처리를 해제하고 구성해야 합니다 `fs.s3a.endpoint.region`.

```

apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example-session-cluster
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    #fs.s3a.endpoint.region: OPT_IN_AWS_REGION_NAME
    state.checkpoints.dir: CHECKPOINT_S3_STORAGE_PATH
    state.savepoints.dir: SAVEPOINT_S3_STORAGE_PATH
  flinkVersion: v1_17
  executionRoleArn: JOB_EXECUTION_ROLE_ARN
  emrReleaseLabel: "emr-6.15.0-flink-latest"
  jobManager:
    storageDir: HIGH_AVAILABILITY_S3_STORAGE_PATH
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  monitoringConfiguration:
    s3MonitoringConfiguration:
      logUri:
    cloudWatchMonitoringConfiguration:
      logGroupName: LOG_GROUP_NAME

```

2. 다음 명령으로 Flink 배포를 제출합니다. 이렇게 하면 `FlinkDeployment` 객체(`basic-example-session-cluster`)도 생성됩니다.

```
kubectl create -f basic-example-app-cluster.yaml -n NAMESPACE
```

3. 다음과 같은 명령을 사용하여 세션 클러스터 LIFECYCLE이 `STABLE`인지 확인하세요.

```
kubectl get flinkdeployments.flink.apache.org basic-example-session-cluster -n NAMESPACE
```

출력은 다음 예시와 비슷해야 합니다.

| NAME                          | JOB STATUS | LIFECYCLE STATE |
|-------------------------------|------------|-----------------|
| basic-example-session-cluster |            | STABLE          |

- 다음 콘텐츠 예제가 포함된 FlinkSessionJob 사용자 지정 정의 리소스 파일 `basic-session-job.yaml`을 생성합니다.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkSessionJob
metadata:
  name: basic-session-job
spec:
  deploymentName: basic-session-deployment
  job:
    # If you have your job jar in an S3 bucket you can use that path.
    # To use jar in S3 bucket, set
    # OPERATOR_EXECUTION_ROLE_ARN (--set emrContainers.operatorExecutionRoleArn=
    $OPERATOR_EXECUTION_ROLE_ARN)
    # when you install Spark operator
    jarURI: https://repo1.maven.org/maven2/org/apache/flink/flink-examples-
    streaming_2.12/1.16.1/flink-examples-streaming_2.12-1.16.1-TopSpeedWindowing.jar
    parallelism: 2
    upgradeMode: stateless
```

- 다음 명령으로 Flink 세션 작업을 제출합니다. 이렇게 하면 FlinkSessionJob 객체 `basic-session-job`이 생성됩니다.

```
kubectl apply -f basic-session-job.yaml -n $NAMESPACE
```

- 다음과 같은 명령을 사용하여 세션 클러스터 LIFECYCLE이 STABLE이고 JOB STATUS가 RUNNING인지 확인하세요.

```
kubectl get flinkdeployments.flink.apache.org basic-example-session-cluster -
n NAMESPACE
```

출력은 다음 예시와 비슷해야 합니다.

| NAME                          | JOB STATUS | LIFECYCLE STATE |
|-------------------------------|------------|-----------------|
| basic-example-session-cluster | RUNNING    | STABLE          |

## 7. Flink UI에 액세스합니다.

```
kubectl port-forward deployments/basic-example-session-cluster 8081 -n NAMESPACE
```

## 8. localhost:8081을 열어서 Flink 작업을 로컬에서 확인합니다.

## 9. 작업을 정리합니다. 이 작업을 위해 생성된 S3 아티팩트(예: 검사, 고가용성, 메타데이터 세이브포인트 및 CloudWatch 로그)를 정리해야 합니다.

# Flink 애플리케이션을 실행하기 위한 보안 역할 권한

이 주제에서는 Flink 애플리케이션을 배포하고 실행하기 위한 보안 역할에 대해 설명합니다. 배포를 관리하고 작업을 생성 및 관리하는 데 필요한 두 가지 역할(작업자 역할과 작업 역할)이 있습니다. 이 주제에서는 이러한 역할을 소개하고 해당 권한을 나열합니다.

## 역할 기반 액세스 제어

운영자를 배포하고 Flink 작업을 실행하려면 두 개의 Kubernetes 역할(즉, 운영자 하나와 작업 역할 하나)을 생성해야 합니다. Amazon EMR은 운영자를 설치할 때 기본적으로 두 가지 역할을 생성합니다.

## 운영자 역할

운영자 역할을 사용하여 flinkdeployments를 관리함으로써 각 Flink 작업 및 기타 리소스(예: 서비스)에 대한 JobManager를 생성하고 관리합니다.

운영자 역할의 기본 이름은 `emr-containers-sa-flink-operator`이며, 다음과 같은 권한이 필요합니다.

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - services
  - events
  - configmaps
  - secrets
  - serviceaccounts
  verbs:
  - '*'
- apiGroups:
  - rbac.authorization.k8s.io
```

```
resources:
- roles
- rolebindings
verbs:
- '*'
- apiGroups:
  - apps
  resources:
  - deployments
  - deployments/finalizers
  - replicasets
  verbs:
  - '*'
- apiGroups:
  - extensions
  resources:
  - deployments
  - ingresses
  verbs:
  - '*'
- apiGroups:
  - flink.apache.org
  resources:
  - flinkdeployments
  - flinkdeployments/status
  - flinksessionjobs
  - flinksessionjobs/status
  verbs:
  - '*'
- apiGroups:
  - networking.k8s.io
  resources:
  - ingresses
  verbs:
  - '*'
- apiGroups:
  - coordination.k8s.io
  resources:
  - leases
  verbs:
  - '*'
```

## 작업 역할

JobManager는 작업 역할을 사용하여 각 작업에 대한 TaskManager 및 ConfigMap을 생성하고 관리합니다.

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
  - configmaps
  verbs:
  - '*'
- apiGroups:
  - apps
  resources:
  - deployments
  - deployments/finalizers
  verbs:
  - '*'
```

## Amazon EMR on EKS에 대한 Flink Kubernetes 운영자 제거

다음 단계에 따라 Flink Kubernetes 운영자를 제거합니다.

1. 운영자를 삭제합니다.

```
helm uninstall flink-kubernetes-operator -n <NAMESPACE>
```

2. Helm에서 제거하지 않는 Kubernetes 리소스를 삭제합니다.

```
kubectl delete serviceaccounts, roles, rolebindings -l emr-
containers.amazonaws.com/component=flink.operator --namespace <namespace>
kubectl delete crd flinkdeployments.flink.apache.org
flinksessionjobs.flink.apache.org
```

3. (선택 사항) cert-manager를 삭제합니다.

```
kubectl delete -f https://github.com/jetstack/cert-manager/releases/download/
v1.12.0/cert-manager.yaml
```



# Flink 네이티브 Kubernetes 사용

Amazon EMR 릴리스 6.13.0 이상에서는 Flink 애플리케이션을 Amazon EMR on EKS 클러스터에 제출하고 해당 애플리케이션을 실행하는 데 사용할 수 있는 명령줄 도구로 Flink 네이티브 Kubernetes를 지원합니다.

## 주제

- [Amazon EMR on EKS에서 Flink 네이티브 Kubernetes 설정](#)
- [Amazon EMR on EKS용 Flink 네이티브 Kubernetes 시작하기](#)
- [네이티브 Kubernetes에 대한 Flink JobManager 서비스 계정 보안 요구 사항](#)

## Amazon EMR on EKS에서 Flink 네이티브 Kubernetes 설정

Amazon EMR on EKS에서 Flink CLI를 사용하여 애플리케이션을 실행하기 전에 다음 작업을 완료합니다. Amazon Web Services(AWS)에 이미 가입했고 Amazon EKS를 사용하고 있는 경우 Amazon EMR on EKS를 사용할 준비를 거의 마친 상태입니다. 필수 조건 중 하나를 이미 완료한 경우 해당 조건을 건너뛰고 다음 조건으로 넘어갈 수 있습니다.

- [의 최신 버전 설치 또는 업데이트 AWS CLI](#) -를 이미 설치한 경우 최신 버전이 있는지 AWS CLI 확인합니다.
- [Amazon EKS 시작하기 – eksctl](#) - 관련 단계를 수행하여 Amazon EKS에서 노드를 포함하는 새 Kubernetes 클러스터를 생성합니다.
- [Amazon EMR 기본 이미지 URI 선택](#)(릴리스 6.13.0 이상) - Flink Kubernetes 명령은 Amazon EMR 릴리스 6.13.0 이상에서 지원됩니다.
- JobManager 서비스 계정에 TaskManager 포드를 생성하고 감시할 수 있는 적절한 권한이 있는지 확인합니다. 자세한 내용은 [네이티브 Kubernetes에 대한 Flink JobManager 서비스 계정 보안 요구 사항](#)을 참조하세요.
- 로컬 [AWS 보안 인증 프로파일](#)을 설정합니다.
- Flink 애플리케이션을 실행하려는 [Amazon EKS 클러스터용 kubeconfig 파일을 생성 또는 업데이트](#)합니다.

## Amazon EMR on EKS용 Flink 네이티브 Kubernetes 시작하기

이 단계에서는 Flink 애플리케이션을 구성하고, 이에 대한 서비스 계정을 설정하며, 실행하는 방법을 보여줍니다. Flink 네이티브 Kubernetes는 실행 중인 Kubernetes 클러스터에 Flink를 배포하는 데 사용됩니다.

### Flink 애플리케이션 구성 및 실행

Amazon EMR 6.13.0 이상은 Amazon EKS 클러스터에서 Flink 애플리케이션을 실행하기 위한 Flink 네이티브 Kubernetes를 지원합니다. Flink 애플리케이션을 실행하려면 다음 단계를 수행합니다.

1. Flink 네이티브 Kubernetes 명령으로 Flink 애플리케이션을 실행하려면 먼저 [the section called “설정”](#)의 단계를 완료합니다.
2. [Flink 다운로드 및 설치](#).
3. 다음과 같은 환경 변수의 값을 설정합니다.

```
#Export the FLINK_HOME environment variable to your local installation of Flink
export FLINK_HOME=/usr/local/bin/flink #Will vary depending on your installation
export NAMESPACE=flink
export CLUSTER_ID=flink-application-cluster
export IMAGE=<123456789012.dkr.ecr.sample-AWS ##-.amazonaws.com/flink/emr-6.13.0-flink:latest>
export FLINK_SERVICE_ACCOUNT=emr-containers-sa-flink
export FLINK_CLUSTER_ROLE_BINDING=emr-containers-crb-flink
```

4. Kubernetes 리소스를 관리할 서비스 계정을 생성합니다.

```
kubectl create serviceaccount $FLINK_SERVICE_ACCOUNT -n $NAMESPACE
kubectl create clusterrolebinding $FLINK_CLUSTER_ROLE_BINDING --clusterrole=edit --serviceaccount=$NAMESPACE:$FLINK_SERVICE_ACCOUNT
```

5. run-application CLI 명령을 실행합니다.

```
$FLINK_HOME/bin/flink run-application \
  --target kubernetes-application \
  -Dkubernetes.namespace=$NAMESPACE \
  -Dkubernetes.cluster-id=$CLUSTER_ID \
  -Dkubernetes.container.image.ref=$IMAGE \
  -Dkubernetes.service-account=$FLINK_SERVICE_ACCOUNT \
  local:///opt/flink/examples/streaming/Iteration.jar
```

```

2022-12-29 21:13:06,947 INFO  org.apache.flink.kubernetes.utils.KubernetesUtils
    [] - Kubernetes deployment requires a fixed port. Configuration
    blob.server.port will be set to 6124
2022-12-29 21:13:06,948 INFO  org.apache.flink.kubernetes.utils.KubernetesUtils
    [] - Kubernetes deployment requires a fixed port. Configuration
    taskmanager.rpc.port will be set to 6122
2022-12-29 21:13:07,861 WARN
    org.apache.flink.kubernetes.KubernetesClusterDescriptor    [] - Please note that
    Flink client operations(e.g. cancel, list, stop, savepoint, etc.) won't work from
    outside the Kubernetes cluster since 'kubernetes.rest-service.exposed.type' has
    been set to ClusterIP.
2022-12-29 21:13:07,868 INFO
    org.apache.flink.kubernetes.KubernetesClusterDescriptor    [] - Create flink
    application cluster flink-application-cluster successfully, JobManager Web
    Interface: http://flink-application-cluster-rest.flink:8081
  
```

## 6. 생성된 Kubernetes 리소스를 검사합니다.

```

kubectl get all -n <namespace>
NAME READY STATUS RESTARTS AGE
pod/flink-application-cluster-546687cb47-w2p2z 1/1 Running 0 3m37s
pod/flink-application-cluster-taskmanager-1-1 1/1 Running 0 3m24s

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE
service/flink-application-cluster ClusterIP None <none> 6123/TCP,6124/TCP 3m38s
service/flink-application-cluster-rest ClusterIP 10.100.132.158 <none> 8081/TCP
3m38s

NAME READY UP-TO-DATE AVAILABLE AGE
deployment.apps/flink-application-cluster 1/1 1 1 3m38s

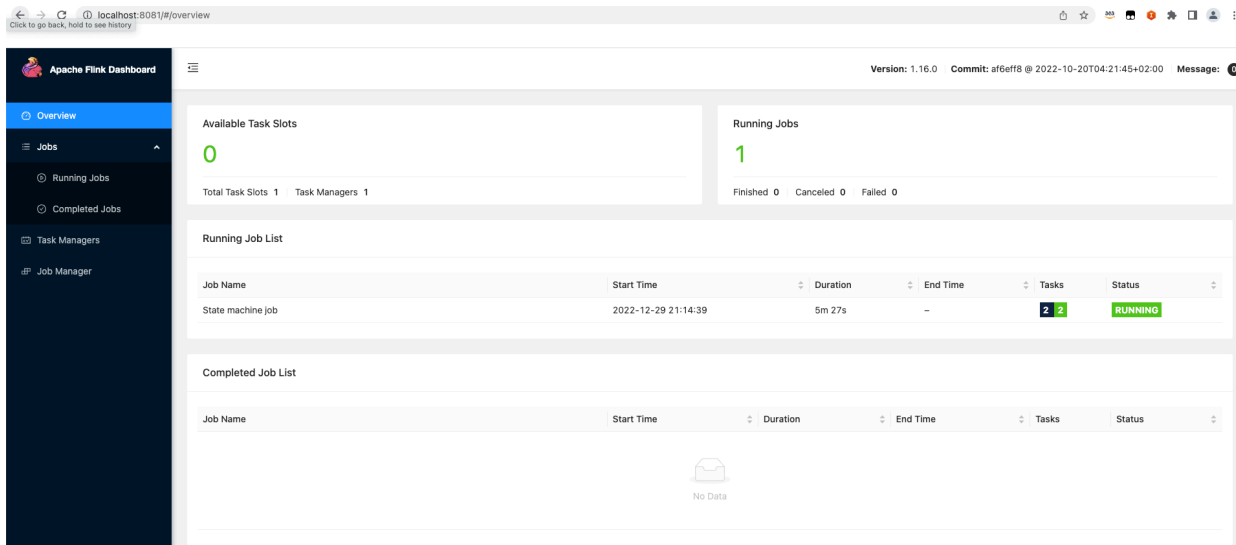
NAME DESIRED CURRENT READY AGE
replicaset.apps/flink-application-cluster-546687cb47 1 1 1 3m38s
  
```

## 7. 포트는 8081로 포워딩됩니다.

```

kubectl port-forward service/flink-application-cluster-rest 8081 -n <namespace>
Forwarding from 127.0.0.1:8081 -> 8081
  
```

## 8. Flink UI에 로컬로 액세스합니다.



## 9. Flink 애플리케이션을 삭제합니다.

```
kubectl delete deployment.apps/flink-application-cluster -n <namespace>
deployment.apps "flink-application-cluster" deleted
```

Flink에 애플리케이션을 제출하는 방법에 대한 자세한 내용은 Apache Flink 설명서에서 [Native Kubernetes](#)를 참조하세요.

## 네이티브 Kubernetes에 대한 Flink JobManager 서비스 계정 보안 요구 사항

Flink JobManager 포드는 Kubernetes 서비스 계정을 사용함으로써 Kubernetes API 서버에 액세스하여 TaskManager 포드를 생성하고 감시합니다. JobManager 서비스 계정에는 TaskManager 포드를 생성 및 삭제할 수 있는 적절한 권한이 있어야 하며, 클러스터에서 JobManager 및 ResourceManager의 주소를 검색하기 위해 TaskManager에서 리더 ConfigMaps를 감시하도록 허용해야 합니다.

이 서비스 계정에는 다음 규칙이 적용됩니다.

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - "*"
- apiGroups:
  - ""
```

```
resources:
- services
verbs:
- "*"
- apiGroups:
- ""
resources:
- configmaps
verbs:
- "*"
- apiGroups:
- "apps"
resources:
- deployments
verbs:
- "*"

```

## Flink 및 FluentD에 대한 Docker 이미지 사용자 지정

다음 단계를 수행하여 Apache Flink 또는 FluentD 이미지를 사용해 Amazon EMR on EKS에 대한 Docker 이미지를 사용자 지정합니다. 여기에는 기본 이미지 가져오기, 사용자 지정, 게시 및 워크로드 제출에 대한 기술 지침이 포함됩니다.

### 주제

- [사전 조건](#)
- [1단계: Amazon Elastic Container Registry에서 기본 이미지 검색](#)
- [2단계: 기본 이미지 사용자 지정](#)
- [3단계: 사용자 지정 이미지 게시](#)
- [4단계: 사용자 지정 이미지를 사용하여 Amazon EMR에서 Flink 워크로드 제출](#)

## 사전 조건

Docker 이미지를 사용자 지정하기 전에 다음 사전 조건을 완료했는지 확인합니다.

- [Amazon EMR on EKS에 대한 Flink Kubernetes 연산자 설정](#)을 완료했습니다.
- 환경에 Docker를 설치했습니다. 자세한 내용은 [Get Docker](#)를 참조하세요.

## 1단계: Amazon Elastic Container Registry에서 기본 이미지 검색

기본 이미지에는 다른 AWS 서비스에 액세스하는 데 사용되는 Amazon EMR 런타임 및 커넥터가 포함되어 있습니다. Flink 버전 6.14.0 이상의 EKS에서 Amazon EMR on EKS를 사용하는 경우 Amazon ECR 퍼블릭 갤러리에서 기본 이미지를 가져올 수 있습니다. 갤러리를 탐색하여 이미지 링크를 찾은 다음, 이미지를 로컬 Workspace로 가져옵니다. 예를 들어 Amazon EMR 6.14.0 릴리스의 경우 다음 `docker pull` 명령은 최신 표준 기본 이미지를 반환합니다. `emr-6.14.0:latest`를 원하는 릴리스 버전으로 바꿉니다.

```
docker pull public.ecr.aws/emr-on-eks/flink/emr-6.14.0-flink:latest
```

다음은 Flink 갤러리 이미지 및 Fluentd 갤러리 이미지에 대한 링크입니다.

- [emr-on-eks/flink/emr-6.14.0-flink](#)
- [emr-on-eks/fluentd/emr-6.14.0](#)

## 2단계: 기본 이미지 사용자 지정

다음 단계에서는 Amazon ECR에서 가져온 기본 이미지를 사용자 지정하는 방법을 설명합니다.

1. 로컬 Workspace에 새 Dockerfile을 생성합니다.
2. Dockerfile을 수정하고 다음 콘텐츠를 추가합니다. 이 Dockerfile에서는 `public.ecr.aws/emr-on-eks/flink/emr-7.13.0-flink:latest`에서 가져온 컨테이너 이미지를 사용합니다.

```
FROM public.ecr.aws/emr-on-eks/flink/emr-7.13.0-flink:latest
USER root
### Add customization commands here ###
USER hadoop:hadoop
```

Fluentd를 사용하는 경우 다음 구성을 사용합니다.

```
FROM public.ecr.aws/emr-on-eks/fluentd/emr-7.13.0:latest
USER root
### Add customization commands here ###
USER hadoop:hadoop
```

3. Dockerfile에 명령을 추가하여 기본 이미지를 사용자 지정합니다. 다음 명령은 Python 라이브러리를 설치하는 명령을 시연합니다.

```
FROM public.ecr.aws/emr-on-eks/flink/emr-7.13.0-flink:latest
USER root
RUN pip3 install --upgrade boto3 pandas numpy // For python 3
USER hadoop:hadoop
```

4. DockerFile을 생성한 같은 디렉터리에서 다음 명령을 실행하여 Docker 이미지를 빌드합니다. -t 플래그 다음에 제공하는 필드는 이미지의 사용자 지정 이름입니다.

```
docker build -t <YOUR_ACCOUNT_ID>.dkr.ecr.<YOUR_ECR_REGION>.amazonaws.com/
<ECR_REPO>:<ECR_TAG>
```

### 3단계: 사용자 지정 이미지 게시

이제 Amazon ECR 레지스트리에 새 Docker 이미지를 게시할 수 있습니다.

1. 다음 명령을 실행하여 Docker 이미지를 저장할 Amazon ECR 리포지토리를 생성합니다. 리포지토리의 이름을 입력합니다(예: emr\_custom\_repo.). 자세한 내용은 Amazon Elastic Container Registry 사용 설명서의 [리포지토리 생성](#)을 참조하세요.

```
aws ecr create-repository \
    --repository-name emr_custom_repo \
    --image-scanning-configuration scanOnPush=true \
    --region <AWS_REGION>
```

2. 다음 명령을 실행하여 기본 레지스트리에 인증합니다. 자세한 내용은 Amazon Elastic Container Registry 사용 설명서의 [기본 레지스트리에 대해 인증](#)을 참조하세요.

```
aws ecr get-login-password --region <AWS_REGION> | docker login --username AWS --
password-stdin <AWS_ACCOUNT_ID>.dkr.ecr.<YOUR_ECR_REGION>.amazonaws.com
```

3. 이미지를 푸시합니다. 자세한 내용은 Amazon Elastic Container Registry 사용 설명서의 [Amazon ECR에 이미지 푸시](#)를 참조하세요.

```
docker push <YOUR_ACCOUNT_ID>.dkr.ecr.<YOUR_ECR_REGION>.amazonaws.com/
<ECR_REPO>:<ECR_TAG>
```

## 4단계: 사용자 지정 이미지를 사용하여 Amazon EMR에서 Flink 워크로드 제출

사용자 지정 이미지를 사용하려면 FlinkDeployment 사양을 다음과 같이 변경합니다. 이를 수행하려면 배포 사양의 spec.image 줄에 자체 이미지를 입력합니다.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example
spec:
  flinkVersion: v1_18
  image: <YOUR_ACCOUNT_ID>.dkr.ecr.<YOUR_ECR_REGION>.amazonaws.com/
  <ECR_REPO>:<ECR_TAG>
  imagePullPolicy: Always
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "1"
```

Fluentd 작업에 사용자 지정 이미지를 사용하려면 배포 사양의 monitoringConfiguration.image 줄에 자체 이미지를 입력합니다.

```
monitoringConfiguration:
  image: <YOUR_ACCOUNT_ID>.dkr.ecr.<YOUR_ECR_REGION>.amazonaws.com/
  <ECR_REPO>:<ECR_TAG>
  cloudWatchMonitoringConfiguration:
    logGroupName: flink-log-group
    logStreamNamePrefix: custom-fluentd
```

## Flink Kubernetes 운영자 및 Flink 작업 모니터링

이 섹션에서는 Amazon EMR on EKS를 사용하여 Flink 작업을 모니터링할 수 있는 여러 가지 방법을 설명합니다. 여기에는 Amazon Managed Service for Prometheus와의 Flink 통합, 작업 상태 및 지표를 제공하는 Flink 웹 대시보드 사용 또는 모니터링 구성을 사용하여 Amazon S3 및 Amazon CloudWatch로 로그 데이터 전송이 포함됩니다.

### 주제

- [Amazon Managed Service for Prometheus를 사용하여 Flink 작업 모니터링](#)
- [Flink UI를 사용하여 Flink 작업 모니터링](#)



- [모니터링 구성을 사용하여 Flink Kubernetes 연산자 및 Flink 작업 모니터링](#)

## Amazon Managed Service for Prometheus를 사용하여 Flink 작업 모니터링

Amazon Managed Service for Prometheus(관리 포털)와 Apache Flink를 통합할 수 있습니다. Amazon Managed Service for Prometheus는 Amazon EKS에서 실행되는 클러스터에서 Amazon Managed Service for Prometheus 서버의 지표 수집을 지원합니다. Amazon Managed Service for Prometheus는 Amazon EKS 클러스터에서 이미 실행 중인 Prometheus 서버와 함께 작동합니다. Amazon EMR Flink 운영자와의 Amazon Managed Service for Prometheus 통합을 실행하면 Amazon Managed Service for Prometheus와 통합되도록 Prometheus 서버를 자동으로 배포하고 구성합니다.

1. [Amazon Managed Service for Prometheus Workspace를 생성합니다](#). 이 Workspace는 수집 엔드 포인트 역할을 합니다. 나중에 원격 쓰기 URL이 필요합니다.
2. 서비스 계정에 대한 IAM 역할 설정.

이 온보딩 방법에서는 Prometheus 서버가 실행되는 Amazon EKS 클러스터의 서비스 계정에 대한 IAM 역할을 사용합니다. 이러한 역할을 서비스 역할이라고도 합니다.

아직 역할이 없는 경우 [Amazon EKS 클러스터의 지표 수집을 위한 서비스 역할을 설정](#)합니다.

계속하기 전에 `amp-iamproxy-ingest-role`이라는 IAM 역할을 생성합니다.

3. Amazon Managed Service for Prometheus와 함께 Amazon EMR Flink 운영자를 설치합니다.

이제 Amazon Managed Service for Prometheus Workspace, Amazon Managed Service for Prometheus에 대한 전용 IAM 역할 및 필요한 권한이 확보되었으므로 Amazon EMR Flink 운영자를 설치할 수 있습니다.

`enable-amp.yaml` 파일을 생성합니다. 이 파일을 사용하면 사용자 지정 구성을 사용하여 Amazon Managed Service for Prometheus 설정을 재정의할 수 있습니다. 고유한 역할을 사용해야 합니다.

```
kube-prometheus-stack:
  prometheus:
    serviceAccount:
      create: true
      name: "amp-iamproxy-ingest-service-account"
      annotations:
        eks.amazonaws.com/role-arn: "arn:aws:iam::<AWS_ACCOUNT_ID>:role/amp-iamproxy-ingest-role"
      remoteWrite:
```

```
- url: <AMAZON_MANAGED_PROMETHEUS_REMOTE_WRITE_URL>
sigv4:
  region: <AWS_REGION>
queueConfig:
  maxSamplesPerSend: 1000
  maxShards: 200
  capacity: 2500
```

[Helm Install --set](#) 명령을 사용하여 flink-kubernetes-operator 차트에 재정의의 전달합니다.

```
helm upgrade -n <namespace> flink-kubernetes-operator \
oci://public.ecr.aws/emr-on-eks/flink-kubernetes-operator \
--set prometheus.enabled=true
-f enable-amp.yaml
```

이 명령은 포트 9999의 연산자에 Prometheus 리포터를 자동으로 설치합니다. 향후 FlinkDeployment는 9249의 metrics 포트를 공개합니다.

- Flink 운영자 지표는 Prometheus의 flink\_k8soperator\_ 레이블 아래에 표시됩니다.
- Flink 태스크 관리자 지표는 Prometheus의 flink\_taskmanager\_ 레이블 아래에 표시됩니다.
- Flink 작업 관리자 지표는 Prometheus의 flink\_jobmanager\_ 레이블 아래에 표시됩니다.

## Flink UI를 사용하여 Flink 작업 모니터링

실행 중인 Flink 애플리케이션의 상태와 성능을 모니터링하려면 Flink 웹 대시보드를 사용합니다. 이 대시보드는 작업 상태, TaskManager 수, 작업에 대한 지표 및 로그에 대한 정보를 제공합니다. 또한 Flink 작업의 구성을 확인 및 수정하고 Flink 클러스터와 상호 작용하여 작업을 제출하거나 취소할 수 있습니다.

Kubernetes에서 실행 중인 Flink 애플리케이션의 Flink 웹 대시보드에 액세스하는 방법:

1. `kubectl port-forward` 명령을 사용하여 Flink 애플리케이션의 TaskManager 포트에서 Flink 웹 대시보드가 실행되는 포트에 로컬 포트를 전달합니다. 기본적으로 이 포트는 8081입니다. `deployment-name`을 위에서 언급한 Flink 애플리케이션 배포 이름으로 바꿉니다.

```
kubectl get deployments -n namespace
```

출력 예시:

```
kubectl get deployments -n flink-namespace
```

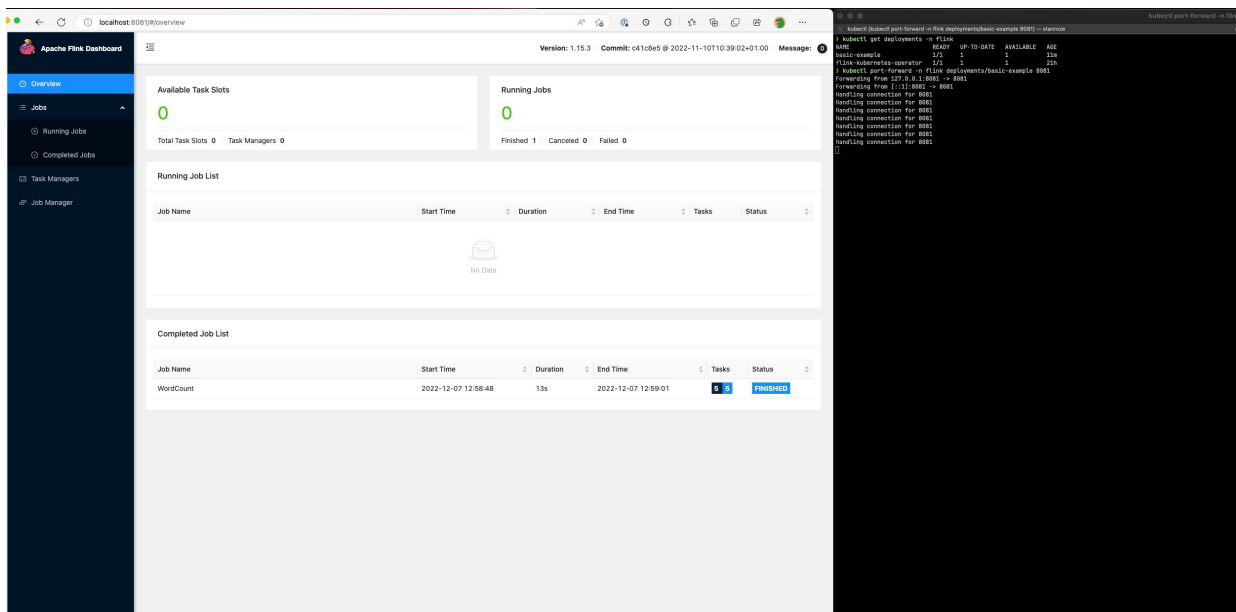
| NAME                      | READY | UP-TO-DATE | AVAILABLE | AGE |
|---------------------------|-------|------------|-----------|-----|
| basic-example             | 1/1   | 1          | 1         | 11m |
| flink-kubernetes-operator | 1/1   | 1          | 1         | 21h |

```
kubectl port-forward deployments/deployment-name 8081 -n namespace
```

2. 로컬에서 다른 포트를 사용하려면 *local-port*:8081 파라미터를 사용합니다.

```
kubectl port-forward -n flink deployments/basic-example 8080:8081
```

3. 웹 브라우저에서 Flink 웹 대시보드에 액세스하려면 `http://localhost:8081` 또는 사용자 지정 로컬 포트를 사용하는 경우 `http://localhost:local-port`로 이동합니다. 이 대시보드에는 작업 상태, TaskManager 수, 작업에 대한 지표 및 로그와 같은 실행 중인 Flink 애플리케이션에 대한 정보가 표시됩니다.



## 모니터링 구성을 사용하여 Flink Kubernetes 연산자 및 Flink 작업 모니터링

모니터링 구성을 사용하면 Flink 애플리케이션 및 운영자 로그의 로그 아카이브를 S3 및/또는 CloudWatch로 쉽게 설정할 수 있습니다(둘 중 하나 또는 둘 다 선택 가능). 이렇게 하면 JobManager 및 TaskManager 포드에 FluentD sidecar가 추가되고 이후에 이러한 구성 요소의 로그가 구성된 싱크로 전달됩니다.

**Note**

이 기능을 사용하려면 다른 AWS 서비스와 상호 작용해야 하므로 Flink 운영자의 서비스 계정 및 Flink 작업(서비스 계정)에 대해 IAM 역할을 설정해야 합니다. [Amazon EMR on EKS에 대한 Flink Kubernetes 운영자 설정](#)에서 IRSA를 사용하여 설정해야 합니다.

## Flink 애플리케이션 로그

다음과 같은 방법으로 이 구성을 정의할 수 있습니다.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example
spec:
  image: FLINK IMAGE TAG
  imagePullPolicy: Always
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
  executionRoleArn: JOB EXECUTION ROLE
  jobManager:
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: local:///opt/flink/examples/streaming/StateMachineExample.jar
  monitoringConfiguration:
    s3MonitoringConfiguration:
      logUri: S3 BUCKET
    cloudWatchMonitoringConfiguration:
      logGroupName: LOG GROUP NAME
      logStreamNamePrefix: LOG GROUP STREAM PREFIX
  sideCarResources:
    limits:
      cpuLimit: 500m
      memoryLimit: 250Mi
```

```
containerLogRotationConfiguration:
  rotationSize: 2GB
  maxFilesToKeep: 10
```

유효한 구성 옵션은 다음과 같습니다.

- `s3MonitoringConfiguration` - S3로의 전달을 설정하기 위한 구성 키
  - `logUri`(필수) - 로그를 저장할 S3 버킷 경로.
  - 로그가 업로드되고 나면 S3의 경로는 다음과 같습니다.
  - 로그 로테이션이 활성화되지 않았습니다.

```
s3://${logUri}/${POD_NAME}/STDOUT or STDERR.gz
```

- 로그 로테이션이 활성화되었습니다. 로테이션된 파일과 현재 파일(날짜 스탬프가 없는 파일)을 모두 사용할 수 있습니다.

```
s3://${logUri}/${POD_NAME}/STDOUT or STDERR.gz
```

다음 형식은 증가하는 숫자입니다.

```
s3://${logUri}/${POD_NAME}/stdout_YYYYMMDD_index.gz
```

- 이 전달자를 사용하려면 다음 IAM 권한이 필요합니다.

```
{
  "Effect": "Allow",
  "Action": [
    "s3:PutObject"
  ],
  "Resource": [
    "${S3_BUCKET_URI}/*",
    "${S3_BUCKET_URI}"
  ]
}
```

- `cloudWatchMonitoringConfiguration` - CloudWatch로의 전달을 설정하기 위한 구성 키.
  - `logGroupName`(필수) - 로그를 전송하려는 CloudWatch 로그 그룹의 이름(그룹이 없는 경우 자동으로 생성됨).

- `logStreamNamePrefix`(선택 사항) - 로그를 보낼 로그 스트림의 이름입니다. 기본값은 빈 문자열입니다. 형식은 다음과 같습니다.

```
${logStreamNamePrefix}/${POD_NAME}/STDOUT or STDERR
```

- 이 전달자를 사용하려면 다음 IAM 권한이 필요합니다.

```
{
  "Effect": "Allow",
  "Action": [
    "logs:CreateLogStream",
    "logs:CreateLogGroup",
    "logs:PutLogEvents"
  ],
  "Resource": [
    "arn:aws:logs:REGION:ACCOUNT-ID:log-group:{YOUR_LOG_GROUP_NAME}:",
    "arn:aws:logs:REGION:ACCOUNT-ID:log-group:{YOUR_LOG_GROUP_NAME}"
  ]
}
```

- `sideCarResources`(선택 사항) - 시작된 Fluentbit sidecar 컨테이너에서 리소스 한도를 설정하기 위한 구성 키.
- `memoryLimit`(선택 사항) - 기본값은 512Mi입니다. 필요에 따라 조정합니다.
- `cpuLimit`(선택 사항) - 이 옵션에는 기본값이 없습니다. 필요에 따라 조정합니다.
- `containerLogRotationConfiguration`(선택 사항) - 컨테이너 로그 로테이션 동작을 제어합니다. 기본적으로 활성화됩니다.
  - `rotationSize`(필수) - 로그 로테이션을 위한 파일 크기를 지정합니다. 가능한 값 범위는 2KB에서 2GB 사이입니다. `rotationSize` 파라미터의 숫자 단위 부분은 정수로 전달됩니다. 십진수는 지원되지 않으므로 로테이션 크기를 1.5GB(예: 1,500MB 값)로 지정할 수 있습니다. 기본값은 2GB입니다.
  - `maxFilesToKeep`(필수) - 로테이션을 수행한 후 컨테이너에서 보존할 최대 파일 수를 지정합니다. 최솟값은 1이고 최댓값은 50입니다. 기본값은 10입니다.

## Flink 운영자 로그

또한 차트 Helm 설치에 관한 `values.yaml` 파일에서 다음 옵션을 사용하여 운영자를 위한 로그 아카이브를 활성화할 수 있습니다. S3, CloudWatch 또는 둘 다 활성화할 수 있습니다.

```
monitoringConfiguration:
  s3MonitoringConfiguration:
    logUri: "S3-BUCKET"
    totalFileSize: "1G"
    uploadTimeout: "1m"
  cloudWatchMonitoringConfiguration:
    logGroupName: "flink-log-group"
    logStreamNamePrefix: "example-job-prefix-test-2"
  sideCarResources:
    limits:
      cpuLimit: 1
      memoryLimit: 800Mi
    memoryBufferLimit: 700M
```

monitoringConfiguration 아래에서 다음과 같은 구성 옵션을 사용할 수 있습니다.

- s3MonitoringConfiguration - S3에 아카이브하려면 이 옵션을 설정합니다.
- logUri(필수) - 로그를 저장할 S3 버킷 경로.
- 다음은 로그가 업로드된 후의 S3 버킷 경로 형식입니다.
- 로그 로테이션이 활성화되지 않았습니다.

```
s3://{logUri}/{POD NAME}/OPERATOR or WEBHOOK/STDOUT or STDERR.gz
```

- 로그 로테이션이 활성화되었습니다. 로테이션된 파일과 현재 파일(날짜 스탬프가 없는 파일)을 모두 사용할 수 있습니다.

```
s3://{logUri}/{POD NAME}/OPERATOR or WEBHOOK/STDOUT or STDERR.gz
```

다음 형식 인덱스는 증가하는 숫자입니다.

```
s3://{logUri}/{POD NAME}/OPERATOR or WEBHOOK/stdout_YYYYMMDD_index.gz
```

- cloudWatchMonitoringConfiguration - CloudWatch로의 전달을 설정하기 위한 구성 키.
- logGroupName(필수) - 로그를 보낼 CloudWatch 로그 그룹의 이름. 그룹이 없으면 그룹이 자동으로 생성됩니다.
- logStreamNamePrefix(선택 사항) - 로그를 보낼 로그 스트림의 이름. 기본값은 빈 문자열입니다. CloudWatch의 형식은 다음과 같습니다.

```
${logStreamNamePrefix}/${POD_NAME}/STDOUT or STDERR
```

- `sidecarResources`(선택 사항) - 시작된 Fluentbit sidecar 컨테이너에서 리소스 한도를 설정하기 위한 구성 키.
- `memoryLimit`(선택 사항) - 메모리 한도. 필요에 따라 조정합니다. 기본값은 512Mi입니다.
- `cpuLimit` - CPU 한도. 필요에 따라 조정합니다. 기본값은 없습니다.
- `containerLogRotationConfiguration`(선택 사항) - 컨테이너 로그 로테이션 동작을 제어합니다. 기본적으로 활성화됩니다.
- `rotationSize`(필수) - 로그 로테이션을 위한 파일 크기를 지정합니다. 가능한 값 범위는 2KB에서 2GB 사이입니다. `rotationSize` 파라미터의 숫자 단위 부분은 정수로 전달됩니다. 십진수는 지원되지 않으므로 로테이션 크기를 1.5GB(예: 1,500MB 값)로 지정할 수 있습니다. 기본값은 2GB입니다.
- `maxFilesToKeep`(필수) - 로테이션을 수행한 후 컨테이너에서 보존할 최대 파일 수를 지정합니다. 최솟값은 1이고 최댓값은 50입니다. 기본값은 10입니다.

## Flink에서 고가용성 및 작업 복원력을 지원하는 방법

다음 섹션에서는 Flink 작업의 신뢰성과 가용성을 높이는 방법을 간략하게 설명합니다. Flink 고가용성 및 장애가 발생할 경우 다양한 복구 기능과 같은 내장 기능을 통해 이를 수행합니다.

### 주제

- [Flink 운영자 및 Flink 애플리케이션을 위한 고가용성\(HA\) 사용](#)
- [Amazon EMR on EKS로 태스크 복구 및 작업 규모 조정을 위한 Flink 작업 재시작 시간 최적화](#)
- [Amazon EMR on EKS에서 Flink를 사용한 스팟 인스턴스의 정상적인 서비스 해제](#)

## Flink 운영자 및 Flink 애플리케이션을 위한 고가용성(HA) 사용

이 주제에서는 고가용성을 구성하는 방법 및 몇 가지 사용 사례에 대한 작동 방식을 설명합니다. 여기에는 작업 관리자를 사용하는 경우와 Flink 네이티브 kubernetes를 사용하는 경우가 포함됩니다.

### Flink 운영자 고가용성

Flink 운영자의 고가용성을 활성화하여 장애 발생 시 대기 중인 Flink 운영자로 장애 조치하여 운영자 제어 루프에서 가동 중단을 최소화할 수 있습니다. 고가용성은 기본적으로 활성화되어 있으며 시작 운



영자 복제본의 기본 수는 2입니다. 차트 Helm에 대해 `values.yaml` 파일의 복제본 필드를 구성할 수 있습니다.

다음 필드를 사용자 지정 가능합니다.

- `replicas`(선택 사항, 기본값: 2): 이 숫자를 1보다 크게 설정하면 다른 대기 운영자가 생성되고 작업을 더 빠르게 복구할 수 있습니다.
- `highAvailabilityEnabled`(선택 사항, 기본값: true): HA 활성화 여부를 제어합니다. 이 파라미터를 true로 지정하면 다중 AZ 배포를 지원하고 올바른 `flink-conf.yaml` 파라미터를 설정할 수 있습니다.

`values.yaml` 파일에서 다음 구성을 설정하여 운영자의 HA를 비활성화할 수 있습니다.

```
...
imagePullSecrets: []

replicas: 1

# set this to false if you don't want HA
highAvailabilityEnabled: false
...
```

## 다중 AZ 배포

여러 가용 영역에서 운영자 포드를 생성합니다. 이는 약한 제약 조건이며, 다른 AZ에 충분한 리소스가 없는 경우 운영자 포드가 동일한 AZ에서 예약됩니다.

## 리더 복제본 결정

HA가 활성화된 경우 복제본은 리스 기능을 사용하여 어떤 JM이 리더인지 결정하고 리더 선택에 K8s Lease를 사용합니다. 리스 기능을 설명하고 `.Spec.Holder Identity` 필드를 확인하여 현재 리더를 결정할 수 있습니다.

```
kubectl describe lease <Helm Install Release Name>-<NAMESPACE>-lease -n <NAMESPACE> |
grep "Holder Identity"
```

## Flink-S3 상호 작용

## 액세스 보안 인증 구성

S3 버킷에 액세스할 수 있는 적절한 IAM 권한으로 IRSA를 구성했는지 확인하세요.

### S3 애플리케이션 모드에서 작업 jar 가져오기

Flink 운영자는 S3에서 애플리케이션 jar 가져오기도 지원합니다. FlinkDeployment 사양에서 JarURI의 S3 위치를 제공하면 됩니다.

이 기능을 사용하여 PyFlink 스크립트와 같은 다른 아티팩트를 다운로드할 수도 있습니다. 결과 Python 스크립트는 /opt/flink/usrlib/ 경로 아래에 배치됩니다.

다음 예제에서는 PyFlink 작업에 대해 이 기능을 사용하는 방법을 보여줍니다. jarURI 및 args 필드를 기록합니다.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: python-example
spec:
  image: <YOUR CUSTOM PYFLINK IMAGE>
  emrReleaseLabel: "emr-6.12.0-flink-latest"
  flinkVersion: v1_16
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "1"
  serviceAccount: flink
  jobManager:
    highAvailabilityEnabled: false
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: "s3://<S3-BUCKET>/scripts/pyflink.py" # Note, this will trigger the
    artifact download process
    entryClass: "org.apache.flink.client.python.PythonDriver"
    args: ["-pyclientexec", "/usr/local/bin/python3", "-py", "/opt/flink/usrlib/
    pyflink.py"]
    parallelism: 1
    upgradeMode: stateless
```

## Flink S3 커넥터

Flink는 두 개의 S3 커넥터(아래 목록 참조)와 함께 제공됩니다. 다음 섹션에서는 어떤 커넥터를 언제 사용해야 하는지를 설명합니다.

검사: Presto S3 커넥터

- S3 스키마를 `s3p://`로 설정
- s3에 대한 검사에 사용할 권장 커넥터. 자세한 내용은 Apache Flink 설명서의 [S3-specific](#)을 참조하세요.

FlinkDeployment 사양 예제:

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    state.checkpoints.dir: s3p://<BUCKET-NAME>/flink-checkpoint/
```

## S3: Hadoop S3 커넥터에 대한 읽기 및 쓰기

- S3 스키마를 `s3://` 또는 `s3a://`로 설정
- S3에서 파일을 읽고 쓰는 데 권장되는 커넥터([Flinks 파일 시스템 인터페이스](#)를 구현하는 S3 커넥터만 해당).
- 기본적으로 `flink-conf.yaml` 파일에서 `fs.s3a.aws.credentials.provider`를 설정합니다 (`com.amazonaws.auth.WebIdentityTokenCredentialsProvider`). 기본값(`flink-conf`)을 완전히 재정의하고 S3와 상호 작용하는 경우 이 제공업체를 사용해야 합니다.

FlinkDeployment 사양 예제

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example
spec:
  job:
```

```

jarURI: local:///opt/flink/examples/streaming/WordCount.jar
args: [ "--input", "s3a://<INPUT BUCKET>/PATH", "--output", "s3a://<OUTPUT BUCKET>/PATH" ]
parallelism: 2
upgradeMode: stateless

```

## Flink 작업 관리자

Flink 배포용 고가용성(HA)을 통해 일시적인 오류가 발생하여 JobManager가 충돌하더라도 작업을 계속 진행할 수 있습니다. 작업은 HA가 활성화된 상태에서 마지막으로 성공한 체크포인트부터 다시 시작됩니다. HA를 활성화하지 않으면 Kubernetes는 JobManager를 다시 시작하지만 작업은 새 작업으로 시작되고 진행 상황은 사라집니다. HA를 구성한 후에 JobManager에서 일시적인 오류가 발생할 경우 참조할 수 있도록 HA 메타데이터를 영구 스토리지에 저장한 다음, 마지막으로 성공한 체크포인트에서 작업을 재개하도록 Kubernetes에 지시할 수 있습니다.

Flink 작업에 대해 HA는 기본적으로 활성화되어 있습니다(복제본 수는 2로 설정되어 있으며, 이 경우 HA 메타데이터가 지속되려면 S3 스토리지 위치를 제공해야 함).

## HA 구성

```

apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: basic-example
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
  executionRoleArn: "<JOB EXECUTION ROLE ARN>"
  emrReleaseLabel: "emr-6.13.0-flink-latest"
  jobManager:
    resource:
      memory: "2048m"
      cpu: 1
    replicas: 2
    highAvailabilityEnabled: true
    storageDir: "s3://<S3 PERSISTENT STORAGE DIR>"
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1

```

다음은 작업 관리자(.spec.jobManager에서 정의됨)에서 위 HA 구성에 대한 설명입니다.

- `highAvailabilityEnabled`(선택 사항, 기본값: `true`): HA를 활성화하지 않고 제공된 HA 구성을 사용하지 않으려면 이 옵션을 `false` 로 설정합니다. 여전히 '복제본' 필드를 조작하여 HA를 수동으로 구성할 수 있습니다.
- `replicas`(선택 사항, 기본값: 2): 이 숫자를 1보다 크게 설정하면 다른 대기 `JobManagers`가 생성되고 작업을 더 빠르게 복구할 수 있습니다. HA를 비활성화하는 경우 복제본 수를 1로 설정해야 합니다. 그렇지 않으면 검증 오류가 계속 발생합니다(HA가 활성화되지 않은 경우 복제본 1개만 지원됨).
- `storageDir`(필수): 기본적으로 복제본 수를 2로 사용하기 때문에 영구 `storageDir`을 제공해야 합니다. 현재 이 필드에는 스토리지 위치로 S3 경로만 허용합니다.

## 포드 지역성

또한 HA를 활성화하는 경우 동일한 AZ에 파드를 배치하려고 시도하므로 성능이 개선됩니다(동일한 AZ에 포드를 배치하여 네트워크 지연 시간 감소). 이는 최대한의 원칙이 적용되는 프로세스입니다. 즉, 대부분의 포드가 예약된 AZ에 충분한 리소스가 없는 경우 나머지 포드는 여전히 예약되지만 결국 이 AZ 외부의 노드에 배치될 수 있습니다.

## 리더 복제본 결정

HA가 활성화된 경우 복제본은 리스 기능을 통해 어떤 JM이 리더인지 결정하고 이 메타데이터를 저장할 데이터 스토어로 K8s Configmap을 사용합니다. 리더를 결정하려면 Configmap의 콘텐츠를 살펴보고 데이터 아래에 있는 `org.apache.flink.k8s.leader.restserver` 키를 확인하여 IP 주소가 있는 K8s 포드를 찾을 수 있습니다. 다음과 같은 `bash` 명령을 사용할 수 있습니다.

```
ip=$(kubectl get configmap -n <NAMESPACE> <JOB-NAME>-cluster-config-map -o json | jq -r ".data[\"org.apache.flink.k8s.leader.restserver\"]" | awk -F: '{print $2}' | awk -F '/' '{print $3}')
kubectl get pods -n NAMESPACE -o json | jq -r ".items[] | select(.status.podIP == \"$ip\") | .metadata.name"
```

## Flink 작업 - 네이티브 Kubernetes

Amazon EMR 6.13.0 이상은 Amazon EKS 클러스터에서 고가용성 모드의 Flink 애플리케이션을 실행하기 위한 Flink 네이티브 Kubernetes를 지원합니다.

### Note

Flink 작업을 제출할 때 고가용성 메타데이터를 저장할 Amazon S3 버킷을 생성해야 합니다. 이 기능을 사용하고 싶지 않은 경우 비활성화할 수 있습니다. 기본적으로 활성화됩니다.

Flink 고가용성 특성을 활성화하려면 [run-application CLI 명령](#)을 실행할 때 다음 Flink 파라미터를 입력하세요. 파라미터는 예제 아래에 정의되어 있습니다.

```
-Dhigh-availability.type=kubernetes \
-Dhigh-availability.storageDir=S3://DOC-EXAMPLE-STORAGE-BUCKET \
-
Dfs.s3a.aws.credentials.provider="com.amazonaws.auth.WebIdentityTokenCredentialsProvider" \
-Dkubernetes.jobmanager.replicas=3 \
-Dkubernetes.cluster-id=example-cluster
```

- **Dhigh-availability.storageDir** – 작업을 위한 고가용성 메타데이터를 저장할 Amazon S3 버킷입니다.

**Dkubernetes.jobmanager.replicas** – 1보다 큰 정수로 생성할 작업 관리자 포드의 수입니다.

**Dkubernetes.cluster-id** – Flink 클러스터를 식별하는 고유한 ID입니다.

## Amazon EMR on EKS로 태스크 복구 및 작업 규모 조정을 위한 Flink 작업 재시작 시간 최적화

태스크가 실패하거나 규모 조정 작업이 발생할 경우 Flink는 마지막으로 완료된 체크포인트의 태스크를 다시 재실행하려고 시도합니다. 체크포인트 상태의 크기와 병렬 태스크의 수에 따라 재시작 프로세스를 실행하는 데 1분 이상 소요될 수 있습니다. 프로세스를 다시 시작하는 동안에는 작업에 대한 백로그 태스크가 누적될 수 있습니다. 그렇지만 Flink는 실행 그래프의 복구 및 재시작 속도를 최적화하여 작업 안정성을 향상시킬 수 있는 방법이 몇 가지 있습니다.

이 페이지에서는 Amazon EMR Flink를 사용하여 스팟 인스턴스에서의 태스크 복구 및 조정 작업 중에 작업 재시작 시간을 개선할 수 있는 몇 가지 방법을 설명합니다. 스팟 인스턴스는 할인된 가격으로 사용할 수 있는 미사용 컴퓨팅 용량입니다. 가끔 중단되는 것을 포함하여 고유한 동작이 있으므로, Amazon EMR on EKS가 서비스 해제를 수행하고 작업을 재시작하는 방법을 비롯해 Amazon EMR on EKS에서 이러한 동작을 처리하는 방법을 이해하는 것이 중요합니다.

### 주제

- [태스크-로컬 복구](#)
- [Amazon EBS 볼륨 마운트를 통한 태스크-로컬 복구](#)
- [일반 로그 기반 증분 체크포인트](#)
- [세분화된 복구](#)

- [적응형 스케줄러의 결합된 재시작 메커니즘](#)

## 태스크-로컬 복구

### Note

태스크-로컬 복구는 EKS 6.14.0 이상의 Amazon EMR on EKS에서 Flink를 통해 지원됩니다.

Flink 체크포인트를 사용할 경우 각 태스크에서 Flink가 Amazon S3와 같은 분산 스토리지에 기록하는 상태 스냅샷을 만듭니다. 복구의 경우 태스크는 분산 스토리지를 통해 해당 상태를 복원합니다. 분산 스토리지에서는 내결함성을 제공하며 모든 노드에서 액세스가 가능하기 때문에 크기 재조정이 이뤄지는 동안 상태를 재분배할 수 있습니다.

하지만 원격 분산 저장소에는 모든 태스크에서 네트워크를 통해 원격 위치에서 해당 상태를 읽어야 한다는 단점도 있습니다. 이러한 한계로 인해 태스크 복구 또는 규모 조정 작업 중에 대규모 상태의 복구 시간이 길어질 수 있습니다.

이와 같은 긴 복구 시간 문제는 태스크-로컬 복구를 통해 해결됩니다. 태스크에서는 체크포인트의 상태를 로컬 디스크와 같이 해당 작업에 대해 로컬인 보조 스토리지에 기록합니다. 또한 태스크는 기본 스토리지(이 경우 Amazon S3)에 상태를 저장합니다. 복구가 진행되는 동안 스케줄러는 태스크가 이전에 실행된 동일한 태스크 관리자에서 태스크를 예약하기 때문에 원격 상태 저장소에서 데이터를 읽는 대신 로컬 상태 저장소에서 복구할 수 있습니다. 자세한 내용을 알아보려면 Apache Flink 설명서의 [태스크 로컬 복구](#)를 참조하세요.

샘플 작업을 사용한 벤치마크 테스트 결과에 따르면 태스크-로컬 복구가 활성화된 상태에서는 복구 시간이 몇 분에서 몇 초로 단축된 것으로 확인되었습니다.

태스크-로컬 복구를 활성화하려면 `flink-conf.yaml` 파일에 다음과 같은 구성을 설정하세요. 체크포인트 간격의 값을 밀리초 단위로 지정하세요.

```
state.backend.local-recovery: true
state.backend: hasmap or rocksdb
state.checkpoints.dir: s3://STORAGE-BUCKET-PATH/checkpoint
execution.checkpointing.interval: 15000
```

## Amazon EBS 볼륨 마운트를 통한 태스크-로컬 복구

### Note

Amazon EBS를 통한 태스크-로컬 복구는 Amazon EMR on EKS 6.15.0 이상의 Flink를 통해 지원됩니다.

Amazon EMR on EKS에서 Flink를 사용할 경우 태스크 로컬 복구를 위해 Amazon EBS 볼륨을 TaskManager 포드에 자동 프로비저닝할 수 있습니다. 기본 오버레이 마운트에는 10GB 볼륨이 함께 제공되어 상태가 낮은 작업에 충분합니다. 상태가 큰 작업에서는 자동 EBS 볼륨 마운트 옵션을 활성화할 수 있습니다. TaskManager 포드는 포드 생성 과정에서 자동으로 생성되어 마운트되며 포드 삭제 중에는 제거됩니다.

다음 단계를 따라 Amazon EMR on EKS에서 Flink용 자동 EBS 볼륨 마운트를 활성화하세요.

1. 이후 단계에서 사용할 다음 변수의 값을 내보내세요.

```
export AWS_REGION=aa-example-1
export FLINK_EKS_CLUSTER_NAME=my-cluster
export AWS_ACCOUNT_ID=111122223333
```

2. 클러스터에 대해 kubeconfig YAML 파일을 생성 또는 업데이트합니다.

```
aws eks update-kubeconfig --name $FLINK_EKS_CLUSTER_NAME --region $AWS_REGION
```

3. Amazon EKS 클러스터에서 Amazon EBS CSI(컨테이너 스토리지 인터페이스) 드라이버에 대한 IAM 서비스 계정을 생성합니다.

```
eksctl create iamserviceaccount \
  --name ebs-csi-controller-sa \
  --namespace kube-system \
  --region $AWS_REGION \
  --cluster $FLINK_EKS_CLUSTER_NAME \
  --role-name TLR_${AWS_REGION}_${FLINK_EKS_CLUSTER_NAME} \
  --role-only \
  --attach-policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy \
  --approve
```

4. 다음 명령에 따라 Amazon EBS CSI 드라이버를 생성합니다.



```
eksctl create addon \
  --name aws-ebs-csi-driver \
  --region $AWS_REGION \
  --cluster $FLINK_EKS_CLUSTER_NAME \
  --service-account-role-arn arn:aws:iam::${AWS_ACCOUNT_ID}:role/TLR_
${AWS_REGION}_${FLINK_EKS_CLUSTER_NAME}
```

5. 다음 명령에 따라 Amazon EBS 스토리지 클래스를 생성합니다.

```
cat # EOF # storage-class.yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: ebs-sc
provisioner: ebs.csi.aws.com
volumeBindingMode: WaitForFirstConsumer
EOF
```

그런 후 다음과 같이 클래스를 적용합니다.

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

6. Helm에서는 서비스 계정을 생성할 수 있는 옵션과 함께 Amazon EMR Flink Kubernetes 연산자를 설치합니다. 그러면 Flink 배포에 사용할 수 있는 `emr-containers-sa-flink`가 생성됩니다.

```
helm install flink-kubernetes-operator flink-kubernetes-operator/ \
  --set jobServiceAccount.create=true \
  --set rbac.jobRole.create=true \
  --set rbac.jobRoleBinding.create=true
```

7. Flink 작업을 제출하고 태스크-로컬 복구를 위한 EBS 볼륨의 자동 프로비저닝을 활성화하려면 `flink-conf.yaml` 파일에 다음과 같은 구성을 설정하세요. 작업의 상태 크기에 맞게 크기 제한을 조정하세요. `serviceAccount`를 `emr-containers-sa-flink`으로 설정합니다. 체크포인트 간격의 값을 밀리초 단위로 지정하세요. `executionRoleArn`은 생략하세요.

```
flinkConfiguration:
  task.local-recovery.ebs.enable: true
  kubernetes.taskmanager.local-recovery.persistentVolumeClaim.sizeLimit: 10Gi
  state.checkpoints.dir: s3://BUCKET-PATH/checkpoint
  state.backend.local-recovery: true
```

```
state.backend: hasmap or rocksdb
state.backend.incremental: "true"
execution.checkpointing.interval: 15000
serviceAccount: emr-containers-sa-flink
```

Amazon EBS CSI 드라이버 플러그인을 삭제할 준비가 끝나면 다음과 같은 명령을 사용하세요.

```
# Detach Attached Policy
aws iam detach-role-policy --role-name TLR_${AWS_REGION}_${FLINK_EKS_CLUSTER_NAME}
--policy-arn arn:aws:iam::aws:policy/service-role/AmazonEBSCSIDriverPolicy
# Delete the created Role
aws iam delete-role --role-name TLR_${AWS_REGION}_${FLINK_EKS_CLUSTER_NAME}
# Delete the created service account
eksctl delete iamserviceaccount --name ebs-csi-controller-sa --namespace kube-system
--cluster $FLINK_EKS_CLUSTER_NAME --region $AWS_REGION
# Delete Addon
eksctl delete addon --name aws-ebs-csi-driver --cluster $FLINK_EKS_CLUSTER_NAME --
region $AWS_REGION
# Delete the EBS storage class
kubectl delete -f storage-class.yaml
```

## 일반 로그 기반 증분 체크포인트

### Note

Amazon EMR on EKS 6.14.0 이상에서 Flink를 사용하면 일반 로그 기반 증분 체크포인트가 지원됩니다.

체크포인트의 속도를 높이기 위해 일반 로그 기반 증분 체크포인트가 Flink 1.16에 추가되었습니다. 체크포인트 간격을 빠르게 하면 복구 후 다시 처리해야 하는 이벤트가 줄어들기 때문에 복구 작업이 적어지는 경우가 많습니다. 자세한 내용은 Apache Flink 블로그에서 [일반 로그 기반 증분 체크포인트로 체크포인트의 속도 및 안정성 강화](#)를 참조하세요.

샘플 작업을 이용하여 수행한 벤치마크 테스트에서 일반 로그 기반 증분 체크포인트를 사용하면 체크포인트 시간이 몇 분에서 몇 초로 단축된 것이 확인되었습니다.

일반 로그 기반 증분 체크포인트를 활성화하려면 `flink-conf.yaml` 파일에 다음 구성을 설정하세요. 체크포인트 간격의 값을 밀리초 단위로 지정하세요.

```
state.backend.changelog.enabled: true
state.backend.changelog.storage: filesystem
dstl.dfs.base-path: s3://bucket-path/changelog
state.backend.local-recovery: true
state.backend: rocksdb
state.checkpoints.dir: s3://bucket-path/checkpoint
execution.checkpointing.interval: 15000
```

## 세분화된 복구

### Note

Amazon EMR on EKS 6.14.0 이상에서 Flink를 사용할 경우 기본 스케줄러에 대해 세분화된 복구 지원이 제공됩니다. 적응형 스케줄러에서의 세분화된 복구 지원은 Amazon EMR on EKS 6.15.0 이상에서 Flink를 통해 이용할 수 있습니다.

실행 중에 태스크가 실패하면 Flink에서는 전체 실행 그래프를 재설정하고 마지막으로 완료된 체크포인트에서 전체 재실행을 트리거합니다. 이 방식은 실패한 태스크를 재실행하는 것보다 비용이 많이 듭니다. 세분화된 복구에서는 실패한 태스크의 파이프라인으로 연결된 구성 요소만 재시작합니다. 다음 예제의 작업 그래프에는 버텍스가 5개(A~E) 있습니다. 버텍스 사이에 있는 모든 연결은 포인트별 분포로 파이프라인되며 작업의 `parallelism.default`는 2로 설정됩니다.

```
A # B # C # D # E
```

이 예시에서는 태스크가 총 10개 실행 중입니다. 첫 번째 파이프라인(a1~e1)은 TaskManager(TM1)에서 실행되고 두 번째 파이프라인(a2~e2)은 또 다른 TaskManager(TM2)에서 실행됩니다.

```
a1 # b1 # c1 # d1 # e1
a2 # b2 # c2 # d2 # e2
```

a1 # e1 및 a2 # e2라는 두 개의 구성 요소가 파이프라인으로 연결되어 있습니다. TM1 또는 TM2 중에 하나가 실패하면 실패는 TaskManager가 중이던 파이프라인에 있는 태스크 5개에만 영향을 미칩니다. 재시작 전략에 따라 영향을 받는 파이프라인 구성 요소만 시작됩니다.

세분화된 복구는 완벽히 병렬화된 Flink 작업에서만 작동합니다. `keyBy()` 또는 `redistribute()` 작업에서는 지원되지 않습니다. 자세한 내용은 Flink 개선 제안 Jira 프로젝트의 [FLIP-1: 태스크 실패에서 세분화된 복구](#)를 참조하세요.

세분화된 복구를 활성화하려면 `flink-conf.yaml` 파일에 다음과 같은 구성을 설정하세요.

```
jobmanager.execution.failover-strategy: region
restart-strategy: exponential-delay or fixed-delay
```

## 적응형 스케줄러의 결합된 재시작 메커니즘

### Note

적응형 스케줄러의 결합된 재시작 메커니즘은 Amazon EMR on EKS 6.15.0 이상의 Flink에서 지원됩니다.

적응형 스케줄러에서는 가용 슬롯을 기반으로 작업 병렬성을 조정할 수 있습니다. 이 스케줄러는 구성된 작업 병렬 처리에 적합한 가용 슬롯이 충분하지 않은 경우 병렬 처리의 수를 자동으로 줄입니다. 새 슬롯이 가용 상태가 되면 작업은 구성된 작업 병렬 처리로 다시 확장됩니다. 적응형 스케줄러는 가용 리소스가 충분하지 않은 경우 작업에서 가동 중지가 발생하는 것을 방지합니다. Flink Autoscaler에 대해 지원되는 스케줄러입니다. 이러한 이유들로 인해 적응형 스케줄러를 Amazon EMR Flink와 함께 사용하는 것이 좋습니다. 단, 적응형 스케줄러는 짧은 시간 내에 여러 번 재시작을 수행할 수 있으며, 새 리소스가 추가될 때마다 한 번씩 다시 시작됩니다. 이로 인해 작업 성능이 떨어질 수 있습니다.

Amazon EMR 6.15.0 이상에서는 Flink에 첫 번째 리소스가 추가될 때 재시작 기간을 연 다음, 구성된 기본 1분 간격까지 기다리는 적응형 스케줄러의 결합된 재시작 메커니즘이 있습니다. 이 메커니즘에서는 구성된 병렬 처리로 작업을 실행하기 위한 가용 리소스가 충분하거나 간격 제한 시간이 초과될 경우 단일 재시작을 수행합니다.

샘플 작업을 이용한 벤치마크 테스트에서는 적응형 스케줄러와 Flink Autoscaler를 사용할 경우 이 기능이 기본 동작보다 10% 더 많은 레코드를 처리하는 것이 입증되었습니다.

결합된 재시작 메커니즘을 활성화하려면 `flink-conf.yaml` 파일에 다음 구성을 설정하세요.

```
jobmanager.adaptive-scheduler.combined-restart.enabled: true
jobmanager.adaptive-scheduler.combined-restart.window-interval: 1m
```

## Amazon EMR on EKS에서 Flink를 사용한 스팟 인스턴스의 정상적인 서비스 해제

Amazon EMR on EKS가 포함된 Flink를 사용하면 작업 복구 및 작업 규모 조정 과정에서 작업 재시작 시간을 개선할 수 있습니다.

## 개요

Amazon EMR on EKS 릴리스 6.15.0 이상에서는 Apache Flink를 사용하는 Amazon EMR on EKS의 스팟 인스턴스에 대한 태스크 관리자의 정상적인 서비스 해제를 지원합니다. 이 기능의 일부로 Flink가 포함된 Amazon EMR on EKS에서는 다음과 같은 기능을 제공합니다.

- 적시 체크포인트 – Flink 스트리밍 작업은 스팟 인스턴스 중단에 대응하고, 실행 중인 작업에 대한 JIT(적시) 체크포인트를 수행하고, 이러한 스팟 인스턴스에 추가 태스크를 예약하는 것을 방지할 수 있습니다. JIT 체크포인트는 기본 및 적응형 스케줄러에서 지원됩니다.
- 결합된 재시작 메커니즘 – 결합된 재시작 메커니즘은 작업이 대상 리소스 병렬 처리에 도달하거나 현재 구성된 기간의 종료 시점에 도달한 후 작업을 재시작하기 위해 최대한 노력합니다. 또한 복수의 스팟 인스턴스 종료로 인해 발생할 수 있는 연속된 작업 재시작을 방지할 수도 있습니다. 결합된 재시작 메커니즘은 적응형 스케줄러에서만 이용 가능합니다.

이러한 기능에는 다음과 같은 이점이 있습니다.

- 스팟 인스턴스를 활용하여 태스크 관리자를 실행하고 클러스터 지출을 줄일 수 있습니다.
- 스팟 인스턴스 태스크 관리자의 활성 상태가 개선되면 복원력이 높아지고 작업 예약의 효율성도 증가합니다.
- 스팟 인스턴스 종료 후 재시작 횟수가 감소하기 때문에 Flink 작업의 가동 시간이 증가합니다.

## 정상적인 서비스 해제 작동 방식

Apache Flink가 실행되는 Amazon EMR on EKS 클러스터에서 Amazon EMR을 프로비저닝하고 작업 관리자에는 온디맨드 노드를, 태스크 관리자에는 스팟 인스턴스 노드를 각각 지정하는 사례를 생각해 보세요. 태스크 관리자는 종료 2분 전에 중단 알림을 받습니다.

이 시나리오에서 작업 관리자는 스팟 인스턴스 중단 신호를 처리하고, 스팟 인스턴스에서 추가 작업 예약을 차단하고, 스트리밍 작업에 대한 JIT 체크포인트를 시작합니다.

그러면 작업 관리자는 현재 재시작 간격 기간에서 현재 작업 병렬 처리를 이행할 수 있을 만큼 가용 신규 리소스가 충분히 존재하는 경우에만 작업 그래프를 다시 시작합니다. 재시작 기간 간격은 스팟 인스턴스 교체 기간, 새 태스크 관리자 포드 생성 및 작업 관리자 등록을 근거로 정해집니다.

## 사전 조건

정상적인 서비스 해제를 사용하려면 Apache Flink가 실행되는 Amazon EMR on EKS 클러스터에서 스트리밍 작업을 생성하고 실행합니다. 다음 예시와 같이 하나 이상의 스팟 인스턴스에 대해 Adaptive

Scheduler 및 태스크 관리자가 예약되도록 활성화하세요. 작업 관리자에는 온디맨드 노드를 사용해야 하며, 스팟 인스턴스도 하나 이상 존재하는 경우에는 태스크 관리자에도 온디맨드 노드를 사용할 수 있습니다.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: deployment_name
spec:
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    cluster.taskmanager.graceful-decommission.enabled: "true"
    execution.checkpointing.interval: "240s"
    jobmanager.adaptive-scheduler.combined-restart.enabled: "true"
    jobmanager.adaptive-scheduler.combined-restart.window-interval : "1m"
  serviceAccount: flink
  jobManager:
    resource:
      memory: "2048m"
      cpu: 1
    nodeSelector:
      'eks.amazonaws.com/capacityType': 'ON_DEMAND'
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
    nodeSelector:
      'eks.amazonaws.com/capacityType': 'SPOT'
  job:
    jarURI: flink_job_jar_path
```

## 구성

이 섹션에서는 서비스 해제 요구 사항을 위해 지정할 수 있는 구성들을 대부분 설명합니다.

| Key(키)                       | 설명                            | 기본값  | 허용되는 값      |
|------------------------------|-------------------------------|------|-------------|
| cluster.taskmanager.graceful | 태스크 관리자의 정상적인 서비스 해제를 활성화합니다. | true | true, false |

| Key(키)                                                                                 | 설명                                                            | 기본값   | 허용되는 값             |
|----------------------------------------------------------------------------------------|---------------------------------------------------------------|-------|--------------------|
| l-decommi<br>ssion.ena<br>bled                                                         |                                                               |       |                    |
| jobmanage<br>r.adaptiv<br>e-schedul<br>er.combin<br>ed-restar<br>t.enabled             | 적응형 스케줄러에서 결합된 재시작 메커니즘을 활성화합니다.                              | false | true, false        |
| jobmanage<br>r.adaptiv<br>e-schedul<br>er.combin<br>ed-restar<br>t.window-<br>interval | 작업에 대해 병합된 재시작을 수행하는 결합된 재시작 기간 간격입니다. 단위가 없는 정수는 밀리초로 해석합니다. | 1m    | 예: 30, 60s, 3m, 1h |

## Flink 애플리케이션에 대한 Autoscaler 사용

운영자의 Autoscaler를 사용하면 Flink 작업에서 지표를 수집하고 작업 버텍스 수준에서 병렬 처리를 자동으로 조정하여 역압을 완화할 수 있습니다. 다음은 구성에 대한 예제입니다.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  ...
spec:
  ...
  flinkVersion: v1_18
  flinkConfiguration:
    job.autoscaler.enabled: "true"
    job.autoscaler.stabilization.interval: 1m
    job.autoscaler.metrics.window: 5m
    job.autoscaler.target.utilization: "0.6"
    job.autoscaler.target.utilization.boundary: "0.2"
```

```
job.autoscaler.restart.time: 2m
job.autoscaler.catch-up.duration: 5m
pipeline.max-parallelism: "720"
...
```

이 구성은 Amazon EMR의 최신 릴리스에 대한 기본값을 사용합니다. 다른 버전을 사용하는 경우 값이 다를 수 있습니다.

#### Note

Amazon EMR 7.2.0부터는 구성에 `kubernetes.operator` 접두사를 포함하지 않아도 됩니다. 7.1.0 이하를 사용하는 경우 각 구성 전에 접두사를 사용해야 합니다. 예를 들어, `kubernetes.operator.job.autoscaler.scaling.enabled`를 지정해야 합니다.

다음은 Autoscaler의 구성 옵션입니다.

- `job.autoscaler.scaling.enabled` - 오토스케일러에서 버텍스 조정 실행을 활성화할지 여부를 지정합니다. 기본값은 `true`입니다. 이 구성을 비활성화하면 오토스케일러는 지표만 수집하고 각 버텍스에 대해 제안된 병렬 처리를 평가하지만 작업을 업그레이드하지는 않습니다.
- `job.autoscaler.stabilization.interval` - 새로운 조정이 실행되지 않는 안정화 기간. 기본값은 5분입니다.
- `job.autoscaler.metrics.window` - 조정 지표 집계 기간. 기간이 길수록 더 원활하고 안정적이지만 갑작스러운 로드 변경에 대응하는 경우 Autoscaler 속도가 느려질 수 있습니다. 기본값은 15분입니다. 3~60분의 값을 사용하여 실험해 보는 것이 좋습니다.
- `job.autoscaler.target.utilization` - 안정적인 작업 성능을 제공하고 로드 변동을 위한 약간의 버퍼를 제공하기 위한 목표 버텍스 사용률. 기본값은 0.7로, 작업 버텍스의 사용률/로드를 70%로 설정합니다.
- `job.autoscaler.target.utilization.boundary` - 로드 변동에서 즉각적인 조정을 피하기 위해 추가 버퍼 역할을 하는 목표 버텍스 사용률 경계. 기본값은 0.3이며, 조정 작업을 트리거하기 전에 목표 사용률과 30%의 편차가 허용됨을 의미합니다.
- `ob.autoscaler.restart.time` - 애플리케이션을 다시 시작하는 데 걸리는 예상 시간. 기본값은 5분입니다.
- `job.autoscaler.catch-up.duration` - 예상 캐치업 시간으로, 조정 작업이 완료된 후 모든 백로그를 완전히 처리합니다. 기본값은 5분입니다. 캐치업 기간을 줄임으로써 Autoscaler는 조정 작업을 위한 추가 용량을 예약해야 합니다.



- `pipeline.max-parallelism` - Autoscaler에서 사용할 수 있는 최대 병렬 처리. 이 한도가 Flink 구성 또는 각 운영자에 구성된 최대 병렬 처리보다 더 높은 경우 Autoscaler는 이 한도를 무시합니다. 기본값은 -1입니다. Autoscaler는 최대 병렬 처리의 나눗수로 병렬 처리를 계산하므로, Flink에서 제공하는 기본값을 사용하는 대신, 나눗수가 많은 최대 병렬 처리를 선택하는 것이 좋습니다. 이 구성에서는 120, 180, 240, 360, 720 등과 같이 60의 배수를 사용하는 것이 좋습니다.

구성 참조 페이지에 대한 자세한 내용은 [Autoscaler configuration](#)을 참조하세요.

## 오토스케일러 파라미터 자동 조정

이 섹션에서는 다양한 Amazon EMR 버전에 대한 자동 조정 동작을 설명합니다. 또한 다양한 오토 스케일링 구성도 자세히 설명합니다.

### Note

Amazon EMR 7.2.0 이상은 오픈 소스 구성 `job.autoscaler.restart.time-tracking.enabled`를 사용하여 재조정 시간 예측을 활성화합니다. 재조정 시간 예측은 Amazon EMR 자동 조정과 동일한 기능을 제공하므로, 재시작 시간에 경험적 값을 수동으로 할당하지 않아도 됩니다.

Amazon EMR 7.1.0 이하를 사용하는 경우에도 Amazon EMR 자동 조정을 계속 사용할 수 있습니다.

### 7.2.0 and higher

Amazon EMR 7.2.0 이상은 오토 스케일링 결정을 적용하는 데 필요한 실제 재시작 시간을 측정합니다. 릴리스 7.1.0 이하에서는 `job.autoscaler.restart.time` 구성을 사용하여 예상 최대 재시작 시간을 수동으로 구성해야 했습니다. `job.autoscaler.restart.time-tracking.enabled` 구성을 사용하면 첫 번째 조정에 대한 재시작 시간만 입력하면 됩니다. 이후 작업자는 실제 재시작 시간을 기록하고 후속 조정에 사용합니다.

이 추적을 활성화하려면 다음 명령을 사용합니다.

```
job.autoscaler.restart.time-tracking.enabled: true
```

다음은 재조정 시간 예측을 위한 관련 구성입니다.

| 구성                                           | 필수  | 기본값   | 설명                                                                                                                  |
|----------------------------------------------|-----|-------|---------------------------------------------------------------------------------------------------------------------|
| job.autoscaler.restart.time-tracking.enabled | 아니요 | False | Flink 오토스케일러가 시간 경과에 따라 구성을 자동으로 조정하여 조정 결정을 최적화해야 하는지 여부를 나타냅니다. 오토스케일러는 오토스케일러 파라미터 restart.time 만 자동 조정할 수 있습니다. |
| job.autoscaler.restart.time                  | 아니요 | 5분    | 연산자가 이전 조정에서 실제 재시작 시간을 확인할 수 있을 때까지 Amazon EMR on EKS 에서 사용하는 예상 재시작 시간.                                           |
| job.autoscaler.restart.time-tracking.limit   | 아니요 | 15분   | job.autoscaler.restart.time-tracking.enabled 가 true로 설정된 경우 관찰된 최대 재시작 시간.                                          |

다음은 재조정 시간 예측을 시도하는 데 사용할 수 있는 배포 사양의 예입니다.

```

apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: autoscaling-example
spec:
  flinkVersion: v1_18
  flinkConfiguration:

    # Autoscaler parameters
    job.autoscaler.enabled: "true"
    job.autoscaler.scaling.enabled: "true"
    job.autoscaler.stabilization.interval: "5s"
    job.autoscaler.metrics.window: "1m"

```

```

job.autoscaler.restart.time-tracking.enabled: "true"
job.autoscaler.restart.time: "2m"
job.autoscaler.restart.time-tracking.limit: "10m"

jobmanager.scheduler: adaptive
taskmanager.numberOfTaskSlots: "1"
pipeline.max-parallelism: "12"

executionRoleArn: <JOB_ARN>
emrReleaseLabel: emr-7.13.0-flink-latest
jobManager:
  highAvailabilityEnabled: false
  storageDir: s3://<s3_bucket>/flink/autoscaling/ha/
  replicas: 1
  resource:
    memory: "1024m"
    cpu: 0.5
taskManager:
  resource:
    memory: "1024m"
    cpu: 0.5
job:
  jarURI: s3://<s3_bucket>/some-job-with-back-pressure
  parallelism: 1
  upgradeMode: stateless

```

역압을 시뮬레이션하려면 다음 배포 사양을 사용합니다.

```

job:
  jarURI: s3://<s3_bucket>/pyflink-script.py
  entryClass: "org.apache.flink.client.python.PythonDriver"
  args: ["-py", "/opt/flink/usrlib/pyflink-script.py"]
  parallelism: 1
  upgradeMode: stateless

```

다음 Python 스크립트를 S3 버킷에 업로드합니다.

```

import logging
import sys
import time
import random

```

```

from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import StreamTableEnvironment

TABLE_NAME="orders"
QUERY=f"""
CREATE TABLE {TABLE_NAME} (
    id INT,
    order_time AS CURRENT_TIMESTAMP,
    WATERMARK FOR order_time AS order_time - INTERVAL '5' SECONDS
)
WITH (
    'connector' = 'datagen',
    'rows-per-second'='10',
    'fields.id.kind'='random',
    'fields.id.min'='1',
    'fields.id.max'='100'
);
"""

def create_backpressure(i):
    time.sleep(2)
    return i

def autoscaling_demo():
    env = StreamExecutionEnvironment.get_execution_environment()
    t_env = StreamTableEnvironment.create(env)
    t_env.execute_sql(QUERY)
    res_table = t_env.from_path(TABLE_NAME)

    stream = t_env.to_data_stream(res_table) \
        .shuffle().map(lambda x: create_backpressure(x)) \
        .print()
    env.execute("Autoscaling demo")

if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format="%(message)s")
    autoscaling_demo()

```

재조정 시간 예측이 작동하는지 확인하려면 Flink 연산자의 DEBUG 수준 로깅이 활성화되어 있는지 확인합니다. 아래 예제에서는 헬름 차트 파일(values.yaml)을 업데이트하는 방법을 보여줍니다. 그런 다음, 업데이트된 헬름 차트를 다시 설치하고 Flink 작업을 다시 실행합니다.

```
log4j-operator.properties: |+
```

```
# Flink Operator Logging Overrides
rootLogger.level = DEBUG
```

리더 포드의 이름을 가져옵니다.

```
ip=$(kubectl get configmap -n $NAMESPACE <job-name>-cluster-config-map -o json | jq
-r ".data[\"org.apache.flink.k8s.leader.restserver\"]" | awk -F: '{print $2}' | awk
-F '/' '{print $3}')

kubectl get pods -n $NAMESPACE -o json | jq -r ".items[] | select(.status.podIP ==
\"$ip\") | .metadata.name"
```

다음 명령을 실행하여 지표 평가에 사용된 실제 재시작 시간을 가져옵니다.

```
kubectl logs <FLINK-OPERATOR-POD-NAME> -c flink-kubernetes-operator -n <OPERATOR-
NAMESPACE> -f | grep "Restart time used in scaling summary computation"
```

다음과 비슷한 로그가 표시됩니다. 첫 번째 조정에서만 `job.autoscaler.restart.time`을 사용합니다. 후속 조정에서는 관찰된 재시작 시간을 사용합니다.

```
2024-05-16 17:17:32,590 o.a.f.a.ScalingExecutor [DEBUG][default/autoscaler-
example] Restart time used in scaling summary computation: PT2M
2024-05-16 17:19:03,787 o.a.f.a.ScalingExecutor [DEBUG][default/autoscaler-
example] Restart time used in scaling summary computation: PT14S
2024-05-16 17:19:18,976 o.a.f.a.ScalingExecutor [DEBUG][default/autoscaler-
example] Restart time used in scaling summary computation: PT14S
2024-05-16 17:20:50,283 o.a.f.a.ScalingExecutor [DEBUG][default/autoscaler-
example] Restart time used in scaling summary computation: PT14S
2024-05-16 17:22:21,691 o.a.f.a.ScalingExecutor [DEBUG][default/autoscaler-
example] Restart time used in scaling summary computation: PT14S
```

## 7.0.0 and 7.1.0

오픈 소스 기본 제공 Flink 오토스케일러는 다양한 지표를 사용하여 최적의 조정 결정을 내립니다. 그러나 계산에 사용하는 기본값은 대부분의 워크로드에 적용 가능한 값이지만, 주어진 작업에 적합하지 않을 수 있습니다. Flink 연산자의 Amazon EMR on EKS 버전에 추가된 자동 조정 기능은 캡처된 특정 지표에서 관찰된 과거 추세를 검토한 다음, 해당 작업에 맞게 조정된 가장 최적의 값을 계산하려고 시도합니다.

| 구성                                                                    | 필수  | 기본값   | 설명                                                                                                                            |
|-----------------------------------------------------------------------|-----|-------|-------------------------------------------------------------------------------------------------------------------------------|
| kubernetes.operator.job.autoscaler.autotune.enable                    | 아니요 | False | Flink 오토스케일러가 시간 경과에 따라 구성을 자동으로 조정하여 오토스케일러 조정 결정을 최적화해야 하는지 여부를 나타냅니다. 현재 오토스케일러는 오토스케일러 파라미터 restart.time 만 자동 조정할 수 있습니다. |
| kubernetes.operator.job.autoscaler.autotune.metrics.history.max.count | 아니요 | 3     | 오토스케일러가 Amazon EMR on EKS 지표 구성 맵에 보관하는 Amazon EMR on EKS의 기록 지표 수를 나타냅니다.                                                    |
| kubernetes.operator.job.autoscaler.autotune.metrics.restart.count     | 아니요 | 3     | 주어진 작업의 평균 재시작 시간 계산을 시작하기 전에 오토스케일러가 수행하는 재시작 횟수를 나타냅니다.                                                                     |

자동 조정을 활성화하려면 다음을 완료해야 합니다.

- kubernetes.operator.job.autoscaler.autotune.enable:를 true로 설정합니다.
- metrics.job.status.enable:를 TOTAL\_TIME로 설정합니다.
- 자동 조정을 활성화하기 위해 [Flink 애플리케이션에 대해 오토스케일러 사용](#) 설정을 따랐습니다.

다음은 자동 조정을 시도하는 데 사용할 수 있는 배포 사양에 대한 예제입니다.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: autoscaling-example
spec:
  flinkVersion: v1_18
  flinkConfiguration:
```

```

# Autotuning parameters
kubernetes.operator.job.autoscaler.autotune.enable: "true"
kubernetes.operator.job.autoscaler.autotune.metrics.history.max.count: "2"
kubernetes.operator.job.autoscaler.autotune.metrics.restart.count: "1"
metrics.job.status.enable: TOTAL_TIME

# Autoscaler parameters
kubernetes.operator.job.autoscaler.enabled: "true"
kubernetes.operator.job.autoscaler.scaling.enabled: "true"
kubernetes.operator.job.autoscaler.stabilization.interval: "5s"
kubernetes.operator.job.autoscaler.metrics.window: "1m"

jobmanager.scheduler: adaptive

taskmanager.numberOfTaskSlots: "1"
state.savepoints.dir: s3://<S3_bucket>/autoscaling/savepoint/
state.checkpoints.dir: s3://<S3_bucket>/flink/autoscaling/checkpoint/
pipeline.max-parallelism: "4"

executionRoleArn: <JOB ARN>
emrReleaseLabel: emr-6.14.0-flink-latest
jobManager:
  highAvailabilityEnabled: true
  storageDir: s3://<S3_bucket>/flink/autoscaling/ha/
  replicas: 1
  resource:
    memory: "1024m"
    cpu: 0.5
taskManager:
  resource:
    memory: "1024m"
    cpu: 0.5
job:
  jarURI: s3://<S3_bucket>/some-job-with-back-pressure
  parallelism: 1
  upgradeMode: last-state

```

역압을 시뮬레이션하려면 다음 배포 사양을 사용합니다.

```

job:
  jarURI: s3://<S3_bucket>/pyflink-script.py
  entryClass: "org.apache.flink.client.python.PythonDriver"

```

```
args: ["-py", "/opt/flink/usrlib/pyflink-script.py"]
parallelism: 1
upgradeMode: last-state
```

다음 Python 스크립트를 S3 버킷에 업로드합니다.

```
import logging
import sys
import time
import random

from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import StreamTableEnvironment

TABLE_NAME="orders"
QUERY=f"""
CREATE TABLE {TABLE_NAME} (
    id INT,
    order_time AS CURRENT_TIMESTAMP,
    WATERMARK FOR order_time AS order_time - INTERVAL '5' SECONDS
)
WITH (
    'connector' = 'datagen',
    'rows-per-second'='10',
    'fields.id.kind'='random',
    'fields.id.min'='1',
    'fields.id.max'='100'
);
"""

def create_backpressure(i):
    time.sleep(2)
    return i

def autoscaling_demo():
    env = StreamExecutionEnvironment.get_execution_environment()
    t_env = StreamTableEnvironment.create(env)
    t_env.execute_sql(QUERY)
    res_table = t_env.from_path(TABLE_NAME)

    stream = t_env.to_data_stream(res_table) \
        .shuffle().map(lambda x: create_backpressure(x)) \
        .print()
```



```
env.execute("Autoscaling demo")

if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format="%(message)s")
    autoscaling_demo()
```

자동 튜너가 작동하는지 확인하려면 다음 명령을 사용합니다. Flink 연산자에 대한 자체 리더 포트 정보를 사용해야 합니다.

먼저 리더 포트의 이름을 가져옵니다.

```
ip=$(kubectl get configmap -n $NAMESPACE <job-name>-cluster-config-map -o json | jq
-r ".data[\"org.apache.flink.k8s.leader.restserver\"]" | awk -F: '{print $2}' | awk
-F '/' '{print $3}')

kubectl get pods -n $NAMESPACE -o json | jq -r ".items[] | select(.status.podIP ==
\"$ip\") | .metadata.name"
```

리더 포트의 이름을 지정한 후 다음 명령을 실행할 수 있습니다.

```
kubectl logs -n $NAMESPACE -c flink-kubernetes-operator --follow <YOUR-FLINK-
OPERATOR-POD-NAME> | grep -E 'EmrEks|autotun|calculating|restart|autoscaler'
```

다음과 비슷한 로그가 표시됩니다.

```
[m[33m2023-09-13 20:10:35,941[m [36mc.a.c.f.k.o.a.EmrEksMetricsAutotuner[m
[36m[DEBUG][flink/autoscaling-example] Using the latest
Emr Eks Metric for calculating restart.time for autotuning:
EmrEksMetrics(restartMetric=RestartMetric(restartingTime=65, numRestarts=1))

[m[33m2023-09-13 20:10:35,941[m [36mc.a.c.f.k.o.a.EmrEksMetricsAutotuner[m
[32m[INFO ][flink/autoscaling-example] Calculated average restart.time metric via
autotuning to be: PT0.065S
```

## Amazon EMR on EKS에서 Flink 작업에 대한 유지 관리 및 문제 해결

다음 섹션에서는 장기 실행 중인 Flink 작업을 유지 관리하는 방법을 개략적으로 살펴보고 자주 발생하는 Flink 관련 문제 몇 가지를 해결하는 방법에 대한 지침을 제공합니다.

# Flink 애플리케이션 유지 관리

## 주제

- [업그레이드 모드](#)

Flink 애플리케이션은 일반적으로 몇 주, 몇 달 또는 몇 년 등 장기간 실행되도록 설계되었습니다. 모든 장기 실행 서비스와 마찬가지로 Flink 스트리밍 애플리케이션에도 유지 관리가 필요합니다. 유지 관리는 버그 수정, 개선 및 이후 버전의 Flink 클러스터로의 마이그레이션을 포함합니다.

FlinkDeployment 및 FlinkSessionJob 리소스에 대한 사양이 변경될 경우 실행 중인 애플리케이션을 업그레이드해야 합니다. 이 작업을 수행하기 위해 운영자는 (이미 일시 중단된 경우를 제외하고) 실행 중인 작업을 중지하고 해당 작업을 최신 사양을 이용해 다시 배포하고, 상태 저장 애플리케이션의 경우에는 이전 실행의 상태를 이용해 다시 배포합니다.

사용자는 상태 저장 애플리케이션이 중지되고 JobSpec의 upgradeMode 설정을 이용해 복원될 때 상태를 관리하는 방법을 제어합니다.

## 업그레이드 모드

### 선택적 도입

### 상태 비저장

빈 상태에서의 상태 비저장 애플리케이션 업그레이드입니다.

### 마지막 상태

모든 애플리케이션 상태(실패한 작업 포함)의 간편 업그레이드에서는 항상 가장 최근에 성공한 체크포인트를 사용하기 때문에 정상 작업이 필요하지 않습니다. HA 메타데이터가 손실된 경우에는 수동으로 복구해야 할 수 있습니다. 최신 체크포인트를 선택할 때 작업이 대체될 수 있는 시간을 제한하기 위해 `kubernetes.operator.job.upgrade.last-state.max.allowed.checkpoint.age`를 구성할 수 있습니다. 체크포인트가 구성된 값보다 오래된 경우 정상 작업에 대한 세이프포인트가 대신 사용됩니다. 세션 모드에서는 이 작업이 지원되지 않습니다.

### 세이프포인트

업그레이드를 위해 세이프포인트를 사용하여 최대한의 안전성과 백업/포크 포인트로 사용될 가능성을 제공합니다. 업그레이드 프로세스가 진행되는 동안 세이프포인트가 생성됩니다. 세이프포인트가 생성되도록 하려면 Flink 작업이 실행 중이어야 합니다. 작업이 비정상 상태인 경우 마지막 체

크포인트가 사용됩니다(kubernetes.operator.job.upgrade.last-state-fallback.enabled가 false로 설정된 경우는 제외). 마지막 체크포인트를 사용할 수 없다면 작업 업그레이드가 실패합니다.

## 문제 해결

이 섹션에서는 Amazon EMR on EKS의 문제를 해결하는 방법을 설명합니다. Amazon EMR과 관련된 일반적인 문제를 해결하는 방법에 대한 자세한 내용은 Amazon EMR 관리 안내서에서 [클러스터 문제 해결](#)을 참조하세요.

- [PersistentVolumeClaims\(PVC\)를 사용하는 작업 문제 해결](#)
- [Amazon EMR on EKS 수직 자동 조정 문제 해결](#)
- [Amazon EMR on EKS Spark 운영자 문제 해결](#)

## Amazon EMR on EKS에서 Apache Flink 문제 해결

차트 Helm 설치 시 리소스 매핑을 찾을 수 없음

차트 Helm 설치 시 다음과 같은 오류 메시지가 나타날 수 있다.

```
Error: INSTALLATION FAILED: pulling from host 1234567890.dkr.ecr.us-west-2.amazonaws.com failed with status code [manifests 6.13.0]: 403 Forbidden Error:
INSTALLATION FAILED: unable to build kubernetes objects from release manifest:
[resource mapping not found for name: "flink-operator-serving-cert" namespace: "<the namespace to install your operator>" from "": no matches for kind "Certificate" in
version "cert-manager.io/v1"

ensure CRDs are installed first, resource mapping not found for name: "flink-operator-selfsigned-issuer" namespace: "<the namespace to install your operator>" " from "": no
matches for kind "Issuer" in version "cert-manager.io/v1"

ensure CRDs are installed first].
```

이 오류를 해결하려면 웹훅 구성 요소 추가를 활성화하도록 cert-manager를 설치합니다. 사용하는 각 Amazon EKS 클러스터에 cert-manager를 설치해야 합니다.

```
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/download/v1.12.0
```

## AWS 서비스 액세스 거부 오류

access denied 오류가 표시되는 경우 차트 Helm values.yaml 파일에서 operatorExecutionRoleArn에 대한 IAM 역할에 올바른 권한이 있는지 확인합니다. 또한 FlinkDeployment 사양에서 executionRoleArn 아래 IAM 역할에 올바른 권한이 있는지 확인합니다.

### FlinkDeployment가 멈춤

FlinkDeployment가 중단된 상태로 멈추면 다음 단계에 따라 배포를 강제로 삭제합니다.

1. 배포 실행을 편집합니다.

```
kubectl edit -n Flink Namespace flinkdeployments/App Name
```

2. 이 파이널라이저를 제거합니다.

```
finalizers:
  - flinkdeployments.flink.apache.org/finalizer
```

3. 배포를 삭제합니다.

```
kubectl delete -n Flink Namespace flinkdeployments/App Name
```

오프라인에서 Flink 애플리케이션을 실행할 때 s3a AWSBadRequestException 문제 AWS 리전

[오프라인 AWS 리전](#)에서 Flink 애플리케이션을 실행하는 경우 다음 오류가 표시될 수 있습니다.

```
Caused by: org.apache.hadoop.fs.s3a.AWSBadRequestException: getFileStatus on
s3://flink.txt: com.amazonaws.services.s3.model.AmazonS3Exception: Bad Request
(Service: Amazon S3; Status Code: 400; Error Code: 400 Bad Request; Request ID:
ABCDEF GHIJKL; S3 Extended Request ID:
ABCDEF GHIJKLMNOP=; Proxy: null), S3 Extended Request ID: ABCDEF GHIJKLMNOP=:400 Bad
Request: Bad Request
(Service: Amazon S3; Status Code: 400; Error Code: 400 Bad Request; Request ID:
ABCDEF GHIJKL; S3 Extended Request ID: ABCDEF GHIJKLMNOP=; Proxy: null)
```

```
Caused by: org.apache.hadoop.fs.s3a.AWSBadRequestException: getS3Region on flink-
application: software.amazon.awssdk.services.s3.model.S3Exception: null
(Service: S3, Status Code: 400, Request ID: ABCDEF GHIJKLMNOP, Extended Request ID:
ABCDEF GHIJKLMNOPQRST==):null: null
```

```
(Service: S3, Status Code: 400, Request ID: ABCDEFGHIJKLMNOP, Extended Request ID:
AHl42uDNaTUF0us/5IIVNvSakBcMjMCH7dd37ky0vE6jhABCDEFGHIJKLMN0PQRST==)
```

이러한 오류를 해결하려면 FlinkDeployment 정의 파일에서 다음 구성을 사용합니다.

```
spec:
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    fs.s3a.endpoint.region: OPT_IN_AWS_REGION_NAME
```

또한 SDKv2 자격 증명 제공업체를 사용하는 것이 좋습니다.

```
fs.s3a.aws.credentials.provider:
  software.amazon.awssdk.auth.credentials.WebIdentityTokenFileCredentialsProvider
```

SDKv1 자격 증명 제공업체를 사용하려면 SDK가 옵트인 리전을 지원하는지 확인합니다. 자세한 내용은 [aws-sdk-java GitHub repository](#)를 참조하세요.

옵트인 리전에서 Flink SQL 문을 실행할 때 S3 AWSBadRequestException이 표시되면 Flink 구성 사양에서 fs.s3a.endpoint.region: *OPT\_IN\_AWS\_REGION\_NAME* 구성을 설정해야 합니다.

CN 리전에서 Flink 세션 작업을 실행하는 경우 S3A AWSBadRequestException

Amazon EMR 릴리스 6.15.0~7.2.0의 경우 CN 리전에서 Flink 세션 작업을 실행할 때 다음과 같은 오류 메시지가 표시될 수 있습니다. 여기에는 중국(베이징) 및 중국(닝샤)이 포함됩니다.

```
Error:
{"type":"org.apache.flink.kubernetes.operator.exception.ReconciliationException","message":"or
      getFileStatus on s3://ABCDPath:
software.amazon.awssdk.services.s3.model.S3Exception: null (Service: S3, Status Code:
400, Request ID: ABCDEFGH, Extended Request ID:
      ABCDEFGH:null: null (Service: S3, Status Code: 400, Request ID:
ABCDEFGH, Extended Request ID: ABCDEFGH","additionalMetadata":{"},"throwableList":

[{"type":"org.apache.hadoop.fs.s3a.AWSBadRequestException","message":"getFileStatus on
s3://ABCDPath: software.amazon.awssdk.services.s3.model.S3Exception:
      null (Service: S3, Status Code: 400, Request ID: ABCDEFGH, Extended
Request ID: ABCDEFGH:null: null (Service: S3, Status Code: 400, Request ID: ABCDEFGH,
      Extended Request ID: ABCDEFGH","additionalMetadata":{"}},
{"type":"software.amazon.awssdk.services.s3.model.S3Exception","message":"null
(Service: S3, Status Code: 400,
```

```
Request ID: ABCDEFGH, Extended Request ID:
ABCDEFGH", "additionalMetadata": {}}}]}
```

이 문제를 인지하고 있습니다. 팀은 이러한 모든 릴리스 버전에 대해 flink 연산자를 패치하는 작업을 하고 있습니다. 그러나 패치를 완료하기 전에 이 오류를 해결하려면 Flink 연산자 헬름 차트를 다운로드하고, 압축 해제하며(압축된 파일 추출), 헬름 차트에서 구성을 변경해야 합니다.

특정 단계는 다음과 같습니다.

1. 특히 디렉터리를 헬름 차트의 로컬 폴더로 변경하고, 다음 명령줄을 실행하여 헬름 차트를 가져와 압축 해제(추출)합니다.

```
helm pull oci://public.ecr.aws/emr-on-eks/flink-kubernetes-operator \
--version $VERSION \
--namespace $NAMESPACE
```

```
tar -zxvf flink-kubernetes-operator-$VERSION.tgz
```

2. 헬름 차트 폴더로 이동하여 templates/flink-operator.yaml 파일을 찾습니다.
3. flink-operator-config ConfigMap을 찾아 flink-conf.yaml의 다음 fs.s3a.endpoint.region 구성을 추가합니다. 예제:

```
{{- if .Values.defaultConfiguration.create }}
apiVersion: v1
kind: ConfigMap
metadata:
  name: flink-operator-config
  namespace: {{ .Release.Namespace }}
  labels:
    {{- include "flink-operator.labels" . | nindent 4 }}
data:
  flink-conf.yaml: |+
fs.s3a.endpoint.region: {{ .Values.emrContainers.awsRegion }}
```

4. 로컬 헬름 차트를 설치하고 작업을 실행합니다.

# Apache Flink가 포함된 Amazon EMR on EKS에 대해 지원되는 릴리스

Apache Flink는 다음 Amazon EMR on EKS 릴리스에서 사용할 수 있습니다. 사용 가능한 모든 릴리스에 대한 자세한 내용은 [Amazon EMR on EKS 릴리스](#) 섹션을 참조하세요.

| 릴리스 레이블                              | Java | Flink  | Flink 운영자 |
|--------------------------------------|------|--------|-----------|
| emr-7.2.0-flink-latest               | 17   | 1.18.1 | -         |
| emr-7.2.0-flink-k8s-operator-latest  | 11   | -      | 1.8.0     |
| emr-7.1.0-flink-latest               | 17   | 1.18.1 | -         |
| emr-7.1.0-flink-k8s-operator-latest  | 11   | -      | 1.6.1     |
| emr-7.0.0-flink-latest               | 11   | 1.18.0 | -         |
| emr-7.0.0-flink-k8s-operator-latest  | 11   | -      | 1.6.1     |
| emr-6.15.0-flink-latest              | 11   | 1.17.1 | -         |
| emr-6.15.0-flink-k8s-operator-latest | 11   | -      | 1.6.0     |
| emr-6.14.0-flink-latest              | 11   | 1.17.1 | -         |
| emr-6.14.0-flink-k8s-operator-latest | 11   | -      | 1.6.0     |
| emr-6.13.0-flink-latest              | 11   | 1.17.0 | -         |
| emr-6.13.0-flink-k8s-operator-latest | 11   | -      | 1.5.0     |

# Amazon EMR on EKS를 사용하여 Spark 작업 실행

작업 실행은 Spark jar, PySpark 스크립트 또는 SparkSQL 쿼리와 같이 Amazon EMR on EKS에 제출하는 작업 단위입니다. 이 주제에서는 이를 사용하여 작업 실행을 관리하고 AWS CLI, Amazon EMR 콘솔을 사용하여 작업 실행을 보고, 일반적인 작업 실행 오류를 해결하는 방법에 대한 개요를 제공합니다.

Amazon EMR on EKS에서는 IPv6 Spark 작업을 실행할 수 없습니다.

## Note

Amazon EMR on EKS를 사용하여 작업 실행을 제출하기 전에 [Amazon EMR on EKS 설정](#)의 단계를 완료해야 합니다.

## 주제

- [StartJobRun을 사용하여 Spark 작업 실행](#)
- [Spark 운영자에서 Spark 작업 실행](#)
- [spark-submit을 사용하여 Spark 작업 실행](#)
- [Amazon EMR on EKS에서 Apache Livy 사용](#)
- [Amazon EMR on EKS 작업 실행 관리](#)
- [작업 템플릿 사용](#)
- [포드 템플릿 사용](#)
- [작업 재시도 정책 사용](#)
- [Spark 이벤트 로그 로테이션 사용](#)
- [Spark 컨테이너 로그 로테이션 사용](#)
- [Amazon EMR Spark 작업에 수직 자동 조정 사용](#)

## StartJobRun을 사용하여 Spark 작업 실행

이 섹션은 Spark 작업을 실행할 수 있도록 환경을 준비하기 위한 자세한 설정 단계를 포함하며, 지정된 파라미터로 작업 실행을 제출하기 위한 단계별 지침을 제공합니다.

## 주제

- [Amazon EMR on EKS 설정](#)



- [StartJobRun을 사용하여 작업 실행 제출](#)
- [작업 제출자 분류 사용](#)
- [Amazon EMR 컨테이너 기본 분류 사용](#)

## Amazon EMR on EKS 설정

Amazon EMR on EKS에 대해 설정하려면 다음 작업을 수행합니다. Amazon Web Services(AWS)에 이미 가입했고 Amazon EKS를 사용하고 있는 경우 Amazon EMR on EKS를 사용할 준비를 거의 마친 상태입니다. 이미 완료한 작업은 건너뛰니다.

### Note

또한 [Amazon EMR on EKS Workshop](#)에 따라 Amazon EMR on EKS에서 Spark 작업을 실행하는 데 필요한 모든 리소스를 설정할 수 있습니다. 또한 워크숍에서는 CloudFormation 템플릿을 사용하여 시작하는 데 필요한 리소스를 생성하는 자동화 기능을 제공합니다. 다른 템플릿과 모범 사례는 GitHub의 [EMR Containers Best Practices Guide](#)를 참조하세요.

1. [의 최신 버전 설치 또는 업데이트 AWS CLI](#)
2. [kubectl 및 eksctl 설정](#)
3. [Amazon EKS 시작하기 – eksctl](#)
4. [Amazon EMR on EKS에 대한 클러스터 액세스 활성화](#)
5. [EKS 클러스터에 IAM 역할 활성화](#)
6. [Amazon EMR on EKS에 대한 액세스 권한 부여](#)
7. [Amazon EMR에 Amazon EKS 클러스터 등록](#)

## Amazon EMR on EKS에 대한 클러스터 액세스 활성화

다음 섹션에서는 클러스터 액세스를 활성화하는 몇 가지 방법을 보여줍니다. 첫 번째에서는 Amazon EKS 클러스터 액세스 관리(CAM)를 사용하며, 두 번째에서는 클러스터 액세스를 활성화하기 위한 수동 단계를 수행하는 방법을 보여줍니다.

## EKS 액세스 항목을 사용하여 클러스터 액세스 활성화(권장)

### Note

aws-auth ConfigMap은 더 이상 사용되지 않습니다. Kubernetes API에 대한 액세스를 관리하는 데 권장되는 방법은 [액세스 항목](#)입니다.

Amazon EMR은 [Amazon EKS 클러스터 액세스 관리\(CAM\)](#)와 통합되어 있으므로, 필요한 AuthN 및 AuthZ 정책의 구성을 자동화하여 Amazon EKS 클러스터의 네임스페이스에서 Amazon EMR Spark 작업을 실행할 수 있습니다. Amazon EKS 클러스터 네임스페이스에서 가상 클러스터를 생성하는 경우 Amazon EMR은 필요한 모든 권한을 자동으로 구성하므로 현재 워크플로에 단계를 추가하지 않아도 됩니다.

### Note

Amazon EKS CAM과 Amazon EMR의 통합은 새로운 Amazon EMR on EKS 가상 클러스터에 대해서만 지원됩니다. 이 통합을 사용하기 위해 기존 가상 클러스터를 마이그레이션할 수 없습니다.

## 사전 조건

- 의 버전 2.15.3 이상을 실행 중인지 확인합니다. AWS CLI
- Amazon EKS 클러스터는 버전 1.23 이상에 있어야 합니다.

## 설정

Amazon EKS에서 AccessEntry API 작업과 Amazon EMR 간 통합을 설정하려면 다음 항목을 완료해야 합니다.

- Amazon EKS 클러스터의 authenticationMode가 API\_AND\_CONFIG\_MAP으로 설정되어 있는지 확인합니다.

```
aws eks describe-cluster --name <eks-cluster-name>
```

아직 아닌 경우 authenticationMode를 API\_AND\_CONFIG\_MAP으로 설정합니다.

```
aws eks update-cluster-config
  --name <eks-cluster-name>
  --access-config authenticationMode=API_AND_CONFIG_MAP
```

인증 모드에 대한 자세한 내용은 [클러스터 인증 모드 섹션](#)을 참조하세요.

- CreateVirtualCluster 및 DeleteVirtualCluster API 작업을 실행하는 데 사용하는 [IAM 역할](#)에 다음 권한도 있는지 확인합니다.

```
{
  "Effect": "Allow",
  "Action": [
    "eks:CreateAccessEntry"
  ],
  "Resource":
    "arn:<AWS_PARTITION>:eks:<AWS_REGION>:<AWS_ACCOUNT_ID>:cluster/<EKS_CLUSTER_NAME>"
},
{
  "Effect": "Allow",
  "Action": [
    "eks:DescribeAccessEntry",
    "eks>DeleteAccessEntry",
    "eks>ListAssociatedAccessPolicies",
    "eks:AssociateAccessPolicy",
    "eks:DisassociateAccessPolicy"
  ],
  "Resource": "arn:<AWS_PARTITION>:eks:<AWS_REGION>:<AWS_ACCOUNT_ID>:access-entry/
    <EKS_CLUSTER_NAME>/role/<AWS_ACCOUNT_ID>/AWSServiceRoleForAmazonEMRContainers/*"
}
```

## 개념 및 용어

다음은 Amazon EKS CAM과 관련된 용어 및 개념 목록입니다.

- 가상 클러스터(VC) - Amazon EKS에서 생성된 네임스페이스의 논리적 표현. Amazon EKS 클러스터 네임스페이스에 대한 1:1 링크입니다. 이를 사용하여 지정된 네임스페이스 내 Amazon EKS 클러스터에서 Amazon EMR 워크로드를 실행할 수 있습니다.
- 네임스페이스 - 단일 EKS 클러스터 내에서 리소스 그룹을 격리하는 메커니즘.
- 액세스 정책 - EKS 클러스터 내의 IAM 역할에 액세스 및 작업을 부여하는 권한.

- 액세스 항목 - 역할 ARN으로 생성된 항목. 액세스 항목을 액세스 정책에 연결하여 Amazon EKS 클러스터에서 특정 권한을 할당할 수 있습니다.
- EKS 액세스 항목 통합 가상 클러스터 - Amazon EKS의 [액세스 항목 API 작업](#)을 사용하여 생성된 가상 클러스터.

### aws-auth를 사용하여 클러스터 액세스 활성화

Kubernetes 역할을 생성하고, 이 역할을 Kubernetes 사용자에게 바인딩하며, Kubernetes 사용자를 서비스 연결 역할 [AWSServiceRoleForAmazonEMRContainers](#)에 매핑하는 등의 작업을 수행함으로써 Amazon EMR on EKS에서 클러스터의 특정 네임스페이스에 액세스할 수 있도록 허용해야 합니다. emr-containers에서 IAM ID 매핑 명령을 서비스 이름으로 사용하면 이러한 작업이 eksctl에서 자동화됩니다. 다음 명령을 사용하여 이러한 작업을 쉽게 수행할 수 있습니다.

```
eksctl create iamidentitymapping \
  --cluster my_eks_cluster \
  --namespace kubernetes_namespace \
  --service-name "emr-containers"
```

*my\_eks\_cluster*를 Amazon EKS 클러스터 이름으로, *kubernetes\_namespace*를 Amazon EMR 워크로드 실행을 위해 생성한 Kubernetes 네임스페이스로 바꿉니다.

#### Important

이 기능을 사용하려면 이전 단계([kubectl 및 eksctl 설정](#))를 사용하여 최신 eksctl을 다운로드해야 합니다.

### Amazon EMR on EKS에서 클러스터 액세스를 활성화하는 수동 단계

다음 수동 단계를 사용하여 Amazon EMR on EKS에 대한 클러스터 액세스를 활성화할 수도 있습니다.

#### 1. 특정 네임스페이스에서 Kubernetes 역할 생성

##### Amazon EKS 1.22 - 1.29

Amazon EKS 1.22~1.29에서는 다음 명령을 실행하여 특정 네임스페이스에 Kubernetes 역할을 생성합니다. 이 역할은 Amazon EMR on EKS에 필요한 RBAC 권한을 부여합니다.

```
namespace=my-namespace
```

```

cat - >>EOF | kubectl apply -f - >>namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: emr-containers
  namespace: ${namespace}
rules:
  - apiGroups: [""]
    resources: ["namespaces"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["serviceaccounts", "services", "configmaps", "events", "pods",
"pods/log"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["create", "patch", "delete", "watch"]
  - apiGroups: ["apps"]
    resources: ["statefulsets", "deployments"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"annotate", "patch", "label"]
  - apiGroups: ["batch"]
    resources: ["jobs"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"annotate", "patch", "label"]
  - apiGroups: ["extensions", "networking.k8s.io"]
    resources: ["ingresses"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"annotate", "patch", "label"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["roles", "rolebindings"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletecollection", "annotate", "patch", "label"]
EOF

```

## Amazon EKS 1.21 and below

Amazon EKS 1.21 이하에서는 다음 명령을 실행하여 특정 네임스페이스에 Kubernetes 역할을 생성합니다. 이 역할은 Amazon EMR on EKS에 필요한 RBAC 권한을 부여합니다.

```
namespace=my-namespace
cat - >>EOF | kubectl apply -f - >>namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: emr-containers
  namespace: ${namespace}
rules:
- apiGroups: [""]
  resources: ["namespaces"]
  verbs: ["get"]
- apiGroups: [""]
  resources: ["serviceaccounts", "services", "configmaps", "events", "pods",
"pods/log"]
  verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletcollection", "annotate", "patch", "label"]
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["create", "patch", "delete", "watch"]
- apiGroups: ["apps"]
  resources: ["statefulsets", "deployments"]
  verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"annotate", "patch", "label"]
- apiGroups: ["batch"]
  resources: ["jobs"]
  verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"annotate", "patch", "label"]
- apiGroups: ["extensions"]
  resources: ["ingresses"]
  verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"annotate", "patch", "label"]
- apiGroups: ["rbac.authorization.k8s.io"]
  resources: ["roles", "rolebindings"]
  verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletcollection", "annotate", "patch", "label"]
- apiGroups: [""]
  resources: ["persistentvolumeclaims"]
```

```
verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletecollection", "annotate", "patch", "label"]
EOF
```

## 2. 네임스페이스로 범위가 지정된 Kubernetes 역할 바인딩 생성

다음 명령을 실행하여 지정된 네임스페이스에서 Kubernetes 역할 바인딩을 생성합니다. 이 역할 바인딩은 이전 단계에서 생성한 역할에 정의된 권한을 `emr-containers` 사용자에게 부여합니다. 이 사용자는 [Amazon EMR on EKS에 대한 서비스 연결 역할](#)을 식별하고, 이를 통해 Amazon EMR on EKS에서 사용자가 생성한 역할로 정의된 작업을 수행할 수 있습니다.

```
namespace=my-namespace

cat - <<EOF | kubectl apply -f - --namespace "${namespace}"
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: emr-containers
  namespace: ${namespace}
subjects:
- kind: User
  name: emr-containers
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: emr-containers
  apiGroup: rbac.authorization.k8s.io
EOF
```

## 3. Kubernetes **aws-auth** 구성 맵 업데이트

다음 옵션 중 하나를 사용하여 Amazon EMR on EKS 서비스 연결 역할을 이전 단계에서 Kubernetes 역할에 바인딩된 `emr-containers` 사용자와 매핑할 수 있습니다.

### 옵션 1: **eksctl** 사용

다음 `eksctl` 명령을 실행하여 Amazon EMR on EKS 서비스 연결 역할을 `emr-containers` 사용자와 매핑합니다.

```
eksctl create iamidentitymapping \
  --cluster my-cluster-name \
```

```
--arn "arn:aws:iam::my-account-id:role/AWSServiceRoleForAmazonEMRContainers" \
--username emr-containers
```

## 옵션 2: eksctl 사용 안 함

1. 다음 명령을 실행하여 aws-auth 구성 맵을 텍스트 편집기에서 엽니다.

```
kubectl edit -n kube-system configmap/aws-auth
```

### Note

Error from server (NotFound): configmaps "aws-auth" not found와 같은 오류 메시지가 표시되면 Amazon EKS 사용 설명서에서 [사용자 역할 추가](#)의 단계를 참조하여 재고 ConfigMap을 적용합니다.

2. Amazon EMR on EKS 서비스 연결 역할 세부 정보를 data 아래 ConfigMap의 mapRoles 섹션에 추가합니다. 파일에 이미 존재하지 않는 경우 이 섹션을 추가합니다. 데이터 아래의 업데이트된 mapRoles 섹션은 다음 예제와 같습니다.

```
apiVersion: v1
data:
  mapRoles: |
    - rolearn: arn:aws:iam::<your-account-id>:role/
      AWSServiceRoleForAmazonEMRContainers
      username: emr-containers
    - ... <other previously existing role entries, if there's any>.
```

3. 파일을 저장하고 텍스트 편집기를 종료합니다.

## Amazon SageMaker Unified Studio에 대한 클러스터 액세스 활성화

Amazon EMR on EKS 및 Amazon SageMaker Unified Studio에는 Amazon EKS 클러스터에 대한 액세스 권한이 필요합니다. [EMR on EKS 및 SageMaker Unified Studio에 대한 EKS 클러스터 액세스 활성화](#)의 단계에 따라 액세스를 제공하세요.

## EKS 클러스터에 IAM 역할 활성화

다음 항목에서는 IAM 역할을 활성화하기 위한 옵션을 자세히 설명합니다.

## 주제



- [옵션 1: EKS 클러스터에서 EKS Pod Identity 활성화](#)
- [옵션 2: EKS 클러스터에서 서비스 계정에 대한 IAM 역할\(IRSA\) 활성화](#)

### 옵션 1: EKS 클러스터에서 EKS Pod Identity 활성화

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 EMR on EKS는 emr-7.3.0 릴리스부터 StartJobRun 제출 모델에 대해 EKS 포드 ID를 지원하기 시작했습니다.

EKS Pod Identity에 대한 자세한 내용은 [EKS Pod Identity의 작동 방식 이해](#)를 참조하세요.

EKS Pod Identity를 사용해야 하는 이유는 무엇인가요?

EMR 설정의 일환으로 작업 실행 역할은 IAM 역할과 특정 네임스페이스(EMR 가상 클러스터)의 서비스 계정 간에 신뢰 경계를 설정합니다. IRSA를 사용하면 EMR 작업 실행 역할의 신뢰 정책을 업데이트하여 이를 수행할 수 있습니다. 그러나 IAM 신뢰 정책 길이의 4096자 하드 제한으로 인해 최대 열두(12개) EKS 클러스터에서 단일 작업 실행 IAM 역할을 공유하는 제약이 있었습니다.

포드 ID에 대한 EMR의 지원을 통해 이제 EKS 팀이 EKS 포드 ID의 연결 API를 통해 IAM 역할과 서비스 계정 간의 신뢰 경계를 관리합니다.

#### Note

EKS Pod Identity의 보안 경계는 포드 수준이 아니라 서비스 계정 수준에 위치합니다.

### Pod Identity 고려 사항

Pod Identity 제한 사항에 대한 자세한 내용은 [EKS Pod Identity 고려 사항](#)을 참조하세요.

### EKS 클러스터에서 EKS Pod Identity 준비

#### 필수 권한이 NodeInstanceRole에 있는지 확인

노드 역할 NodeInstanceRole에는 에이전트가 EKS 인증 API에서 AssumeRoleForPodIdentity 작업을 수행할 수 있는 권한이 필요합니다. Amazon EKS 사용 설명서에 정의된 [AmazonEKSWorkerNodePolicy](#)에 다음을 추가하거나 사용자 지정 정책을 사용할 수 있습니다.

EKS 클러스터가 0.181.0보다 높은 eksctl 버전으로 생성된 경우, 필요한 AssumeRoleForPodIdentity 권한을 포함한 AmazonEKSWorkerNodePolicy가 노드 역할에 자동으로 첨부됩니다. 권한이 없는 경우 Pod Identity에 대한 역할 인계를 허용하는 다음 권한을 AmazonEKSWorkerNodePolicy에 수동으로 추가합니다. 이 권한은 EKS Pod Identity 에이전트가 포드의 자격 증명을 검색하는 데 필요합니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks-auth:AssumeRoleForPodIdentity"
      ],
      "Resource": [
        "*"
      ],
      "Sid": "AllowEKSAUTHAssumeroleforpodidentity"
    }
  ]
}
```

## EKS Pod Identity 에이전트 추가 기능 만들기

다음 명령을 사용하여 EKS Pod Identity Agent 추가 기능을 최신 버전으로 생성합니다.

```
aws eks create-addon --cluster-name cluster-name --addon-name eks-pod-identity-agent

kubectl get pods -n kube-system | grep 'eks-pod-identity-agent'
```

다음 단계를 사용하여 Amazon EKS 콘솔에서 EKS Pod Identity Agent 추가 기능을 생성합니다.

1. [Amazon EKS 콘솔](#)에서 Amazon EKS 콘솔을 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.
6. 선택한 추가 기능 설정 구성 페이지의 버전 드롭다운 목록에서 임의의 버전을 선택합니다.
7. (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.
8. 다음을 선택합니다.
9. CLI를 통해 에이전트 포드가 클러스터에서 실행 중인지 확인합니다.

```
kubect1 get pods -n kube-system | grep 'eks-pod-identity-agent'
```

예시 출력은 다음과 같습니다.

| NAME                         | READY | STATUS  | RESTARTS    | AGE |
|------------------------------|-------|---------|-------------|-----|
| eks-pod-identity-agent-gmqp7 | 1/1   | Running | 1 (24h ago) | 24h |
| eks-pod-identity-agent-prnsh | 1/1   | Running | 1 (24h ago) | 24h |

이 작업을 수행하면 kube-system 네임스페이스에 새 DaemonSet가 설정됩니다. 각 EKS 노드에서 실행되는 Amazon EKS Pod Identity 에이전트는 [AssumeRoleForPodIdentity](#) 작업을 사용하여 EKS Auth API에서 임시 자격 증명을 검색합니다. 그런 다음 컨테이너 내에서 실행하는 AWS SDKs에 이러한 자격 증명을 사용할 수 있습니다.

자세한 내용은 퍼블릭 문서의 사전 조건인 [Amazon EKS Pod Identity Agent 설정](#)을 참조하세요.

## 작업 실행 역할 생성

### EKS Pod Identity를 허용하는 작업 실행 역할 만들기 또는 업데이트

Amazon EMR를 사용하여 EKS에서 워크로드를 실행하려면 IAM 역할을 생성해야 합니다. 이 설명서에서는 이 역할을 작업 실행 역할이라고 합니다. IAM 역할 생성에 대한 자세한 내용은 [IAM 역할 생성](#)을 참조하세요.

또한 작업 실행 역할에 필요한 권한을 지정하는 IAM 정책을 생성한 다음이 정책을 역할에 연결하여 EKS Pod Identity를 활성화합니다.

예를 들어, 다음과 같은 작업 실행 역할이 있습니다. 자세한 내용은 [작업 실행 역할 만들기](#)를 참조하세요.

```
arn:aws:iam::111122223333:role/PodIdentityJobExecutionRole
```

### ⚠ Important

Amazon EMR on EKS는 작업 실행 역할 이름을 기반으로 Kubernetes 서비스 계정을 자동으로 생성합니다. `cluster_name`, `pod_name` 및 `service_account_name`의 조합이 길이 제한을 초과할 경우 작업이 실패할 수 있으므로 역할 이름이 너무 길지 않은지 확인합니다.

작업 실행 역할 구성 - 작업 실행 역할이 EKS Pod Identity에 대한 아래 신뢰 권한으로 생성되었는지 확인합니다. 기존 작업 실행 역할을 업데이트하려면 다음 EKS 서비스 보안 주체를 신뢰 정책의 추가 권한으로 신뢰하도록 구성합니다. 이 신뢰 권한은 기존 IRSA 신뢰 정책과 함께 사용할 수 있습니다.

```
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
```

사용자 권한: 사용자에게는 `StartJobRun` API 호출을 실행하거나 작업을 제출할 수 있는 `iam:PassRole` 권한이 필요합니다. 이 권한을 통해 사용자는 작업 실행 역할을 EMR on EKS에 전달할 수 있습니다. 기본적으로 작업 관리자에게는 권한이 있어야 합니다.

사용자에게 필요한 권한은 다음과 같습니다.

```
{
  "Effect": "Allow",
```

```

    "Action": "iam:PassRole",
    "Resource": "arn:aws:iam::111122223333:role/PodIdentityJobExecutionRole",
    "Condition": {
      "StringEquals": {
        "iam:PassedToService": "pods.eks.amazonaws.com"
      }
    }
  }
}

```

특정 EKS 클러스터에 대한 사용자 액세스를 추가적으로 제한하려면 IAM 정책에 AssociatedResourceArn 속성 필터를 추가합니다. 역할 가정을 승인된 EKS 클러스터로 제한하여 리소스 수준 보안 제어를 강화합니다.

```

"Condition": {
  "ArnLike": {
    "iam:AssociatedResourceARN": [
      "arn:aws:eks:us-west-2:111122223333:cluster/*"
    ]
  }
}

```

## EKS Pod Identity 연결 설정

### 사전 조건

EKS 관리자 사용자와 같은 Pod Identity 연결을 생성하는 IAM 자격 증명에 eks:CreatePodIdentityAssociation 및 iam:PassRole 권한이 있는지 확인합니다.

### JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks:CreatePodIdentityAssociation"
      ],
      "Resource": [
        "arn:aws:eks:*:*:cluster/*"
      ],
      "Sid": "AllowEKSCreatepodidentityassociation"
    }
  ]
}

```

```

    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/*"
      ],
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": "pods.eks.amazonaws.com"
        }
      },
      "Sid": "AllowIAMPassrole"
    }
  ]
}

```

## 역할 및 EMR 서비스 계정에 대한 연결 생성

### Create EMR role associations through the AWS CLI

작업을 Kubernetes 네임스페이스에 제출할 때 관리자는 작업 실행 역할과 EMR 관리형 서비스 계정의 자격 증명 간에 연결을 생성합니다. 단, EMR 관리형 서비스 계정은 작업 제출 시 자동으로 생성되며 작업이 제출된 네임스페이스로 범위가 지정됩니다.

AWS CLI (버전 2.24.0 이상)에서 다음 명령을 실행하여 포드 자격 증명과의 역할 연결을 생성합니다.

다음 명령을 실행하여 Pod Identity와 역할 연결을 생성합니다.

```

aws emr-containers create-role-associations \
  --cluster-name mycluster \
  --namespace mynamespace \
  --role-name JobExecutionRoleIRSAv2

```

### 참고:

- 각 클러스터에는 1,000개의 연결 제한이 있을 수 있습니다. 각 작업 실행 역할 - 네임스페이스 매핑에는 작업 제출자, 드라이버 및 실행기 포드에 대한 3개의 연결이 필요합니다.

- 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 및에 대한 자습서는 [IAM 자습서: IAM 역할을 사용하여 AWS 계정 간 액세스 권한 위임을 AssumeRole](#) 참조하세요.

## Create EMR role associations through Amazon EKS

EMR은 작업이 제출될 때 특정 명명 패턴이 있는 서비스 계정을 생성합니다. 수동으로 연결하거나 이 워크플로를 AWS SDK와 통합하려면 다음 단계를 따르세요.

구문 서비스 계정 이름:

```
emr-containers-sa-spark-%(SPARK_ROLE)s-%(AWS_ACCOUNT_ID)s-
%(BASE36_ENCODED_ROLE_NAME)s
```

아래 예제에서는 샘플 작업 실행 역할 JobExecutionRoleIRSAv2의 역할 연결을 생성합니다.

역할 연결 예제:

```
RoleName: JobExecutionRoleIRSAv2
Base36EncodingOfRoleName: 2eum5fah1jc1kwyjc19ikdhdkdegh1n26vbe
```

샘플 CLI 명령:

```
# setup for the client service account (used by job runner pod)
# emr-containers-sa-spark-client-111122223333-2eum5fah1jc1kwyjc19ikdhdkdegh1n26vbe
aws eks create-pod-identity-association --cluster-name mycluster
--role-arn arn:aws:iam::111122223333:role/JobExecutionRoleIRSAv2
--namespace mynamespace --service-account emr-containers-sa-spark-
client-111122223333-2eum5fah1jc1kwyjc19ikdhdkdegh1n26vbe

# driver service account
# emr-containers-sa-spark-driver-111122223333-2eum5fah1jc1kwyjc19ikdhdkdegh1n26vbe

aws eks create-pod-identity-association --cluster-name mycluster
--role-arn arn:aws:iam::111122223333:role/JobExecutionRoleIRSAv2
--namespace mynamespace --service-account emr-containers-sa-spark-
driver-111122223333-2eum5fah1jc1kwyjc19ikdhdkdegh1n26vbe

# executor service account
```

```
# emr-containers-sa-spark-executor-111122223333-2eum5fah1jc1kwyjc19ikdhdkdegh1n26vbe
aws eks create-pod-identity-association --cluster-name mycluster
--role-arn arn:aws:iam::111122223333:role/JobExecutionRoleIRSAv2
--namespace mynamespace --service-account emr-containers-sa-spark-
executor-111122223333-2eum5fah1jc1kwyjc19ikdhdkdegh1n26vbe
```

EKS Pod Identity에 필요한 모든 단계를 완료한 후에는 IRSA 설정에서 다음 단계를 건너뛸 수 있습니다.

- [EKS 클러스터에서 서비스 계정에 대한 IAM 역할\(IRSA\) 활성화](#)
- [작업 실행 역할 생성](#)
- [작업 실행 역할의 신뢰 정책 업데이트](#)

[사용자에게 Amazon EMR on EKS에 대한 액세스 권한 부여](#) 단계로 직접 건너뛸 수 있습니다.

### 역할 연결 삭제

가상 클러스터 또는 작업 실행 역할을 삭제하고 더 이상 서비스 계정에 EMR에 대한 액세스 권한을 부여하지 않으려는 경우 해당 역할에 대한 연결을 삭제합니다. 그 이유는 EKS가 존재하지 않는 리소스(네임스페이스 및 서비스 계정)와의 연결을 허용하기 때문입니다. Amazon EMR on EKS는 네임스페이스가 삭제되거나 역할이 더 이상 사용되지 않는 경우 연결을 삭제하여 다른 연결을 위한 공간을 확보할 것을 권장합니다.

#### Note

EKS에는 생성할 수 있는 연결 수에 제한이 있으므로(소프트 제한: 클러스터당 연결 1,000개), 연결을 삭제하지 않으면 잔여 연결이 확장 기능에 영향을 미칠 수 있습니다. 지정된 네임스페이스에 Pod Identity 연결을 나열하여 정리해야 하는 잔여 연결이 있는지 확인할 수 있습니다.

```
aws eks list-pod-identity-associations --cluster-name mycluster --namespace mynamespace
```

AWS CLI (버전 2.24.0 이상)에서 다음 emr-containers 명령을 실행하여 EMR의 역할 연결을 삭제합니다.

```
aws emr-containers delete-role-associations \
--cluster-name mycluster \
```



```
--namespace mynamespace \
--role-name JobExecutionRoleIRSAv2
```

## 기존 IRSA를 Pod Identity로 자동 마이그레이션

eksctl 도구를 사용하여 서비스 계정에 대한 기존 IAM 역할(IRSA)을 Pod Identity 연결로 마이그레이션할 수 있습니다.

```
eksctl utils migrate-to-pod-identity \
  --cluster mycluster \
  --remove-oidc-provider-trust-relationship \
  --approve
```

--approve 플래그를 지정하지 않고 명령을 실행하면 마이그레이션 단계를 반영하는 계획만 출력되며 실제 마이그레이션이 발생하지 않습니다.

## 문제 해결

자격 증명 공급자에 대한 NoClassDefinitionFound 또는 ClassNotFound 예외로 인해 작업이 실패했거나 자격 증명 공급자를 가져오지 못했습니다.

EKS Pod Identity는 컨테이너 자격 증명 공급자를 사용하여 필요한 자격 증명을 검색합니다. 사용자 지정 자격 증명 공급자를 지정한 경우 올바르게 작동하는지 확인합니다. 또는 EKS Pod Identity를 지원하는 올바른 AWS SDK 버전을 사용하고 있는지 확인합니다. 자세한 내용은 [Amazon EKS 시작하기](#)를 참조하세요.

eks-pod-identity-agent 로그에 표시된 "[x] 크기 제한으로 인해 자격 증명을 검색하지 못함" 오류로 인해 작업이 실패했습니다.

EMR on EKS는 Kubernetes 서비스 계정을 작업 실행 역할 이름을 기반으로 생성합니다. 역할 이름이 너무 길면, cluster\_name pod\_name 및 service\_account\_name의 조합이 길이 제한을 초과하기 때문에 EKS Auth가 자격 증명을 검색할 수 없습니다. 공간을 가장 많이 차지하는 구성 요소를 식별하고 그에 따라 크기를 조정합니다.

eks-pod-identity 로그에 표시된 "xxx 자격 증명 검색에 실패했습니다." 오류로 인해 작업이 실패했습니다.

이 문제의 한 가지 가능한 원인은 클러스터에 대해 PrivateLink를 올바르게 구성하지 않고 프라이빗 서브넷에 EKS 클러스터가 구성되어 있기 때문일 수 있습니다. 클러스터가 프라이빗 네트워크에 있는지

확인하고 문제를 해결하도록 AWS PrivateLink 구성합니다. 자세한 지침은 [Amazon EKS 시작하기](#)를 참조하세요.

## 옵션 2: EKS 클러스터에서 서비스 계정에 대한 IAM 역할(IRSA) 활성화

서비스 계정에 대한 IAM 역할 기능은 Amazon EKS 버전 1.14 이상 및 2019년 9월 3일 이후 버전 1.13 이상으로 업데이트된 EKS 클러스터에서 사용할 수 있습니다. 이 기능을 사용하기 위해 기존 EKS 클러스터를 버전 1.14 이상으로 업데이트할 수 있습니다. 자세한 내용은 [Amazon EKS 클러스터 Kubernetes 버전 업데이트](#)를 참조하세요.

클러스터에서 서비스 계정에 대한 IAM 역할을 지원하는 경우 클러스터는 이 역할에 연결된 [OpenID Connect](#) 발행자 URL을 갖게 됩니다. Amazon EKS 콘솔에서 이 URL을 보거나 다음 AWS CLI 명령을 사용하여 검색할 수 있습니다.

### Important

이 명령에서 적절한 출력을 받으려면 최신 버전의 AWS CLI 를 사용해야 합니다.

```
aws eks describe-cluster --name cluster_name --query "cluster.identity.oidc.issuer" --output text
```

예상 출력은 다음과 같습니다.

```
https://oidc.eks.<region-code>.amazonaws.com/id/EXAMPLED539D4633E53DE1B716D3041E
```

클러스터에서 서비스 계정에 대한 IAM 역할을 사용하려면 [eksctl](#) 또는 [AWS Management Console](#)을 사용하여 OIDC ID 제공업체를 생성해야 합니다.

**eksctl**로 클러스터의 IAM OIDC 자격 증명 공급자를 생성하려면

eksctl 버전은 다음 명령을 통해 확인할 수 있습니다. 이 절차에서는 eksctl을 설치했으며 eksctl 버전이 0.32.0 이상인 것으로 가정합니다.

```
eksctl version
```

eksctl 설치 또는 업그레이드에 관한 자세한 내용은 [Installing or upgrading eksctl](#)을 참조하세요.

다음 명령을 사용하여 클러스터용 OIDC 자격 증명 공급자를 생성합니다. *cluster\_name*을 고유한 값으로 바꿉니다.

```
eksctl utils associate-iam-oidc-provider --cluster cluster_name --approve
```

를 사용하여 클러스터에 대한 IAM OIDC 자격 증명 공급자를 생성하려면 AWS Management Console 클러스터의 Amazon EKS 콘솔 설명에서 OIDC 발급자 URL을 검색하거나 다음 AWS CLI 명령을 사용합니다.

AWS CLI에서 OIDC 발급자 URL을 검색하려면 다음 명령을 사용합니다.

```
aws eks describe-cluster --name <cluster_name> --query "cluster.identity.oidc.issuer" --output text
```

Amazon EKS 콘솔에서 OIDC 발급자 URL을 검색하려면 다음 단계를 사용합니다.

1. IAM 콘솔(<https://console.aws.amazon.com/iam/>)을 엽니다.
2. 탐색 창에서 ID 제공업체를 선택하고 제공업체 생성을 선택합니다.
  1. 공급자 유형에서 공급자 유형 선택을 선택한 다음 OpenID Connect를 선택합니다.
  2. Provider URL(공급자 URL)에 클러스터의 OIDC 발행자 URL을 붙여넣습니다.
  3. 대상에서 sts.amazonaws.com을 입력한 후 다음 단계를 선택합니다.
3. 공급자 정보가 올바른지 확인한 다음 생성을 선택하여 자격 증명 공급자를 생성합니다.

## 작업 실행 역할 생성

Amazon EMR on EKS에서 워크로드를 실행하려면 IAM 역할을 생성해야 합니다. 이 설명서에서는 이 역할을 작업 실행 역할이라고 합니다. IAM 역할 생성에 대한 자세한 내용은 IAM 사용 설명서에서 [IAM 역할 생성](#)을 참조하세요.

또한 작업 실행 역할에 대한 권한을 지정하는 IAM 정책을 생성한 다음, IAM 정책을 작업 실행 역할에 연결해야 합니다.

작업 실행 역할에 대한 다음 정책은 리소스 대상, Amazon S3 및 CloudWatch에 대한 액세스를 허용합니다. 이러한 권한은 작업 및 액세스 로그를 모니터링하는 데 필요합니다. AWS CLI를 사용하여 동일한 프로세스를 수행하려면:

작업 실행을 위한 IAM 역할 만들기: EMR이 작업 실행에 사용할 역할을 만들어 보겠습니다. 이 역할은 EKS에서 실행될 때 EMR 작업이 수행하게 될 역할입니다.

```
cat <<EoF > ~/environment/emr-trust-policy.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "elasticmapreduce.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EoF

aws iam create-role --role-name EMRContainers-JobExecutionRole --assume-role-policy-
document file://~/environment/emr-trust-policy.json
```

다음으로 s3 및 cloudwatch에 로그를 쓸 수 있도록 필요한 IAM 정책을 역할에 연결합니다.

```
cat <<EoF > ~/environment/EMRContainers-JobExecutionRole.json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": "arn:aws:s3:::amzn-s3-demo-bucket"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogStream",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
```

```

        "arn:aws:logs:*:*:*"
    ]
}
]
}
EoF
aws iam put-role-policy --role-name EMRContainers-JobExecutionRole --policy-name
EMR-Containers-Job-Execution --policy-document file://~/environment/EMRContainers-
JobExecutionRole.json

```

### Note

액세스 범위는 적절해야 하며, 작업 실행 역할의 모든 S3 객체에 부여되지 않아야 합니다.

## JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket"
      ],
      "Sid": "AllowS3Putobject"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogStream",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams"
      ],
      "Resource": [

```

```

        "arn:aws:logs:*:*:*"
    ],
    "Sid": "AllowLOGSPutlogevents"
  }
]
}

```

자세한 내용은 [작업 실행 역할 사용](#), [S3 로그를 사용하도록 작업 실행 구성](#) 및 [CloudWatch Logs를 사용하도록 작업 실행 구성](#)을 참조하세요.

### 작업 실행 역할의 신뢰 정책 업데이트

서비스 계정에 대한 IAM 역할(IRSA)을 사용하여 Kubernetes 네임스페이스에서 작업을 실행하는 경우 관리자는 작업 실행 역할과 EMR 관리형 서비스 계정의 자격 증명 사이에서 신뢰 관계를 생성해야 합니다. 신뢰 관계는 작업 실행 역할의 신뢰 정책을 업데이트하여 생성할 수 있습니다. 단, EMR 관리형 서비스 계정은 작업 제출 시 자동으로 생성되며 작업이 제출된 네임스페이스로 범위가 지정됩니다.

다음 명령을 실행하여 신뢰 정책을 업데이트합니다.

```

aws emr-containers update-role-trust-policy \
  --cluster-name cluster \
  --namespace namespace \
  --role-name iam_role_name_for_job_execution

```

자세한 내용은 [Amazon EMR on EKS에서 작업 실행 역할 사용](#) 단원을 참조하십시오.

#### Important

위 명령을 실행하는 운영자에게 eks:DescribeCluster, iam:GetRole, iam:UpdateAssumeRolePolicy 권한이 있어야 합니다.

## Amazon EMR on EKS에 대한 액세스 권한 부여

Amazon EMR on EKS에서 수행하는 모든 작업에는 해당 작업에 해당하는 IAM 권한이 필요합니다. Amazon EMR on EKS 작업을 수행하는 데 필요한 IAM 정책을 생성하고 이 정책을 사용하고 있는 IAM 사용자 또는 역할에 연결해야 합니다.

이 주제에서는 새 정책을 생성하여 사용자에게 연결하는 단계를 제공합니다. 또한 여기에서는 Amazon EMR on EKS 환경을 설정하는 데 필요한 기본 권한도 다룹니다. 비즈니스 요구 사항에 따라 가능한 한 항상 특정 리소스에 대한 권한을 구체화하는 것이 좋습니다.

새 IAM 정책을 생성하고 IAM 콘솔에서 사용자에게 연결

### 새 IAM 정책 생성

1. 에 로그인 AWS Management Console 하고 <https://console.aws.amazon.com/iam/> IAM 콘솔을 엽니다.
2. IAM 콘솔의 왼쪽 탐색 창에서 정책을 선택합니다.
3. 정책 페이지에서 정책 생성을 선택합니다.
4. 정책 생성 창에서 JSON 편집 탭으로 이동합니다. 이 절차 다음의 예제와 같이 하나 이상의 JSON 명령문이 포함된 정책 문서를 생성합니다. 다음으로 정책 검토를 선택합니다.
5. [정책 검토(Review Policy)] 화면에서 [정책 이름(Policy Name)]을 입력합니다(예: AmazonEMR0nEKSPolicy). 정책의 이름과 설명(선택 사항)을 입력하고 정책 생성을 선택합니다.

### 사용자 또는 역할에 정책 연결

1. 에 로그인 AWS Management Console 하고 <https://console.aws.amazon.com/iam/> IAM 콘솔을 엽니다.
2. 탐색 창에서 Policies를 선택합니다.
3. 정책 목록에서, 이전 섹션에서 생성한 정책 옆의 확인란을 선택합니다. [Filter] 메뉴와 검색 상자를 사용하여 정책 목록을 필터링할 수 있습니다.
4. 정책 조치를 선택한 후 연결을 선택합니다.
5. 정책을 연결할 사용자 또는 역할을 선택합니다. 필터 메뉴와 검색 상자를 사용하면 보안 주체 개체 목록을 필터링할 수 있습니다. 정책을 추가할 사용자 또는 역할을 선택한 후 정책 연결을 선택합니다.

### 가상 클러스터 관리를 위한 권한

AWS 계정에서 가상 클러스터를 관리하려면 다음 권한을 사용하여 IAM 정책을 생성합니다. 이러한 권한을 통해 AWS 계정에서 가상 클러스터를 생성, 나열, 설명 및 삭제할 수 있습니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:CreateServiceLinkedRole"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "emr-containers.amazonaws.com"
        }
      },
      "Sid": "AllowIAMCreateservicelinkedrole"
    },
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:CreateVirtualCluster",
        "emr-containers:ListVirtualClusters",
        "emr-containers:DescribeVirtualCluster",
        "emr-containers>DeleteVirtualCluster"
      ],
      "Resource": [
        "*"
      ],
      "Sid": "AllowEMRCONTAINERSCreatevirtualcluster"
    }
  ]
}
```

Amazon EMR은 Amazon EKS 클러스터 액세스 관리(CAM)와 통합되어 있으므로, 필요한 AuthN 및 AuthZ 정책의 구성을 자동화하여 Amazon EKS 클러스터의 네임스페이스에서 Amazon EMR Spark 작업을 실행할 수 있습니다. 이를 수행하려면 다음 권한이 있어야 합니다.

```
{
```



```

"Effect": "Allow",
"Action": [
    "eks:CreateAccessEntry"
],
"Resource":
"arn:<AWS_PARTITION>:eks:<AWS_REGION>:<AWS_ACCOUNT_ID>:cluster/<EKS_CLUSTER_NAME>"
},
{
    "Effect": "Allow",
    "Action": [
        "eks:DescribeAccessEntry",
        "eks>DeleteAccessEntry",
        "eks:ListAssociatedAccessPolicies",
        "eks:AssociateAccessPolicy",
        "eks:DisassociateAccessPolicy"
    ],
    "Resource": "arn:<AWS_PARTITION>:eks:<AWS_REGION>:<AWS_ACCOUNT_ID>:access-
entry/<EKS_CLUSTER_NAME>/role/<AWS_ACCOUNT_ID>/AWSServiceRoleForAmazonEMRContainers/*"
}

```

자세한 내용은 [Amazon EMR on EKS에 대한 클러스터 액세스 활성화 자동화](#)를 참조하세요.

AWS 계정에서 CreateVirtualCluster 작업을 처음 호출하는 경우 Amazon EMR on EKS에 대한 서비스 연결 역할을 생성할 수 있는 CreateServiceLinkedRole 권한도 필요합니다. 자세한 내용은 [Amazon EMR on EKS에 대한 서비스 연결 역할 사용](#) 단원을 참조하십시오.

작업을 제출하기 위한 권한

AWS 계정의 가상 클러스터에서 작업을 제출하려면 다음 권한이 있는 IAM 정책을 생성합니다. 이러한 권한을 통해 계정의 모든 가상 클러스터에 대한 작업 실행을 시작, 나열, 설명 및 취소할 수 있습니다. 작업을 제출하기 전에 가상 클러스터의 상태를 확인할 수 있도록 가상 클러스터를 나열하거나 설명하기 위한 권한을 추가하는 것을 고려해야 합니다.

JSON

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "emr-containers:StartJobRun",

```

```

        "emr-containers:ListJobRuns",
        "emr-containers:DescribeJobRun",
        "emr-containers:CancelJobRun"
    ],
    "Resource": [
        "*"
    ],
    "Sid": "AllowEMRCONTAINERSStartjobrun"
}
]
}

```

## 디버깅 및 모니터링을 위한 권한

Amazon S3 및 CloudWatch로 푸시된 로그에 액세스하거나 Amazon EMR 콘솔에서 애플리케이션 이벤트 로그를 보려면 다음 권한이 포함된 IAM 정책을 생성합니다. 비즈니스 요구 사항에 따라 가능하면 항상 특정 리소스에 대한 권한을 구체화하는 것이 좋습니다.

### Important

Amazon S3 버킷을 생성하지 않은 경우 정책 명령문에 `s3:CreateBucket` 권한을 추가해야 합니다. 로그 그룹을 생성하지 않은 경우 정책 명령문에 `logs:CreateLogGroup`를 추가해야 합니다.

## JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DescribeJobRun",
        "elasticmapreduce:CreatePersistentAppUI",
        "elasticmapreduce:DescribePersistentAppUI",
        "elasticmapreduce:GetPersistentAppUIPresignedURL"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}

```

```

    ],
    "Sid": "AllowEMRCONTAINERSDescribejobrun"
  },
  {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:ListBucket"
    ],
    "Resource": [
      "*"
    ],
    "Sid": "AllowS3GetObject"
  },
  {
    "Effect": "Allow",
    "Action": [
      "logs:Get*",
      "logs:DescribeLogGroups",
      "logs:DescribeLogStreams"
    ],
    "Resource": [
      "*"
    ],
    "Sid": "AllowLOGSGet"
  }
]
}

```

Amazon S3 및 CloudWatch에 로그를 푸시하도록 작업 실행을 구성하는 방법에 대한 자세한 내용은 [S3 로그를 사용하도록 작업 실행 구성 및 CloudWatch Logs를 사용하도록 작업 실행 구성을 참조하세요](#).

## Amazon EMR에 Amazon EKS 클러스터 등록

클러스터 등록은 워크로드 실행을 위해 Amazon EMR on EKS를 설정하는 마지막 필수 단계입니다.

다음 명령을 사용하여 Amazon EKS 클러스터에 대해 선택한 이름과 이전 단계에서 설정한 네임스페이스로 가상 클러스터를 생성합니다.

**Note**

각 가상 클러스터는 모든 EKS 클러스터에서 이름이 고유해야 합니다. 두 가상 클러스터의 이름이 같은 경우 두 가상 클러스터가 서로 다른 EKS 클러스터에 속해 있더라도 배포 프로세스에 실패합니다.

```
aws emr-containers create-virtual-cluster \
--name virtual_cluster_name \
--container-provider '{
  "id": "cluster_name",
  "type": "EKS",
  "info": {
    "eksInfo": {
      "namespace": "namespace_name"
    }
  }
}'
```

또는 가상 클러스터에 필요한 파라미터가 포함된 JSON 파일을 생성하고 JSON 파일 경로와 함께 `create-virtual-cluster` 명령을 실행할 수 있습니다. 자세한 내용은 [가상 클러스터 관리](#) 단원을 참조하십시오.

**Note**

가상 클러스터가 생성되었는지 확인하려면 `list-virtual-clusters` 작업을 사용하거나 Amazon EMR 콘솔의 가상 클러스터 페이지로 이동하여 가상 클러스터의 상태를 확인합니다.

## StartJobRun을 사용하여 작업 실행 제출

지정된 파라미터를 포함하는 JSON 파일을 사용하여 작업 실행을 제출하는 방법

1. 다음 예제 JSON 파일에서 볼 수 있듯이 `start-job-run-request.json` 파일을 생성하고 작업 실행에 필요한 파라미터를 지정합니다. 파라미터에 대한 자세한 내용은 [작업 실행 구성 옵션](#)를 참조하세요.

```
{
  "name": "myjob",
```

```

"virtualClusterId": "123456",
"executionRoleArn": "iam_role_name_for_job_execution",
"releaseLabel": "emr-6.2.0-latest",
"jobDriver": {
  "sparkSubmitJobDriver": {
    "entryPoint": "entryPoint_location",
    "entryPointArguments": ["argument1", "argument2", ...],
    "sparkSubmitParameters": "--class <main_class> --conf
spark.executor.instances=2 --conf spark.executor.memory=2G --conf
spark.executor.cores=2 --conf spark.driver.cores=1"
  }
},
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "spark-defaults",
      "properties": {
        "spark.driver.memory": "2G"
      }
    }
  ],
  "monitoringConfiguration": {
    "persistentAppUI": "ENABLED",
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "my_log_group",
      "logStreamNamePrefix": "log_stream_prefix"
    },
    "s3MonitoringConfiguration": {
      "logUri": "s3://my_s3_log_location"
    }
  }
}
}

```

- 로컬에 저장된 start-job-run-request.json 파일 경로와 함께 start-job-run 명령을 사용합니다.

```

aws emr-containers start-job-run \
--cli-input-json file:///./start-job-run-request.json

```

## start-job-run 명령을 사용하여 작업 실행을 시작하는 방법

1. 다음 예제에서 볼 수 있듯이 StartJobRun 명령에 지정된 모든 파라미터를 제공합니다.

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--execution-role-arn execution-role-arn \
--release-label emr-6.2.0-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "entryPoint_location",
"entryPointArguments": ["argument1", "argument2", ...], "sparkSubmitParameters":
"--class <main_class> --conf spark.executor.instances=2 --conf
spark.executor.memory=2G --conf spark.executor.cores=2 --conf
spark.driver.cores=1"}}' \
--configuration-overrides '{"applicationConfiguration": [{"classification":
"spark-defaults", "properties": {"spark.driver.memory": "2G"}}],
"monitoringConfiguration": {"cloudWatchMonitoringConfiguration":
{"logGroupName": "log_group_name", "logStreamNamePrefix": "log_stream_prefix"},
"persistentAppUI": "ENABLED", "s3MonitoringConfiguration": {"logUri":
"s3://my_s3_log_location" }}}
```

2. Spark SQL의 경우 다음 예제에서 볼 수 있듯이 StartJobRun 명령에 지정된 모든 파라미터를 제공합니다.

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--execution-role-arn execution-role-arn \
--release-label emr-6.7.0-latest \
--job-driver '{"sparkSqlJobDriver": {"entryPoint": "entryPoint_location",
"sparkSqlParameters": "--conf spark.executor.instances=2 --conf
spark.executor.memory=2G --conf spark.executor.cores=2 --conf
spark.driver.cores=1"}}' \
--configuration-overrides '{"applicationConfiguration": [{"classification":
"spark-defaults", "properties": {"spark.driver.memory": "2G"}}],
"monitoringConfiguration": {"cloudWatchMonitoringConfiguration":
{"logGroupName": "log_group_name", "logStreamNamePrefix": "log_stream_prefix"},
"persistentAppUI": "ENABLED", "s3MonitoringConfiguration": {"logUri":
"s3://my_s3_log_location" }}}
```

## 작업 제출자 분류 사용

### 개요

Amazon EMR on EKS StartJobRun 요청은 Spark 드라이버를 생성하기 위해 작업 제출자 포드(job-runner 포드라고도 함)를 생성합니다. emr-job-submitter 분류를 사용하여 노드 선택기를 구성하고, 허용 오차를 추가하고, 로깅을 사용자 지정하고, 작업 제출자 포드를 기타 수정할 수 있습니다.

emr-job-submitter 분류에서 사용할 수 있는 설정은 다음과 같습니다.

#### `jobsubmitter.node.selector.[selectorKey]`

키 *selectorKey* 및 값을 구성 값으로 사용하여 작업 제출자 포드의 노드 선택기에를 추가합니다. 예를 들어를 `jobsubmitter.node.selector.identifier`로 설정할 수 `myIdentifier` 있으며 작업 제출자 포드에는 키 `identifier`와 값이 있는 노드 선택기가 있습니다 `myIdentifier`. 이는 작업 제출자 포드를 배치할 수 있는 노드를 지정하는 데 사용할 수 있습니다. 여러 노드 선택기 키를 추가하려면 이 접두사를 사용하여 여러 구성을 설정합니다.

#### `jobsubmitter.label.[labelKey]`

key *labelKey* 및 값을 구성 값으로 사용하여 작업 제출자 포드의 레이블에를 추가합니다. 여러 레이블을 추가하려면 이 접두사를 사용하여 여러 구성을 설정합니다.

#### `jobsubmitter.annotation.[annotationKey]`

작업 제출자 포드의 주석에 키 *annotationKey* 및 값을 구성 값으로 사용하여 추가합니다. 여러 주석을 추가하려면 이 접두사를 사용하여 여러 구성을 설정합니다.

#### `jobsubmitter.node.toleration.[tolerationKey]`

작업 제출자 포드에 [허용치](#)를 추가합니다. 기본적으로 포드에 추가된 허용 오차는 없습니다. 톨러레이션의 키는 *tolerationKey*이고 톨러레이션의 값은 구성 값입니다. 구성 값이 비어 있지 않은 문자열로 설정된 경우 연산자가 됩니다 `Equals`. 구성 값이 로 설정된 경우 연산자 `Exists`가 됩니다.

#### `jobsubmitter.node.toleration.[tolerationKey].[effect]`

접두사가 붙은 *tolerationKey* 효과를 추가합니다. 이 필드는 허용치를 추가할 때 필요합니다. 효과 필드에 허용되는 값은 `NoExecute`, `NoSchedule` 및 `PreferNoSchedule`입니다.

#### `jobsubmitter.node.toleration.[tolerationKey].[tolerationSeconds]`

*tolerationKey* 접두사에 *tolerationSeconds*를 추가합니다. *tolerationKey* 선택 필드입니다. 효과가 인 경우에만 적용됩니다 `NoExecute`.

## **jobsubmitter.scheduler.name**

작업 제출자 포드에 대한 사용자 지정 schedulerName 설정합니다.

## **jobsubmitter.logging**

작업 제출자 포드에 대한 로깅을 활성화하거나 비활성화합니다. 이 값이 로 설정되면 작업 제출자 포드에서 DISABLED 로깅 컨테이너가 제거되어 s3MonitoringConfiguration 또는 monitoringConfiguration과 같이 지정된 포드에 대한 로깅이 비활성화됩니다. cloudWatchMonitoringConfiguration. 이 설정이 설정되지 않았거나 다른 값으로 설정된 경우 작업 제출자 포드에 대한 로깅이 활성화됩니다.

## **jobsubmitter.logging.image**

작업 제출자 포드의 로깅 컨테이너에 사용할 사용자 지정 이미지를 설정합니다.

## **jobsubmitter.logging.request.cores**

작업 제출자 포드의 로깅 컨테이너에 대한 CPU 수의 사용자 지정 값을 CPU 단위로 설정합니다. 기본적으로 이 값은 100m로 설정되어 있습니다.

## **jobsubmitter.logging.request.memory**

작업 제출자 포드의 로깅 컨테이너에 대한 메모리 양에 대한 사용자 지정 값을 바이트 단위로 설정합니다. 기본적으로 이 값은 200Mi로 설정되어 있습니다. 메비바이트는 메가바이트와 유사한 측정 단위입니다.

## **jobsubmitter.container.image**

작업 제출자 포드의 job-runner 컨테이너에 대한 사용자 지정 이미지를 설정합니다.

## **jobsubmitter.container.image.pullPolicy**

작업 제출자 포드의 컨테이너에 대한 [imagePullPolicy](#)를 설정합니다.

작업 제출자 포드는 온디맨드 인스턴스에 배치하는 것이 좋습니다. 작업 제출자 포드를 스팟 인스턴스에 배치하면 작업 제출자 포드가 실행되는 인스턴스에서 스팟 인스턴스 종단이 발생하는 경우 작업이 실패할 수 있습니다. [작업 제출자 포드를 단일 가용 영역에 배치하거나 노드에 적용되는 Kubernetes 레이블을 사용할 수도 있습니다.](#)

## 작업 제출자 분류 예제

### 이 섹션의 내용

- [작업 제출자 포드의 온디맨드 노드 배치를 포함한 StartJobRun 요청](#)



- [StartJobRun 작업 제출자 포드에 대한 단일 AZ 노드 배치 및 Amazon EC2 인스턴스 유형 배치를 사용한 요청](#)
- [StartJobRun 작업 제출자 포드에 대한 레이블, 주석 및 사용자 지정 스케줄러가 포함된 요청](#)
- [StartJobRun 키, 값 dedicated, 효과 graviton\\_machines, 60초tolerationSeconds의 NoExecute가 있는 작업 제출자 포드에 허용 오차가 적용된 요청](#)
- [StartJobRun 작업 제출자 포드에 대해 로깅이 비활성화된 요청](#)
- [StartJobRun 작업 제출자 포드에 대한 사용자 지정 로깅 컨테이너 이미지, CPU 및 메모리를 사용한 요청](#)
- [StartJobRun 사용자 지정 작업 제출자 컨테이너 이미지 및 폴 정책을 사용한 요청](#)

작업 제출자 포드의 온디맨드 노드 배치를 포함한 **StartJobRun** 요청

```
cat >spark-python-in-s3-nodeselector-job-submitter.json << EOF
{
  "name": "spark-python-in-s3-nodeselector",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py",
      "sparkSubmitParameters": "--conf spark.driver.cores=5 --conf
spark.executor.memory=20G --conf spark.driver.memory=15G --conf
spark.executor.cores=6"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.dynamicAllocation.enabled": "false"
        }
      },
      {
        "classification": "emr-job-submitter",
        "properties": {
          "jobsubmitter.node.selector.eks.amazonaws.com/capacityType": "ON_DEMAND"
        }
      }
    ]
  }
}
```

```

    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
EOF
aws emr-containers start-job-run --cli-input-json file:///spark-python-in-s3-
nodeselector-job-submitter.json

```

**StartJobRun** 작업 제출자 포드에 대한 단일 AZ 노드 배치 및 Amazon EC2 인스턴스 유형 배치를 사용한 요청

```

"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "emr-job-submitter",
      "properties": {
        "jobsubmitter.node.selector.topology.kubernetes.io/zone": "Availability Zone",
        "jobsubmitter.node.selector.node.kubernetes.io/instance-type": "m5.4xlarge"
      }
    }
  ]
}

```

**StartJobRun** 작업 제출자 포드에 대한 레이블, 주석 및 사용자 지정 스케줄러가 포함된 요청

```

"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "emr-job-submitter",
      "properties": {
        "jobsubmitter.label.label1": "value1",
        "jobsubmitter.label.label2": "value2",
        "jobsubmitter.annotation.ann1": "value1",
        "jobsubmitter.annotation.ann2": "value2",
        "jobsubmitter.scheduler.name": "custom-scheduler"
      }
    }
  ]
}

```

```
    }
  }
]
}
```

**StartJobRun** 키, 값 **dedicated**, 효과 **graviton\_machines**, 60초 **tolerationSeconds**의 **NoExecute**가 있는 작업 제출자 포드에 허용 오차가 적용된 요청

```
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "emr-job-submitter",
      "properties": {
        "jobsubmitter.node.toleration.dedicated": "graviton_machines",
        "jobsubmitter.node.toleration.dedicated.effect": "NoExecute",
        "jobsubmitter.node.toleration.dedicated.tolerationSeconds": "60"
      }
    }
  ]
}
```

**StartJobRun** 작업 제출자 포드에 대해 로깅이 비활성화된 요청

```
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "emr-job-submitter",
      "properties": {
        "jobsubmitter.logging": "DISABLED"
      }
    }
  ],
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "/emr-containers/jobs",
      "logStreamNamePrefix": "demo"
    },
    "s3MonitoringConfiguration": {
      "logUri": "s3://joblogs"
    }
  }
}
```

## StartJobRun 작업 제출자 포드에 대한 사용자 지정 로깅 컨테이너 이미지, CPU 및 메모리를 사용한 요청

```
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "emr-job-submitter",
      "properties": {
        "jobsubmitter.logging.image": "YOUR_ECR_IMAGE_URL",
        "jobsubmitter.logging.request.memory": "200Mi",
        "jobsubmitter.logging.request.cores": "0.5"
      }
    }
  ],
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "/emr-containers/jobs",
      "logStreamNamePrefix": "demo"
    },
    "s3MonitoringConfiguration": {
      "logUri": "s3://joblogs"
    }
  }
}
```

## StartJobRun 사용자 지정 작업 제출자 컨테이너 이미지 및 풀 정책을 사용한 요청

```
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "emr-job-submitter",
      "properties": {
        "jobsubmitter.container.image": "123456789012.dkr.ecr.us-west-2.amazonaws.com/emr6.11_custom_repo",
        "jobsubmitter.container.image.pullPolicy": "kubernetes pull policy"
      }
    }
  ]
}
```

# Amazon EMR 컨테이너 기본 분류 사용

## 개요

emr-containers-defaults 분류에서 사용할 수 있는 설정은 다음과 같습니다.

### job-start-timeout

기본적으로, 작업을 시작할 수 없고 SUBMITTED 상태로 15분 동안 대기하면 작업이 시간 초과됩니다. 이 구성은 작업 시간이 초과되기 전에 대기할 초 수를 변경합니다.

### executor.logging

실행기 포드에 대한 로깅을 활성화하거나 비활성화합니다. 이 값을 로 설정하면 DISABLED 로깅 컨테이너가 실행기 포드에서 제거되어 s3MonitoringConfiguration 또는 monitoringConfiguration과 같이 지정된 이러한 포드에 대한 로깅이 비활성화됩니다. cloudWatchMonitoringConfiguration. 이 설정이 설정되지 않았거나 다른 값으로 설정된 경우 실행기 포드에 대한 로깅이 활성화됩니다.

### logging.image

드라이버 및 실행기 포드의 로깅 컨테이너에 사용할 사용자 지정 이미지를 설정합니다.

### logging.request.cores

드라이버 및 실행기 포드의 CPU 수의 사용자 지정 값을 CPU 단위로 설정합니다. 기본적으로 이 값은 설정되어 있지 않습니다.

### logging.request.memory

드라이버 및 실행기 포드에서 로깅 컨테이너에 할당할 메모리 양(바이트 단위)에 대한 사용자 지정 값을 설정합니다. 기본적으로 이 값은 512Mi로 설정되어 있습니다. 메비바이트는 메가바이트와 유사한 측정 단위입니다.

## 작업 제출자 분류 예제

### 이 섹션의 내용

- [사용자 지정 작업 제한 시간이 있는 StartJobRun 요청](#)
- [StartJobRun 실행기 포드에 대해 로깅이 비활성화된 요청](#)
- [StartJobRun 드라이버 및 실행기 포드에 대한 사용자 지정 로깅 컨테이너 이미지, CPU 및 메모리를 사용한 요청](#)

## 사용자 지정 작업 제한 시간이 있는 **StartJobRun** 요청

```
{
  "name": "spark-python",
  "virtualClusterId": "virtual-cluster-id",
  "executionRoleArn": "execution-role-arn",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "s3://S3-prefix/trip-count.py"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "emr-containers-defaults",
        "properties": {
          "job-start-timeout": "1800"
        }
      }
    ],
    "monitoringConfiguration": {
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "/emr-containers/jobs",
        "logStreamNamePrefix": "demo"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://joblogs"
      }
    }
  }
}
```

## **StartJobRun** 실행기 포드에 대해 로깅이 비활성화된 요청

```
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "emr-containers-defaults",
      "properties": {
        "executor.logging": "DISABLED"
      }
    }
  ]
}
```

```

],
"monitoringConfiguration": {
  "cloudWatchMonitoringConfiguration": {
    "logGroupName": "/emr-containers/jobs",
    "logStreamNamePrefix": "demo"
  },
  "s3MonitoringConfiguration": {
    "logUri": "s3://joblogs"
  }
}
}

```

**StartJobRun** 드라이버 및 실행기 포드에 대한 사용자 지정 로깅 컨테이너 이미지, CPU 및 메모리를 사용한 요청

```

"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "emr-containers-defaults",
      "properties": {
        "logging.image": "YOUR_ECR_IMAGE_URL",
        "logging.request.memory": "200Mi",
        "logging.request.cores": "0.5"
      }
    }
  ],
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "/emr-containers/jobs",
      "logStreamNamePrefix": "demo"
    },
    "s3MonitoringConfiguration": {
      "logUri": "s3://joblogs"
    }
  }
}

```

## Spark 운영자에서 Spark 작업 실행

Amazon EMR 릴리스 6.10.0 이상에서는 Amazon EMR on EKS의 작업 제출 모델로 Apache Spark의 Kubernetes 운영자 또는 Spark 운영자를 지원합니다. Spark 운영자를 사용하면 Amazon EMR 릴리

스 런타임으로 자체 Amazon EKS 클러스터에서 Spark 애플리케이션을 배포하고 관리할 수 있습니다. Amazon EKS 클러스터에 Spark 운영자를 배포한 후에는 운영자를 통해 Spark 애플리케이션을 직접 제출할 수 있습니다. 운영자는 Spark 애플리케이션의 수명 주기를 관리합니다.

#### Note

Amazon EMR은 vCPU 및 메모리 소비를 기반으로 Amazon EKS에 대한 요금을 계산합니다. 이 계산은 드라이버 및 실행기 포드에 적용됩니다. 이 계산은 Amazon EMR 애플리케이션 이미지를 다운로드한 시점부터 시작하여 Amazon EKS 포드가 종료될 때까지 수행되며, 가장 가까운 초로 반올림됩니다.

### 주제

- [Amazon EMR on EKS에 대한 Spark 운영자 구성](#)
- [Amazon EMR on EKS에 대한 Spark 운영자 시작하기](#)
- [Amazon EMR on EKS에 대한 Spark 운영자를 사용하여 수직 자동 조정 사용](#)
- [Amazon EMR on EKS에 대한 Spark 운영자 제거](#)
- [모니터링 구성을 사용하여 Spark Kubernetes 운영자 및 Spark 작업 모니터링을 참조하세요.](#)
- [Amazon EMR on EKS에서 보안 및 Spark 운영자](#)

## Amazon EMR on EKS에 대한 Spark 운영자 구성

Amazon EKS에서 Spark 운영자를 설치하려면 먼저 설정을 위해 다음 작업을 수행합니다. Amazon Web Services(AWS)에 이미 가입했고 Amazon EKS를 사용하고 있는 경우 Amazon EMR on EKS를 사용할 준비를 거의 마친 상태입니다. Amazon EKS에서 Spark 운영자를 설정하려면 다음 작업을 수행합니다. 필수 조건 중 하나를 이미 완료한 경우 해당 조건을 건너뛰고 다음 조건으로 넘어갈 수 있습니다.

- [의 최신 버전 설치 또는 업데이트 AWS CLI](#) -를 이미 설치한 경우 최신 버전이 있는지 AWS CLI 확인합니다.
- [kubecti 및 eksctl 설치](#) - eksctl은 Amazon EKS와 통신하는 데 사용하는 명령줄 도구입니다.
- [Install Helm](#) - Kubernetes용 Helm 패키지 관리자는 Kubernetes 클러스터에서 애플리케이션을 설치하고 관리하는 데 도움이 됩니다.
- [Amazon EKS 시작하기 – eksctl](#) - 관련 단계를 수행하여 Amazon EKS에서 노드를 포함하는 새 Kubernetes 클러스터를 생성합니다.



- [Amazon EMR 기본 이미지 URI 선택](#)(릴리스 6.10.0 이상) - Spark 운영자는 Amazon EMR 릴리스 6.10.0 이상에서 지원됩니다.

## Amazon EMR on EKS에 대한 Spark 운영자 시작하기

이 주제는 Spark 애플리케이션과 Schedule Spark 애플리케이션을 배포하여 Amazon EKS에서 Spark 운영자를 사용하기 시작하는 데 도움이 됩니다.

### Spark 운영자 설치

다음 단계를 사용하여 Apache Spark용 Kubernetes 운영자를 설치합니다.

1. 아직 실행하지 않았다면, [Amazon EMR on EKS에 대한 Spark 운영자 구성](#)의 단계를 완료합니다.
2. Helm 클라이언트를 Amazon ECR 레지스트리에 인증합니다. 다음 명령에서 *region-id* 값을 원하는 AWS 리전 및 [리전별 Amazon ECR 레지스트리 계정](#) 페이지에 있는 해당 리전에 대응하는 *ECR-registry-account* 값으로 바꿉니다.

```
aws ecr get-login-password \
--region region-id | helm registry login \
--username AWS \
--password-stdin ECR-registry-account.dkr.ecr.region-id.amazonaws.com
```

3. 다음 명령을 사용하여 Spark 운영자를 설치합니다.

차트 Helm `--version` 파라미터의 경우 `emr-` 접두사 및 날짜 접미사가 제거된 Amazon EMR 릴리스 레이블을 사용합니다. 예를 들어, `emr-6.12.0-java17-latest` 릴리스에서 `6.12.0-java17`을 지정합니다. 다음 명령의 예제에서는 `emr-7.13.0-latest` 릴리스를 사용하므로 차트 Helm `--version`에 대해 `7.13.0`을 지정합니다.

```
helm install spark-operator-demo \
oci://895885662937.dkr.ecr.region-id.amazonaws.com/spark-operator \
--set emrContainers.awsRegion=region-id \
--version 7.13.0 \
--namespace spark-operator \
--create-namespace
```

기본적으로 이 명령은 Spark 운영자에 대한 서비스 계정 `emr-containers-sa-spark-operator`를 생성합니다. 다른 서비스 계정을 사용하려면 `serviceAccounts.sparkoperator.name` 인수를 제공합니다. 예제:

```
--set serviceAccounts.sparkoperator.name my-service-account-for-spark-operator
```

[Spark 운영자에서 수직 자동 조정을 사용](#)하려면 운영자에게 웹훅을 허용하도록 설치 명령에 다음 줄을 추가합니다.

```
--set webhook.enable=true
```

4. `helm list` 명령을 사용하여 차트 Helm을 설치했는지 확인합니다.

```
helm list --namespace spark-operator -o yaml
```

`helm list` 명령은 새로 배포된 차트 Helm 릴리스 정보를 반환해야 합니다.

```
app_version: v1beta2-1.3.8-3.1.1
chart: spark-operator-7.13.0
name: spark-operator-demo
namespace: spark-operator
revision: "1"
status: deployed
updated: 2023-03-14 18:20:02.721638196 +0000 UTC
```

5. 필요한 추가 옵션을 사용하여 설치를 완료합니다. 자세한 내용은 GitHub의 [spark-on-k8s-operator](#) 설명서를 참조하세요.

## Spark 애플리케이션 실행

Spark 운영자는 Amazon EMR 6.10.0 이상에서 지원됩니다. Spark 운영자를 설치하면 기본적으로 Spark 애플리케이션을 실행하기 위한 서비스 계정 `emr-containers-sa-spark`가 생성됩니다. 다음 단계를 사용하여 Amazon EMR on EKS 6.10.0 이상에서 Spark 운영자를 사용하여 Spark 애플리케이션을 실행합니다.

1. Spark 운영자와 함께 Spark 애플리케이션을 실행하려면 먼저 [Amazon EMR on EKS에 대한 Spark 운영자 구성](#) 및 [Spark 운영자 설치](#)의 단계를 완료합니다.
2. 다음 콘텐츠 예제가 포함된 SparkApplication 정의 파일 `spark-pi.yaml`을 생성합니다.

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
```

```
name: spark-pi
namespace: spark-operator
spec:
  type: Scala
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.3.1"
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.3.1
    serviceAccount: emr-containers-sa-spark
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.3.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
```

3. 이제, 다음 명령을 사용하여 Spark 애플리케이션을 제출합니다. 이렇게 하면 이름이 spark-pi인 SparkApplication 객체도 생성됩니다.

```
kubectl apply -f spark-pi.yaml
```

4. 다음 명령을 사용하여 SparkApplication 객체에 대한 이벤트를 확인합니다.

```
kubectl describe sparkapplication spark-pi --namespace spark-operator
```

Spark 운영자를 통해 Spark에 애플리케이션을 제출하는 방법에 대한 자세한 내용은 GitHub의 [spark-on-k8s-operator](#) 설명서에서 [Using a SparkApplication](#)를 참조하세요.

## 스토리지로 Amazon S3 사용

Amazon S3를 파일 스토리지 옵션으로 사용하려면 YAML 파일에 다음 구성을 추가합니다.

```
hadoopConf:
# EMRFS filesystem
  fs.s3.customAWSCredentialsProvider:
com.amazonaws.auth.WebIdentityTokenCredentialsProvider
  fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
  fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
  fs.s3.buffer.dir: /mnt/s3
  fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"
  mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
"2"
  mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
sparkConf:
# Required for EMR Runtime
  spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
  spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
  spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
  spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
```

Amazon EMR 릴리스 7.2.0 이상을 사용하는 경우 구성이 기본적으로 포함됩니다. 이 경우 Spark 애플리케이션 YAML 파일 `local://<file_path>` 대신 `s3://<bucket_name>/<file_path>`로 파일 경로를 설정할 수 있습니다.

그런 다음, Spark 애플리케이션을 정상적으로 제출합니다.

## Amazon EMR on EKS에 대한 Spark 운영자를 사용하여 수직 자동 조정 사용

Amazon EMR 7.0부터 Amazon EMR on EKS 수직 자동 조정을 사용하여 리소스 관리를 간소화할 수 있습니다. 이 기능은 Amazon EMR Spark 애플리케이션에 제공하는 워크로드 요구 사항에 맞게 메모리와 CPU 리소스를 자동으로 튜닝합니다. 자세한 내용은 [Amazon EMR Spark 작업에 수직 자동 조정 사용](#) 단원을 참조하십시오.

이 섹션에서는 수직 자동 조정을 사용하도록 Spark 운영자를 구성하는 방법을 설명합니다.

### 사전 조건

모니터링을 구성하기 전에 다음 설정 작업을 완료하세요.

- [Amazon EMR on EKS에 대한 Spark 운영자 구성](#)의 단계를 수행하세요.
- (선택 사항) 이미 이전 버전의 Spark 운영자를 설치한 경우 SparkApplication/ScheduledSparkApplication CRD를 삭제합니다.

```
kubectl delete crd sparkApplication
kubectl delete crd scheduledSparkApplication
```

- [Spark 운영자 설치](#)의 단계를 수행하세요. 3단계에서 운영자에서 웹훅을 허용하도록 설치 명령에 다음 줄을 추가합니다.

```
--set webhook.enable=true
```

- [Amazon EMR on EKS에서 수직 자동 조정 설정](#)의 단계를 수행하세요.
- Amazon S3 위치의 파일에 대한 액세스 권한을 부여합니다.

1. S3 권한이 있는 JobExecutionRole로 드라이버 및 연산자 서비스 계정에 주석을 지정합니다.

```
kubectl annotate serviceaccount -n spark-operator emr-containers-sa-spark
eks.amazonaws.com/role-arn=JobExecutionRole
kubectl annotate serviceaccount -n spark-operator emr-containers-sa-spark-
operator eks.amazonaws.com/role-arn=JobExecutionRole
```

2. 해당 네임스페이스에서 작업 실행 역할의 신뢰 정책을 업데이트합니다.

```
aws emr-containers update-role-trust-policy \
--cluster-name cluster \
--namespace ${Namespace}\
--role-name iam_role_name_for_job_execution
```

3. 작업 실행 역할의 IAM 역할 신뢰 정책을 편집하고 serviceaccount를 emr-containers-sa-spark-\*-\*-\*xxxx에서 emr-containers-sa-\*로 업데이트합니다.

```
{
  "Effect": "Allow",
  "Principal": {
    "Federated": "OIDC-provider"
  },
  "Action": "sts:AssumeRoleWithWebIdentity",
  "Condition": {
    "StringLike": {
      "OIDC": "system:serviceaccount:${Namespace}:emr-containers-sa-*"
    }
  }
}
```

4. Amazon S3를 파일 스토리지로 사용하는 경우 yaml 파일에 다음 기본값을 추가합니다.

```
hadoopConf:
# EMRFS filesystem
  fs.s3.customAWSCredentialsProvider:
    com.amazonaws.auth.WebIdentityTokenCredentialsProvider
  fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
  fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
  fs.s3.buffer.dir: /mnt/s3
  fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"

  mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
    "2"
  mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
sparkConf:
  # Required for EMR Runtime
  spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/
aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
```

```
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*
spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/
lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native
spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*
spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-
lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native
```

Spark 운영자에서 수직 자동 조정 기능을 사용하여 작업을 실행합니다.

Spark 운영자와 함께 Spark 애플리케이션을 실행하려면 먼저 [사전 조건](#)의 단계를 완료해야 합니다.

Spark 운영자와 함께 수직 자동 조정을 사용하려면 Spark 애플리케이션 사양의 드라이버에 다음 구성을 추가하여 수직 자동 조정을 켭니다.

```
dynamicSizing:
  mode: Off
  signature: "my-signature"
```

이 구성은 수직 자동 크기 조정을 활성화하며, 작업에 대한 서명을 선택할 수 있는 필수 서명 구성입니다.

구성 및 파라미터 값에 대한 자세한 내용은 [Amazon EMR on EKS에 대한 수직 자동 조정 구성](#)을 참조하세요. 기본적으로 작업은 수직 자동 조정의 모니터링 전용 꺼짐 모드로 제출됩니다. 이 모니터링 상태를 사용하면 자동 조정을 수행하지 않고도 리소스 권장 사항을 계산하고 볼 수 있습니다. 자세한 내용은 [수직 오토 스케일링 모드](#)를 참조하세요.

다음 예제에서는 수직 자동 조정을 사용하는 데 필요한 구성이 포함된 SparkApplication 정의 파일 spark-pi.yaml을 보여줍니다.

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
```

```
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: spark-operator
spec:
  type: Scala
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-7.13.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.4.1"
  dynamicSizing:
    mode: Off
    signature: "my-signature"
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.4.1
    serviceAccount: emr-containers-sa-spark
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.4.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
```



이제, 다음 명령을 사용하여 Spark 애플리케이션을 제출합니다. 이렇게 하면 이름이 `spark-pi`인 `SparkApplication` 객체도 생성됩니다.

```
kubectl apply -f spark-pi.yaml
```

Spark 운영자를 통해 Spark에 애플리케이션을 제출하는 방법에 대한 자세한 내용은 GitHub의 `spark-on-k8s-operator` 설명서에서 [Using a SparkApplication](#)를 참조하세요.

## 수직 자동 조정 기능 확인

제출된 작업에 대해 수직 자동 조정이 올바르게 작동하는지 확인하려면 `kubectl`을 사용하여 `verticalpodautoscaler` 사용자 지정 리소스를 가져오고 조정 권장 사항을 확인합니다.

```
kubectl get verticalpodautoscalers --all-namespaces \
-l=emr-containers.amazonaws.com/dynamic.sizing.signature=my-signature
```

이 쿼리에 대한 출력은 다음과 비슷합니다.

| NAMESPACE      | NAME                                                        | MODE |
|----------------|-------------------------------------------------------------|------|
| CPU MEM        | PROVIDED AGE                                                |      |
| spark-operator | ds-p73j6mkosvc4xeb3gr7x4xol2bfcw5evqimzqojrlysvj3giozuq-vpa | Off  |
| 580026651      | True 15m                                                    |      |

출력이 비슷하지 않거나 오류 코드를 포함하는 경우 문제 해결을 위한 단계는 [Amazon EMR on EKS 수직 자동 조정 문제 해결](#) 섹션을 참조하세요.

포드 및 애플리케이션을 제거하려면 다음 명령을 실행합니다.

```
kubectl delete sparkapplication spark-pi
```

## Amazon EMR on EKS에 대한 Spark 운영자 제거

다음 단계를 사용하여 Spark 운영자를 제거합니다.

- 올바른 네임스페이스를 사용하여 Spark 운영자를 삭제합니다. 이 예제에서 네임스페이스는 `spark-operator-demo`입니다.

```
helm uninstall spark-operator-demo -n spark-operator
```

- Spark 운영자 서비스 계정을 삭제합니다.

```
kubectl delete sa emr-containers-sa-spark-operator -n spark-operator
```

3. Spark 운영자 CustomResourceDefinitions(CRD)를 삭제합니다.

```
kubectl delete crd sparkapplications.sparkoperator.k8s.io
kubectl delete crd scheduledsparkapplications.sparkoperator.k8s.io
```

## 모니터링 구성을 사용하여 Spark Kubernetes 운영자 및 Spark 작업 모니터링을 참조하세요.

모니터링 구성을 사용하면 Spark 애플리케이션 및 운영자 로그의 로그 아카이빙을 Amazon S3 또는 Amazon CloudWatch에 쉽게 설정할 수 있습니다. 사용자는 하나 또는 둘 다를 선택할 수 있습니다. 이 작업을 수행하면 Spark 운영자 포드, 드라이버 및 실행기 포드에 로그 에이전트 사이드카가 추가되고 이후에 이러한 구성 요소의 로그가 구성된 싱크에 전달됩니다.

### 사전 조건

모니터링을 구성하기 전에 다음 설정 작업을 완료하세요.

1. (선택 사항) 이미 이전 버전의 Spark 운영자를 설치한 경우 SparkApplication/ScheduledSparkApplication CRD를 삭제합니다.

```
kubectl delete crd scheduledsparkapplications.sparkoperator.k8s.io
kubectl delete crd sparkapplications.sparkoperator.k8s.io
```

2. IAM에 운영자/작업 실행 역할이 아직 없는 경우 생성합니다.
3. 다음 명령을 실행하여 방금 생성한 운영자/작업 실행 역할의 신뢰 정책을 업데이트합니다.

```
aws emr-containers update-role-trust-policy \
--cluster-name cluster \
--namespace namespace \
--role-name iam_role_name_for_operator/job_execution_role
```

4. 운영자/작업 실행 역할의 IAM 역할 신뢰 정책을 다음과 같이 편집합니다.

```
{
  "Effect": "Allow",
  "Principal": {
```

```

    "Federated": "${OIDC-provider}"
  },
  "Action": "sts:AssumeRoleWithWebIdentity",
  "Condition": {
    "StringLike": {
      "OIDC_PROVIDER:sub": "system:serviceaccount:${Namespace}:emr-
containers-sa-*"
    }
  }
}

```

5. 다음 권한을 사용하여 IAM에서 monitoringConfiguration 정책을 생성합니다.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogStreams",
        "logs:CreateLogStream",
        "logs:CreateLogGroup",
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:log_group_name",
        "arn:aws:logs:*:*:log-group:log_group_name:"
      ],
      "Sid": "AllowLOGSDescribeLogStreams"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "*"
      ],
      "Sid": "AllowLOGSDescribeLogGroups"
    }
  ]
}

```

```

    "Effect": "Allow",
    "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket"
    ],
    "Resource": [
        "arn:aws:s3:::bucket_name",
        "arn:aws:s3:::bucket_name/*"
    ],
    "Sid": "AllowS3Putobject"
  }
]
}

```

6. 위의 정책을 운영자/작업 실행 역할에 첨부하세요.

## Spark 운영자 로그

helm install을 수행할 때 다음과 같은 방법으로 모니터링 구성을 정의할 수 있습니다.

```

helm install spark-operator spark-operator \
--namespace namespace \
--set emrContainers.awsRegion=aws_region \
--set emrContainers.monitoringConfiguration.image=log_agent_image_url \
--set
  emrContainers.monitoringConfiguration.s3MonitoringConfiguration.logUri=S3_bucket_uri \
--set
  emrContainers.monitoringConfiguration.cloudWatchMonitoringConfiguration.logGroupName=log_group
\
--set
  emrContainers.monitoringConfiguration.cloudWatchMonitoringConfiguration.logStreamNamePrefix=log_stream_name_prefix
\
--set emrContainers.monitoringConfiguration.sideCarResources.limits.cpuLimit=500m \
--set emrContainers.monitoringConfiguration.sideCarResources.limits.memoryLimit=512Mi \
--set
  emrContainers.monitoringConfiguration.containerLogRotationConfiguration.rotationSize=2GB
\
--set
  emrContainers.monitoringConfiguration.containerLogRotationConfiguration.maxFilesToKeep=10
\
--set webhook.enable=true \

```

```
--set emrContainers.operatorExecutionRoleArn=operator_execution_role_arn
```

## 모니터링 구성

monitoringConfiguration 아래에서 다음과 같은 구성 옵션을 사용할 수 있습니다.

- 이미지(선택 사항) - 에이전트 이미지 URL을 기록합니다. 제공되지 않으면 emrReleaseLabel로 가져옵니다.
- s3MonitoringConfiguration – Amazon S3에 아카이브하려면 이 옵션을 설정합니다.
  - logUri(필수) - 로그를 저장할 Amazon S3 버킷 경로입니다.
  - 다음은 로그가 업로드된 후 Amazon S3 버킷 경로의 샘플 형식입니다. 첫 번째 예제에서는 로그 순환이 활성화되지 않았음을 보여줍니다.

```
s3://${logUri}/${POD_NAME}/operator/stdout.gz
s3://${logUri}/${POD_NAME}/operator/stderr.gz
```

로그 순환은 기본적으로 활성화됩니다. 증분 인덱스가 있는 순환된 파일과 이전 샘플과 동일한 현재 파일을 모두 볼 수 있습니다.

```
s3://${logUri}/${POD_NAME}/operator/stdout_YYYYMMDD_index.gz
s3://${logUri}/${POD_NAME}/operator/stderr_YYYYMMDD_index.gz
```

- cloudWatchMonitoringConfiguration – Amazon CloudWatch로의 전달을 설정하기 위한 구성 키입니다.
  - logGroupName(필수) - Amazon CloudWatch 로그를 전송할 로그 그룹의 이름입니다. 그룹이 없으면 그룹이 자동으로 생성됩니다.
  - logStreamNamePrefix(선택 사항) - 로그를 보낼 로그 스트림의 이름입니다. 기본값은 빈 문자열입니다. 이 형식 Amazon CloudWatch 은 다음과 같습니다.

```
${logStreamNamePrefix}/${POD_NAME}/STDOUT or STDERR
```

- sideCarResources(선택 사항) - 시작된 Fluentbit sidecar 컨테이너에서 리소스 한도를 설정하기 위한 구성 키입니다.
  - memoryLimit(선택 사항) - 메모리 제한입니다. 필요에 따라 조정합니다. 기본값은 512Mi입니다.
  - cpuLimit(선택 사항) - CPU 제한입니다. 필요에 따라 조정합니다. 기본값은 500m입니다.
- containerLogRotationConfiguration(선택 사항) - 컨테이너 로그 로테이션 동작을 제어합니다. 기본적으로 활성화됩니다.

- rotationSize(필수) - 로그 로테이션을 위한 파일 크기를 지정합니다. 가능한 값 범위는 2KB에서 2GB 사이입니다. rotationSize 파라미터의 숫자 단위 부분은 정수로 전달됩니다. 십진수는 지원되지 않으므로 로테이션 크기를 1.5GB(예: 1,500MB 값)로 지정할 수 있습니다. 기본값은 2GB입니다.
- maxFilesToKeep(필수) - 로테이션을 수행한 후 컨테이너에서 보존할 최대 파일 수를 지정합니다. 최솟값은 1이고 최댓값은 50입니다. 기본값은 10입니다.

monitoringConfiguration을 구성한 후에는 Amazon S3 버킷 Amazon CloudWatch 또는 둘 다에서 Spark 운영자 포드 로그를 확인할 수 있어야 합니다. Amazon S3 버킷의 경우 첫 번째 로그 파일이 풀러시될 때까지 2분 정도 기다려야 합니다.

로그인을 찾으려면 CloudWatch > 로그 그룹 > **## ## ##** > **## ##**/작업자/stderr로 이동할 Amazon CloudWatch 수 있습니다.

또는 CloudWatch > 로그 그룹 > **## ## ##** > **## ##**/작업자/stdout으로 이동하면 됩니다.

## Spark 애플리케이션 로그

다음과 같은 방법으로 이 구성을 정의할 수 있습니다.

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: namespace
spec:
  type: Scala
  mode: cluster
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
  sparkVersion: "3.3.1"
  emrReleaseLabel: emr_release_label
  executionRoleArn: job_execution_role_arn
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
```

```

driver:
  cores: 1
  coreLimit: "1200m"
  memory: "512m"
  labels:
    version: 3.3.1
  volumeMounts:
    - name: "test-volume"
      mountPath: "/tmp"
executor:
  cores: 1
  instances: 1
  memory: "512m"
  labels:
    version: 3.3.1
  volumeMounts:
    - name: "test-volume"
      mountPath: "/tmp"
monitoringConfiguration:
  image: "log_agent_image"
  s3MonitoringConfiguration:
    logUri: "S3_bucket_uri"
  cloudWatchMonitoringConfiguration:
    logGroupName: "log_group_name"
    logStreamNamePrefix: "log_stream_prefix"
  sideCarResources:
    limits:
      cpuLimit: "500m"
      memoryLimit: "250Mi"
  containerLogRotationConfiguration:
    rotationSize: "2GB"
    maxFilesToKeep: "10"

```

monitoringConfiguration 아래에서 다음과 같은 구성 옵션을 사용할 수 있습니다.

- 이미지(선택 사항) - 에이전트 이미지 URL을 기록합니다. 제공되지 않으면 emrReleaseLabel로 가져옵니다.
- s3MonitoringConfiguration – Amazon S3에 아카이브하려면 이 옵션을 설정합니다.
- logUri(필수) - 로그를 저장할 Amazon S3 버킷 경로입니다. 첫 번째 예제에서는 로그 순환이 활성화되지 않았음을 보여줍니다.

```
s3://${logUri}/${APPLICATION_NAME}-${APPLICATION UID}/${POD NAME}/stdout.gz
```

```
s3://${logUri}/${APPLICATION_NAME}-${APPLICATION_UID}/${POD_NAME}/stderr.gz
```

로그 순환은 기본적으로 활성화됩니다. 로테이션된 파일(인덱스 증가)과 현재 파일(날짜 스탬프 없음)을 모두 사용할 수 있습니다.

```
s3://${logUri}/${APPLICATION_NAME}-${APPLICATION_UID}/${POD_NAME}/
stdout_YYYYMMDD_index.gz
s3://${logUri}/${APPLICATION_NAME}-${APPLICATION_UID}/${POD_NAME}/
stderr_YYYYMMDD_index.gz
```

- `cloudWatchMonitoringConfiguration` – Amazon CloudWatch로의 전달을 설정하기 위한 구성 키입니다.
- `logGroupName`(필수) - 로그를 보낼 CloudWatch 로그 그룹의 이름. 그룹이 없으면 그룹이 자동으로 생성됩니다.
- `logStreamNamePrefix`(선택 사항) - 로그를 보낼 로그 스트림의 이름. 기본값은 빈 문자열입니다. CloudWatch의 형식은 다음과 같습니다.

```
${logStreamNamePrefix}/${APPLICATION_NAME}-${APPLICATION_UID}/${POD_NAME}/stdout
${logStreamNamePrefix}/${APPLICATION_NAME}-${APPLICATION_UID}/${POD_NAME}/stderr
```

- `sideCarResources`(선택 사항) - 시작된 Fluentbit sidecar 컨테이너에서 리소스 한도를 설정하기 위한 구성 키입니다.
- `memoryLimit`(선택 사항) - 메모리 제한입니다. 필요에 따라 조정합니다. 기본값은 250Mi입니다.
- `cpuLimit` - CPU 제한입니다. 필요에 따라 조정합니다. 기본값은 500m입니다.
- `containerLogRotationConfiguration`(선택 사항) - 컨테이너 로그 로테이션 동작을 제어합니다. 기본적으로 활성화됩니다.
- `rotationSize`(필수) - 로그 로테이션을 위한 파일 크기를 지정합니다. 가능한 값 범위는 2KB에서 2GB 사이입니다. `rotationSize` 파라미터의 숫자 단위 부분은 정수로 전달됩니다. 십진수는 지원하지 않으므로 로테이션 크기를 1.5GB(예: 1,500MB 값)로 지정할 수 있습니다. 기본값은 2GB입니다.
- `maxFilesToKeep`(필수) - 로테이션을 수행한 후 컨테이너에서 보존할 최대 파일 수를 지정합니다. 최소값은 1입니다. 최대값은 50입니다. 기본값은 10입니다.

`monitoringConfiguration`을 구성한 후 Amazon S3 버킷이나 CloudWatch 또는 둘 다에서 Spark 애플리케이션 드라이버와 실행기 로그를 확인할 수 있습니다. Amazon S3 버킷의 경우 첫 번째 로그 파일이



풀러시될 때까지 2분 정도 기다려야 합니다. 예를 들어, Amazon S3에서 버킷 경로는 다음과 같이 나타납니다.

Amazon S3 > 버킷 > **## ##** > **Spark ##### ## - UUID** > **## ##** > stderr.gz

또는 다음과 같습니다.

Amazon S3 > 버킷 > **## ##** > **Spark ##### ## - UUID** > **## ##** > stdout.gz

CloudWatch에서 경로는 다음과 같습니다.

CloudWatch > 로그 그룹 > **## ## ##** > **Spark ##### ## - UUID/## ##**/stderr

또는 다음과 같습니다.

CloudWatch > 로그 그룹 > **## ## ##** > **Spark ##### ## - UUID/## ##**/stdout

## Amazon EMR on EKS에서 보안 및 Spark 운영자

Spark 운영자를 사용하는 경우 클러스터 액세스 권한을 설정하는 몇 가지 방법이 있습니다. 첫 번째에서는 역할 기반 액세스 제어를 사용합니다. 역할 기반 액세스 제어(RBAC)는 조직 내 개인의 역할에 따라 액세스를 제한합니다. 액세스를 처리하는 기본적인 방법이 되었습니다. 두 번째 액세스 방법은 할당된 특정 권한을 통해 리소스 액세스를 제공하는 AWS Identity and Access Management 역할을 수임하는 것입니다.

### 주제

- [역할 기반 액세스 제어\(RBAC\)를 사용하여 클러스터 액세스 권한 설정](#)
- [서비스 계정에 대한 IAM 역할\(IRSA\)을 사용하여 클러스터 액세스 권한 설정](#)

## 역할 기반 액세스 제어(RBAC)를 사용하여 클러스터 액세스 권한 설정

Spark 운영자를 배포하기 위해 Amazon EMR on EKS는 Spark 운영자와 Spark 앱을 위한 두 가지 역할 및 서비스 계정을 생성합니다.

### 주제

- [운영자 서비스 계정 및 역할](#)
- [Spark 서비스 계정 및 역할](#)

## 운영자 서비스 계정 및 역할

Amazon EMR on EKS는 Spark 작업 및 기타 리소스(예: 서비스)와 관련된 SparkApplications를 관리할 운영자 서비스 계정 및 역할을 생성합니다.

이 서비스 계정의 기본 이름은 `emr-containers-sa-spark-operator`입니다.

이 서비스 역할에는 다음 규칙이 적용됩니다.

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - services
  - configmaps
  - secrets
  verbs:
  - create
  - get
  - delete
  - update
- apiGroups:
  - extensions
  - networking.k8s.io
  resources:
  - ingresses
  verbs:
  - create
  - get
  - delete
- apiGroups:
  - ""
  resources:
  - nodes
  verbs:
  - get
- apiGroups:
  - ""
```

```
resources:
- events
verbs:
- create
- update
- patch
- apiGroups:
- ""
resources:
- resourcequotas
verbs:
- get
- list
- watch
- apiGroups:
- apiextensions.k8s.io
resources:
- customresourcedefinitions
verbs:
- create
- get
- update
- delete
- apiGroups:
- admissionregistration.k8s.io
resources:
- mutatingwebhookconfigurations
- validatingwebhookconfigurations
verbs:
- create
- get
- update
- delete
- apiGroups:
- sparkoperator.k8s.io
resources:
- sparkapplications
- sparkapplications/status
- scheduledsparkapplications
- scheduledsparkapplications/status
verbs:
- "*"
{{- if .Values.batchScheduler.enable }}
# required for the `volcano` batch scheduler
```

```

- apiGroups:
  - scheduling.incubator.k8s.io
  - scheduling.sigs.dev
  - scheduling.volcano.sh
resources:
- podgroups
verbs:
- "*"
{{- end }}
{{ if .Values.webhook.enable }}
- apiGroups:
  - batch
resources:
- jobs
verbs:
- delete
{{- end }}

```

## Spark 서비스 계정 및 역할

Spark 드라이버 포드에는 포드와 동일한 네임스페이스에 있는 Kubernetes 서비스 계정이 필요합니다. 이 서비스 계정에는 실행기 포드의 생성, 가져오기, 나열, 패치, 삭제 권한과 드라이버용 Kubernetes 헤드리스 서비스의 생성 권한이 필요합니다. 포드 네임스페이스의 기본 서비스 계정에 필요한 권한이 없으면 드라이버가 실패하고 서비스 계정 없이 종료됩니다.

이 서비스 계정의 기본 이름은 `emr-containers-sa-spark`입니다.

이 서비스 역할에는 다음 규칙이 적용됩니다.

```

rules:
- apiGroups:
  - ""
resources:
- pods
verbs:
- "*"
- apiGroups:
  - ""
resources:
- services
verbs:
- "*"
- apiGroups:

```

```

- ""
resources:
- configmaps
verbs:
- "*"
- apiGroups:
- ""
resources:
- persistentvolumeclaims
verbs:
- "*"

```

## 서비스 계정에 대한 IAM 역할(IRSA)을 사용하여 클러스터 액세스 권한 설정

이 섹션에서는 예를 사용하여 AWS Identity and Access Management 역할을 수임하도록 Kubernetes 서비스 계정을 구성하는 방법을 보여줍니다. 그런 다음 서비스 계정을 사용하는 포드는 역할에 액세스 권한이 있는 모든 AWS 서비스에 액세스할 수 있습니다.

다음 예제에서는 Spark 애플리케이션을 실행하여 Amazon S3에 있는 파일의 단어 수를 계산합니다. 이를 위해 서비스 계정에 대한 IAM 역할(IRSA)을 설정하여 Kubernetes 서비스 계정을 인증하고 권한을 부여할 수 있습니다.

### Note

이 예제에서는 Spark 운영자의 'spark-operator' 네임스페이스와 Spark 애플리케이션을 제출하는 네임스페이스를 사용합니다.

## 사전 조건

이 페이지의 예제를 사용하기 전에 다음 필수 조건을 완료합니다.

- [Spark 운영자를 설정합니다.](#)
- [Spark 운영자 설치.](#)
- [Amazon S3 버킷을 생성](#)합니다.
- 좋아하는 시를 poem.txt 텍스트 파일에 저장하고 파일을 S3 버킷에 업로드합니다. 이 페이지에서 생성한 Spark 애플리케이션이 텍스트 파일의 내용을 읽습니다. S3에 파일을 업로드하는 방법에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서에서 [버킷에 객체 업로드](#)를 참조하세요.

## IAM 역할을 수임하도록 Kubernetes 서비스 계정 구성

다음 단계를 사용하여 포드가 액세스 권한이 있는 서비스에 액세스하는 데 사용할 수 있는 IAM 역할을 수임하도록 Kubernetes AWS 서비스 계정을 구성합니다.

1. 를 완료한 후 [사전 조건](#) AWS Command Line Interface 를 사용하여 Amazon S3에 업로드한 example-policy.json 파일에 대한 읽기 전용 액세스를 허용하는 파일을 생성합니다.

```
cat >example-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::my-pod-bucket",
        "arn:aws:s3:::my-pod-bucket/*"
      ]
    }
  ]
}
EOF
```

2. IAM 정책 example-policy를 생성합니다.

```
aws iam create-policy --policy-name example-policy --policy-document file://
example-policy.json
```

3. 다음으로 IAM 역할 example-role을 생성하고 이를 Spark 드라이버의 Kubernetes 서비스 계정과 연결합니다.

```
eksctl create iamserviceaccount --name driver-account-sa --namespace spark-operator \
--cluster my-cluster --role-name "example-role" \
--attach-policy-arn arn:aws:iam::111122223333:policy/example-policy --approve
```

4. Spark 드라이버 서비스 계정에 필요한 클러스터 역할 바인딩이 포함된 yaml 파일을 생성합니다.

```
cat >spark-rbac.yaml <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: driver-account-sa
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: spark-role
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: edit
subjects:
- kind: ServiceAccount
  name: driver-account-sa
  namespace: spark-operator
EOF
```

## 5. 클러스터 역할 바인딩 구성을 적용합니다.

```
kubectl apply -f spark-rbac.yaml
```

kubectl 명령은 성공적인 계정 생성을 확인합니다.

```
serviceaccount/driver-account-sa created
clusterrolebinding.rbac.authorization.k8s.io/spark-role configured
```

## Spark 운영자에서 애플리케이션 실행

[Kubernetes 서비스 계정 구성](#) 후 [사전 조건](#)의 일부로 업로드한 텍스트 파일의 단어 수를 계산하는 Spark 애플리케이션을 실행할 수 있습니다.

1. Amazon EMR 버전 6을 기반으로 단어 수 계산 애플리케이션을 위한 SparkApplication 정의를 사용하여 새 파일 word-count.yaml을 만듭니다.

```
cat >word-count.yaml <<EOF
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
```

```

metadata:
  name: word-count
  namespace: spark-operator
spec:
  type: Java
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.JavaWordCount
  mainApplicationFile: local:///usr/lib/spark/examples/jars/spark-examples.jar
  arguments:
    - s3://my-pod-bucket/poem.txt
  hadoopConf:
    # EMRFS filesystem
    fs.s3.customAWSCredentialsProvider:
com.amazonaws.auth.WebIdentityTokenCredentialsProvider
    fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
    fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
    fs.s3.buffer.dir: /mnt/s3
    fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"

  mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
"2"
    mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
  sparkConf:
    # Required for EMR Runtime
    spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*
    spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/
lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
    spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*

```



```

    spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-
lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native
    sparkVersion: "3.3.1"
    restartPolicy:
      type: Never
    driver:
      cores: 1
      coreLimit: "1200m"
      memory: "512m"
      labels:
        version: 3.3.1
      serviceAccount: my-spark-driver-sa
    executor:
      cores: 1
      instances: 1
      memory: "512m"
      labels:
        version: 3.3.1
EOF

```

버전 7 릴리스에서 Spark 운영자를 사용하는 경우 구성 값 중 일부를 조정합니다.

```

cat >word-count.yaml <<EOF
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: word-count
  namespace: spark-operator
spec:
  type: Java
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-7.7.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.JavaWordCount
  mainApplicationFile: local:///usr/lib/spark/examples/jars/spark-examples.jar
  arguments:
    - s3://my-pod-bucket/poem.txt
  hadoopConf:
    # EMRFS filesystem
    fs.s3.customAWSCredentialsProvider:
com.amazonaws.auth.WebIdentityTokenCredentialsProvider
    fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem

```

```

fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
fs.s3.buffer.dir: /mnt/s3
fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"

mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
"2"
mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
sparkConf:
  # Required for EMR Runtime
  spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/aws-java-sdk-v2/*:/usr/share/
aws/emr/emrfs/conf:/usr/share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/
*/usr/share/aws/emr/security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/
aws/hmclient/lib/aws-glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-
Serde/hive-openx-serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-
sdk.jar:/home/hadoop/extrajars/*
  spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/
lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
  spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/aws-java-sdk-v2/*:/usr/
share/aws/emr/emrfs/conf:/usr/share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/
auxlib/*:/usr/share/aws/emr/security/conf:/usr/share/aws/emr/security/lib/*:/usr/
share/aws/hmclient/lib/aws-glue-datacatalog-spark-client.jar:/usr/share/java/Hive-
JSON-Serde/hive-openx-serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-
spark-sdk.jar:/home/hadoop/extrajars/*
  spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-
lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native
sparkVersion: "3.3.1"
restartPolicy:
  type: Never
driver:
  cores: 1
  coreLimit: "1200m"
  memory: "512m"
  labels:
    version: 3.3.1
  serviceAccount: my-spark-driver-sa
executor:
  cores: 1
  instances: 1
  memory: "512m"
  labels:

```

```
version: 3.3.1
EOF
```

## 2. Spark 애플리케이션을 제출합니다.

```
kubectl apply -f word-count.yaml
```

kubectl 명령은 word-count라고 하는 SparkApplication 객체를 성공적으로 생성했다는 확인을 반환합니다.

```
sparkapplication.sparkoperator.k8s.io/word-count configured
```

## 3. SparkApplication 객체에 대한 이벤트를 확인하려면 다음 명령을 실행합니다.

```
kubectl describe sparkapplication word-count -n spark-operator
```

kubectl 명령은 이벤트와 함께 SparkApplication에 대한 설명을 반환합니다.

```
Events:
  Type            Reason                                     Age           From
  Message
  ----
  Normal          SparkApplicationSpecUpdateProcessed      3m2s (x2 over 17h)    spark-operator
  Successfully processed spec update for SparkApplication word-count
  Warning         SparkApplicationPendingRerun             3m2s (x2 over 17h)    spark-operator
  SparkApplication word-count is pending rerun
  Normal          SparkApplicationSubmitted                2m58s (x2 over 17h)    spark-operator
  SparkApplication word-count was submitted successfully
  Normal          SparkDriverRunning                      2m56s (x2 over 17h)    spark-operator
  Driver word-count-driver is running
  Normal          SparkExecutorPending                    2m50s              spark-operator
  Executor [javawordcount-fdd1698807392c66-exec-1] is pending
  Normal          SparkExecutorRunning                    2m48s              spark-operator
  Executor [javawordcount-fdd1698807392c66-exec-1] is running
  Normal          SparkDriverCompleted                    2m31s (x2 over 17h)    spark-operator
  Driver word-count-driver completed
  Normal          SparkApplicationCompleted                2m31s (x2 over 17h)    spark-operator
  SparkApplication word-count completed
  Normal          SparkExecutorCompleted                    2m31s (x2 over 2m31s)  spark-operator
  Executor [javawordcount-fdd1698807392c66-exec-1] completed
```

이제 애플리케이션이 S3 파일에 있는 단어 수를 계산합니다. 단어 수를 확인하려면 드라이버의 로그 파일을 참조하세요.

```
kubectl logs pod/word-count-driver -n spark-operator
```

kubectl 명령은 word-count 애플리케이션의 결과와 함께 로그 파일의 콘텐츠를 반환해야 합니다.

```
INFO DAGScheduler: Job 0 finished: collect at JavaWordCount.java:53, took 5.146519 s
Software: 1
```

Spark 운영자를 통해 Spark에 애플리케이션을 제출하는 방법에 대한 자세한 내용은 GitHub의 Kubernetes Operator for Apache Spark (spark-on-k8s-operator) 설명서에서 [Using a SparkApplication](#)을 참조하세요.

## spark-submit을 사용하여 Spark 작업 실행

Amazon EMR 릴리스 6.10.0 이상에서는 Spark 애플리케이션을 Amazon EMR on EKS 클러스터에 제출하고 해당 애플리케이션을 실행하는 데 사용할 수 있는 명령줄 도구로 spark-submit을 지원합니다.

### Note

Amazon EMR은 vCPU 및 메모리 소비를 기반으로 Amazon EKS에 대한 요금을 계산합니다. 이 계산은 드라이버 및 실행기 포드에 적용됩니다. 이 계산은 Amazon EMR 애플리케이션 이미지를 다운로드한 시점부터 시작하여 Amazon EKS 포드가 종료될 때까지 수행되며, 가장 가까운 초로 반올림됩니다.

### 주제

- [Amazon EMR on EKS에서 spark-submit 설정](#)
- [Amazon EMR on EKS에서 spark-submit 시작하기](#)
- [spark-submit에 대한 Spark 드라이버 서비스 계정 보안 요구 사항 확인](#)

## Amazon EMR on EKS에서 spark-submit 설정

Amazon EMR on EKS에서 spark-submit을 사용하여 애플리케이션을 실행하기 전에 다음 작업을 완료합니다. Amazon Web Services(AWS)에 이미 가입했고 Amazon EKS를 사용하고 있는 경우 Amazon

EMR on EKS를 사용할 준비를 거의 마친 상태입니다. 필수 조건 중 하나를 이미 완료한 경우 해당 조건을 건너뛰고 다음 조건으로 넘어갈 수 있습니다.

- [의 최신 버전 설치 또는 업데이트 AWS CLI](#) -를 이미 설치한 경우 최신 버전이 있는지 AWS CLI 확인합니다.
- [kubectl 및 eksctl 설치](#) - eksctl은 Amazon EKS와 통신하는 데 사용하는 명령줄 도구입니다.
- [Amazon EKS 시작하기 – eksctl](#) - 관련 단계를 수행하여 Amazon EKS에서 노드를 포함하는 새 Kubernetes 클러스터를 생성합니다.
- [Amazon EMR 기본 이미지 URI 선택](#)(릴리스 6.10.0 이상) - spark-submit 명령은 Amazon EMR 릴리스 6.10.0 이상에서 지원됩니다.
- 드라이버 서비스 계정에 실행기 포드를 생성하고 감시할 수 있는 적절한 권한이 있는지 확인합니다. 자세한 내용은 [spark-submit에 대한 Spark 드라이버 서비스 계정 보안 요구 사항 확인](#) 단원을 참조하십시오.
- 로컬 [AWS 보안 인증 프로파일](#)을 설정합니다.
- Amazon EKS 콘솔에서 EKS 클러스터를 선택한 다음, 개요, 세부 정보, API 서버 엔드포인트 아래에 있는 EKS 클러스터 엔드포인트를 찾습니다.

## Amazon EMR on EKS에서 spark-submit 시작하기

Amazon EMR 6.10.0 이상에서는 Amazon EKS 클러스터에서 Spark 애플리케이션을 실행하기 위해 spark-submit을 지원합니다. 다음 섹션에서는 Spark 애플리케이션에 대한 명령을 제출하는 방법을 보여줍니다.

### Spark 애플리케이션 실행

Spark 애플리케이션을 실행하려면 다음 단계를 수행합니다.

1. spark-submit 명령으로 Spark 애플리케이션을 실행하려면 먼저 [Amazon EMR on EKS에서 spark-submit 설정](#)의 단계를 완료합니다.
2. Amazon EMR on EKS 기본 이미지를 사용하여 컨테이너를 실행합니다. 자세한 내용은 [기본 이미지 URI를 선택하는 방법](#)을 참조하세요.

```
kubectl run -it containerName --image=EMRonEKSIImage --command -n namespace /bin/
bash
```

3. 다음과 같은 환경 변수의 값을 설정합니다.

```
export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon EKS-cluster-endpoint
```

4. 이제 다음 명령을 사용하여 Spark 애플리케이션을 제출합니다.

```
$SPARK_HOME/bin/spark-submit \
--class org.apache.spark.examples.SparkPi \
--master $MASTER_URL \
--conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest \
--conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
--deploy-mode cluster \
--conf spark.kubernetes.namespace=spark-operator \
local:///usr/lib/spark/examples/jars/spark-examples.jar 20
```

Spark로 애플리케이션을 제출하는 방법에 대한 자세한 내용은 Apache Spark 설명서에서 [Submitting applications](#)를 참조하세요.

#### Important

spark-submit에서는 제출 메커니즘으로 클러스터 모드만 지원합니다.

## spark-submit에 대한 Spark 드라이버 서비스 계정 보안 요구 사항 확인

Spark 드라이버 포드는 Kubernetes 서비스 계정을 사용함으로써 Kubernetes API 서버에 액세스하여 실행기 포드를 생성하고 감시합니다. 드라이버 서비스 계정에 클러스터의 포드를 나열, 생성, 편집, 패치 및 삭제할 수 있는 적절한 권한이 있어야 합니다. 다음 명령을 실행하여 이러한 리소스를 나열할 수 있는지 확인할 수 있습니다.

```
kubectl auth can-i list/create/edit/delete/patch pods
```

각 명령을 실행하여 필요한 권한이 있는지 확인합니다.

```
kubectl auth can-i list pods
kubectl auth can-i create pods
kubectl auth can-i edit pods
kubectl auth can-i delete pods
```

```
kubectl auth can-i patch pods
```

이 서비스 역할에는 다음 규칙이 적용됩니다.

```
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - "*"

```

## spark-submit과 관련된 서비스 계정에 대한 IAM 역할(IRSA) 설정

다음 섹션에서는 서비스 계정에 대한 IAM 역할(IRSA)을 설정하여 Amazon S3에 저장된 Spark 애플리케이션을 실행할 수 있도록 Kubernetes 서비스 계정을 인증하고 권한을 부여하는 방법을 설명합니다.

### 사전 조건

이 설명서의 예제 중 하나를 시도하기 전에 다음 사전 조건을 완료했는지 확인합니다.

- [spark-submit 설정 완료됨](#)
- [S3 버킷을 생성](#)하고 Spark 애플리케이션 jar을 [업로드](#)함

## IAM 역할을 수임하도록 Kubernetes 서비스 계정 구성

다음 단계에서는 AWS Identity and Access Management (IAM) 역할을 수임하도록 Kubernetes 서비스 계정을 구성하는 방법을 다룹니다. 서비스 계정을 사용하도록 포드를 구성한 후 해당 포드는 역할에 액세스 권한이 AWS 서비스 있는 모든에 액세스할 수 있습니다.

1. [업로드](#)한 Amazon S3 객체에 대한 읽기 전용 액세스를 허용하는 정책 파일을 생성합니다.

```
cat >my-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::<my-spark-jar-bucket>",
        "arn:aws:s3:::<my-spark-jar-bucket>/*"
      ]
    }
  ]
}
EOF
```

2. IAM 정책을 생성합니다.

```
aws iam create-policy --policy-name my-policy --policy-document file://my-policy.json
```

3. IAM 역할 생성 및 Spark 드라이버의 Kubernetes 서비스 계정에 연결

```
eksctl create iamserviceaccount --name my-spark-driver-sa --namespace spark-operator \
--cluster my-cluster --role-name "my-role" \
--attach-policy-arn arn:aws:iam::111122223333:policy/my-policy --approve
```

4. Spark 드라이버 서비스 계정에 필요한 [권한](#)이 있는 YAML 파일을 생성합니다.

```
cat >spark-rbac.yaml <<EOF
apiVersion: rbac.authorization.k8s.io/v1
```



```
kind: Role
metadata:
  namespace: default
  name: emr-containers-role-spark
rules:
- apiGroups:
  - ""
  resources:
  - pods
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - services
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - configmaps
  verbs:
  - "*"
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - "*"
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: spark-role-binding
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: emr-containers-role-spark
subjects:
- kind: ServiceAccount
  name: emr-containers-sa-spark
  namespace: default
```

EOF

5. 클러스터 역할 바인딩 구성을 적용합니다.

```
kubectl apply -f spark-rbac.yaml
```

6. kubectl 명령은 생성된 계정의 확인을 반환해야 합니다.

```
serviceaccount/emr-containers-sa-spark created
clusterrolebinding.rbac.authorization.k8s.io/emr-containers-role-spark configured
```

## Spark 애플리케이션 실행

Amazon EMR 6.10.0 이상에서는 Amazon EKS 클러스터에서 Spark 애플리케이션을 실행하기 위해 spark-submit을 지원합니다. Spark 애플리케이션을 실행하려면 다음 단계를 수행합니다.

1. [Amazon EMR on EKS에 대한 spark-submit 설정](#)의 단계를 완료했는지 확인합니다.
2. 다음과 같은 환경 변수의 값을 설정합니다.

```
export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon EKS-cluster-endpoint
```

3. 이제 다음 명령을 사용하여 Spark 애플리케이션을 제출합니다.

```
$SPARK_HOME/bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master $MASTER_URL \
  --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.15.0:latest \
  --conf spark.kubernetes.authenticate.driver.serviceAccountName=emr-containers-sa-spark \
  --deploy-mode cluster \
  --conf spark.kubernetes.namespace=default \
  --conf "spark.driver.extraClassPath=/usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*" \
```

```
--conf "spark.driver.extraLibraryPath=/usr/lib/hadoop/lib/native:/usr/lib/hadoop-
lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/
native" \
--conf "spark.executor.extraClassPath=/usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/
hadoop-aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/
share/aws/emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/
security/conf:/usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-
glue-datacatalog-spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-
serde.jar:/usr/share/aws/sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/
hadoop/extrajars/*" \
--conf "spark.executor.extraLibraryPath=/usr/lib/hadoop/lib/native:/usr/lib/
hadoop-lzo/lib/native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/
lib/native" \
--conf
spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.auth.WebIdentityTokenCredent
\
--conf spark.hadoop.fs.s3.impl=com.amazon.ws.emr.hadoop.fs.EmrFileSystem \
--conf
spark.hadoop.fs.AbstractFileSystem.s3.impl=org.apache.hadoop.fs.s3.EMRFSDelegate \
--conf spark.hadoop.fs.s3.buffer.dir=/mnt/s3 \
--conf spark.hadoop.fs.s3.getObject.initialSocketTimeoutMilliseconds="2000" \
--conf
spark.hadoop.mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFile
\
--conf spark.hadoop.mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem="true" \
s3://my-pod-bucket/spark-examples.jar 20
```

4. Spark 드라이버가 Spark 작업을 완료한 후 Spark 작업이 완료되었음을 나타내는 로그 줄이 제출 끝에 표시됩니다.

```
23/11/24 17:02:14 INFO LoggingPodStatusWatcherImpl: Application
org.apache.spark.examples.SparkPi with submission ID default:org-apache-spark-
examples-sparkpi-4980808c03ff3115-driver finished
23/11/24 17:02:14 INFO ShutdownHookManager: Shutdown hook called
```

## 정리

애플리케이션 실행이 완료되면 다음 명령을 사용하여 정리를 수행할 수 있습니다.

```
kubectl delete -f spark-rbac.yaml
```

# Amazon EMR on EKS에서 Apache Livy 사용

Amazon EMR 릴리스 7.1.0 이상에서는 Apache Livy를 사용하여 Amazon EMR on EKS에서 작업을 제출할 수 있습니다. Apache Livy를 사용하면 자체 Apache Livy REST 엔드포인트를 설정하고 이를 사용해 Amazon EKS 클러스터에서 Spark 애플리케이션을 배포하고 관리할 수 있습니다. Amazon EKS 클러스터에 Livy를 설치한 후 Livy 엔드포인트를 사용하여 Spark 애플리케이션을 Livy 서버에 제출할 수 있습니다. 서버는 Spark 애플리케이션의 수명 주기를 관리합니다.

## Note

Amazon EMR은 vCPU 및 메모리 소비를 기반으로 Amazon EKS에 대한 요금을 계산합니다. 이 계산은 드라이버 및 실행기 포드에 적용됩니다. 이 계산은 Amazon EMR 애플리케이션 이미지를 다운로드한 시점부터 시작하여 Amazon EKS 포드가 종료될 때까지 수행되며, 가장 가까운 초로 반올림됩니다.

## 주제

- [Amazon EMR on EKS에 대한 Apache Livy 설정](#)
- [Amazon EMR on EKS에서 Apache Livy 시작하기](#)
- [Amazon EMR on EKS에 대한 Apache Livy를 사용하여 Spark 애플리케이션 실행](#)
- [Amazon EMR on EKS에서 Apache Livy 제거](#)
- [Amazon EMR on EKS에서 Apache Livy에 대한 보안](#)
- [Amazon EMR on EKS 릴리스에서 Apache Livy에 대한 설치 속성](#)
- [일반적인 환경 변수 형식 오류 문제 해결](#)

## Amazon EMR on EKS에 대한 Apache Livy 설정

Amazon EKS 클러스터에 Apache Livy를 설치하려면 먼저 사전 조건 도구 세트를 설치하고 구성해야 합니다. 여기에는 AWS 리소스 작업을 위한 기본 명령줄 도구 AWS CLI인 , Amazon EKS 작업을 위한 명령줄 도구, 이 사용 사례에서 클러스터 애플리케이션을 인터넷에 제공하고 네트워크 트래픽을 라우팅하는 데 사용되는 컨트롤러가 포함됩니다.

- [의 최신 버전 설치 또는 업데이트 AWS CLI](#) -를 이미 설치한 경우 최신 버전이 있는지 AWS CLI 확인합니다.
- [kubectl 및 eksctl 설치](#) - eksctl은 Amazon EKS와 통신하는 데 사용하는 명령줄 도구입니다.

- [Install Helm](#) - Kubernetes용 Helm 패키지 관리자는 Kubernetes 클러스터에서 애플리케이션을 설치하고 관리하는 데 도움이 됩니다.
- [Amazon EKS 시작하기 - eksctl](#) - 관련 단계를 수행하여 Amazon EKS에서 노드를 포함하는 새 Kubernetes 클러스터를 생성합니다.
- [Amazon EMR 릴리스 레이블 선택](#) - Amazon EMR 릴리스 7.1.0 이상에서 Apache Livy가 지원됩니다.
- [ALB 컨트롤러 설치](#) - ALB 컨트롤러는 Kubernetes 클러스터용 AWS Elastic Load Balancing을 관리합니다. Apache Livy를 설정하는 동안 Kubernetes 수신을 생성할 때 AWS Network Load Balancer(NLB)를 생성합니다.

## Amazon EMR on EKS에서 Apache Livy 시작하기

다음 단계를 완료하여 Apache Livy를 설치합니다. 여기에는 패키지 관리자 구성, Spark 워크로드 실행을 위한 네임스페이스 생성, Livy 설치, 로드 밸런싱 설정 및 확인 단계가 포함됩니다. Spark에서 배치 작업을 실행하려면 다음 단계를 완료해야 합니다.

1. 아직 하지 않은 경우 [Amazon EMR에 대한 Apache Livy on EKS](#)를 설정합니다.
2. Helm 클라이언트를 Amazon ECR 레지스트리에 인증합니다. [리전별로 Amazon ECR 레지스트리 계정](#) AWS 리전 에서에 해당하는 ECR-registry-account 값을 찾을 수 있습니다.

```
aws ecr get-login-password --region <AWS_REGION> | helm registry login \
--username AWS \
--password-stdin <ECR-registry-account>.dkr.ecr.<region-id>.amazonaws.com
```

3. Livy를 설정하면 Livy 서버에 대한 한 서비스 계정과 Spark 애플리케이션에 대한 다른 계정이 생성됩니다. 서비스 계정에 대한 IRSA를 설정하려면 [서비스 계정에 대한 IAM 역할\(IRSA\)을 사용하여 액세스 권한 설정](#)을 참조하세요.
4. Spark 워크로드를 실행할 네임스페이스를 생성합니다.

```
kubectl create ns <spark-ns>
```

5. 다음 명령을 사용하여 Livy를 설치합니다.

이 Livy 엔드포인트는 EKS 클러스터의 VPC에 대해서만 내부적으로 사용할 수 있습니다. VPC 이상의 액세스를 활성화하려면 Helm 설치 명령에서 `--set loadbalancer.internal=false`를 설정합니다.

**Note**

기본적으로 SSL은 이 Livy 엔드포인트 내에서 활성화되지 않으며, 엔드포인트는 EKS 클러스터의 VPC 내에서만 표시됩니다. `loadbalancer.internal=false` 및 `ssl.enabled=false`를 설정하면 비보안 엔드포인트가 VPC 외부로 노출됩니다. 보안 Livy 엔드포인트를 설정하려면 [TLS/SSL을 사용하여 보안 Apache Livy 엔드포인트 구성](#)을 참조하세요.

```
helm install livy-demo \
  oci://895885662937.dkr.ecr.region-id.amazonaws.com/livy \
  --version 7.13.0 \
  --namespace livy-ns \
  --set image=ECR-registry-account.dkr.ecr.region-id.amazonaws.com/livy/
emr-7.13.0:latest \
  --set sparkNamespace=<spark-ns> \
  --create-namespace
```

다음과 같이 출력되어야 합니다.

```
NAME: livy-demo
LAST DEPLOYED: Mon Mar 18 09:23:23 2024
NAMESPACE: livy-ns
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The Livy server has been installed.
Check installation status:
1. Check Livy Server pod is running
   kubectl --namespace livy-ns get pods -l "app.kubernetes.io/instance=livy-demo"
2. Verify created NLB is in Active state and it's target groups are healthy (if
   loadbalancer.enabled is true)

Access LIVY APIs:
  # Ensure your NLB is active and healthy
  # Get the Livy endpoint using command:
  LIVY_ENDPOINT=$(kubectl get svc -n livy-ns -l app.kubernetes.io/
instance=livy-demo,emr-containers.amazonaws.com/type=loadbalancer -o
```

```
jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}' | awk '{printf "%s:8998\n", $0}'
# Access Livy APIs using http://$LIVY_ENDPOINT or https://$LIVY_ENDPOINT (if
SSL is enabled)
# Note: While uninstalling Livy, makes sure the ingress and NLB are deleted
after running the helm command to avoid dangling resources
```

Livy 서버 및 Spark 세션의 기본 서비스 계정 이름은 `emr-containers-sa-livy` 및 `emr-containers-sa-spark-livy`입니다. 사용자 지정 이름을 사용하려면 `serviceAccounts.name` 및 `sparkServiceAccount.name` 파라미터를 사용합니다.

```
--set serviceAccounts.name=my-service-account-for-livy
--set sparkServiceAccount.name=my-service-account-for-spark
```

## 6. 헬름 차트를 설치했는지 확인합니다.

```
helm list -n livy-ns -o yaml
```

`helm list` 명령은 새 헬름 차트에 대한 정보를 반환해야 합니다.

```
app_version: 0.7.1-incubating
chart: livy-emr-7.13.0
name: livy-demo
namespace: livy-ns
revision: "1"
status: deployed
updated: 2024-02-08 22:39:53.539243 -0800 PST
```

## 7. Network Load Balancer가 활성화 상태인지 확인합니다.

```
LIVY_NAMESPACE=<livy-ns>
LIVY_APP_NAME=<livy-app-name>
AWS_REGION=<AWS_REGION>

# Get the NLB Endpoint URL
NLB_ENDPOINT=$(kubectl --namespace $LIVY_NAMESPACE get svc -l "app.kubernetes.io/instance=$LIVY_APP_NAME,emr-containers.amazonaws.com/type=loadbalancer" -o jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}')

# Get all the load balancers in the account's region
ELB_LIST=$(aws elbv2 describe-load-balancers --region $AWS_REGION)
```

```
# Get the status of the NLB that matching the endpoint from the Kubernetes service
NLB_STATUS=$(echo $ELB_LIST | grep -A 8 "\"DNSName\": \"$NLB_ENDPOINT\"" | awk '/
Code/{print $2}/' | tr -d '",'')
echo $NLB_STATUS
```

## 8. 이제 Network Load Balancer의 대상 그룹이 정상인지 확인합니다.

```
LIVY_NAMESPACE=<livy-ns>
LIVY_APP_NAME=<livy-app-name>
AWS_REGION=<AWS_REGION>

# Get the NLB endpoint
NLB_ENDPOINT=$(kubectl --namespace $LIVY_NAMESPACE get svc -l "app.kubernetes.io/
instance=$LIVY_APP_NAME,emr-containers.amazonaws.com/type=loadbalancer" -o
jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}')

# Get all the load balancers in the account's region
ELB_LIST=$(aws elbv2 describe-load-balancers --region $AWS_REGION)

# Get the NLB ARN from the NLB endpoint
NLB_ARN=$(echo $ELB_LIST | grep -B 1 "\"DNSName\": \"$NLB_ENDPOINT\"" | awk
'/"LoadBalancerArn":/,/,/' | awk '/:/{print $2}' | tr -d \,)

# Get the target group from the NLB. Livy setup only deploys 1 target group
TARGET_GROUP_ARN=$(aws elbv2 describe-target-groups --load-balancer-arn $NLB_ARN
--region $AWS_REGION | awk '/"TargetGroupArn":/,/,/' | awk '/:/{print $2}' | tr -d
\,,)

# Get health of target group
aws elbv2 describe-target-health --target-group-arn $TARGET_GROUP_ARN
```

다음은 대상 그룹의 상태를 보여주는 샘플 출력입니다.

```
{
  "TargetHealthDescriptions": [
    {
      "Target": {
        "Id": "<target IP>",
        "Port": 8998,
        "AvailabilityZone": "us-west-2d"
      },
      "HealthCheckPort": "8998",
```



```

        "TargetHealth": {
            "State": "healthy"
        }
    }
]
}

```

NLB가 active 상태가 되고 대상 그룹이 healthy 상태가 되면 계속할 수 있습니다. 몇 분 정도 걸릴 수 있습니다.

9. Helm 설치에서 Livy 엔드포인트를 검색합니다. Livy 엔드포인트의 보안 여부는 SSL을 활성화했는지 여부에 따라 달라집니다.

```

LIVY_NAMESPACE=<livy-ns>
LIVY_APP_NAME=<livy-app-name>
LIVY_ENDPOINT=$(kubectl get svc -n livy-ns -l app.kubernetes.io/
instance=<livy-app-name>,emr-containers.amazonaws.com/type=loadbalancer -o
jsonpath='{.items[0].status.loadBalancer.ingress[0].hostname}' | awk '{printf
"%s:8998\n", $0}')
echo "$LIVY_ENDPOINT"

```

10. Helm 설치에서 Spark 서비스 계정 검색

```

SPARK_NAMESPACE=spark-ns
LIVY_APP_NAME=<livy-app-name>
SPARK_SERVICE_ACCOUNT=$(kubectl --namespace $SPARK_NAMESPACE
get sa -l "app.kubernetes.io/instance=$LIVY_APP_NAME" -o
jsonpath='{.items[0].metadata.name}')
echo "$SPARK_SERVICE_ACCOUNT"

```

다음 출력과 유사한 결과가 나타날 것입니다.

```
emr-containers-sa-spark-livy
```

11. VPC 외부에서 액세스를 활성화하도록 `internalALB=true`를 설정한 경우 Amazon EC2 인스턴스를 생성하고 Network Load Balancer가 EC2 인스턴스의 수신 네트워크 트래픽을 허용하는지 확인합니다. 인스턴스가 Livy 엔드포인트에 액세스할 수 있으려면 이를 수행해야 합니다. VPC 외부에서 엔드포인트를 안전하게 노출하는 방법에 대한 자세한 내용은 [TLS/SSL을 사용하여 보안 Apache Livy 엔드포인트 설정](#)을 참조하세요.

12. Livy를 설치하면 Spark 애플리케이션을 실행할 `emr-containers-sa-spark` 서비스 계정이 생성됩니다. Spark 애플리케이션이 S3와 같은 AWS 리소스를 사용하거나 AWS API 또는 CLI 작업을 호출하는 경우 IAM 역할을 Spark 서비스 계정에 필요한 권한과 연결해야 합니다. 자세한 내용은 [서비스 계정에 대한 IAM 역할\(IRSA\)을 사용하여 클러스터 액세스 권한 설정](#)을 참조하세요.

Apache Livy는 Livy를 설치하는 동안 사용할 수 있는 추가 구성을 지원합니다. 자세한 내용은 Amazon EMR on EKS 릴리스의 Apache Livy 설치 속성을 참조하세요.

## Amazon EMR on EKS에 대한 Apache Livy를 사용하여 Spark 애플리케이션 실행

Apache Livy로 Spark 애플리케이션을 실행하기 전에 [Amazon EMR on EKS에 대한 Apache Livy 설정](#) 및 [Amazon EMR on EKS에 대한 Apache Livy 시작하기](#)의 단계를 완료했는지 확인합니다.

Apache Livy를 사용하여 두 가지 유형의 애플리케이션을 실행할 수 있습니다.

- 배치 세션 - Spark 배치 작업을 제출하는 Livy 워크로드의 한 유형.
- 대화형 세션 - Spark 쿼리를 실행할 수 있는 프로그래밍 방식 및 시각적 인터페이스를 제공하는 Livy 워크로드의 한 유형.

### Note

서로 다른 세션의 드라이버 및 실행기 포드는 서로 통신할 수 있습니다. 네임스페이스는 포드 간 보안을 보장하지 않습니다. Kubernetes는 지정된 네임스페이스 내 포드 하위 세트에 대한 선택적 권한을 허용하지 않습니다.

## 배치 세션 실행

배치 작업을 제출하려면 다음 명령을 사용합니다.

```
curl -s -k -H 'Content-Type: application/json' -X POST \
  -d '{
    "name": "my-session",
    "file": "entryPoint_location (S3 or local)",
    "args": ["argument1", "argument2", ...],
    "conf": {
      "spark.kubernetes.namespace": "<spark-namespace>",
```

```

        "spark.kubernetes.container.image": "public.ecr.aws/emr-on-eks/spark/
emr-7.13.0:latest",
        "spark.kubernetes.authenticate.driver.serviceAccountName": "<spark-
service-account>"
    }
}' <livy-endpoint>/batches

```

배치 작업을 모니터링하려면 다음 명령을 사용합니다.

```

curl -s -k -H 'Content-Type: application/json' -X GET <livy-endpoint>/batches/my-
session

```

## 대화형 세션 구성

Apache Livy를 사용하여 대화형 세션을 실행하려면 다음 단계를 참조하세요.

1. 자체 호스팅 또는 SageMaker AI Jupyter Notebook과 같은 관리형 Jupyter Notebook에 액세스할 수 있는지 확인합니다. Jupyter Notebook에 [sparkmagic](#)이 설치되어 있어야 합니다.
2. Spark 구성 `spark.kubernetes.file.upload.path`에 대한 버킷을 생성합니다. Spark 서비스 계정이 버킷에 대한 읽기 및 쓰기 액세스 권한을 보유하는지 확인합니다. Spark 서비스 계정을 구성하는 방법에 대한 자세한 내용은 서비스 계정에 대한 IAM 역할(IRSA)을 사용하여 액세스 권한 설정을 참조하세요.
3. `%load_ext sparkmagic.magics` 명령을 사용하여 Jupyter Notebook에 `sparkmagic`을 로드합니다.
4. `%manage_spark` 명령을 실행하여 Jupyter Notebook으로 Livy 엔드포인트를 설정합니다. 엔드포인트 추가 탭을 선택하고 구성된 인증 유형을 선택하며 노트북에 Livy 엔드포인트를 추가한 다음, 엔드포인트 추가를 선택합니다.
5. 다시 `%manage_spark`를 실행하여 Spark 컨텍스트를 생성한 다음, 세션 생성으로 이동합니다. Livy 엔드포인트를 선택하고 고유한 세션 이름을 지정하며 언어를 선택한 후 다음 속성을 추가합니다.

```

{
  "conf": {
    "spark.kubernetes.namespace": "livy-namespcae",
    "spark.kubernetes.container.image": "public.ecr.aws/emr-on-eks/spark/
emr-7.13.0:latest",
    "spark.kubernetes.authenticate.driver.serviceAccountName": "<spark-service-
account>",

```

```
"spark.kubernetes.file.upload.path": "<URI_TO_S3_LOCATION_"
}
}
```

6. 애플리케이션을 제출하고 Spark 컨텍스트가 생성될 때까지 기다립니다.
7. 대화형 세션의 상태를 모니터링하려면 다음 명령을 실행합니다.

```
curl -s -k -H 'Content-Type: application/json' -X GET livy-endpoint/sessions/my-interactive-session
```

## Spark 애플리케이션 모니터링

Livy UI를 사용하여 Spark 애플리케이션의 진행 상황을 모니터링하려면 `http://<livy-endpoint>/ui` 링크를 사용합니다.

## Amazon EMR on EKS에서 Apache Livy 제거

다음 단계를 수행하여 Apache Livy를 제거합니다.

1. 네임스페이스의 이름 및 애플리케이션 이름을 사용하여 Livy 설정을 삭제합니다. 이 예제에서 애플리케이션 이름은 `livy-demo`이고 네임스페이스는 `livy-ns`입니다.

```
helm uninstall livy-demo -n livy-ns
```

2. 제거 시 Amazon EMR on EKS는 Livy의 Kubernetes 서비스, AWS 로드 밸런서 및 설치 중에 생성한 대상 그룹을 삭제합니다. 리소스를 삭제하는 데 몇 분 정도 걸릴 수 있습니다. 네임스페이스에 Livy를 다시 설치하기 전에 리소스가 삭제되었는지 확인합니다.
3. Spark 네임스페이스를 삭제합니다.

```
kubectl delete namespace spark-ns
```

## Amazon EMR on EKS에서 Apache Livy에 대한 보안

Amazon EMR on EKS에서 Apache Livy에 대한 보안을 구성하는 방법에 대한 자세한 내용은 다음 주제를 참조하세요. 이러한 옵션에는 전송 계층 보안, 조직 내 개인의 역할을 기반으로 하는 액세스인 역할 기반 액세스 제어, 부여된 권한을 기반으로 리소스에 대한 액세스를 제공하는 IAM 역할 사용이 포함됩니다.

## 주제

- [TLS/SSL을 사용하여 보안 Apache Livy 엔드포인트 설정](#)
- [역할 기반 액세스 제어\(RBAC\)를 사용하여 Apache Livy 및 Spark 애플리케이션 권한 설정](#)
- [서비스 계정에 대한 IAM 역할\(IRSA\)을 사용하여 액세스 권한 설정](#)

## TLS/SSL을 사용하여 보안 Apache Livy 엔드포인트 설정

엔드투엔드 TLS 및 SSL 암호화를 사용하여 Amazon EMR on EKS에 대해 Apache Livy를 설정하는 방법에 대한 자세한 내용은 다음 섹션을 참조하세요.

### TLS 및 SSL 암호화 설정

Apache Livy 엔드포인트에서 SSL 암호화를 설정하려면 다음 단계를 수행합니다.

- [Secrets Store CSI Driver and AWS Secrets and Configuration Provider\(ASCP\) 설치](#) - Secrets Store CSI Driver 및 ASCP는 Livy 서버 포드가 SSL을 활성화하는 데 필요한 Livy의 JKS 인증서와 암호를 안전하게 저장합니다. Secrets Store CSI 드라이버만 설치하고 지원되는 다른 보안 암호 제공업체를 사용할 수도 있습니다.
- [ACM 인증서 생성](#) - 이 인증서는 클라이언트와 ALB 엔드포인트 간 연결을 보안하는 데 필요합니다.
- ALB 엔드포인트와 Livy 서버 간의 연결을 보호하는 데 AWS Secrets Manager 필요한에 대한 JKS 인증서, 키 암호 및 키 스토어 암호를 설정합니다.
- Livy 서비스 계정에 보안 암호를 검색할 수 있는 권한을 추가합니다 AWS Secrets Manager . Livy 서버는 ASCP에서 보안 암호를 검색하고 Livy 구성을 추가하여 Livy 서버를 보호하려면 이러한 권한이 필요합니다. 서비스 계정에 IAM 권한을 추가하려면 서비스 계정에 대한 IAM 역할(IRSA)을 사용하여 액세스 권한 설정을 참조하세요.

에 대한 키 및 키 스토어 암호를 사용하여 JKS 인증서 설정 AWS Secrets Manager

다음 단계를 수행하여 키 및 키 저장소 암호를 사용해 JKS 인증서를 설정합니다.

1. Livy 서버의 키 저장소 파일을 생성합니다.

```
keytool -genkey -alias <host> -keyalg RSA -keysize 2048 -dname
CN=<host>,OU=hw,0=hw,L=<your_location>,ST=<state>,C=<country> -
keypass <keyPassword> -keystore <keystore_file> -storepass <storePassword> --
validity 3650
```

## 2. 인증서를 생성합니다.

```
keytool -export -alias <host> -keystore mykeystore.jks -rfc -
file mycertificate.cert -storepass <storePassword>
```

## 3. 신뢰 저장소 파일을 생성합니다.

```
keytool -import -noprompt -alias <host>-file <cert_file> -
keystore <truststore_file> -storepass <truststorePassword>
```

## 4. JKS 인증서에 저장합니다 AWS Secrets Manager. livy-jks-secret을 보안 암호로, fileb://mykeystore.jks를 키 저장소 JKS 인증서의 경로로 바꿉니다.

```
aws secretsmanager create-secret \
--name livy-jks-secret \
--description "My Livy keystore JKS secret" \
--secret-binary fileb://mykeystore.jks
```

## 5. Secrets Manager에 키 저장소 및 키 암호를 저장합니다. 자체 파라미터를 사용해야 합니다.

```
aws secretsmanager create-secret \
--name livy-jks-secret \
--description "My Livy key and keystore password secret" \
--secret-string "{\"keyPassword\":\"<test-key-password>\", \"keyStorePassword\": \"<test-key-store-password>\"}"
```

## 6. 다음 명령으로 Livy 서버 네임스페이스를 생성합니다.

```
kubectl create ns <livy-ns>
```

## 7. JKS 인증서와 암호가 있는 Livy 서버에 대한 ServiceProviderClass 객체를 생성합니다.

```
cat >livy-secret-provider-class.yaml << EOF
apiVersion: secrets-store.csi.x-k8s.io/v1
kind: SecretProviderClass
metadata:
  name: aws-secrets
spec:
  provider: aws
  parameters:
    objects: |
      - objectName: "livy-jks-secret"
```

```

      objectType: "secretsmanager"
    - objectName: "livy-passwords"
      objectType: "secretsmanager"

```

EOF

```
kubectl apply -f livy-secret-provider-class.yaml -n <livy-ns>
```

## SSL 지원 Apache Livy 시작하기

Livy 서버에서 SSL을 활성화한 후 AWS Secrets Manager에서 keyStore 및 keyPasswords 보안 암호에 액세스할 수 있도록 serviceAccount를 설정해야 합니다.

1. Livy 서버 네임스페이스를 생성합니다.

```
kubectl create namespace <livy-ns>
```

2. Secrets Manager의 보안 암호에 액세스할 수 있도록 Livy 서비스 계정을 설정합니다. IRSA 설정에 대한 자세한 내용은 [Apache Livy를 설치하는 동안 IRSA 설정](#)을 참조하세요.

```

aws ecr get-login-password --region region-id | helm registry login \
--username AWS \
--password-stdin ECR-registry-account.dkr.ecr.region-id.amazonaws.com

```

3. Livy를 설치합니다. 헬름 차트 --version 파라미터의 경우 Amazon EMR 릴리스 레이블(7.1.0)을 사용합니다. 또한 Amazon ECR 레지스트리 계정 ID 및 리전 ID를 자체 ID로 바꾸어야 합니다. [리전별로 Amazon ECR 레지스트리 계정](#) AWS 리전 에서에 해당하는 ECR-registry-account 값을 찾을 수 있습니다.

```

helm install <livy-app-name> \
oci://895885662937.dkr.ecr.region-id.amazonaws.com/livy \
--version 7.13.0 \
--namespace <livy-namespace-name> \
--set image=<ECR-registry-account.dkr.ecr>.<region>.amazonaws.com/livy/emr-7.13.0:latest \
--set sparkNamespace=<spark-namespace> \
--set ssl.enabled=true
--set ssl.CertificateArn=livy-acm-certificate-arn
--set ssl.secretProviderClassName=aws-secrets
--set ssl.keyStoreObjectName=livy-jks-secret
--set ssl.keyPasswordsObjectName=livy-passwords

```

```
--create-namespace
```

#### 4. [Amazon EMR on EKS에 Apache Livy 설치](#)의 5단계부터 계속합니다.

### 역할 기반 액세스 제어(RBAC)를 사용하여 Apache Livy 및 Spark 애플리케이션 권한 설정

Livy를 배포하기 위해 Amazon EMR on EKS는 서버 서비스 계정 및 역할과 Spark 서비스 계정 및 역할을 생성합니다. 이러한 역할에는 설정을 완료하고 Spark 애플리케이션을 실행하는 데 필요한 RBAC 권한이 있어야 합니다.

#### 서버 서비스 계정 및 역할에 대한 RBAC 권한

Amazon EMR on EKS는 Spark 작업에 대한 Livy 세션을 관리하고 수신 및 기타 리소스 사이에서 트래픽을 라우팅하도록 Livy 서버 서비스 계정 및 역할을 생성합니다.

이 서비스 계정의 기본 이름은 `emr-containers-sa-livy`입니다. 이 경우 다음 권한이 있어야 합니다.

```
rules:
- apiGroups:
  - ""
  resources:
  - "namespaces"
  verbs:
  - "get"
- apiGroups:
  - ""
  resources:
  - "serviceaccounts"
    "services"
    "configmaps"
    "events"
    "pods"
    "pods/log"
  verbs:
  - "get"
    "list"
    "watch"
    "describe"
    "create"
    "edit"
```



```

    "delete"
    "deletecollection"
    "annotate"
    "patch"
    "label"
  - apiGroups:
    - ""
    resources:
    - "secrets"
    verbs:
    - "create"
    - "patch"
    - "delete"
    - "watch"
  - apiGroups:
    - ""
    resources:
    - "persistentvolumeclaims"
    verbs:
    - "get"
    - "list"
    - "watch"
    - "describe"
    - "create"
    - "edit"
    - "delete"
    - "annotate"
    - "patch"
    - "label"

```

## Spark 서비스 계정 및 역할에 대한 RBAC 권한

Spark 드라이버 포드에는 포드와 동일한 네임스페이스에 있는 Kubernetes 서비스 계정이 필요합니다. 이 서비스 계정에는 실행기 포드 그리고 드라이버 포드에 필요한 모든 리소스를 관리할 수 있는 권한이 필요합니다. 네임스페이스의 기본 서비스 계정에 필요한 권한이 없으면 드라이버가 실패하고 종료됩니다. 다음 RBAC 권한이 필요합니다.

```

rules:
- apiGroups:
  - ""
  resources:
  - "batch"
  - "extensions"
  - "apps"

```

```
resources:
- "configmaps"
  "serviceaccounts"
  "events"
  "pods"
  "pods/exec"
  "pods/log"
  "pods/portforward"
  "secrets"
  "services"
  "persistentvolumeclaims"
  "statefulsets"
verbs:
- "create"
  "delete"
  "get"
  "list"
  "patch"
  "update"
  "watch"
  "describe"
  "edit"
  "deletecollection"
  "patch"
  "label"
```

## 서비스 계정에 대한 IAM 역할(IRSA)을 사용하여 액세스 권한 설정

기본적으로 Livy 서버와 Spark 애플리케이션의 드라이버 및 실행기는 AWS 리소스에 액세스할 수 없습니다. 서버 서비스 계정과 Spark 서비스 계정은 Livy 서버 및 Spark 애플리케이션의 포드에 대한 AWS 리소스 액세스를 제어합니다. 액세스 권한을 부여하려면 서비스 계정을 필요한 AWS 권한이 있는 IAM 역할로 매핑해야 합니다.

Apache Livy를 설치하기 전, 설치 중 또는 설치를 완료한 후 IRSA 매핑을 설정할 수 있습니다.

### Apache Livy 설치 중 IRSA 설정(서버 서비스 계정용)

#### Note

이 매핑은 서버 서비스 계정에서만 지원됩니다.

1. [Amazon EMR on EKS에 대한 Apache Livy 설정](#)을 완료하고 [Amazon EMR on EKS에서 Apache Livy 설치](#)를 수행하는 중인지 확인합니다.
2. Livy 서버에 대한 Kubernetes 네임스페이스를 생성합니다. 이 예제에서 네임스페이스의 이름은 `livy-ns`입니다.
3. 포드가 액세스할에 AWS 서비스 대한 권한이 포함된 IAM 정책을 생성합니다. 다음 예제에서는 Spark 진입점에 대한 Amazon S3 리소스를 가져오는 IAM 정책을 생성합니다.

```
cat >my-policy.json <<EOF{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::my-spark-entripoint-bucket"
    }
  ]
}
EOF

aws iam create-policy --policy-name my-policy --policy-document file://my-policy.json
```

4. 다음 명령을 사용하여 AWS 계정 ID를 변수로 설정합니다.

```
account_id=$(aws sts get-caller-identity --query "Account" --output text)
```

5. 클러스터의 OpenID Connect(OIDC) ID 제공업체를 환경 변수로 설정합니다.

```
oidc_provider=$(aws eks describe-cluster --name my-cluster --region $AWS_REGION --query "cluster.identity.oidc.issuer" --output text | sed -e "s/^https:\\\\//")
```

6. 서비스 계정의 네임스페이스 및 이름에 대한 변수를 설정합니다. 자체 값을 사용해야 합니다.

```
export namespace=default
export service_account=my-service-account
```

7. 다음 명령으로 신뢰 정책 파일을 생성합니다. 네임스페이스 내 모든 서비스 계정에 역할의 액세스 권한을 부여하려면 다음 명령을 복사하고 `StringEquals`를 `StringLike`로 바꾸고 `$service_account`를 `*`로 바꿉니다.

```
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
```

## 8. 역할을 생성합니다.

```
aws iam create-role --role-name my-role --assume-role-policy-document file://trust-relationship.json --description "my-role-description"
```

9. 다음 Helm 설치 명령을 사용하여 IRSA를 매핑하도록 `serviceAccount.executionRoleArn`을 설정합니다. 다음은 Helm 설치 명령의 예제입니다. [리전별로 Amazon ECR 레지스트리 계정](#) AWS 리전 에서에 해당하는 ECR-registry-account 값을 찾을 수 있습니다.

```
helm install livy-demo \
  oci://895885662937.dkr.ecr.us-west-2.amazonaws.com/livy \
  --version 7.13.0 \
  --namespace livy-ns \
  --set image=ECR-registry-account.dkr.ecr.region-id.amazonaws.com/livy/emr-7.13.0:latest \
  --set sparkNamespace=spark-ns \
  --set serviceAccount.executionRoleArn=arn:aws:iam::123456789012:role/my-role
```

## Spark 서비스 계정에 IRSA 매핑

IRSA를 Spark 서비스 계정에 매핑하기 전에 다음 항목을 완료했는지 확인합니다.

- [Amazon EMR on EKS에 대한 Apache Livy 설정](#)을 완료하고 [Amazon EMR on EKS에서 Apache Livy 설치](#)를 수행하는 중인지 확인합니다.
- 클러스터에 대한 기존 IAM OpenID Connect(OIDC) 제공업체가 있어야 합니다. 이미 있는지 확인하거나 새로 생성하는 방법을 알아보려면 [Create an IAM OIDC provider for your cluster](#)를 참조하세요.
- eksctl CLI의 버전 0.171.0 이상 또는 AWS CloudShell을 설치했는지 확인합니다. eksctl을 설치 또는 업그레이드하려면 eksctl 설명서의 [Installation](#)을 참조하세요.

다음 단계를 수행하여 IRSA를 Spark 서비스 계정에 매핑합니다.

1. 다음 명령을 사용하여 Spark 서비스 계정을 가져옵니다.

```
SPARK_NAMESPACE=<spark-ns>
LIVY_APP_NAME=<livy-app-name>
kubectl --namespace $SPARK_NAMESPACE describe sa -l "app.kubernetes.io/instance=$LIVY_APP_NAME" | awk '/^Name:/ {print $2}'
```

2. 서비스 계정의 네임스페이스 및 이름에 대한 변수를 설정합니다.

```
export namespace=default
export service_account=my-service-account
```

3. 다음 명령을 사용하여 IAM 역할에 대한 신뢰 정책 파일을 생성합니다. 다음 예제에서는 네임스페이스 내 모든 서비스 계정에 역할을 사용할 수 있는 권한을 부여합니다. 이를 수행하려면 StringEquals를 StringLike로 바꾸고 \$service\_account를 \*로 바꿉니다.

```
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

```

#### 4. 역할을 생성합니다.

```
aws iam create-role --role-name my-role --assume-role-policy-document file://trust-relationship.json --description "my-role-description"
```

#### 5. 다음 eksctl 명령을 사용하여 서버 또는 Spark 서비스 계정을 매핑합니다. 자체 값을 사용해야 합니다.

```
eksctl create iamserviceaccount --name spark-sa \
--namespace spark-namespace --cluster livy-eks-cluster \
--attach-role-arn arn:aws:iam::0123456789012:role/my-role \
--approve --override-existing-serviceaccounts
```

## Amazon EMR on EKS 릴리스에서 Apache Livy에 대한 설치 속성

Apache Livy 설치에서는 Livy 헬름 차트 버전을 선택할 수 있습니다. 헬름 차트는 설치 및 설정 환경을 사용자 지정할 수 있는 다양한 속성을 제공합니다. 이러한 속성은 Amazon EMR on EKS 릴리스 7.1.0 이상에서 지원됩니다.

### 주제

- [Amazon EMR 7.1.0 설치 속성](#)

## Amazon EMR 7.1.0 설치 속성

다음 표에서는 지원되는 모든 Livy 속성을 설명합니다. Apache Livy를 설치하는 경우 Livy 헬름 차트 버전을 선택할 수 있습니다. 설치 중에 속성을 설정하려면 `--set <property>=<value>` 명령을 사용합니다.

| 속성    | 설명                                     | 기본값 |
|-------|----------------------------------------|-----|
| image | Livy 서버의 Amazon EMR 릴리스 URI. 필수 구성입니다. | ""  |

| 속성                          | 설명                                                                | 기본값    |
|-----------------------------|-------------------------------------------------------------------|--------|
| sparkNamespace              | Livy Spark 세션을 실행할 네임스페이스. 예를 들어, "livy"를 지정합니다. 필수 구성입니다.        | ""     |
| nameOverride                | livy 대신 이름을 제공합니다. 이름은 모든 Livy 리소스에 대한 레이블로 설정됩니다.                | "livy" |
| fullnameOverride            | 리소스의 전체 이름 대신 사용할 이름을 제공합니다.                                      | ""     |
| ssl.enabled                 | Livy 엔드포인트에서 Livy 서버로 엔드투엔드 SSL을 활성화합니다.                          | FALSE  |
| ssl.certificateArn          | SSL이 활성화된 경우 서비스에서 생성한 NLB에 대한 ACM 인증서 ARN입니다.                    | ""     |
| ssl.secretProviderClassName | SSL이 활성화된 경우 SSL과의 Livy 서버 연결을 위해 NLB를 보안하는 보안 암호 제공업체 클래스 이름입니다. | ""     |
| ssl.keyStoreObjectName      | SSL이 활성화된 경우 보안 암호 제공업체 클래스의 키 저장소 인증서에 대한 객체 이름입니다.              | ""     |
| ssl.keyPasswordsObjectName  | SSL이 활성화된 경우 키 저장소 및 키 암호가 있는 보안 암호의 객체 이름.                       | ""     |
| rbac.create                 | true인 경우 RBAC 리소스를 생성합니다.                                         | FALSE  |

| 속성                              | 설명                                                                                                                                 | 기본값                            |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------|--------------------------------|
| serviceAccount.create           | true인 경우 Livy 서비스 계정을 생성합니다.                                                                                                       | TRUE                           |
| serviceAccount.name             | Livy에 대해 사용할 서비스 계정의 이름. 이 속성을 설정하지 않고 서비스 계정을 생성하면 Amazon EMR on EKS는 fullname 재정의 속성을 사용하여 이름을 자동 생성합니다.                         | "emr-containers-sa-livy"       |
| serviceAccount.executionRoleArn | Livy 서비스 계정의 실행 역할 ARN.                                                                                                            | ""                             |
| sparkServiceAccount.create      | true인 경우 .Release.Namespace 에서 Spark 서비스 계정 생성                                                                                     | TRUE                           |
| sparkServiceAccount.name        | Spark에 대해 사용할 서비스 계정의 이름. 이 속성을 설정하지 않고 Spark 서비스 계정을 생성하면 Amazon EMR on EKS는 -spark-livy 접미사의 fullnameOverride 속성으로 이름을 자동 생성합니다. | "emr-containers-sa-spark-livy" |
| service.name                    | Livy 서비스의 이름                                                                                                                       | "emr-containers-livy"          |
| service.annotations             | Livy 서비스 주석                                                                                                                        | {}                             |
| loadbalancer.enabled            | Amazon EKS 클러스터 외부에 Livy 엔드포인트를 노출하는 데 사용되는 Livy 서비스에 대한 로드 밸런서를 생성할지 여부.                                                          | FALSE                          |



| 속성                        | 설명                                                                                                                                                                    | 기본값   |
|---------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| loadbalancer.internal     | Livy 엔드포인트를 VPC 내부 또는 외부로 구성할지 여부.<br><br>이 속성을 FALSE로 설정하면 엔드포인트가 VPC 외부의 소스에 노출됩니다. 엔드포인트를 TLS/SSL로 보안하는 것이 좋습니다. 자세한 내용은 <a href="#">TLS 및 SSL 암호화 설정</a> 을 참조하세요. | FALSE |
| imagePullSecrets          | 프라이빗 리포지토리에서 Livy 이미지를 가져오는 데 사용할 imagePullSecret 이름 목록.                                                                                                              | []    |
| 리소스                       | Livy 컨테이너에 대한 리소스 요청 및 제한.                                                                                                                                            | {}    |
| nodeSelector              | Livy 포드를 예약할 노드.                                                                                                                                                      | {}    |
| 허용 오차                     | 정의할 Livy 포드 허용치를 포함하는 목록.                                                                                                                                             | []    |
| affinity                  | Livy 포드 선호도 규칙.                                                                                                                                                       | {}    |
| persistence.enabled       | true인 경우 세션 디렉터리의 지속성을 활성화합니다.                                                                                                                                        | FALSE |
| persistence.subPath       | 세션 디렉터리에 탑재할 PVC 하위 경로.                                                                                                                                               | ""    |
| persistence.existingClaim | 새 PVC를 생성하는 대신 사용할 PVC.                                                                                                                                               | {}    |

| 속성                       | 설명                                                                                                                                                                                                                                                                   | 기본값           |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|
| persistence.storageClass | 사용할 스토리지 클래스.<br>이 파라미터를 정의하려면<br>storageClassName:<br><i>&lt;storageClass&gt;</i> 형식을<br>사용합니다. 이 파라미터를<br>"- "로 설정하면 동적 프로비<br>저닝이 비활성화됩니다. 이<br>파라미터를 null로 설정하거<br>나 아무 것도 지정하지 않으<br>면 Amazon EMR on EKS는<br>storageClassName을 설정하<br>지 않으며 기본 프로비저너를<br>사용합니다. | ""            |
| persistence.accessMode   | PVC 액세스 모드.                                                                                                                                                                                                                                                          | ReadWriteOnce |
| persistence.size         | PVC 크기.                                                                                                                                                                                                                                                              | 20Gi          |
| persistence.annotations  | PVC에 대한 추가 주석.                                                                                                                                                                                                                                                       | {}            |
| env.*                    | Livy 컨테이너로 설정할 추<br>가 env. 자세한 내용은 <a href="#">Livy를<br/>설치하는 동안 자체 Livy 및<br/>Spark 구성 입력</a> 을 참조하세<br>요.                                                                                                                                                          | {}            |
| envFrom.*                | Kubernetes 구성 맵 또는 보안<br>암호에서 Livy로 설정할 추가<br>envs.                                                                                                                                                                                                                  | []            |
| livyConf.*               | 탑재된 Kubernetes 구성 맵 또<br>는 보안 암호에서 설정할 추가<br>livy.conf 항목.                                                                                                                                                                                                           | {}            |

| 속성                  | 설명                                                            | 기본값 |
|---------------------|---------------------------------------------------------------|-----|
| sparkDefaultsConf.* | 탑재된 Kubernetes 구성 맵 또는 보안 암호에서 설정할 추가 spark-defaults.conf 항목. | {}  |

## 일반적인 환경 변수 형식 오류 문제 해결

Livy 및 Spark 구성을 입력하는 경우 지원되지 않으며 오류를 일으킬 수 있는 환경 변수 형식이 있습니다. 이 절차에서는 올바른 형식을 사용하는 데 도움이 되는 일련의 단계를 안내합니다.

Livy를 설치하는 동안 자체 Livy 및 Spark 구성 입력

env.\* Helm 속성을 사용하여 모든 Apache Livy 또는 Apache Spark 환경 변수를 구성할 수 있습니다. 아래 단계를 수행하여 example.config.with-dash.withUppercase 예제 구성을 지원되는 환경 변수 형식으로 변환합니다.

1. 대문자를 1 및 소문자를 바꿉니다. 예를 들어, example.config.with-dash.withUppercase는 example.config.with-dash.with1uppercase가 됩니다.
2. 대시(-)를 0으로 바꿉니다. 예를 들어, example.config.with-dash.with1uppercase는 example.config.with0dash.with1uppercase가 됩니다.
3. 점(.)을 밑줄(\_)로 바꿉니다. 예를 들어, example.config.with0dash.with1uppercase는 example\_config\_with0dash\_with1uppercase가 됩니다.
4. 모든 소문자를 대문자로 바꿉니다.
5. 변수 이름에 LIVY\_ 접두사를 추가합니다.
6. --set env.YOUR\_VARIABLE\_NAME.value=yourvalue 형식을 사용하여 헬름 차트를 통해 Livy를 설치하는 동안 변수를 사용합니다.

예를 들어, Livy 및 Spark 구성 livy.server.recovery.state-store = filesystem 및 spark.kubernetes.executor.podNamePrefix = my-prefix를 설정하려면 다음 Helm 속성을 사용합니다.

```
--set env.LIVY_LIVY_SERVER_RECOVERY_STATE0STORE.value=filesystem
--set env.LIVY_SPARK_KUBERNETES_EXECUTOR_POD0NAME0PREFIX.value=myprefix
```

## Amazon EMR on EKS 작업 실행 관리

다음 섹션에서는 Amazon EMR on EKS 작업 실행을 관리하는 데 도움이 되는 주제를 다룹니다. 여기에는를 사용할 때 작업 실행 파라미터 구성 AWS CLI, 로그 데이터 저장 방법 구성, Spark SQL 스크립트를 실행하여 쿼리 실행, 작업 실행 상태 이해, 작업 모니터링 방법 파악이 포함됩니다. 데이터를 처리하기 위해 작업 실행을 설정하고 완료하려는 경우 일반적으로 순서대로 이러한 주제를 수행할 수 있습니다.

### 주제

- [를 사용하여 작업 실행 관리 AWS CLI](#)
- [StartJobRun API를 통해 Spark SQL 스크립트 실행](#)
- [작업 실행 상태](#)
- [Amazon EMR 콘솔에서 작업 보기](#)
- [작업 실행 시 발생하는 일반적인 오류](#)

## 를 사용하여 작업 실행 관리 AWS CLI

이 주제에서는 AWS Command Line Interface ()를 사용하여 작업 실행을 관리하는 방법을 다룹니다 AWS CLI. 보안 파라미터, 드라이버 및 다양한 재정의 설정과 같은 속성을 자세히 설명합니다. 로깅을 구성하는 다양한 방법을 다루는 하위 주제도 포함되어 있습니다.

### 주제

- [작업 실행 구성 옵션](#)
- [Amazon S3 로그를 사용하도록 작업 실행 구성](#)
- [Amazon CloudWatch Logs를 사용하도록 작업 실행 구성](#)
- [작업 실행 나열](#)
- [작업 실행 설명](#)
- [작업 실행 취소](#)

## 작업 실행 구성 옵션

다음 옵션을 사용하여 작업 실행 파라미터를 구성합니다.

- `--execution-role-arn`: 작업 실행에 사용되는 IAM 역할을 제공해야 합니다. 자세한 내용은 [Amazon EMR on EKS에서 작업 실행 역할 사용](#) 단원을 참조하십시오.

- `--release-label`: Amazon EMR 버전 5.32.0 및 6.2.0 이상을 사용하여 Amazon EMR on EKS를 배포할 수 있습니다. Amazon EMR on EKS는 이전 Amazon EMR 릴리스 버전에서 지원되지 않습니다. 자세한 내용은 [Amazon EMR on EKS 릴리스](#) 단원을 참조하십시오.
- `--job-driver`: 작업 드라이버는 기본 작업에 대한 입력을 제공하는 데 사용됩니다. 실행하려는 작업 유형에 대한 값 중 하나만 전달할 수 있는 집합 유형 필드입니다. 다음은 지원되는 유형입니다.
  - Spark 제출 작업 - Spark 제출을 통해 명령을 실행하는 데 사용됩니다. 이 작업 유형을 사용하여 Spark 제출을 통해 Scala, PySpark, SparkR, SparkSQL 및 기타 지원되는 작업을 실행할 수 있습니다. 이 작업 유형에는 다음과 같은 파라미터가 있습니다.
    - Entrypoint - 실행하려는 기본 jar/py 파일에 대한 Hadoop 호환 파일 시스템(HCFS) 참조입니다.
    - EntryPointArguments - 기본 jar/py 파일에 전달하려는 인수의 배열입니다. entrypoint 코드를 사용하여 이러한 파라미터를 읽는 작업을 처리해야 합니다. 배열의 각 인수는 쉼표로 분리해야 합니다. EntryPointArguments 인수에는 (), {} 또는 []와 같은 괄호를 포함할 수 없습니다.
    - SparkSubmitParameters - 작업에 전송하려는 추가 Spark 파라미터입니다. 이 파라미터를 사용하여 드라이버 메모리 및 실행기 수와 같은 기본 Spark 속성(예: `-conf` 또는 `-class`)을 재정의합니다. 자세한 내용은 [Launching Applications with spark-submit](#)을 참조하세요.
  - Spark SQL 작업 - Spark SQL을 통해 SQL 쿼리 파일을 실행하는 데 사용됩니다. 이 작업 유형을 사용하여 SparkSQL 작업을 실행할 수 있습니다. 이 작업 유형에는 다음과 같은 파라미터가 있습니다.
    - Entrypoint - 실행하려는 SQL 쿼리 파일에 대한 Hadoop 호환 파일 시스템(HCFS) 참조입니다.

Spark SQL 작업에 사용할 수 있는 추가 Spark 파라미터 목록은 [StartJobRun API를 통해 Spark SQL 스크립트 실행](#) 섹션을 참조하세요.

- `--configuration-overrides`: 애플리케이션에 대한 구성 객체를 제공하여 애플리케이션의 기본 구성을 재정의할 수 있습니다. 간편 구문을 사용하여 구성을 제공하거나 JSON 파일의 구성 객체를 참조할 수 있습니다. 구성 객체는 분류, 속성 및 선택적 중첩 구성으로 이루어져 있습니다. 속성은 해당 파일에서 재정의하려는 설정으로 구성됩니다. 단일 JSON 객체에서 여러 애플리케이션에 대해 다양한 분류를 지정할 수 있습니다. 사용 가능한 구성 분류는 Amazon EMR 릴리스 버전에 따라 달라집니다. Amazon EMR의 각 릴리스 버전에 사용할 수 있는 구성 분류 목록은 [Amazon EMR on EKS 릴리스](#) 섹션을 참조하세요.

애플리케이션 재정의 및 Spark 제출 파라미터에서 동일한 구성을 전달하는 경우 Spark 제출 파라미터가 우선합니다. 전체 구성 우선순위 목록은 가장 높은 우선순위부터 가장 낮은 우선순위 순으로 나열됩니다.

- SparkSession 생성 시 제공되는 구성.
- `-conf`를 사용하는 `sparkSubmitParameters`의 일부로 제공되는 구성.

- 애플리케이션 재정의의 일부로 제공되는 구성.
- 해당 릴리스에서 Amazon EMR이 선택하는 최적화된 구성.
- 애플리케이션의 기본 오픈 소스 구성.

Amazon CloudWatch 또는 Amazon S3를 사용하여 작업 실행을 모니터링하려면 CloudWatch에 대한 구성 세부 정보를 제공해야 합니다. 자세한 내용은 [Amazon S3 로그를 사용하도록 작업 실행 구성](#) 및 [Amazon CloudWatch Logs를 사용하도록 작업 실행 구성](#) 섹션을 참조하세요. S3 버킷 또는 CloudWatch 로그 그룹이 없는 경우 Amazon EMR은 로그를 버킷에 업로드하기 전에 그룹을 생성합니다.

- Kubernetes 구성 옵션의 추가 목록은 [Spark Properties on Kubernetes](#)를 참조하세요.

다음 Spark 구성은 지원되지 않습니다.

- `spark.kubernetes.authenticate.driver.serviceAccountName`
- `spark.kubernetes.authenticate.executor.serviceAccountName`
- `spark.kubernetes.namespace`
- `spark.kubernetes.driver.pod.name`
- `spark.kubernetes.container.image.pullPolicy`
- `spark.kubernetes.container.image`

#### Note

사용자 지정 도커 이미지에 `spark.kubernetes.container.image`를 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS에 대한 도커 이미지 사용자 지정](#) 단원을 참조하십시오.

## Amazon S3 로그를 사용하도록 작업 실행 구성

작업 진행 상황을 모니터링하고 실패 문제를 해결하려면 Amazon S3, Amazon CloudWatch Logs 또는 둘 다로 로그 정보를 전송하도록 작업을 구성해야 합니다. 이 주제는 Amazon EMR on EKS에서 시작된 작업에 Amazon S3에 애플리케이션 로그를 게시하는 작업을 시작하는 데 도움이 됩니다.

### S3 로그 IAM 정책

작업에서 Amazon S3로 로그 데이터를 전송하려면 먼저 작업 실행 역할에 대한 권한 정책에 다음 권한이 포함되어야 합니다. `amzn-s3-demo-logging-bucket`을 로깅 버킷의 이름으로 바꿉니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket",
        "arn:aws:s3:::amzn-s3-demo-bucket/*"
      ],
      "Sid": "AllowS3Putobject"
    }
  ]
}
```

 Note

Amazon EMR on EKS는 Amazon S3 버킷을 생성할 수도 있습니다. Amazon S3 버킷을 사용할 수 없는 경우 IAM 정책에 “s3:CreateBucket” 권한을 포함합니다.

Amazon S3에 로그를 전송할 수 있는 적절한 권한을 실행 역할에 부여한 후 [클러스터를 사용하여 작업 실행 관리 AWS CLI](#)에서와 같이 start-job-run 요청의 monitoringConfiguration 섹션에 s3MonitoringConfiguration이 전달되면 로그 데이터가 다음 Amazon S3 위치로 전송됩니다.

- 제출자 로그 - `/logUri/virtual-cluster-id/jobs/job-id/containers/pod-name/(stderr.gz/stdout.gz)`
- 드라이버 로그 - `/logUri/virtual-cluster-id/jobs/job-id/containers/spark-application-id/spark-job-id-driver/(stderr.gz/stdout.gz)`
- 실행기 로그 - `/logUri/virtual-cluster-id/jobs/job-id/containers/spark-application-id/executor-pod-name/(stderr.gz/stdout.gz)`

## Amazon CloudWatch Logs를 사용하도록 작업 실행 구성

작업 진행 상황을 모니터링하고 실패 문제를 해결하려면 Amazon S3, Amazon CloudWatch Logs 또는 둘 다로 로그 정보를 전송하도록 작업을 구성해야 합니다. 이 주제는 Amazon EMR on EKS에서 시작된 작업에서 CloudWatch Logs를 사용하는 데 도움이 됩니다. CloudWatch Logs에 대한 자세한 정보는 Amazon CloudWatch 사용 설명서의 [로그 파일 모니터링](#)을 참조하세요.

### CloudWatch Logs IAM 정책

작업에서 CloudWatch Log로 로그 데이터를 전송하려면 먼저 작업 실행 역할에 대한 권한 정책에 다음 권한이 포함되어야 합니다. *my\_log\_group\_name* 및 *my\_log\_stream\_prefix*를 각각 CloudWatch 로그 그룹 및 로그 스트림 이름으로 바꿉니다. 실행 역할 ARN에 적절한 권한이 있는 한, Amazon EMR on EKS는 로그 그룹과 로그 스트림이 없는 경우 해당 로그 그룹과 로그 스트림을 생성합니다.

### JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogStream",
        "logs:DescribeLogGroups",
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "arn:aws:logs:*:*:*"
      ],
      "Sid": "AllowLOGSCreatelogstream"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents"
      ],
      "Resource": [
        "arn:aws:logs:*:*:log-group:my_log_group_name:log-stream:my_log_stream_prefix/*"
      ],
      "Sid": "AllowLOGSPutlogevents"
    }
  ]
}
```



```
]
}
```

### Note

Amazon EMR on EKS는 로그 스트림을 생성할 수도 있습니다. 로그 스트림이 없는 경우 IAM 정책에 "logs:CreateLogGroup" 권한이 포함되어야 합니다.

실행 역할에 적절한 권한을 부여한 후 [클러스터를 사용하여 작업 실행 관리 AWS CLI](#)에서와 같이 애플리케이션은 start-job-run 요청의 monitoringConfiguration 섹션에 cloudWatchMonitoringConfiguration이 전달되면 로그 데이터를 CloudWatch Logs로 보냅니다.

StartJobRun API에서 *log\_group\_name*은 CloudWatch의 로그 그룹 이름이고 *log\_stream\_prefix*는 CloudWatch의 로그 스트림 이름 접두사입니다. AWS Management Console에서 이러한 로그를 보고 검색할 수 있습니다.

- 제출자 로그 - *logGroup/logStreamPrefix/virtual-cluster-id/jobs/job-id/containers/pod-name/(stderr/stdout)*
- 드라이버 로그 - *logGroup/logStreamPrefix/virtual-cluster-id/jobs/job-id/containers/spark-application-id/spark-job-id-driver/(stderr/stdout)*
- 실행기 로그 - *logGroup/logStreamPrefix/virtual-cluster-id/jobs/job-id/containers/spark-application-id/executor-pod-name/(stderr/stdout)*

## 작업 실행 나열

다음 예제에서 볼 수 있듯이 list-job-run을 실행하여 작업 실행 상태를 표시할 수 있습니다.

```
aws emr-containers list-job-runs --virtual-cluster-id <cluster-id>
```

## 작업 실행 설명

다음 예제에서 볼 수 있듯이 describe-job-run을 실행하여 작업 상태, 상태 세부 정보, 작업 이름 등 작업에 대한 추가 세부 정보를 가져올 수 있습니다.

```
aws emr-containers describe-job-run --virtual-cluster-id cluster-id --id job-run-id
```

## 작업 실행 취소

다음 예제에서 볼 수 있듯이 `cancel-job-run`을 실행하여 실행 중인 작업을 취소할 수 있습니다.

```
aws emr-containers cancel-job-run --virtual-cluster-id cluster-id --id job-run-id
```

## StartJobRun API를 통해 Spark SQL 스크립트 실행

Amazon EMR on EKS 릴리스 6.7.0 이상에는 StartJobRun API를 통해 Spark SQL 스크립트를 실행할 수 있도록 Spark SQL 작업 드라이버가 포함되어 있습니다. 기존 Spark SQL 스크립트를 수정하지 않고도 StartJobRun API를 사용하여 Amazon EMR on EKS에서 Spark SQL 쿼리를 직접 실행하도록 SQL 진입점 파일을 제공할 수 있습니다. 다음 테이블에는 StartJobRun API를 통해 Spark SQL 작업에 지원되는 Spark 파라미터가 나와 있습니다.

다음 Spark 파라미터 중에서 Spark SQL 작업에 전송할 파라미터를 선택할 수 있습니다. 이러한 파라미터를 사용하여 기본 Spark 속성을 재정의합니다.

| 옵션                                  | 설명                                                                                                             |
|-------------------------------------|----------------------------------------------------------------------------------------------------------------|
| <code>--name NAME</code>            | 애플리케이션 이름                                                                                                      |
| <code>--jars JARS</code>            | 드라이버 및 실행 클래스 경로에 포함할 jar의 쉼표로 구분된 목록.                                                                         |
| <code>--packages</code>             | 드라이버 및 실행기 클래스 경로에 포함할 jar의 maven 좌표에 대한 쉼표로 구분된 목록.                                                           |
| <code>--exclude-packages</code>     | 종속 항목 충돌을 피하기 위해 <code>-packages</code> 에서 제공하는 종속 항목을 해결하는 동안 제외할 쉼표로 구분된 <code>groupId:artifactId</code> 목록. |
| <code>--repositories</code>         | <code>-packages</code> 로 지정된 maven 좌표를 검색하기 위한 추가 원격 리포지토리의 쉼표로 구분된 목록.                                        |
| <code>--files FILES</code>          | 각 실행기의 작업 디렉터리에 배치할 쉼표로 구분된 파일 목록.                                                                             |
| <code>--conf PROP=VALUE</code>      | Spark 구성 속성.                                                                                                   |
| <code>--properties-file FILE</code> | 추가 속성을 로드할 파일의 경로.                                                                                             |

| 옵션                         | 설명                                  |
|----------------------------|-------------------------------------|
| --driver-memory MEM        | 드라이버의 메모리. 기본 1,024MB.              |
| --driver-java-options      | 드라이버에 전달할 추가 Java 옵션.               |
| --driver-library-path      | 드라이버에 전달할 추가 라이브러리 경로 항목.           |
| --driver-class-path        | 드라이버에 전달할 추가 클래스 경로 항목.             |
| --executor-memory MEM      | 실행기당 메모리. 기본 1GB.                   |
| --driver-cores NUM         | 드라이버가 사용하는 코어 수.                    |
| --total-executor-cores NUM | 모든 실행기의 총 코어 수.                     |
| --executor-cores NUM       | 각 실행기가 사용하는 코어 수.                   |
| --num-executors NUM        | 실행할 실행기 수.                          |
| -hivevar <key=value>       | Hive 명령에 적용할 변수 대체(예: -hivevar A=B) |
| -hiveconf <property=value> | 지정된 속성에 사용할 값.                      |

Spark SQL 작업의 경우 다음 예제와 같이 start-job-run-request.json 파일을 생성하고 작업 실행에 필요한 파라미터를 지정합니다.

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-6.7.0-latest",
  "jobDriver": {
    "sparkSqlJobDriver": {
      "entryPoint": "entryPoint_location",
      "sparkSqlParameters": "--conf spark.executor.instances=2 --conf
spark.executor.memory=2G --conf spark.executor.cores=2 --conf spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
```

```

    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory": "2G"
        }
      }
    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://my_s3_log_location"
      }
    }
  }
}

```

## 작업 실행 상태

Amazon EMR on EKS 작업 대기열에 작업을 제출하면 작업이 PENDING 상태로 전환됩니다. 그런 다음 작업은 성공(0 코드로 종료)하거나 실패(0이 아닌 코드로 종료)할 때까지 다음 상태를 통과합니다.

다음은 가능한 작업 실행 상태입니다.

- PENDING - Amazon EMR on EKS에 작업 실행이 제출될 때 초기 작업 상태입니다. 작업이 가상 클러스터에 제출되기를 기다리고 있으며 Amazon EMR on EKS는 이 작업을 제출하기 위해 작업 중입니다.
- SUBMITTED - 가상 클러스터에 성공적으로 제출된 작업 실행. 그러면 클러스터 스케줄러가 클러스터에서 이 작업을 실행하려고 시도합니다.
- RUNNING - 가상 클러스터에서 실행 중인 작업 실행. Spark 애플리케이션에서 이는 Spark 드라이버 프로세스가 running 상태임을 의미합니다.
- FAILED - 가상 클러스터에 제출하지 못했거나 성공적으로 완료되지 못한 작업 실행. 이 작업 실패에 대한 추가 정보를 찾으려면 StateDetails 및 FailureReason을 참조하세요.
- COMPLETED - 성공적으로 완료된 작업 실행.
- CANCEL\_PENDING - 취소가 요청된 작업 실행. Amazon EMR on EKS가 가상 클러스터에서 작업을 취소하려고 합니다.

- CANCELLED - 취소된 작업 실행.

## Amazon EMR 콘솔에서 작업 보기

작업 실행 데이터는 볼 수 없으므로 각 작업이 상태를 통과할 때 모니터링할 수 있습니다. Amazon EMR 콘솔에서 작업을 보려면 다음 단계를 수행합니다.

1. Amazon EMR 콘솔 왼쪽 메뉴의 Amazon EMR on EKS에서 가상 클러스터를 선택합니다.
2. 가상 클러스터 목록에서 작업을 보려는 가상 클러스터를 선택합니다.
3. 작업 실행 테이블에서 로그 보기를 선택하여 작업 실행의 세부 정보를 확인합니다.

### Note

원클릭 경험에 대한 지원은 기본적으로 활성화되어 있습니다. 작업 제출 중에 monitoringConfiguration에서 persistentAppUI를 DISABLED로 설정하여 이 기능을 끌 수 있습니다. 자세한 내용은 [영구 애플리케이션 사용자 인터페이스 보기](#)를 참조하세요.

## 작업 실행 시 발생하는 일반적인 오류

StartJobRun API를 실행할 때 다음 오류가 발생할 수 있습니다. 이 표에서는 각 오류를 나열하며, 문제를 빠르게 해결할 수 있도록 완화 단계를 제공합니다.

| 오류 메시지                                                                                                                                                         | 오류 조건             | 권장되는 다음 단계                                                 |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|------------------------------------------------------------|
| error: argument <b>--argument</b> is required                                                                                                                  | 필수 파라미터가 누락되었습니다. | 누락된 인수를 API 요청에 추가합니다.                                     |
| An error occurred (AccessDeniedException) when calling the StartJobRun operation:<br>User: <b>ARN</b> is not authorized to perform: emr-containers:StartJobRun | 실행 역할이 누락되었습니다.   | <a href="#">Amazon EMR on EKS에서 작업 실행 역할 사용</a> 사용을 참조하세요. |

| 오류 메시지                                                                                                                                                                                                                            | 오류 조건                                                            | 권장되는 다음 단계                                                                          |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| An error occurred (AccessDeniedException) when calling the StartJobRun operation:<br>User: <b>ARN</b> is not authorized to perform: emr-containers:StartJobRun                                                                    | 직접 호출자에게 조건 키를 통한 실행 역할[유효한 형식 또는 유효하지 않은 형식]에 대한 권한이 없습니다.      | <a href="#">Amazon EMR on EKS에서 작업 실행 역할 사용</a> 을(를) 참조하세요.                         |
| An error occurred (AccessDeniedException) when calling the StartJobRun operation:<br>User: <b>ARN</b> is not authorized to perform: emr-containers:StartJobRun                                                                    | Job 제출자와 실행 역할 ARN은 서로 다른 계정에 속해 있습니다.                           | 작업 제출자와 실행 역할 ARN이 동일한 AWS 계정에서 생성되었는지 확인합니다.                                       |
| 1 validation error detected:<br>Value <b>Role</b> at 'executionRoleArn' failed to satisfy the ARN regular expression pattern: ^arn:(aws[a-zA-Z0-9-]*):iam::(\d{12})?:(role(\u002F) (\u002F\u0021-\u002F)+\u002F))([w+=,.\u002F]+) | 직접 호출자는 조건 키를 통해 실행 역할에 대한 권한을 갖지만 역할이 ARN 형식의 제약 조건을 충족하지 않습니다. | ARN 형식 뒤에 실행 역할을 제공합니다. <a href="#">Amazon EMR on EKS에서 작업 실행 역할 사용</a> 을(를) 참조하세요. |
| An error occurred (ResourceNotFoundException) when calling the StartJobRun operation: Virtual cluster <b>Virtual Cluster ID</b> doesn't exist.                                                                                    | 가상 클러스터 ID를 찾을 수 없습니다.                                           | Amazon EMR on EKS에 등록된 가상 클러스터 ID를 제공합니다.                                           |
| An error occurred (ValidationException) when calling the StartJobRun operation: Virtual cluster state <b>state</b> is not valid to create resource JobRun.                                                                        | 가상 클러스터에서 작업을 실행할 준비가 되지 않았습니다.                                  | <a href="#">가상 클러스터 상태</a> 을(를) 참조하세요.                                              |

| 오류 메시지                                                                                                                                                                                                        | 오류 조건                            | 권장되는 다음 단계                                                 |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------|------------------------------------------------------------|
| An error occurred (ResourceNotFoundException) when calling the StartJobRun operation: Release <b>RELEASE</b> doesn't exist.                                                                                   | 작업 제출에 지정된 릴리스가 올바르지 않습니다.       | <a href="#">Amazon EMR on EKS 릴리스</a> (를) 참조하세요.           |
| An error occurred (AccessDeniedException) when calling the StartJobRun operation:<br>User: <b>ARN</b> is not authorized to perform: emr-containers:StartJobRun on resource: <b>ARN</b> with an explicit deny. | 사용자에게 StartJobRun을 호출할 권한이 없습니다. | <a href="#">Amazon EMR on EKS에서 작업 실행 역할 사용</a> (를) 참조하세요. |
| An error occurred (AccessDeniedException) when calling the StartJobRun operation:<br>User: <b>ARN</b> is not authorized to perform: emr-containers:StartJobRun on resource: <b>ARN</b>                        |                                  |                                                            |
| An error occurred (ValidationException) when calling the StartJobRun operation: configurationOverrides.monitoringConfiguration.s3MonitoringConfiguration.logUri failed to satisfy constraint : %s             | S3 경로 URI 구문이 유효하지 않습니다.         | logUri는 s3://... 형식이어야 함                                   |

작업을 실행하기 전에 DescribeJobRun API를 실행할 때 다음 오류가 발생할 수 있습니다.

| 오류 메시지                       | 오류 조건                         | 권장되는 다음 단계                                       |
|------------------------------|-------------------------------|--------------------------------------------------|
| 상태 세부 정보: JobRun 제출이 실패했습니다. | StartJobRun의 파라미터가 유효하지 않습니다. | <a href="#">Amazon EMR on EKS 릴리스</a> (를) 참조하세요. |

| 오류 메시지                                                                                                                                        | 오류 조건                                 | 권장되는 다음 단계                                                                                               |
|-----------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------|----------------------------------------------------------------------------------------------------------|
| <p>Classification <i>classification</i> not supported.</p> <p>failureReason: VALIDATION_ERROR</p> <p>state: FAILED.</p>                       |                                       |                                                                                                          |
| <p>stateDetails: Cluster <i>EKS Cluster ID</i> does not exist.</p> <p>failureReason: CLUSTER_UNAVAILABLE</p> <p>state: FAILED</p>             | EKS 클러스터를 사용할 수 없습니다.                 | EKS 클러스터가 존재하고 올바른 권한을 보유하고 있는지 확인합니다. 자세한 내용은 <a href="#">Amazon EMR on EKS 설정</a> 단원을 참조하십시오.          |
| <p>상태 세부 정보: 클러스터 <i>EKS #### ID#</i> 충분한 권한이 없습니다.</p> <p>failureReason: CLUSTER_UNAVAILABLE</p> <p>state: FAILED</p>                        | Amazon EMR에는 EKS 클러스터에 액세스할 권한이 없습니다. | 등록된 네임스페이스에서 Amazon EMR에 대한 권한이 설정되어 있는지 확인합니다. 자세한 내용은 <a href="#">Amazon EMR on EKS 설정</a> 단원을 참조하십시오. |
| <p>stateDetails: Cluster <i>EKS Cluster ID</i> is currently not reachable.</p> <p>failureReason: CLUSTER_UNAVAILABLE</p> <p>state: FAILED</p> | EKS 클러스터에 연결할 수 없습니다.                 | EKS 클러스터가 존재하고 올바른 권한을 보유하고 있는지 확인합니다. 자세한 내용은 <a href="#">Amazon EMR on EKS 설정</a> 단원을 참조하십시오.          |



| 오류 메시지                                                                                                                                       | 오류 조건                               | 권장되는 다음 단계                                                                                      |
|----------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------|-------------------------------------------------------------------------------------------------|
| <p>stateDetails: JobRun submission failed due to an internal error.</p> <p>failureReason: INTERNAL_ERROR</p> <p>state: FAILED</p>            | EKS 클러스터에서 내부 오류가 발생했습니다.           | 해당 사항 없음                                                                                        |
| <p>stateDetails: Cluster <i>EKS Cluster ID</i> does not have sufficient resources.</p> <p>failureReason: USER_ERROR</p> <p>state: FAILED</p> | EKS 클러스터의 리소스가 부족하여 작업을 실행할 수 없습니다. | EKS 노드 그룹에 용량을 추가하거나 EKS Autoscaler를 설정합니다. 자세한 내용은 <a href="#">Cluster Autoscaler</a> 를 참조하세요. |

작업을 실행한 후에 DescribeJobRun API를 실행할 때 다음 오류가 발생할 수 있습니다.

| 오류 메시지                                                                                                                                                                 | 오류 조건                                 | 권장되는 다음 단계                                                                                      |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------|-------------------------------------------------------------------------------------------------|
| <p>stateDetails: JobRun을 모니터링하는 동안 문제가 발생했습니다.</p> <p>Cluster <i>EKS Cluster ID</i> does not exist.</p> <p>failureReason: CLUSTER_UNAVAILABLE</p> <p>state: FAILED</p> | EKS 클러스터가 존재하지 않습니다.                  | EKS 클러스터가 존재하고 올바른 권한을 보유하고 있는지 확인합니다. 자세한 내용은 <a href="#">Amazon EMR on EKS 설정</a> 단원을 참조하십시오. |
| <p>stateDetails: JobRun을 모니터링하는 동안 문제가 발생했습니다.</p>                                                                                                                     | Amazon EMR에는 EKS 클러스터에 액세스할 권한이 없습니다. | 등록된 네임스페이스에서 Amazon EMR에 대한 권한이 설정되어 있는지 확인합니다. 자세한 내용은 <a href="#">Amazon EMR</a>              |

| 오류 메시지                                                                                                                                                                             | 오류 조건                                | 권장되는 다음 단계                                                                                          |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------|-----------------------------------------------------------------------------------------------------|
| <p>Cluster <i>EKS Cluster ID</i> does not have sufficient permissions.</p> <p>failureReason: CLUSTER_UNAVAILABLE</p> <p>state: FAILED</p>                                          |                                      | <p><a href="#">on EKS 설정</a> 단원을 참조하십시오.</p>                                                        |
| <p>stateDetails: JobRun을 모니터링하는 동안 문제가 발생했습니다.</p> <p>Cluster <i>EKS Cluster ID</i> is currently not reachable.</p> <p>failureReason: CLUSTER_UNAVAILABLE</p> <p>state: FAILED</p> | EKS 클러스터에 연결할 수 없습니다.                | <p>EKS 클러스터가 존재하고 올바른 권한을 보유하는지 확인합니다. 자세한 내용은 <a href="#">Amazon EMR on EKS 설정</a> 단원을 참조하십시오.</p> |
| <p>stateDetails: Trouble monitoring your JobRun due to an internal error</p> <p>failureReason: INTERNAL_ERROR</p> <p>state: FAILED</p>                                             | 내부 오류가 발생하여 JobRun 모니터링을 수행할 수 없습니다. | 해당 사항 없음                                                                                            |

작업을 시작할 수 없고 작업이 제출된 상태에서 15분 동안 대기하는 경우 다음 오류가 발생할 수 있습니다. 클러스터 리소스가 부족하기 때문일 수 있습니다.

| 오류 메시지          | 오류 조건                       | 권장되는 다음 단계                                      |
|-----------------|-----------------------------|-------------------------------------------------|
| cluster timeout | 작업이 제출됨 상태로 15분 이상 지속되었습니다. | 아래 표시된 구성 재정의로 이 파라미터의 기본 설정인 15분을 재정의할 수 있습니다. |

다음 구성을 사용하여 클러스터 제한 시간 설정을 30분으로 변경합니다. 새 job-start-timeout 값은 초 단위로 제공합니다.

```
{
  "configurationOverrides": {
    "applicationConfiguration": [{
      "classification": "emr-containers-defaults",
      "properties": {
        "job-start-timeout": "1800"
      }
    }]
  }
}
```

## 작업 템플릿 사용

작업 템플릿은 작업 실행 시작 시 StartJobRun API 간접 호출 간에 공유할 수 있는 값을 저장합니다. 다음과 같은 두 가지 사용 사례를 지원합니다.

- 반복되는 StartJobRun API 요청 값을 방지하려는 경우.
- StartJobRun API 요청을 통해 특정 값을 제공해야 한다는 규칙을 적용하려는 경우.

작업 템플릿을 사용하면 작업 실행에서 재사용 가능한 템플릿을 정의하여 추가 사용자 지정을 적용할 수 있습니다. 예를 들어, 다음과 같습니다.

- 실행기 및 드라이버 컴퓨팅 용량 구성
- IAM 역할과 같은 보안 및 거버넌스 속성 설정
- 여러 애플리케이션 및 데이터 파이프라인에서 사용하도록 도커 이미지 사용자 지정

다음 주제에서는 템플릿을 사용하여 작업 실행을 시작하는 방법 및 템플릿 파라미터를 변경하는 방법을 포함하여 템플릿 사용에 대한 자세한 정보를 제공합니다.

## 주제

- [작업 템플릿을 생성 및 사용하여 작업 실행 시작](#)
- [작업 템플릿 파라미터 정의](#)
- [작업 템플릿에 대한 액세스 제어](#)

## 작업 템플릿을 생성 및 사용하여 작업 실행 시작

이 섹션에서는 작업 템플릿을 생성하고 템플릿을 사용하여 AWS Command Line Interface ()로 작업 실행을 시작하는 방법을 설명합니다AWS CLI.

### 작업 템플릿을 생성하는 방법

1. 다음 예제 JSON 파일에서 볼 수 있듯이 `create-job-template-request.json` 파일을 생성하고 작업 실행에 필요한 파라미터를 지정합니다. 사용 가능한 모든 파라미터에 대한 자세한 내용은 [CreateJobTemplate](#) API를 참조하세요.

StartJobRun API에 필요한 대부분의 값은 `jobTemplateData`에서도 필요합니다. 작업 템플릿을 사용하여 StartJobRun을 간접 호출할 때 파라미터에 대해 자리 표시자를 사용하고 값을 제공하려는 경우 작업 템플릿 파라미터에 대한 다음 섹션을 참조하세요.

```
{
  "name": "mytemplate",
  "jobTemplateData": {
    "executionRoleArn": "iam_role_arn_for_job_execution",
    "releaseLabel": "emr-6.7.0-latest",
    "jobDriver": {
      "sparkSubmitJobDriver": {
        "entryPoint": "entryPoint_location",
        "entryPointArguments": [ "argument1", "argument2", ... ],
        "sparkSubmitParameters": "--class <main_class> --conf
spark.executor.instances=2 --conf spark.executor.memory=2G --conf
spark.executor.cores=2 --conf spark.driver.cores=1"
      }
    },
    "configurationOverrides": {
      "applicationConfiguration": [
        {
          "classification": "spark-defaults",
          "properties": {
            "spark.driver.memory": "2G"
          }
        }
      ]
    }
  }
}
```

```

        }
    },
    "monitoringConfiguration": {
        "persistentAppUI": "ENABLED",
        "cloudWatchMonitoringConfiguration": {
            "logGroupName": "my_log_group",
            "logStreamNamePrefix": "log_stream_prefix"
        },
        "s3MonitoringConfiguration": {
            "logUri": "s3://my_s3_log_location/"
        }
    }
}
}
}
}

```

2. 로컬에 저장된 create-job-template-request.json 파일 경로와 함께 create-job-template 명령을 사용합니다.

```

aws emr-containers create-job-template \
--cli-input-json file:///./create-job-template-request.json

```

작업 템플릿을 사용하여 작업 실행을 시작하는 방법

다음 예제와 같이 StartJobRun 명령에 가상 클러스터 ID, 작업 템플릿 ID 및 작업 이름을 제공합니다.

```

aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
--name myjob \
--job-template-id 1234abcd

```

## 작업 템플릿 파라미터 정의

작업 템플릿 파라미터를 사용하면 작업 템플릿에서 변수를 지정할 수 있습니다. 해당 작업 템플릿을 사용하여 작업 실행을 시작할 때 이러한 파라미터 값을 지정해야 합니다. Job 템플릿 파라미터는 \${parameterName} 형식으로 지정됩니다. jobTemplateData 필드의 값을 작업 템플릿 파라미터로 지정하도록 선택할 수 있습니다. 각 작업 템플릿 파라미터 변수에 대해 해당 데이터 유형(String

또는 NUMBER)과 기본값(선택 사항)을 지정합니다. 아래 예제에서는 진입점 위치, 기본 클래스 및 S3 로그 위치 값에 대한 작업 템플릿 파라미터를 지정하는 방법을 보여줍니다.

진입점 위치, 기본 클래스 및 Amazon S3 로그 위치를 작업 템플릿 파라미터로 지정하는 방법

1. 다음 예제 JSON 파일에서 볼 수 있듯이 `create-job-template-request.json` 파일을 생성하고 작업 실행에 필요한 파라미터를 지정합니다. 파라미터에 대한 자세한 내용은 [CreateJobTemplate](#) API를 참조하세요.

```
{
  "name": "mytemplate",
  "jobTemplateData": {
    "executionRoleArn": "iam_role_arn_for_job_execution",
    "releaseLabel": "emr-6.7.0-latest",
    "jobDriver": {
      "sparkSubmitJobDriver": {
        "entryPoint": "${EntryPointLocation}",
        "entryPointArguments": [ "argument1", "argument2", ... ],
        "sparkSubmitParameters": "--class ${MainClass} --conf
spark.executor.instances=2 --conf spark.executor.memory=2G --conf
spark.executor.cores=2 --conf spark.driver.cores=1"
      }
    },
    "configurationOverrides": {
      "applicationConfiguration": [
        {
          "classification": "spark-defaults",
          "properties": {
            "spark.driver.memory": "2G"
          }
        }
      ],
      "monitoringConfiguration": {
        "persistentAppUI": "ENABLED",
        "cloudWatchMonitoringConfiguration": {
          "logGroupName": "my_log_group",
          "logStreamNamePrefix": "log_stream_prefix"
        },
        "s3MonitoringConfiguration": {
          "logUri": "${LogS3BucketUri}"
        }
      }
    }
  },
}
```

```

    "parameterConfiguration": {
      "EntryPointLocation": {
        "type": "STRING"
      },
      "MainClass": {
        "type": "STRING",
        "defaultValue": "Main"
      },
      "LogS3BucketUri": {
        "type": "STRING",
        "defaultValue": "s3://my_s3_log_location/"
      }
    }
  }
}

```

2. 로컬로 저장되었거나 Amazon S3에 저장된 `create-job-template-request.json` 파일 경로와 함께 `create-job-template` 명령을 사용합니다.

```

aws emr-containers create-job-template \
  --cli-input-json file:///./create-job-template-request.json

```

작업 템플릿 파라미터가 포함된 작업 템플릿을 사용하여 작업을 시작하는 방법

작업 템플릿 파라미터가 포함된 작업 템플릿으로 작업 실행을 시작하려면 아래와 같이 `StartJobRun` API 요청에서 작업 템플릿 ID와 작업 템플릿 파라미터 값을 지정합니다.

```

aws emr-containers start-job-run \
  --virtual-cluster-id 123456 \
  --name myjob \
  --job-template-id 1234abcd \
  --job-template-parameters '{"EntryPointLocation": "entry_point_location", "MainClass": "ExampleMainClass", "LogS3BucketUri": "s3://example_s3_bucket/"}'

```

## 작업 템플릿에 대한 액세스 제어

`StartJobRun` 정책을 사용하면 사용자나 역할이 지정한 작업 템플릿만 사용하여 작업을 실행할 수 있고 지정된 작업 템플릿을 사용하지 않고는 `StartJobRun` 작업을 실행할 수 없도록 적용할 수 있습니다. 이렇게 하려면 먼저 아래와 같이 사용자 또는 역할에 지정된 작업 템플릿에 대한 읽기 권한을 부여해야 합니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DescribeJobRun"
      ],
      "Resource": [
        "arn:aws:emr-containers:*:*:jobtemplate/job_template_1_id",
        "arn:aws:emr-containers:*:*:jobtemplate/job_template_2_id"
      ],
      "Sid": "AllowEMRCONTAINERSDescribejobtemplate"
    }
  ]
}
```

사용자 또는 역할이 지정된 작업 템플릿을 사용할 때만 StartJobRun 작업을 간접 호출할 수 있도록 하려면 지정된 사용자 또는 역할에 다음 StartJobRun 정책 권한을 할당할 수 있습니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:StartJobRun"
      ],
      "Resource": [
        "arn:aws:emr-containers:*:*/virtualclusters/virtual_cluster_id"
      ],
      "Condition": {
        "ArnLike": {
          "emr-containers:JobTemplateArn": [
            "arn:aws:emr-containers:*:*:jobtemplate/job_template_1_id",
            "arn:aws:emr-containers:*:*:jobtemplate/job_template_2_id"
          ]
        }
      }
    }
  ]
}
```



```

    ]
  }
},
  "Sid": "AllowEMRCONTAINERSStartjobrun"
}
]
}

```

작업 템플릿이 실행 역할 ARN 필드 내에 작업 템플릿 파라미터를 지정하는 경우 사용자는 이 파라미터의 값을 제공할 수 있으며, 이를 통해 임의의 실행 역할을 사용해 StartJobRun을 간접 호출할 수 있습니다. 사용자가 제공할 수 있는 실행 역할을 제한하려면 [Amazon EMR on EKS에서 작업 실행 역할 사용](#)에서 실행 역할에 대한 액세스 제어를 참조하세요.

위의 StartJobRun 작업 정책에서 지정된 사용자 또는 역할에 대한 조건이 지정되지 않은 경우 사용자 또는 역할은 읽기 권한이 있는 임의의 작업 템플릿을 사용하거나 임의의 실행 역할을 사용하여 지정된 가상 클러스터에서 StartJobRun 작업을 간접 호출할 수 있습니다.

## 포드 템플릿 사용

Amazon EMR 버전 5.33.0 또는 6.3.0부터 Amazon EMR on EKS는 Spark의 포드 템플릿 기능을 지원합니다. 포드는 공유 스토리지 및 네트워크 리소스가 있는 하나 이상의 컨테이너로 구성된 그룹이자, 컨테이너 실행 방법에 대한 사양입니다. 포드 템플릿은 각 포드의 실행 방법을 결정하는 사양입니다. 포드 템플릿 파일을 사용하여 Spark 구성이 지원하지 않는 드라이버 또는 실행기 포드 구성을 정의할 수 있습니다. Spark의 포드 템플릿 기능에 대한 자세한 내용은 [Pod Template](#)을 참조하세요.

### Note

포드 템플릿 기능은 드라이버 및 실행기 포드에서만 작동합니다. 포드 템플릿을 사용하여 작업 제출기 포드를 구성할 수 없습니다.

## 일반적인 시나리오

Amazon EMR on EKS에서 포드 템플릿을 사용해 공유 EKS 클러스터에서 Spark 작업을 실행하는 방법을 정의하고 비용을 절감하며 리소스 사용률과 성능을 개선할 수 있습니다.

- 비용을 절감하기 위해 Amazon EC2 스팟 인스턴스에서 실행하도록 Spark 실행기 작업을 예약하는 동시에, Amazon EC2 온디맨드 인스턴스에서 실행하도록 Spark 드라이버 작업을 예약할 수 있습니다.

- 리소스 사용률을 높이기 위해 동일한 EKS 클러스터에서 워크로드를 실행하는 여러 팀을 지원할 수 있습니다. 각 팀은 워크로드를 실행하기 위해 지정된 Amazon EC2 노드 그룹을 보유합니다. 포드 템플릿을 사용하여 해당 워크로드에 해당하는 허용 범위를 적용할 수 있습니다.
- 모니터링을 개선하기 위해 별도의 로깅 컨테이너를 실행하여 기존 모니터링 애플리케이션에 로그를 전달할 수 있습니다.

예를 들어, 다음 포드 템플릿 파일은 일반적인 사용 시나리오를 보여줍니다.

```
apiVersion: v1
kind: Pod
spec:
  volumes:
    - name: source-data-volume
      emptyDir: {}
    - name: metrics-files-volume
      emptyDir: {}
  nodeSelector:
    eks.amazonaws.com/nodegroup: emr-containers-nodegroup
  containers:
    - name: spark-kubernetes-driver # This will be interpreted as driver Spark main
      container
      env:
        - name: RANDOM
          value: "random"
      volumeMounts:
        - name: shared-volume
          mountPath: /var/data
        - name: metrics-files-volume
          mountPath: /var/metrics/data
    - name: custom-side-car-container # Sidecar container
      image: <side_car_container_image>
      env:
        - name: RANDOM_SIDE CAR
          value: random
      volumeMounts:
        - name: metrics-files-volume
          mountPath: /var/metrics/data
      command:
        - /bin/sh
        - '-c'
        - <command-to-upload-metrics-files>
  initContainers:
```

```
- name: spark-init-container-driver # Init container
  image: <spark-pre-step-image>
  volumeMounts:
    - name: source-data-volume # Use EMR predefined volumes
      mountPath: /var/data
  command:
    - /bin/sh
    - '-c'
    - <command-to-download-dependency-jars>
```

포드 템플릿은 다음 작업을 완료합니다.

- Spark 기본 컨테이너가 시작되기 전에 실행되는 새 [init 컨테이너](#)를 추가합니다. init 컨테이너는 Spark 기본 컨테이너와 [EmptyDir 볼륨](#)(source-data-volume)을 공유합니다. init 컨테이너를 통해 종속성 다운로드 또는 입력 데이터 생성과 같은 초기화 단계를 실행할 수 있습니다. 그러면 Spark 기본 컨테이너에서 데이터를 소비합니다.
- Spark 기본 컨테이너와 함께 실행되는 다른 [sidecar 컨테이너](#)를 추가합니다. 두 컨테이너는 다른 EmptyDir 볼륨(metrics-files-volume)을 공유합니다. Spark 작업은 Prometheus 지표와 같은 지표를 생성할 수 있습니다. 그러면 Spark 작업에서 지표를 파일에 넣고, 향후 분석을 위해 sidecar 컨테이너에서 파일을 자체 BI 시스템에 업로드하게 합니다.
- Spark 기본 컨테이너에 새 환경 변수를 추가합니다. 작업에서 환경 변수를 사용하게 할 수 있습니다.
- [노드 선택기](#)를 정의합니다. 그러면 포드는 emr-containers-nodegroup 노드 그룹에서만 예약됩니다. 그러면 작업과 팀 사이에서 컴퓨팅 리소스를 분리할 수 있습니다.

## Amazon EMR on EKS에서 포드 템플릿 활성화

Amazon EMR on EKS에서 포드 템플릿 기능을 활성화하려면 Amazon S3의 포드 템플릿 파일을 가리키도록 Spark 속성 `spark.kubernetes.driver.podTemplateFile` 및 `spark.kubernetes.executor.podTemplateFile`을 구성합니다. 그러면 Spark는 포드 템플릿 파일을 다운로드하고 이를 사용하여 드라이버 및 실행기 포드를 구성합니다.

### Note

Spark는 작업 실행 역할을 사용하여 포드 템플릿을 로드하므로 작업 실행 역할에는 Amazon S3에 액세스하여 포드 템플릿을 로드할 수 있는 권한이 있어야 합니다. 자세한 내용은 [작업 실행 역할 생성](#) 단원을 참조하십시오.

다음 작업 실행 JSON 파일에서 볼 수 있듯이 SparkSubmitParameters를 사용하여 포드 템플릿에 대한 Amazon S3 경로를 지정할 수 있습니다.

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "release_label",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2", ...],
      "sparkSubmitParameters": "--class <main_class> \
        --conf
spark.kubernetes.driver.podTemplateFile=s3://path_to_driver_pod_template \
        --conf
spark.kubernetes.executor.podTemplateFile=s3://path_to_executor_pod_template \
        --conf spark.executor.instances=2 \
        --conf spark.executor.memory=2G \
        --conf spark.executor.cores=2 \
        --conf spark.driver.cores=1"
    }
  }
}
```

또는 다음 작업 실행 JSON 파일에서 볼 수 있듯이 configurationOverrides를 사용하여 포드 템플릿에 대한 Amazon S3 경로를 지정할 수 있습니다.

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "release_label",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2", ...],
      "sparkSubmitParameters": "--class <main_class> \
        --conf spark.executor.instances=2 \
        --conf spark.executor.memory=2G \
        --conf spark.executor.cores=2 \
        --conf spark.driver.cores=1"
    }
  }
}
```

```

    },
    "configurationOverrides": {
      "applicationConfiguration": [
        {
          "classification": "spark-defaults",
          "properties": {
            "spark.driver.memory": "2G",
            "spark.kubernetes.driver.podTemplateFile": "s3://path_to_driver_pod_template",
            "spark.kubernetes.executor.podTemplateFile": "s3://path_to_executor_pod_template"
          }
        }
      ]
    }
  }
}

```

#### Note

1. Amazon EMR on EKS에서 포드 템플릿 기능을 사용하는 경우 신뢰할 수 없는 애플리케이션 코드 격리와 같은 보안 지침을 준수해야 합니다. 자세한 내용은 [Amazon EMR on EKS 보안 모범 사례](#) 단원을 참조하십시오.
2. Spark 기본 컨테이너 이름은 `spark.kubernetes.driver.podTemplateContainerName` 및 `spark.kubernetes.executor.podTemplateContainerName`을 사용하여 변경할 수 없습니다. 해당 이름은 `spark-kubernetes-driver` 및 `spark-kubernetes-executors`로 하드 코딩되기 때문입니다. Spark 기본 컨테이너를 사용자 지정하려면 이러한 하드 코딩된 이름을 사용하여 포드 템플릿에서 컨테이너를 지정해야 합니다.

## 포드 템플릿 필드

Amazon EMR on EKS에서 포드 템플릿을 구성할 때는 다음 필드 제한 사항을 고려합니다.

- Amazon EMR on EKS는 적절한 작업 예약을 지원하기 위해 포드 템플릿에서 다음 필드만 허용합니다.

허용되는 포드 수준 필드는 다음과 같습니다.

- `apiVersion`
- `kind`

- metadata
- spec.activeDeadlineSeconds
- spec.affinity
- spec.containers
- spec.enableServiceLinks
- spec.ephemeralContainers
- spec.hostAliases
- spec.hostname
- spec.imagePullSecrets
- spec.initContainers
- spec.nodeName
- spec.nodeSelector
- spec.overhead
- spec.preemptionPolicy
- spec.priority
- spec.priorityClassName
- spec.readinessGates
- spec.runtimeClassName
- spec.schedulerName
- spec.subdomain
- spec.terminationGracePeriodSeconds
- spec.tolerations
- spec.topologySpreadConstraints
- spec.volumes

허용되는 Spark 기본 컨테이너 수준 필드는 다음과 같습니다.

- env
- envFrom
- name

- Lifecycle
- livenessProbe

- readinessProbe
- resources
- startupProbe
- stdin
- stdinOnce
- terminationMessagePath
- terminationMessagePolicy
- tty
- volumeDevices
- volumeMounts
- workingDir

포드 템플릿에서 허용되지 않는 필드를 사용하면 Spark에서 예외가 발생하고 작업이 실패합니다. 다음 예제에서는 허용되지 않는 필드로 인해 Spark 컨트롤러 로그에 표시되는 오류 메시지를 보여줍니다.

```
Executor pod template validation failed.
Field container.command in Spark main container not allowed but specified.
```

- Amazon EMR on EKS에서는 포드 템플릿에서 다음 파라미터를 사전에 정의합니다. 포드 템플릿에서 지정하는 필드는 이 필드와 겹치지 않아야 합니다.

사전 정의된 볼륨 이름은 다음과 같습니다.

- emr-container-communicate
- config-volume
- emr-container-application-log-dir
- emr-container-event-log-dir
- temp-data-dir
- mnt-dir
- home-dir
- emr-container-s3

다음은 Spark 기본 컨테이너에만 적용되는 사전 정의된 볼륨 마운트입니다.

포드 템플릿 필드

- Name: emr-container-communicate, MountPath: /var/log/fluentd

- Name: `emr-container-application-log-dir`, MountPath: `/var/log/spark/user`
- Name: `emr-container-event-log-dir`, MountPath: `/var/log/spark/apps`
- Name: `mnt-dir`, MountPath: `/mnt`
- Name: `temp-data-dir`, MountPath: `/tmp`
- Name: `home-dir`, MountPath: `/home/hadoop`

다음은 Spark 기본 컨테이너에만 적용되는 사전 정의된 환경 변수입니다.

- `SPARK_CONTAINER_ID`
- `K8S_SPARK_LOG_URL_STDERR`
- `K8S_SPARK_LOG_URL_STDOUT`
- `SIDECAR_SIGNAL_FILE`

#### Note

이러한 사전 정의된 볼륨을 계속 사용하고 추가 sidecar 컨테이너에 마운트할 수 있습니다. 예를 들어 `emr-container-application-log-dir`을 사용하고 포드 템플릿에 정의된 sidecar 컨테이너에 이를 마운트할 수 있습니다.

지정한 필드가 포드 템플릿의 사전 정의된 필드와 충돌하는 경우 Spark에서 예외가 발생하고 작업이 실패합니다. 다음 예제는 사전 정의된 필드와의 충돌로 인한 Spark 애플리케이션 로그에 표시되는 오류 메시지를 보여줍니다.

```
Defined volume mount path on main container must not overlap with reserved mount paths: [<reserved-paths>]
```

## sidecar 컨테이너 관련 고려 사항

Amazon EMR은 Amazon EMR on EKS에서 프로비저닝하는 포드의 수명 주기를 제어합니다. sidecar 컨테이너는 Spark 기본 컨테이너와 동일한 수명 주기를 따라야 합니다. 추가 sidecar 컨테이너를 포드에 삽입하는 경우 Amazon EMR에서 정의하는 포드 수명 주기 관리와 통합하여 Spark 기본 컨테이너가 종료될 때 sidecar 컨테이너가 자체적으로 중지될 수 있도록 하는 것이 좋습니다.

비용을 줄이려면 작업이 완료된 후 sidecar 컨테이너가 있는 드라이버 포드가 계속 실행되지 않도록 하는 프로세스를 구현하는 것이 좋습니다. Spark 드라이버는 실행기가 완료되면 실행기 포드를 삭제



합니다. 하지만 드라이버 프로그램이 완료되어도 추가 sidecar 컨테이너는 계속 실행됩니다. 포드 요금은 Amazon EMR on EKS가 드라이버 포드를 정리할 때까지 청구되며, 일반적으로 드라이버 Spark 기본 컨테이너가 완료된 후 1분 이내에 청구됩니다. 비용을 절감하기 위해 다음 섹션에 설명된 대로 Amazon EMR on EKS에서 이 드라이버 및 실행자 포드 모두에 대해 정의한 수명 주기 관리 메커니즘과 추가 sidecar 컨테이너를 통합할 수 있습니다.

드라이버 및 실행기 포드의 Spark 기본 컨테이너는 2초마다 heartbeat를 /var/log/fluentd/main-container-terminated 파일에 전송합니다. Amazon EMR의 사전 정의된 emr-container-communicate 볼륨 마운트를 sidecar 컨테이너에 추가하면 sidecar 컨테이너의 하위 프로세스를 정의하여 이 파일의 마지막 수정 시간을 주기적으로 추적할 수 있습니다. 그런 다음, Spark 기본 컨테이너에서 오랫동안 heartbeat를 중지한 것을 발견하면 하위 프로세스가 저절로 중지됩니다.

다음 예제는 하트비트 파일을 추적하고 자체적으로 중지하는 하위 프로세스를 보여줍니다. *your\_volume\_mount*를 사용자 경로(사전 정의된 볼륨을 마운트한 경로)로 바꿉니다. 스크립트는 sidecar 컨테이너에서 사용하는 이미지 내에 번들로 제공됩니다. 포드 템플릿 파일에서 sub\_process\_script.sh 및 main\_command 명령을 사용하여 sidecar 컨테이너를 지정할 수 있습니다.

```
MOUNT_PATH="your_volume_mount"
FILE_TO_WATCH="$MOUNT_PATH/main-container-terminated"
INITIAL_HEARTBEAT_TIMEOUT_THRESHOLD=60
HEARTBEAT_TIMEOUT_THRESHOLD=15
SLEEP_DURATION=10

function terminate_main_process() {
    # Stop main process
}

# Waiting for the first heartbeat sent by Spark main container
echo "Waiting for file $FILE_TO_WATCH to appear..."
start_wait=$(date +%s)
while ! [[ -f "$FILE_TO_WATCH" ]]; do
    elapsed_wait=$(expr $(date +%s) - $start_wait)
    if [ "$elapsed_wait" -gt "$INITIAL_HEARTBEAT_TIMEOUT_THRESHOLD" ]; then
        echo "File $FILE_TO_WATCH not found after $INITIAL_HEARTBEAT_TIMEOUT_THRESHOLD
seconds; aborting"
        terminate_main_process
        exit 1
    fi
    sleep $SLEEP_DURATION;
```

```
done;
echo "Found file $FILE_TO_WATCH; watching for heartbeats..."

while [[ -f "$FILE_TO_WATCH" ]]; do
    LAST_HEARTBEAT=$(stat -c %Y $FILE_TO_WATCH)
    ELAPSED_TIME_SINCE_AFTER_HEARTBEAT=$(expr $(date +%s) - $LAST_HEARTBEAT)
    if [ "$ELAPSED_TIME_SINCE_AFTER_HEARTBEAT" -gt "$HEARTBEAT_TIMEOUT_THRESHOLD" ];
then
    echo "Last heartbeat to file $FILE_TO_WATCH was more than
$HEARTBEAT_TIMEOUT_THRESHOLD seconds ago at $LAST_HEARTBEAT; terminating"
    terminate_main_process
    exit 0
fi
sleep $SLEEP_DURATION;
done;
echo "Outside of loop, main-container-terminated file no longer exists"

# The file will be deleted once the fluentd container is terminated

echo "The file $FILE_TO_WATCH doesn't exist any more;"
terminate_main_process
exit 0
```

## 작업 재시도 정책 사용

Amazon EMR on EKS 버전 6.9.0 이상에서 작업 실행에 대한 재시도 정책을 설정할 수 있습니다. 재시도 정책에 따라 작업 드라이버 포드가 실패하거나 삭제된 경우 자동으로 다시 시작됩니다. 이를 통해 장기 실행 Spark 스트리밍 작업의 실패에 대한 복원력을 높일 수 있습니다.

## 작업에 대한 재시도 정책 설정

재시도 정책을 구성하려면 [StartJobRun](#) API를 사용하여 `RetryPolicyConfiguration` 필드를 제공합니다. 다음은 `retryPolicyConfiguration` 예제입니다.

```
aws emr-containers start-job-run \
--virtual-cluster-id cluster_id \
--name sample-job-name \
--execution-role-arn execution-role-arn \
--release-label emr-6.9.0-latest \
--job-driver '{
  "sparkSubmitJobDriver": {
    "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py",
```

```

    "entryPointArguments": [ "2" ],
    "sparkSubmitParameters": "--conf spark.executor.instances=2 --conf
spark.executor.memory=2G --conf spark.executor.cores=2 --conf spark.driver.cores=1"
  }
}' \
--retry-policy-configuration '{
  "maxAttempts": 5
}' \
--configuration-overrides '{
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "my_log_group_name",
      "logStreamNamePrefix": "my_log_stream_prefix"
    },
    "s3MonitoringConfiguration": {
      "logUri": "s3://amzn-s3-demo-logging-bucket"
    }
  }
}'

```

#### Note

retryPolicyConfiguration는 AWS CLI 1.27.68 버전 이상에서만 사용할 수 있습니다. AWS CLI 를 최신 버전으로 업데이트하려면 [의 최신 버전 설치 또는 업데이트를 참조하세요.](#)  
[AWS CLI](#)

작업 드라이버 포드가 실패하거나 삭제된 경우 작업 드라이버 포드를 다시 시작할 최대 횟수로 maxAttempts 필드를 구성합니다. [Kubernetes 설명서](#)에 설명된 대로 두 작업 드라이버 재시도 사이의 실행 간격은 지수 재시도 간격(10초, 20초, 40초 등)이며 최대 6분으로 제한됩니다.

#### Note

모든 추가 작업 드라이버 실행은 또 다른 작업 실행으로 요금이 청구되며, [Amazon EMR on EKS 요금](#)이 적용됩니다.

## 재시도 정책 구성 값

- 작업의 기본 재시도 정책: StartJobRun에는 기본적으로 최대 시도 횟수가 1로 설정된 재시도 정책이 있습니다. 원하는 대로 재시도 정책을 구성할 수 있습니다.

**Note**

retryPolicyConfiguration의 maxAttempts를 1로 설정하면 실패 시 드라이버 포드를 불러오려는 재시도가 수행되지 않습니다.

- 작업의 재시도 정책 비활성화: 재시도 정책을 비활성화하려면 retryPolicyConfiguration에서 최대 시도 횟수 값을 1로 설정합니다.

```
"retryPolicyConfiguration": {
  "maxAttempts": 1
}
```

- 작업의 maxAttempts를 유효한 범위로 설정: maxAttempts 값이 유효한 범위를 벗어나면 StartJobRun 직접 호출에 실패합니다. 유효한 maxAttempts 범위는 1~2,147,483,647(32비트 정수)로, Kubernetes의 backOffLimit 구성 설정에서 지원하는 범위입니다. 자세한 내용은 Kubernetes 설명서에서 [Pod backoff failure policy](#)를 참조하세요. maxAttempts 값이 유효하지 않으면 다음 오류 메시지가 반환됩니다.

```
{
  "message": "Retry policy configuration's parameter value of maxAttempts is invalid"
}
```

## 작업의 재시도 정책 상태 검색

[ListJobRuns](#) 및 [DescribeJobRun](#) API를 사용하여 작업의 재시도 상태를 볼 수 있습니다.

활성화된 재시도 정책 구성으로 작업을 요청하면 ListJobRun 및 DescribeJobRun 응답의 RetryPolicyExecution 필드에 재시도 정책 상태가 포함됩니다. 또한 DescribeJobRun 응답에는 작업에 대한 StartJobRun 요청에 입력된 RetryPolicyConfiguration이 포함됩니다.

### 샘플 응답

#### ListJobRuns response

```
{
  "jobRuns": [
    ...
    ...
    "retryPolicyExecution" : {
```

```

    "currentAttemptCount": 2
  }
  ...
  ...
]
}

```

## DescribeJobRun response

```

{
  ...
  ...
  "retryPolicyConfiguration": {
    "maxAttempts": 5
  },
  "retryPolicyExecution" : {
    "currentAttemptCount": 2
  },
  ...
  ...
}

```

아래 [재시도 정책 구성 값](#)의 설명과 같이 작업에서 재시도 정책이 비활성화된 경우 이 필드는 표시되지 않습니다.

## 재시도 정책으로 작업 모니터링

재시도 정책을 활성화하면 생성된 모든 작업 드라이버에 대해 CloudWatch 이벤트가 생성됩니다. 이러한 이벤트를 구독하려면 다음 명령을 사용하여 CloudWatch 이벤트 규칙을 설정합니다.

```

aws events put-rule \
--name cwe-test \
--event-pattern '{"detail-type": ["EMR Job Run New Driver Attempt"]}'

```

이벤트는 작업 드라이버의 newDriverPodName, newDriverCreatedAt 타임스탬프, previousDriverFailureMessage 및 currentAttemptCount에 대한 정보를 반환합니다. 재시도 정책이 비활성화된 경우 이러한 이벤트는 생성되지 않습니다.

CloudWatch 이벤트를 사용하여 작업을 모니터링하는 방법에 대한 자세한 내용은 [Amazon CloudWatch Events를 사용하여 작업 모니터링](#) 섹션을 참조하세요.

## 드라이버 및 실행기의 로그 찾기

드라이버 포드 이름은 `spark-<job id>-driver-<random-suffix>` 형식을 따릅니다. 드라이버가 생성하는 실행기 포드에 동일한 `random-suffix`가 추가됩니다. 이 `random-suffix`를 사용하면 드라이버 및 관련 실행기에 대한 로그를 찾을 수 있습니다. `random-suffix`는 작업에 대해 [재시도 정책이 활성화](#)된 경우에만 표시되고 그렇지 않으면 `random-suffix`는 표시되지 않습니다.

로그를 위해 모니터링 구성을 포함하도록 작업을 구성하는 방법에 대한 자세한 내용은 [Spark 애플리케이션 실행](#) 섹션을 참조하세요.

## Spark 이벤트 로그 로테이션 사용

Amazon EMR 6.3.0 이상에서는 Amazon EMR on EKS에 대한 Spark 이벤트 로그 로테이션 기능을 켤 수 있습니다. 이 기능은 단일 이벤트 로그 파일을 생성하는 대신 구성된 시간 간격에 따라 파일을 로테이션하고 가장 오래된 이벤트 로그 파일을 제거합니다.

Spark 이벤트 로그를 로테이션하면 장기 실행 또는 스트리밍 작업에 대해 생성되는 대용량 Spark 이벤트 로그 파일로 인한 잠재적 문제를 방지할 수 있습니다. 예를 들어, `persistentAppUI` 파라미터로 활성화된 이벤트 로그를 사용하여 장기 실행 Spark 작업을 시작합니다. Spark 드라이버는 이벤트 로그 파일을 생성합니다. 작업이 몇 시간 또는 며칠 동안 실행되고 Kubernetes 노드의 디스크 공간이 제한된 경우 이벤트 로그 파일이 사용 가능한 디스크 공간을 모두 소비할 수 있습니다. Spark 이벤트 로그 로테이션 기능을 켜면 로그 파일을 여러 파일로 분할하고 가장 오래된 파일을 제거하여 문제를 해결할 수 있습니다.

### Note

이 기능은 Amazon EMR on EKS에서만 작동합니다. Amazon EC2에서 실행되는 Amazon EMR은 Spark 이벤트 로그 로테이션을 지원하지 않습니다.

Spark 이벤트 로그 로테이션 기능을 켜려면 다음 Spark 파라미터를 구성합니다.

- `spark.eventLog.rotation.enabled` - 로그 로테이션을 켭니다. 기본적으로는 Spark 구성 파일에서 비활성화되어 있습니다. 이 기능을 켜려면 `true`로 설정합니다.
- `spark.eventLog.rotation.interval` - 로그 로테이션 시간 간격을 지정합니다. 최솟값은 60 초입니다. 기본 값은 300초입니다.
- `spark.eventLog.rotation.minFileSize` - 로그 파일을 로테이션할 최소 파일 크기를 지정합니다. 최소 기본값은 1MB입니다.

- `spark.eventLog.rotation.maxFilesToRetain` - 정리 중에 보관할 로테이션된 로그 파일 수를 지정합니다. 값의 범위는 1~10입니다. 기본값은 2입니다.

다음 예제와 같이 [StartJobRun](#) API의 `sparkSubmitParameters` 섹션에서 이러한 파라미터를 지정할 수 있습니다.

```
"sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf
  spark.eventLog.rotation.enabled=true --conf spark.eventLog.rotation.interval=300 --
  conf spark.eventLog.rotation.minFileSize=1m --conf
  spark.eventLog.rotation.maxFilesToRetain=2"
```

## Spark 컨테이너 로그 로테이션 사용

Amazon EMR 6.11.0 이상에서는 Amazon EMR on EKS에 대한 Spark 컨테이너 로그 로테이션 기능을 켤 수 있습니다. 이 기능은 단일 `stdout` 또는 `stderr` 로그 파일을 생성하는 대신 구성된 로테이션 크기에 따라 파일을 로테이션하고 컨테이너에서 가장 오래된 로그 파일을 제거합니다.

Spark 컨테이너 로그를 로테이션하면 장기 실행 또는 스트리밍 작업에 대해 생성되는 대용량 Spark 로그 파일로 인한 잠재적 문제를 방지할 수 있습니다. 예를 들어 장기 실행 Spark 작업을 시작하면 Spark 드라이버에서 컨테이너 로그 파일을 생성할 수 있습니다. 작업이 몇 시간 또는 며칠 동안 실행되고 Kubernetes 노드의 디스크 공간이 제한된 경우 컨테이너 로그 파일이 사용 가능한 디스크 공간을 모두 소비할 수 있습니다. Spark 컨테이너 로그 로테이션을 켜면 로그 파일을 여러 파일로 분할하고 가장 오래된 파일을 제거합니다.

Spark 컨테이너 로그 로테이션 기능을 켜려면 다음 Spark 파라미터를 구성합니다.

### **containerLogRotationConfiguration**

로그 로테이션을 켜려면 `monitoringConfiguration`에 이 파라미터를 포함합니다. 기본적으로는 비활성화되어 있습니다. `containerLogRotationConfiguration` 이외에도 `s3MonitoringConfiguration`을 사용해야 합니다.

### **rotationSize**

`rotationSize` 파라미터는 로그 로테이션의 파일 크기를 지정합니다. 가능한 값의 범위는 2KB~2GB입니다. `rotationSize` 파라미터의 숫자 단위 부분은 정수로 전달됩니다. 십진수는 지원되지 않으므로 로테이션 크기를 1.5GB(예: 1500MB 값)로 지정할 수 있습니다.

## maxFilesToKeep

maxFilesToKeep 파라미터는 로테이션을 수행한 후 컨테이너에서 보존할 최대 파일 수를 지정합니다. 최솟값은 1이고 최댓값은 50입니다.

다음 예제와 같이 StartJobRun API의 monitoringConfiguration 섹션에서 이러한 파라미터를 지정할 수 있습니다. 이 예제에서 Amazon EMR on EKS는 rotationSize = "10 MB" 및 maxFilesToKeep = 3을 사용하여 로그를 10MB에서 로테이션하고 새 로그 파일을 생성한 후 로그 파일 수가 3개가 되면 가장 오래된 로그 파일을 제거합니다.

```
{
  "name": "my-long-running-job",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-6.11.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2", ...],
      "sparkSubmitParameters": "--class main_class --conf spark.executor.instances=2
--conf spark.executor.memory=2G --conf spark.executor.cores=2 --conf
spark.driver.cores=1"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory": "2G"
        }
      }
    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration": {
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      },
      "s3MonitoringConfiguration": {
        "logUri": "s3://my_s3_log_location"
      }
    }
  }
}
```



```

    "containerLogRotationConfiguration": {
      "rotationSize": "10MB",
      "maxFilesToKeep": "3"
    }
  }
}

```

Spark 컨테이너 로그 로테이션과 함께 작업 실행을 시작하려면 [StartJobRun](#) 명령에서 이 파라미터로 구성된 json 파일의 경로를 포함합니다.

```

aws emr-containers start-job-run \
--cli-input-json file://path-to-json-request-file

```

## Amazon EMR Spark 작업에 수직 자동 조정 사용

Amazon EMR on EKS 수직 자동 조정은 Amazon EMR Spark 애플리케이션에 제공하는 워크로드 요구 사항에 맞게 메모리와 CPU 리소스를 자동으로 튜닝합니다. 이 기능은 리소스 관리를 간소화합니다.

Amazon EMR Spark 애플리케이션의 실시간 및 과거 리소스 사용률을 추적하기 위해 수직 자동 조정에서는 Kubernetes [Vertical Pod Autoscaler\(VPA\)](#)를 활용합니다. 수직 자동 조정 기능은 VPA가 수집한 데이터를 사용하여 Spark 애플리케이션에 할당된 메모리 및 CPU 리소스를 자동으로 조정합니다. 이렇게 간소화된 프로세스는 안정성을 높이고 비용을 최적화합니다.

### 주제

- [Amazon EMR on EKS에서 수직 자동 조정 설정](#)
- [Amazon EMR on EKS에서 수직 자동 조정 시작하기](#)
- [Amazon EMR on EKS에서 수직 자동 조정 구성](#)
- [Amazon EMR on EKS에서 수직 자동 조정 모니터링](#)
- [Amazon EMR on EKS 수직 자동 조정 운영자 제거](#)

## Amazon EMR on EKS에서 수직 자동 조정 설정

이 주제는 수직 자동 조정을 통해 Amazon EMR Spark 작업을 제출할 수 있도록 Amazon EKS 클러스터를 준비하는 데 도움이 됩니다. 설정 프로세스를 진행하려면 다음 섹션의 작업을 확인하거나 완료해야 합니다.

## 주제

- [사전 조건](#)
- [Amazon EKS 클러스터에 Operator Lifecycle Manager\(OLM\) 설치](#)
- [Amazon EMR on EKS 수직 자동 조정 운영자 설치](#)

## 사전 조건

클러스터에 수직 자동 조정 Kubernetes 운영자를 설치하기 전에 다음 작업을 완료합니다. 필수 조건 중 하나를 이미 완료한 경우 해당 조건을 건너뛰고 다음 조건으로 넘어갈 수 있습니다.

- [의 최신 버전 설치 또는 업데이트 AWS CLI](#) -를 이미 설치한 경우 최신 버전이 있는지 AWS CLI 확인합니다.
- [kubectl 설치](#) - kubectl은 Kubernetes API 서버와 통신하기 위해 사용하는 명령줄 도구입니다. Amazon EKS 클러스터에 수직 자동 조정 관련 아티팩트를 설치하고 모니터링하려면 kubectl이 필요합니다.
- [운영자 SDK 설치](#) - Amazon EMR on EKS는 클러스터에 설치하는 수직 자동 조정 운영자의 수명 동안 운영자 SDK를 패키지 관리자로 사용합니다.
- [Docker 설치](#) - Amazon EKS 클러스터에 설치할 수직 자동 조정 관련 도커 이미지를 인증하고 가져오려면 Docker CLI에 액세스해야 합니다.
- [Kubernetes 지표 서버 설치](#) - 먼저 수직 포드 오토스케일러가 Kubernetes API 서버에서 지표를 가져올 수 있도록 지표 서버를 설치해야 합니다.
- [Amazon EKS – eksctl 시작하기](#)(버전 1.24 이상) - 수직 자동 조정은 Amazon EKS 버전 1.24 이상에서 지원됩니다. 클러스터를 생성한 후 [Amazon EMR에서 사용할 수 있도록 등록](#)합니다.
- [Amazon EMR 기본 이미지 URI 선택](#)(릴리스 6.10.0 이상) - 수직 자동 조정은 Amazon EMR 릴리스 6.10.0 이상에서 지원됩니다.

## Amazon EKS 클러스터에 Operator Lifecycle Manager(OLM) 설치

운영자 SDK CLI를 사용하여 다음 예제와 같이 수직 자동 조장을 설정하려는 Amazon EMR on EKS 클러스터에 Operator Lifecycle Manager(OLM)를 설치합니다. 설정 후 OLM을 사용하여 [Amazon EMR 수직 자동 조정 운영자](#)의 수명 주기를 설치하고 관리할 수 있습니다.

```
operator-sdk olm install
```

설치를 검증하려면 `olm status` 명령을 실행합니다.

```
operator-sdk olm status
```

이 명령이 제대로 실행되면 다음과 비슷한 출력이 반환되는지 확인합니다.

```
INFO[0007] Successfully got OLM status for version X.XX
```

설치에 실패한 경우 [Amazon EMR on EKS 수직 자동 조정 문제 해결](#) 섹션을 참조하세요.

## Amazon EMR on EKS 수직 자동 조정 운영자 설치

다음 단계를 사용하여 Amazon EKS 클러스터에 수직 자동 조정 운영자를 설치합니다.

- 설치를 완료하는 데 사용할 다음 환경 변수를 설정합니다.
  - \$REGION**은 클러스터의 AWS 리전 을 가리킵니다. 예를 들어 us-west-2입니다.
  - \$ACCOUNT\_ID**는 리전의 Amazon ECR 계정 ID를 가리킵니다. 자세한 내용은 [리전별 Amazon ECR 레지스트리 계정](#) 단원을 참조하십시오.
  - \$RELEASE**는 클러스터에 사용하려는 Amazon EMR 릴리스를 가리킵니다. 수직 자동 조정을 사용하려면 Amazon EMR 릴리스 6.10.0 이상을 사용해야 합니다.
- 다음으로, 운영자의 [Amazon ECR 레지스트리](#)로 인증 토큰을 가져옵니다.

```
aws ecr get-login-password \
  --region region-id | docker login \
  --username AWS \
  --password-stdin $ACCOUNT_ID.dkr.ecr.region-id.amazonaws.com
```

- 다음 명령으로 Amazon EMR on EKS 수직 자동 조정 운영자를 설치합니다.

```
ECR_URL=$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com && \
REPO_DEST=dynamic-sizing-k8s-operator-olm-bundle && \
BUNDLE_IMG=emr-$RELEASE-dynamic-sizing-k8s-operator && \
operator-sdk run bundle \
$ECR_URL/$REPO_DEST/$BUNDLE_IMG\:latest
```

그러면 Amazon EKS 클러스터의 기본 네임스페이스에 수직 자동 조정 운영자가 릴리스됩니다. 다음 명령을 사용하여 다른 네임스페이스에 설치합니다.

```
operator-sdk run bundle \
```

```
$ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com/dynamic-sizing-k8s-operator-olm-bundle/
emr-$RELEASE-dynamic-sizing-k8s-operator:latest \
-n operator-namespace
```

#### Note

지정한 네임스페이스가 없는 경우 OLM은 운영자를 설치하지 않습니다. 자세한 내용은 [Kubernetes 네임스페이스를 찾을 수 없음](#) 단원을 참조하십시오.

4. kubectl Kubernetes 명령줄 도구를 사용하여 운영자를 성공적으로 설치했는지 확인합니다.

```
kubectl get csv -n operator-namespace
```

kubectl 명령은 단계 상태가 성공인 새로 배포한 수직 오토스케일러 운영자를 반환해야 합니다. 설치 또는 설정에 문제가 있는 경우 [Amazon EMR on EKS 수직 자동 조정 문제 해결](#) 섹션을 참조하세요.

## Amazon EMR on EKS에서 수직 자동 조정 시작하기

Amazon EMR Spark 애플리케이션 워크로드의 요구 사항에 맞게 메모리와 CPU 리소스를 자동으로 조정하려는 경우 Amazon EMR on EKS에 대한 수직 오토스케일링을 사용합니다. 자세한 내용은 [Amazon EMR Spark 작업에 수직 자동 조정 사용](#)을 참조하세요.

### 수직 자동 조정을 사용하여 Spark 작업 제출

[StartJobRun](#) API를 통해 작업을 제출하는 경우 Spark 작업의 드라이버에 다음 두 구성을 추가하여 수직 자동 조정을 활성화합니다.

```
"spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/
dynamic.sizing":"true",
"spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/
dynamic.sizing.signature":"YOUR_JOB_SIGNATURE"
```

위 코드의 첫 번째 줄은 수직 자동 조정 기능을 활성화합니다. 다음 줄은 작업에 사용할 서명을 선택할 수 있는 필수 서명 구성입니다.

이러한 구성 및 허용 가능한 파라미터 값에 대한 자세한 내용은 [Amazon EMR on EKS에서 수직 자동 조정 구성](#) 섹션을 참조하세요. 기본적으로 작업은 수직 자동 조정의 모니터링 전용 꺼짐 모드로 제출됨

니다. 이 모니터링 상태를 사용하면 자동 조정을 수행하지 않고도 리소스 권장 사항을 계산하고 볼 수 있습니다. 자세한 내용은 [수직 자동 조정 모드](#) 단원을 참조하십시오.

다음 예제에서는 수직 자동 조정에서 샘플 `start-job-run` 명령을 완료하는 방법을 보여줍니다.

```
aws emr-containers start-job-run \
--virtual-cluster-id $VIRTUAL_CLUSTER_ID \
--name $JOB_NAME \
--execution-role-arn $EMR_ROLE_ARN \
--release-label emr-6.10.0-latest \
--job-driver '{
  "sparkSubmitJobDriver": {
    "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py"
  }
}' \
--configuration-overrides '{
  "applicationConfiguration": [{
    "classification": "spark-defaults",
    "properties": {
      "spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/
dynamic.sizing": "true",
      "spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/
dynamic.sizing.signature": "test-signature"
    }
  }]
}'
```

## 수직 자동 조정 기능 확인

제출된 작업에 대해 수직 자동 조정이 올바르게 작동하는지 확인하려면 `kubectl`을 사용하여 `verticalpodautoscaler` 사용자 지정 리소스를 가져오고 조정 권장 사항을 확인합니다. 예를 들어 다음 명령은 [수직 자동 조정을 사용하여 Spark 작업 제출](#) 섹션의 예제 작업에 대한 권장 사항을 쿼리합니다.

```
kubectl get verticalpodautoscalers --all-namespaces \
-l=emr-containers.amazonaws.com/dynamic.sizing.signature=test-signature
```

이 쿼리에 대한 출력은 다음과 비슷합니다.

| NAME     | MODE | CPU | MEM |
|----------|------|-----|-----|
| PROVIDED | AGE  |     |     |

```
ds-jceyefkxnhrvdzw6djum3naf2abm6o63a6dvjkkedqtkhlrf25eq-vpa  Off  3304504865  True
87m
```

출력이 비슷하지 않거나 오류 코드를 포함하는 경우 문제 해결을 위한 단계는 [Amazon EMR on EKS 수직 자동 조정 문제 해결](#) 섹션을 참조하세요.

## Amazon EMR on EKS에서 수직 자동 조정 구성

[StartJobRun](#) API를 통해 Amazon EMR Spark 작업을 제출할 때 수직 자동 조정을 구성할 수 있습니다. [수직 자동 조정을 사용하여 Spark 작업 제출](#)의 예제와 같이 Spark 드라이버 포드에서 자동 조정 관련 구성 파라미터를 설정합니다.

Amazon EMR on EKS 수직 자동 조정 운영자는 자동 조정 기능이 있는 드라이버 포드의 신호를 수신한 다음, 드라이버 포드의 설정을 사용하여 Kubernetes Vertical Pod Autoscaler(VPA)와의 통합을 설정합니다. 이를 통해 Spark 실행기 포드의 리소스 추적 및 자동 조정이 용이해집니다.

다음 섹션에서는 Amazon EKS 클러스터의 수직 자동 조정을 구성할 때 사용할 수 있는 파라미터를 설명합니다.

### Note

기능 토글 파라미터를 레이블로 구성하고, 나머지 파라미터는 Spark 드라이버 포드의 주석으로 구성합니다. 자동 조정 파라미터는 `emr-containers.amazonaws.com/` 도메인에 속하며 `dynamic.sizing` 접두사가 붙습니다.

## 필수 파라미터

작업을 제출할 때 Spark 작업 드라이버에 다음 두 파라미터를 포함해야 합니다.

| Key(키)                      | 설명    | 허용되는 값                   | 기본값     | Type | Spark 파라미터 <sup>1</sup>                                    |
|-----------------------------|-------|--------------------------|---------|------|------------------------------------------------------------|
| <code>dynamic.sizing</code> | 기능 전환 | <code>true, false</code> | 설정되지 않음 | 레이블  | <code>spark.kubernetes.driver.label.emr-containers.</code> |

| Key(키)                   | 설명    | 허용되는 값 | 기본값     | Type | Spark 파라미터 <sup>1</sup>                                                                  |
|--------------------------|-------|--------|---------|------|------------------------------------------------------------------------------------------|
|                          |       |        |         |      | amazonaws.com/dynamic.sizing                                                             |
| dynamic.sizing.signature | 작업 서명 | string | 설정되지 않음 | 주석   | spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.signature |

<sup>1</sup> StartJobRun API에서 이 파라미터를 SparkSubmitParameter 또는 ConfigurationOverride로 사용합니다.

- **dynamic.sizing** - dynamic.sizing 레이블을 사용하여 수직 자동 조정을 켜거나 끌 수 있습니다. 수직 자동 조정을 켜려면 Spark 드라이버 포드에서 dynamic.sizing을 true로 설정합니다. 이 레이블을 생략하거나 true 이외의 값으로 설정하면 수직 자동 조정이 꺼집니다.
- **dynamic.sizing.signature** - 드라이버 포드에서 dynamic.sizing.signature 주석을 사용하여 작업 서명을 설정합니다. 수직 자동 조정은 다양한 Amazon EMR Spark 작업 실행의 리소스 사용 데이터를 집계하여 리소스 권장 사항을 도출합니다. 작업을 하나로 묶을 수 있는 고유 식별자를 제공합니다.

#### Note

작업이 매일 또는 매주 같은 고정된 간격으로 반복되는 경우 새 태스크 인스턴스마다 작업 서명이 동일하게 유지되어야 합니다. 이렇게 하면 수직 자동 조정을 통해 다양한 작업 실행에 대한 권장 사항을 계산하고 집계할 수 있습니다.

<sup>1</sup> StartJobRun API에서 이 파라미터를 SparkSubmitParameter 또는 ConfigurationOverride로 사용합니다.

## 선택적 파라미터

수직 자동 조정은 다음과 같은 선택적 파라미터도 지원합니다. 드라이버 포트에서 주석으로 설정합니다.

| Key(키)                                      | 설명              | 허용되는 값             | 기본값   | Type | Spark 파라미터 <sup>1</sup>                                                                     |
|---------------------------------------------|-----------------|--------------------|-------|------|---------------------------------------------------------------------------------------------|
| <a href="#">dynamic.sizing.mode</a>         | 수직 자동 조정        | Off, Initial, Auto | Off   | 주석   | spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.mode         |
| <a href="#">dynamic.sizing.scale.memory</a> | 메모리 조정 활성화      | true, false        | true  | 주석   | spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.memory |
| <a href="#">dynamic.sizing.scale.cpu</a>    | CPU 조정 켜기 또는 끄기 | true, false        | false | 주석   | spark.kubernetes.driver.annotation.e                                                        |



| Key(키)                                          | 설명            | 허용되는 값                                   | 기본값     | Type | Spark 파라미터 <sup>1</sup>                                                                         |
|-------------------------------------------------|---------------|------------------------------------------|---------|------|-------------------------------------------------------------------------------------------------|
|                                                 |               |                                          |         |      | mr-containers.amazonaws.com/dynamic.sizing.scale.cpu                                            |
| <a href="#">dynamic.sizing.scale.memory.min</a> | 메모리 조정의 최소 한도 | 문자열, <a href="#">K8s 리소스 수량</a><br>예: 1G | 설정되지 않음 | 주석   | spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.memory.min |
| <a href="#">dynamic.sizing.scale.memory.max</a> | 메모리 조정의 최대 한도 | 문자열, <a href="#">K8s 리소스 수량</a><br>예: 4G | 설정되지 않음 | 주석   | spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.memory.max |

| Key(키)                                       | 설명            | 허용되는 값                                  | 기본값     | Type | Spark 파라미터 <sup>1</sup>                                                                      |
|----------------------------------------------|---------------|-----------------------------------------|---------|------|----------------------------------------------------------------------------------------------|
| <a href="#">dynamic.sizing.scale.cpu.min</a> | CPU 조정의 최소 한도 | 문자열, <a href="#">K8s 리소스 수량</a><br>예: 1 | 설정되지 않음 | 주석   | spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.cpu.min |
| <a href="#">dynamic.sizing.scale.cpu.max</a> | CPU 조정의 최대 한도 | 문자열, <a href="#">K8s 리소스 수량</a><br>예: 2 | 설정되지 않음 | 주석   | spark.kubernetes.driver.annotation.emr-containers.amazonaws.com/dynamic.sizing.scale.cpu.max |

## 수직 자동 조정 모드

mode 파라미터는 VPA가 지원하는 다양한 자동 조정 모드에 매핑됩니다. 드라이버 포드에서 `dynamic.sizing.mode` 주석을 사용하여 모드를 설정합니다. 다음 값이 이 파라미터에 대해 지원됩니다.

- Off - 권장 사항을 모니터링할 수 있지만 자동 조정은 수행되지 않는 테스트 실행 모드입니다. 수직 자동 조정의 기본 모드입니다. 이 모드에서는 연결된 Vertical Pod Autoscaler 리소스가 권장 사항을 계산하며, kubectl, Prometheus, Grafana와 같은 도구를 통해 권장 사항을 모니터링할 수 있습니다.

- Initial - 이 모드에서는 반복 작업과 같이 작업 실행 기록을 기반으로 권장 사항이 제공되는 경우 작업이 시작될 때 VPA가 리소스를 자동으로 조정합니다.
- Auto - 이 모드에서 VPA는 Spark 실행기 포드를 제거하고 Spark 드라이버 포드가 다시 시작될 때 권장 리소스 설정으로 규모를 자동 조정합니다. 경우에 따라 VPA가 실행 중인 Spark 실행기 포드를 제거하므로 중단된 실행기를 재시도할 때 추가 지연이 발생할 수 있습니다.

## 리소스 조정

수직 자동 조정을 설정할 때 CPU 및 메모리 리소스를 확장할지 여부를 선택할 수 있습니다.

`dynamic.sizing.scale.cpu` 및 `dynamic.sizing.scale.memory` 주석을 `true` 또는 `false`로 설정합니다. 기본적으로 CPU 조정은 `false`로, 메모리 조정은 `true`로 설정됩니다.

## 리소스 최소 및 최대(바운드)

선택적으로 CPU 및 메모리 리소스에 경계를 설정할 수 있습니다. 자동 조정을 활성화할 때

`dynamic.sizing.[memory/cpu].[min/max]` 주석을 사용하여 이러한 리소스의 최솟값 및 최댓값을 선택합니다. 기본적으로 리소스에는 제한이 없습니다. Kubernetes 리소스 수량을 나타내는 문자열 값으로 주석을 설정합니다. 예를 들어 4GB를 나타내도록 `dynamic.sizing.memory.max`를 4G로 설정합니다.

## Amazon EMR on EKS에서 수직 자동 조정 모니터링

`kubectl` Kubernetes 명령줄 도구를 사용하여 클러스터의 활성 수직 조정 관련 권장 사항을 나열할 수 있습니다. 또한 추적된 작업 서명을 확인하고 서명과 관련된 불필요한 리소스를 제거할 수 있습니다.

## 클러스터의 수직 자동 조정 권장 사항 나열

`kubectl`을 사용하여 `verticalpodautoscaler` 리소스를 얻고 현재 상태 및 권장 사항을 확인합니다. 다음 예제 쿼리는 Amazon EKS 클러스터의 모든 활성 리소스를 반환합니다.

```
kubectl get verticalpodautoscalers \
-o custom-columns="NAME:.metadata.name, \"\
\"SIGNATURE:.metadata.labels.emr-containers\.amazonaws\.com/dynamic\.sizing
\.signature, \"\
\"MODE:.spec.updatePolicy.updateMode, \"\
\"MEM:.status.recommendation.containerRecommendations[0].target.memory\" \
--all-namespaces
```

이 쿼리의 출력은 다음과 유사합니다.

| NAME                         | SIGNATURE              | MODE    | MEM         |
|------------------------------|------------------------|---------|-------------|
| ds- <i>example-id-1</i> -vpa | <i>job-signature-1</i> | Off     | <i>none</i> |
| ds- <i>example-id-2</i> -vpa | <i>job-signature-2</i> | Initial | 12936384283 |

## 클러스터의 수직 자동 조정 권장 사항 쿼리 및 삭제

Amazon EMR 수직 자동 조정 작업 실행 리소스를 삭제하면 권장 사항을 추적하고 저장하는 관련 VPA 객체가 자동으로 삭제됩니다.

다음 예제는 kubectl을 사용하여 서명으로 식별되는 작업에 대한 권장 사항을 제거합니다.

```
kubectl delete jobrun -n emr -l=emr-containers\.amazonaws\.com/dynamic\.sizing
\.signature=integ-test
jobrun.dynamicsizing.emr.services.k8s.aws "ds-job-signature" deleted
```

특정 작업 서명을 모르거나 클러스터의 모든 리소스를 제거하려는 경우 다음 예제와 같이 고유한 작업 ID 대신 명령에 --all 또는 --all-namespaces를 사용할 수 있습니다.

```
kubectl delete jobruns --all --all-namespaces
jobrun.dynamicsizing.emr.services.k8s.aws "ds-example-id" deleted
```

## Amazon EMR on EKS 수직 자동 조정 운영자 제거

Amazon EKS 클러스터에서 수직 자동 조정 운영자를 제거하려면 다음 예제와 같이 운영자 SDK CLI와 함께 cleanup 명령을 사용합니다. 이렇게 하면 운영자와 함께 설치한 업스트림 종속 항목(예: Vertical Pod Autoscaler)도 삭제됩니다.

```
operator-sdk cleanup emr-dynamic-sizing
```

운영자를 삭제할 때 클러스터에 실행 중인 작업이 있는 경우 해당 작업은 수직 자동 조정 없이 계속 실행됩니다. 운영자를 삭제한 후 클러스터에서 작업을 제출하면 Amazon EMR on EKS는 [구성](#) 중에 정의했을 수 있는 모든 수직 자동 조정 관련 파라미터를 무시합니다.

# Amazon EMR on EKS에서 대화형 워크로드 실행

대화형 엔드포인트는 대화형 워크로드를 실행할 수 있도록 Amazon EMR Studio를 Amazon EMR on EKS에 연결하여 게이트웨이입니다. EMR Studio의 대화형 엔드포인트를 사용하여 [Amazon S3](#) 및 [Amazon DynamoDB](#)와 같은 데이터 스토어의 데이터 세트와 함께 대화형 분석을 실행할 수 있습니다.

## 사용 사례

- EMR Studio IDE 환경에서 ETL 스크립트를 생성합니다. IDE는 온프레미스 데이터를 수집하여 후속 분석을 위해 변환 후 Amazon S3에 저장합니다.
- 노트북을 사용하여 데이터 세트를 탐색하고 데이터 세트에서 이상을 감지하도록 기계 학습 모델을 학습합니다.
- 비즈니스 대시보드와 같은 분석 애플리케이션을 위한 일일 보고서를 생성하는 스크립트를 생성합니다.

## 주제

- [대화형 엔드포인트 개요](#)
- [Amazon EMR on EKS에서 대화형 엔드포인트를 생성하기 위한 필수 조건](#)
- [가상 클러스터의 대화형 엔드포인트 생성](#)
- [대화형 엔드포인트에 대한 설정 구성](#)
- [대화형 엔드포인트 모니터링](#)
- [자체 호스팅 Jupyter Notebook 사용](#)
- [CLI 명령을 사용하여 대화형 엔드포인트에 대한 정보 가져오기](#)

## 대화형 엔드포인트 개요

대화형 엔드포인트를 통해 Amazon EMR Studio와 같은 대화형 클라이언트에서 Amazon EMR on EKS 클러스터에 연결하여 대화형 워크로드를 실행할 수 있습니다. 대화형 엔드포인트는 대화형 클라이언트에 필요한 원격 커널 수명 주기 관리 기능을 제공하는 Jupyter Enterprise Gateway의 지원을 받습니다. 커널은 Jupyter 기반 Amazon EMR Studio 클라이언트와 상호 작용하여 대화형 워크로드를 실행하는 언어별 프로세스입니다.

대화형 엔드포인트는 다음 커널을 지원합니다.

- Python 3

- Kubernetes 기반 PySpark
- Scala와 함께 사용하는 Apache Spark

### Note

대화형 엔드포인트 및 커널에 대해 Amazon EMR on EKS 요금이 적용됩니다. 자세한 내용은 [Amazon EMR on EKS 요금 페이지](#)를 참조하세요.

EMR Studio에서 Amazon EMR on EKS에 연결하려면 다음 엔터티가 필요합니다.

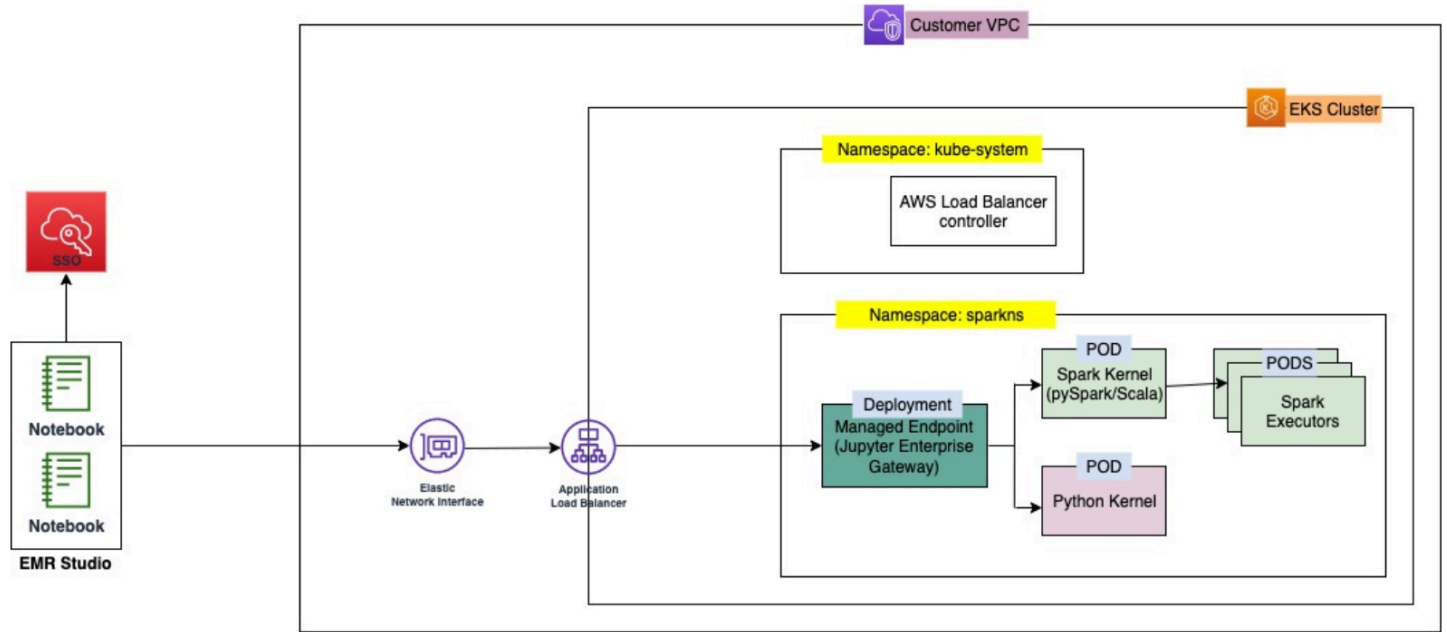
- Amazon EMR on EKS 가상 클러스터 - 가상 클러스터는 Amazon EMR을 등록하는 데 사용하는 Kubernetes 네임스페이스입니다. Amazon EMR은 가상 클러스터를 사용하여 작업을 실행하고 엔드포인트를 호스팅합니다. 동일한 물리적 클러스터로 여러 가상 클러스터를 백업할 수 있습니다. 하지만 각 가상 클러스터는 Amazon EKS 클러스터의 네임스페이스 하나에 매핑됩니다. 가상 클러스터는 청구서에 기여하거나 서비스 외부에서 수명 주기 관리가 필요한 활성 리소스를 생성하지 않습니다.
- Amazon EMR on EKS 대화형 엔드포인트 - 대화형 엔드포인트는 EMR Studio 사용자가 Workspace를 연결할 수 있는 HTTPS 엔드포인트입니다. EMR Studio에서만 HTTPS 엔드포인트에 액세스할 수 있으며, Amazon EKS 클러스터에 대한 Amazon Virtual Private Cloud(VPC)의 프라이빗 서브넷에서 생성할 수 있습니다.

Python, PySpark 및 Spark Scala 커널은 Amazon EMR on EKS 작업 실행 역할에 정의된 권한을 사용하여 다른 AWS 서비스를 간접 호출합니다. 대화형 엔드포인트에 연결하는 모든 커널과 사용자는 엔드포인트를 생성할 때 지정한 역할을 활용합니다. 서로 다른 사용자에게 대해 별도의 엔드포인트를 생성하고 사용자가 서로 다른 AWS Identity and Access Management (IAM) 역할을 갖는 것이 좋습니다.

- AWS Application Load Balancer 컨트롤러 - AWS Application Load Balancer 컨트롤러는 Amazon EKS Kubernetes 클러스터에 대한 Elastic Load Balancing을 관리합니다. Kubernetes Ingress 리소스를 생성할 때 컨트롤러는 Application Load Balancer(ALB)를 프로비저닝합니다. ALB는 대화형 엔드포인트와 같은 Kubernetes 서비스를 Amazon EKS 클러스터 외부에서 동일한 Amazon VPC 내에 노출합니다. 대화형 엔드포인트를 생성하면 대화형 클라이언트가 연결할 수 있도록 ALB를 통해 대화형 엔드포인트를 노출하는 Ingress 리소스도 배포됩니다. 각 Amazon EKS 클러스터에 대해 하나의 AWS Application Load Balancer 컨트롤러만 설치하면 됩니다.

다음 다이어그램은 Amazon EMR on EKS의 대화형 엔드포인트 아키텍처를 보여줍니다. Amazon EKS 클러스터는 분석 워크로드 및 대화형 엔드포인트를 실행하도록 컴퓨팅을 구성합니다. Application

Load Balancer 컨트롤러는 kube-system 네임스페이스에서 실행되고, 워크로드와 대화형 엔드포인트는 가상 클러스터를 생성할 때 지정한 네임스페이스에서 실행됩니다. 대화형 엔드포인트를 생성하면 Amazon EMR on EKS 컨트롤 플레인에서 Amazon EKS 클러스터에 대화형 엔드포인트 배포를 생성합니다. 또한 애플리케이션 로드 밸런서 수신의 인스턴스는 AWS 로드 밸런서 컨트롤러에 의해 생성됩니다. Application Load Balancer는 EMR Studio와 같은 클라이언트가 Amazon EMR 클러스터에 연결하고 대화형 워크로드를 실행하기 위한 외부 인터페이스를 제공합니다.



## Amazon EMR on EKS에서 대화형 엔드포인트를 생성하기 위한 필수 조건

이 섹션에서는 EMR Studio에서 Amazon EMR on EKS 클러스터에 연결하고 대화형 워크로드를 실행하는 데 사용할 수 있는 대화형 엔드포인트를 설정하기 위한 필수 조건을 설명합니다.

### AWS CLI

[설치 또는의 최신 버전으로 업데이트 AWS CLI](#)의 단계에 따라 AWS Command Line Interface ()의 최신 버전을 설치합니다AWS CLI.

### eksctl 설치

[kubectl 설치](#)의 단계를 수행하여 eksctl의 최신 버전을 설치합니다. Amazon EKS 클러스터에 Kubernetes 버전 1.22 이상을 사용하는 경우 0.117.0 이상의 eksctl 버전을 사용합니다.

## Amazon EKS 클러스터

Amazon EKS 클러스터를 생성합니다. 클러스터를 Amazon EMR on EKS에서 가상 클러스터로 등록합니다. 다음은 이 클러스터에 대한 요구 사항 및 고려 사항입니다.

- 클러스터는 EMR Studio와 동일한 Amazon Virtual Private Cloud(VPC)에 있습니다.
- 대화형 엔드포인트를 활성화하고, Git 기반 리포지토리를 연결하며, Application Load Balancer를 프라이빗 모드에서 시작하려면 클러스터에 하나 이상의 프라이빗 서브넷이 있어야 합니다.
- 가상 클러스터를 등록하는 데 사용하는 EMR Studio와 Amazon EKS 클러스터 사이에 하나 이상의 공통된 프라이빗 서브넷이 있어야 합니다. 이를 통해 대화형 엔드포인트가 Studio Workspace에 옵션으로 나타나고 Studio에서 Application Load Balancer로의 연결이 활성화됩니다.

Studio와 Amazon EKS 클러스터를 연결하기 위해 선택할 수 있는 두 가지 방법이 있습니다.

- Amazon EKS 클러스터를 생성하고 EMR Studio에 속한 서브넷에 연결합니다.
- 또는 EMR Studio를 생성하고 Amazon EKS 클러스터의 프라이빗 서브넷을 지정할 수도 있습니다.
- Amazon EMR on EKS 대화형 엔드포인트에 대해 Amazon EKS에서 최적화된 ARM Amazon Linux AMI는 지원되지 않습니다.
- [Amazon EKS 관리형 노드 그룹](#) 및 Karpenter 프로비저닝된 노드만 지원됩니다.

## Amazon EMR on EKS에 대한 클러스터 액세스 권한 부여

[Amazon EMR on EKS에 대한 클러스터 액세스 권한 부여](#)의 단계를 사용하여 Amazon EMR on EKS에 클러스터의 특정 네임스페이스에 대한 액세스 권한을 부여합니다.

## Amazon EMR on EKS에서 IRSA 활성화

Amazon EKS 클러스터에서 서비스 계정에 대한 IAM 역할(IRSA)을 활성화하려면 [서비스 계정에 대한 IAM 역할\(IRSA\) 활성화](#)의 단계를 따릅니다.

## IAM 작업 실행 역할 생성

Amazon EMR on EKS 대화형 엔드포인트에서 워크로드를 실행하려면 IAM 역할을 생성해야 합니다. 이 설명서에서는 이 IAM 역할을 작업 실행 역할이라고 합니다. 이 IAM 역할은 대화형 엔드포인트 컨테이너와 EMR Studio에서 작업을 제출할 때 생성되는 실제 실행 컨테이너 모두에 할당됩니다. Amazon



EMR on EKS 클러스터에 대해 작업 실행 역할의 Amazon 리소스 이름(ARN)이 필요합니다. 이를 위해서는 두 단계가 필요합니다.

- [작업 실행을 위해 IAM 역할 생성.](#)
- [작업 실행 역할의 신뢰 정책 업데이트.](#)

## Amazon EMR on EKS에 대한 액세스 권한 부여

대화형 엔드포인트 생성을 요청하는 IAM 엔터티(사용자 또는 역할)에는 다음과 같은 Amazon EC2 및 `emr-containers` 권한도 있어야 합니다. [Amazon EMR on EKS에 대한 액세스 권한 부여](#)에 설명된 단계에 따라 Amazon EMR on EKS에서 대화형 엔드포인트의 로드 밸런서로 인바운드 트래픽을 제한하는 보안 그룹을 생성, 관리 및 삭제할 수 있는 권한을 부여합니다.

다음 `emr-containers` 권한을 통해 기본적인 대화형 엔드포인트 작업을 수행할 수 있습니다.

```
"ec2:CreateSecurityGroup",
"ec2:DeleteSecurityGroup",
"ec2:AuthorizeSecurityGroupEgress",
"ec2:AuthorizeSecurityGroupIngress",
"ec2:RevokeSecurityGroupEgress",
"ec2:RevokeSecurityGroupIngress"

"emr-containers:CreateManagedEndpoint",
"emr-containers:ListManagedEndpoints",
"emr-containers:DescribeManagedEndpoint",
"emr-containers>DeleteManagedEndpoint"
```

## Amazon EMR에 Amazon EKS 클러스터 등록

가상 클러스터를 설정하고 작업을 실행하려는 Amazon EKS 클러스터의 네임스페이스에 매핑합니다. AWS Fargate 전용 클러스터의 경우 Amazon EMR on EKS 가상 클러스터와 Fargate 프로파일 모두에 동일한 네임스페이스를 사용합니다.

Amazon EMR on EKS 가상 클러스터를 설정하는 방법에 대한 자세한 내용은 [Amazon EMR에 Amazon EKS 클러스터 등록](#) 섹션을 참조하세요.

## Amazon EKS 클러스터에 Deploy AWS Load Balancer 컨트롤러

Amazon EKS 클러스터에는 AWS Application Load Balancer가 필요합니다. Amazon EKS 클러스터당 하나의 Application Load Balancer 컨트롤러만 설정하면 됩니다. AWS Application Load Balancer 컨트롤러

롤러 설정에 대한 자세한 내용은 Amazon EKS 사용 설명서 [의 AWS Load Balancer 컨트롤러 추가 기능 설치를](#) 참조하세요.

## 가상 클러스터의 대화형 엔드포인트 생성

이 주제에서는 AWS 명령줄 인터페이스(AWS CLI)를 사용하여 대화형 엔드포인트를 생성하는 몇 가지 방법을 설명하고 사용 가능한 구성 파라미터에 대한 세부 정보를 포함합니다.

### **create-managed-endpoint** 명령을 사용하여 대화형 엔드포인트 생성

다음과 같이 create-managed-endpoint 명령에서 파라미터를 지정합니다. Amazon EMR on EKS 는 Amazon EMR 릴리스 6.7.0 이상을 사용하여 대화형 엔드포인트를 생성할 수 있도록 지원합니다.

```
aws emr-containers create-managed-endpoint \
--type JUPYTER_ENTERPRISE_GATEWAY \
--virtual-cluster-id 1234567890abcdef0xxxxxxx \
--name example-endpoint-name \
--execution-role-arn arn:aws:iam::444455556666:role/JobExecutionRole \
--release-label emr-6.9.0-latest \
--configuration-overrides '{
  "applicationConfiguration": [{
    "classification": "spark-defaults",
    "properties": {
      "spark.driver.memory": "2G"
    }
  }],
  "monitoringConfiguration": {
    "cloudWatchMonitoringConfiguration": {
      "logGroupName": "log_group_name",
      "logStreamNamePrefix": "log_stream_prefix"
    },
    "persistentAppUI": "ENABLED",
    "s3MonitoringConfiguration": {
      "logUri": "s3://my_s3_log_location"
    }
  }
}'
```

자세한 내용은 [대화형 엔드포인트를 생성하는 파라미터](#) 단원을 참조하십시오.

## JSON 파일에서 지정된 파라미터를 사용하여 대화형 엔드포인트 생성

1. 다음 JSON 파일에 표시된 대로 `create-managed-endpoint-request.json` 파일을 생성하고 엔드포인트에 필요한 파라미터를 지정합니다.

```
{
  "name": "MY_TEST_ENDPOINT",
  "virtualClusterId": "MY_CLUSTER_ID",
  "type": "JUPYTER_ENTERPRISE_GATEWAY",
  "releaseLabel": "emr-6.9.0-latest",
  "executionRoleArn": "arn:aws:iam::444455556666:role/JobExecutionRole",
  "configurationOverrides":
  {
    "applicationConfiguration":
    [
      {
        "classification": "spark-defaults",
        "properties":
        {
          "spark.driver.memory": "8G"
        }
      }
    ],
    "monitoringConfiguration":
    {
      "persistentAppUI": "ENABLED",
      "cloudWatchMonitoringConfiguration":
      {
        "logGroupName": "my_log_group",
        "logStreamNamePrefix": "log_stream_prefix"
      },
      "s3MonitoringConfiguration":
      {
        "logUri": "s3://my_s3_log_location"
      }
    }
  }
}
```

2. 로컬로 저장되었거나 Amazon S3에 저장된 `create-managed-endpoint-request.json` 파일 경로와 함께 `create-managed-endpoint` 명령을 사용합니다.

```
aws emr-containers create-managed-endpoint \
--cli-input-json file:///./create-managed-endpoint-request.json --region AWS-Region
```

## 대화형 엔드포인트 생성 출력

다음과 같은 출력이 터미널에 표시됩니다. 출력에는 새 대화형 엔드포인트의 이름과 식별자가 포함됩니다.

```
{
  "id": "1234567890abcdef0",
  "name": "example-endpoint-name",
  "arn": "arn:aws:emr-containers:us-west-2:111122223333:/
virtualclusters/444455556666/endpoints/444455556666",
  "virtualClusterId": "111122223333xxxxxxxx"
}
```

aws emr-containers create-managed-endpoint를 실행하면 EMR Studio와 대화형 엔드포인트 서버 간의 HTTPS 통신을 허용하는 자체 서명된 인증서가 생성됩니다.

create-managed-endpoint를 실행했지만 필수 조건을 완료하지 않은 경우 Amazon EMR은 계속하기 위해 취해야 하는 작업이 포함된 오류 메시지를 반환합니다.

## 대화형 엔드포인트를 생성하는 파라미터

주제

- [대화형 엔드포인트의 필수 파라미터](#)
- [대화형 엔드포인트의 선택적 파라미터](#)

### 대화형 엔드포인트의 필수 파라미터

대화형 엔드포인트를 생성할 때 다음 파라미터를 지정해야 합니다.

#### **--type**

JUPYTER\_ENTERPRISE\_GATEWAY를 사용합니다. 지원되는 유일한 유형입니다.

#### **--virtual-cluster-id**

Amazon EMR on EKS에 등록된 가상 클러스터의 식별자입니다.

**--name**

EMR Studio 사용자가 드롭다운 목록에서 선택하는 데 도움이 되는 대화형 엔드포인트의 설명 이름.

**--execution-role-arn**

필수 조건의 일부로 생성된 Amazon EMR on EKS에 대한 IAM 작업 실행 역할의 Amazon 리소스 이름(ARN)입니다.

**--release-label**

엔드포인트에 사용할 Amazon EMR 릴리스의 릴리스 레이블입니다. 예를 들어 `emr-6.9.0-latest`입니다. Amazon EMR on EKS는 Amazon EMR 릴리스 6.7.0 이상에서 대화형 엔드포인트를 지원합니다.

## 대화형 엔드포인트의 선택적 파라미터

대화형 엔드포인트를 생성할 때 선택적으로 다음 파라미터도 지정할 수 있습니다.

**--configuration-overrides**

애플리케이션의 기본 구성을 재정의하려면 구성 객체를 제공합니다. 간편 구문을 사용하여 구성을 제공하거나 JSON 파일의 구성 객체를 참조할 수 있습니다.

구성 객체는 분류, 속성 및 선택적 중첩 구성으로 이루어져 있습니다. 속성은 해당 파일에서 재정의하려는 설정으로 구성됩니다. 단일 JSON 객체에서 여러 애플리케이션에 대해 다양한 분류를 지정할 수 있습니다. 사용 가능한 구성 분류는 Amazon EMR on EKS 릴리스에 따라 달라집니다. Amazon EMR on EKS의 각 릴리스에서 사용할 수 있는 구성 분류 목록은 [Amazon EMR on EKS 릴리스](#) 섹션을 참조하세요. 각 릴리스에 대해 나열된 구성 분류 외에도 대화형 엔드포인트는 추가 분류 `jeg-config`를 제공합니다. 자세한 내용은 [Jupyter Enterprise Gateway\(JEG\) 구성 옵션](#) 단원을 참조하십시오.

## 대화형 엔드포인트에 대한 설정 구성

이 섹션에는 대화형 엔드포인트 및 포드 설정에 대한 다양한 구성을 다루는 일련의 주제가 포함되어 있습니다. 이를 통해 장애를 모니터링 및 해결하거나, Amazon S3 또는 로 로그 정보를 보내거나 Amazon CloudWatch Logs, 사용자 지정 포드 템플릿을 지정하는 대화형 엔드포인트를 생성할 수 있습니다.

### 주제

- [Spark 작업 모니터링](#)
- [대화형 엔드포인트가 있는 사용자 지정 포드 템플릿 지정](#)
- [JEG 포드를 노드 그룹에 배포](#)
- [Jupyter Enterprise Gateway\(JEG\) 구성 옵션](#)
- [PySpark 세션 파라미터 수정](#)
- [대화형 엔드포인트가 있는 사용자 지정 커널 이미지](#)

## Spark 작업 모니터링

실패를 모니터링하고 문제를 해결할 수 있도록 엔드포인트에서 시작된 작업이 Amazon S3, Amazon CloudWatch Logs 또는 둘 다로 로그 정보를 전송할 수 있게 대화형 엔드포인트를 구성합니다. 다음 섹션에서는 Amazon EMR on EKS 대화형 엔드포인트에서 Amazon EMR로 시작하는 Spark 작업에 대한 Spark 애플리케이션 로그를 Amazon S3로 보내는 방법을 설명합니다.

### Amazon S3 로그에 대한 IAM 정책 구성

커널에서 Amazon S3로 로그 데이터를 전송하려면 먼저 작업 실행 역할에 대한 권한 정책에 다음 권한 정책이 포함되어야 합니다. *amzn-s3-demo-destination-bucket*을 로깅 버킷 이름으로 바꿉니다.

### JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket",
        "arn:aws:s3:::amzn-s3-demo-bucket/*"
      ],
      "Sid": "AllowS3Putobject"
    }
  ]
}
```

}

**Note**

Amazon EMR on EKS는 S3 버킷도 생성할 수 있습니다. S3 버킷을 사용할 수 없는 경우 IAM 정책에 `s3:CreateBucket` 권한을 포함합니다.

S3 버킷에 로그를 전송하는 데 필요한 권한을 실행 역할에 부여한 후 로그 데이터는 다음 Amazon S3 위치로 전송됩니다. 이는 `create-managed-endpoint` 요청의 `monitoringConfiguration` 섹션에서 `s3MonitoringConfiguration`이 전달될 때 수행됩니다.

- 드라이버 로그 - `logUri/virtual-cluster-id/endpoints/endpoint-id/containers/spark-application-id/spark-application-id-driver/(stderr.gz/stdout.gz)`
- 실행기 로그 - `logUri/virtual-cluster-id/endpoints/endpoint-id/containers/spark-application-id/executor-pod-name-exec-<Number>/(stderr.gz/stdout.gz)`

**Note**

Amazon EMR on EKS는 엔드포인트 로그를 S3 버킷에 업로드하지 않습니다.

## 대화형 엔드포인트가 있는 사용자 지정 포드 템플릿 지정

드라이버와 실행기의 사용자 지정 포드 템플릿을 지정하는 대화형 엔드포인트를 생성할 수 있습니다. 포드 템플릿은 각 포드의 실행 방법을 결정하는 사양입니다. 포드 템플릿 파일을 사용하여 Spark 구성 이 지원하지 않는 드라이버 또는 실행기 포드 구성을 정의할 수 있습니다. 포드 템플릿은 현재 Amazon EMR 릴리스 6.3.0 이상에서 지원됩니다.

포드 템플릿에 대한 자세한 내용은 Amazon EMR on EKS 개발 안내서에서 [포드 템플릿 사용](#)을 참조하세요.

다음 예제에서는 포드 템플릿을 사용해 대화형 엔드포인트를 생성하는 방법을 보여줍니다.

```
aws emr-containers create-managed-endpoint \
  --type JUPYTER_ENTERPRISE_GATEWAY \
  --virtual-cluster-id virtual-cluster-id \
```

```
--name example-endpoint-name \
--execution-role-arn arn:aws:iam::aws-account-id:role/EKSClusterRole \
--release-label emr-6.9.0-latest \
--configuration-overrides '{
  "applicationConfiguration": [
    {
      "classification": "spark-defaults",
      "properties": {
        "spark.kubernetes.driver.podTemplateFile": "path/to/driver/
template.yaml",
        "spark.kubernetes.executor.podTemplateFile": "path/to/executor/
template.yaml"
      }
    }
  ]
}'
```

## JEG 포드를 노드 그룹에 배포

Jupyter Enterprise Gateway(JEG) 포드 배치는 특정 노드 그룹에 대화형 엔드포인트를 배포할 수 있는 기능입니다. 이 기능을 사용하여 대화형 엔드포인트에 대해 `instance type`과 같은 설정을 구성할 수 있습니다.

### 관리형 노드 그룹에 JEG 포드 연결

다음 구성 속성을 사용하면 JEG 포드가 배포될 Amazon EKS 클러스터의 관리형 노드 그룹 이름을 지정할 수 있습니다.

```
//payload
--configuration-overrides '{
  "applicationConfiguration": [
    {
      "classification": "endpoint-configuration",
      "properties": {
        "managed-nodegroup-name": NodeGroupName
      }
    }
  ]
}'
```

노드 그룹에는 노드 그룹에 속하는 모든 노드에 `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` Kubernetes 레이블이 연결되어 있어야 합니다. 이 태그가 있는 노드 그룹의 모든 노드를 나열하려면 다음 명령을 사용합니다.



```
kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-
ng=NodeGroupName
```

위 명령의 출력이 관리형 노드 그룹에 속하는 노드를 반환하지 않는 경우, 노드 그룹에 `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` Kubernetes 레이블이 연결된 노드가 없는 것입니다. 이 경우 아래 단계를 수행하여 노드 그룹의 노드에 해당 레이블을 연결합니다.

1. 다음 명령을 사용하여 관리형 노드 그룹 *NodeGroupName*의 모든 노드에 `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` Kubernetes 레이블을 추가합니다.

```
kubectl label nodes --selector eks:nodegroup-name=NodeGroupName for-use-with-emr-
containers-managed-endpoint-ng=NodeGroupName
```

2. 다음 명령을 사용하여 노드 레이블이 올바르게 지정되었는지 확인합니다.

```
kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-
ng=NodeGroupName
```

관리형 노드 그룹은 Amazon EKS 클러스터의 보안 그룹과 연결되어야 합니다. `eksctl`을 사용하여 클러스터와 관리형 노드 그룹을 생성한 경우 일반적으로 이에 해당합니다. 다음 단계를 사용하여 AWS 콘솔에서 이를 확인할 수 있습니다.

1. Amazon EKS 콘솔에서 클러스터로 이동합니다.
2. 클러스터의 네트워킹 탭으로 이동하여 클러스터 보안 그룹을 기록합니다.
3. 클러스터의 컴퓨팅 탭으로 이동하여 관리형 노드 그룹 이름을 클릭합니다.
4. 관리형 노드 그룹의 세부 정보 탭에서 이전에 기록한 클러스터 보안 그룹이 보안 그룹 아래에 나열되어 있는지 확인합니다.

관리형 노드 그룹이 Amazon EKS 클러스터 보안 그룹에 연결되지 않은 경우 노드 그룹 보안 그룹에 `for-use-with-emr-containers-managed-endpoint-sg=ClusterName/NodeGroupName` 태그를 연결해야 합니다. 아래 단계를 사용하여 이 태그를 연결합니다.

1. Amazon EC2 콘솔로 이동하고 왼쪽 탐색 창에서 보안 그룹을 클릭합니다.
2. 확인란을 클릭하여 관리형 노드 그룹의 보안 그룹을 선택합니다.
3. 태그 탭에서 태그 관리 버튼을 사용하여 `for-use-with-emr-containers-managed-endpoint-sg=ClusterName/NodeGroupName` 태그를 추가합니다.

## 자체 관리형 노드 그룹에 JEG 포드 연결

다음 구성 속성을 사용하면 JEG 포드가 배포될 Amazon EKS 클러스터의 자체 관리형 또는 비관리형 노드 그룹 이름을 지정할 수 있습니다.

```
//payload
--configuration-overrides '{
  "applicationConfiguration": [
    {
      "classification": "endpoint-configuration",
      "properties": {
        "self-managed-nodegroup-name": NodeGroupName
      }
    }
  ]
}'
```

노드 그룹에는 노드 그룹에 속하는 모든 노드에 `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` Kubernetes 레이블이 연결되어 있어야 합니다. 이 태그가 있는 노드 그룹의 모든 노드를 나열하려면 다음 명령을 사용합니다.

```
kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
```

위 명령의 출력이 자체 관리형 노드 그룹에 속하는 노드를 반환하지 않는 경우, 노드 그룹에 `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` Kubernetes 레이블이 연결된 노드가 없는 것입니다. 이 경우 아래 단계를 수행하여 노드 그룹의 노드에 해당 레이블을 연결합니다.

1. `eksctl`을 사용하여 자체 관리형 노드 그룹을 생성한 경우 다음 명령을 사용하여 자체 관리형 노드 그룹 *NodeGroupName*의 모든 노드에 `for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName` Kubernetes 레이블을 한 번에 추가합니다.

```
kubectl label nodes --selector alpha.eksctl.io/nodegroup-name=NodeGroupName for-use-with-emr-containers-managed-endpoint-ng=NodeGroupName
```

자체 관리형 노드 그룹을 생성할 때 `eksctl`을 사용하지 않았다면 위 명령의 선택기를 노드 그룹의 모든 노드에 연결된 다른 Kubernetes 레이블로 바꾸어야 합니다.

2. 다음 명령을 사용하여 노드 레이블이 올바르게 지정되었는지 확인합니다.

```
kubectl get nodes --show-labels | grep for-use-with-emr-containers-managed-endpoint-
ng=NodeGroupName
```

자체 관리형 노드 그룹의 보안 그룹에 `for-use-with-emr-containers-managed-endpoint-sg=ClusterName/NodeGroupName` 태그가 연결되어 있어야 합니다. 다음 단계를 사용하여 AWS Management Console에서 보안 그룹에 태그를 연결합니다.

1. Amazon EC2 콘솔로 이동합니다. 왼쪽 탐색 창에서 보안 그룹을 선택합니다.
2. 자체 관리형 노드 그룹의 보안 그룹 옆에 있는 확인란을 선택합니다.
3. 태그 탭에서 태그 관리 버튼을 사용하여 `for-use-with-emr-containers-managed-endpoint-sg=ClusterName/NodeGroupName` 태그를 추가합니다. *ClusterName* 및 *NodeGroupName*을 적절한 값으로 바꿉니다.

## 온디맨드 인스턴스가 있는 관리형 노드 그룹에 JEG 포드 연결

또한 Kubernetes 레이블 선택기라고 하는 추가 레이블을 정의하여 지정된 노드 또는 노드 그룹에서 대화형 엔드포인트를 실행하기 위한 추가 제약 또는 제한을 지정할 수 있습니다. 다음 예제에서는 JEG 포드에 대한 온디맨드 Amazon EC2 인스턴스를 사용하는 방법을 보여줍니다.

```
--configuration-overrides '{
  "applicationConfiguration": [
    {
      "classification": "endpoint-configuration",
      "properties": {
        "managed-nodegroup-name": NodeGroupName,
        "node-labels": "eks.amazonaws.com/capacityType:ON_DEMAND"
      }
    }
  ]
}'
```

### Note

`managed-nodegroup-name` 또는 `self-managed-nodegroup-name` 속성 중 하나와 함께 `node-labels` 속성을 사용할 수 있습니다.

## Jupyter Enterprise Gateway(JEG) 구성 옵션

Amazon EMR on EKS는 Jupyter Enterprise Gateway(JEG)를 사용하여 대화형 엔드포인트를 컵니다. 엔드포인트를 생성할 때 허용 목록에 있는 JEG 구성에 대해 다음 값을 설정할 수 있습니다.

- **RemoteMappingKernelManager.cull\_idle\_timeout** - 제한 시간(정수 초). 이 기간이 지나면 커널이 유휴 상태이며 컬링할 준비가 된 것으로 간주됩니다. 값이 0 이하인 경우 컬링이 비활성화됩니다. 제한 시간이 짧으면 네트워크 연결 상태가 좋지 않은 사용자의 커널이 컬링될 수 있습니다.
- **RemoteMappingKernelManager.cull\_interval** - 컬링 제한 시간 값을 초과하는 유휴 커널이 있는지 확인하는 간격(정수 초).

## PySpark 세션 파라미터 수정

Amazon EMR on EKS 릴리스 6.9.0부터 Amazon EMR Studio에서는 EMR 노트북 셀에서 %configure 매직 명령을 실행하여 PySpark 세션과 관련된 Spark 구성을 조정할 수 있습니다.

다음 예제에서는 Spark 드라이버 및 실행기의 메모리, 코어 및 기타 속성을 수정하는 데 사용할 수 있는 샘플 페이로드를 보여줍니다. conf 설정의 경우 [Apache Spark 구성 설명서](#)에 언급된 모든 Spark 구성을 구성할 수 있습니다.

```
%%configure -f
{
  "driverMemory": "16G",
  "driverCores": 4,
  "executorMemory" : "32G",
  "executorCores": 2,
  "conf": {
    "spark.dynamicAllocation.maxExecutors" : 10,
    "spark.dynamicAllocation.minExecutors": 1
  }
}
```

다음 예제에서는 Spark 런타임에 파일, pyFiles 및 jar 종속 항목을 추가하는 데 사용할 수 있는 샘플 페이로드를 보여줍니다.

```
%%configure -f
{
  "files": "s3://amzn-s3-demo-bucket-emr-eks/sample_file.txt",
  "pyFiles": : "path-to-python-files",
  "jars" : "path-to-jars"
```

}

## 대화형 엔드포인트가 있는 사용자 지정 커널 이미지

Amazon EMR Studio에서 대화형 워크로드를 실행할 때 애플리케이션에 대한 올바른 종속 항목을 확보하기 위해 대화형 엔드포인트에 맞게 도커 이미지를 사용자 지정하고 사용자 지정된 기본 커널 이미지를 실행할 수 있습니다. 대화형 엔드포인트를 나열하려면 사용자 지정 도커 이미지에 연결하려면 다음 단계를 수행합니다.

### Note

기본 이미지만 재정의할 수 있습니다. 새 커널 이미지 유형을 추가할 수 없습니다.

1. 사용자 지정된 도커 이미지를 생성 및 게시합니다. 기본 이미지에는 Spark 런타임 및 이와 함께 실행되는 노트북 커널이 포함되어 있습니다. 이미지를 생성하려면 [도커 이미지를 사용자 지정하는 방법](#)의 1단계~4단계를 수행하면 됩니다. 1단계에서는 Docker 파일의 기본 이미지 URI는 spark 대신 notebook-spark를 사용해야 합니다.

```
ECR-registry-account.dkr.ecr.Region.amazonaws.com/notebook-spark/container-image-tag
```

AWS 리전 및 컨테이너 이미지 태그를 선택하는 방법에 대한 자세한 내용은 [섹션을 참조하세요 기본 이미지 URI 선택에 관한 세부 정보](#).

2. 사용자 지정 이미지와 함께 사용할 수 있는 대화형 엔드포인트를 생성합니다.
  - a. 다음 콘텐츠를 포함하는 custom-image-managed-endpoint.json JSON 파일을 생성합니다. 이 예제에서는 Amazon EMR 릴리스 6.9.0을 사용합니다.

### Example

```
{
  "name": "endpoint-name",
  "virtualClusterId": "virtual-cluster-id",
  "type": "JUPYTER_ENTERPRISE_GATEWAY",
  "releaseLabel": "emr-6.9.0-latest",
  "executionRoleArn": "execution-role-arn",
  "configurationOverrides": {
    "applicationConfiguration": [
```

```

{
  "classification": "jupyter-kernel-overrides",
  "configurations": [
    {
      "classification": "python3",
      "properties": {
        "container-image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/custom-notebook-python:latest"
      }
    },
    {
      "classification": "spark-python-kubernetes",
      "properties": {
        "container-image": "123456789012.dkr.ecr.us-
west-2.amazonaws.com/custom-notebook-spark:latest"
      }
    }
  ]
}

```

- b. 다음 예제와 같이 JSON 파일에 지정된 구성을 사용하여 대화형 엔드포인트를 생성합니다. 자세한 내용은 [create-managed-endpoint 명령을 사용하여 대화형 엔드포인트 생성](#) 단원을 참조하십시오.

```
aws emr-containers create-managed-endpoint --cli-input-json custom-image-managed-endpoint.json
```

3. EMR Studio를 통해 대화형 엔드포인트에 연결합니다. 자세한 내용과 완료해야 할 단계는 AWS Workshop [Studio 문서의 Amazon EMR on EKS 섹션에서 Studio에서 연결](#)을 참조하세요.

## 대화형 엔드포인트 모니터링

Amazon EMR on EKS 버전 6.10 이상에서 대화형 엔드포인트는 커널 수명 주기 작업을 모니터링하고 관련 문제를 해결하기 위해 Amazon CloudWatch 지표를 생성합니다. 지표는 EMR Studio 또는 자체 호스팅 Jupyter Notebook과 같은 대화형 클라이언트에 의해 트리거됩니다. 대화형 엔드포인트에서 지원하는 각 작업에는 연결된 지표가 있습니다. 작업은 아래 테이블에서와 같이 각 지표에 대한 차원으로 모델링됩니다. 대화형 엔드포인트에서 생성된 지표는 계정의 사용자 지정 네임스페이스인 EMRContainers에서 볼 수 있습니다.

| 지표                  | 설명                                                                 | 단위  |
|---------------------|--------------------------------------------------------------------|-----|
| RequestCount        | 대화형 엔드포인트에서 처리한 작업의 누적 요청 수.                                       | 개수  |
| RequestLatency      | 요청이 대화형 엔드포인트에 도착하고 대화형 엔드포인트에서 응답을 보낸 시점의 시간.                     | 밀리초 |
| 4XXError            | 처리 중 작업 요청으로 인해 4xx 오류가 발생할 때 생성됩니다.                               | 개수  |
| 5XXError            | 작업 요청으로 인해 5Xxx 서버 측 오류가 발생할 때 생성됩니다.                              | 개수  |
| KernelLaunchSuccess | CreateKernel 작업에만 적용됩니다. 이 요청까지 포함하여 성공적으로 실행된 커널 실행의 누적 수를 나타냅니다. | 개수  |
| KernelLaunchFailure | CreateKernel 작업에만 적용됩니다. 이 요청을 포함하여 현재까지 발생한 커널 시작 실패 누적 수를 나타냅니다. | 개수  |

이 대화형 엔드포인트 지표에는 다음과 같은 차원이 연결되어 있습니다.

- **ManagedEndpointId** - 대화형 엔드포인트의 식별자
- **OperationName** - 대화형 클라이언트에서 트리거된 작업

**OperationName**에 대해 사용 가능한 값이 다음 테이블에 나와 있습니다.

| operationName         | 작업 설명                                                       |
|-----------------------|-------------------------------------------------------------|
| CreateKernel          | 대화형 엔드포인트가 커널을 시작하도록 요청합니다.                                 |
| ListKernels           | 대화형 엔드포인트가 이전에 동일한 세션 토큰을 사용하여 시작된 커널을 나열하도록 요청합니다.         |
| GetKernel             | 대화형 엔드포인트가 이전에 시작된 특정 커널에 대한 세부 정보를 가져오도록 요청합니다.            |
| ConnectKernel         | 대화형 엔드포인트가 노트북 클라이언트와 커널 간 연결을 설정하도록 요청합니다.                 |
| ConfigureKernel       | pyspark 커널에 <code>%%configure magic request</code> 를 게시합니다. |
| ListKernelSpecs       | 대화형 엔드포인트에 사용 가능한 커널 사양을 나열하도록 요청합니다.                       |
| GetKernelSpec         | 대화형 엔드포인트가 이전에 시작된 커널의 커널 사양을 가져오도록 요청합니다.                  |
| GetKernelSpecResource | 대화형 엔드포인트가 이전에 시작된 커널 사양과 관련된 특정 리소스를 가져오도록 요청합니다.          |

## 예제

특정 날짜에 대화형 엔드포인트에서 시작된 총 커널 수에 액세스하는 방법:

1. 사용자 지정 네임스페이스 선택: EMRContainers
2. ManagedEndpointId, OperationName - CreateKernel 선택
3. 통계 SUM 및 기간 1 day가 지정된 RequestCount 지표는 지난 24시간 동안 발생한 모든 커널 시작 요청을 제공합니다.



4. 통계 SUM 및 기간 1 day가 지정된 KernelLaunchSuccess 지표는 지난 24시간 동안 성공한 모든 커널 시작 요청을 제공합니다.

특정 날짜에 대화형 엔드포인트에서 실패한 커널 수에 액세스하는 방법:

1. 사용자 지정 네임스페이스 선택: EMRContainers
2. ManagedEndpointId, OperationName - CreateKernel 선택
3. 통계 SUM 및 기간 1 day가 지정된 KernelLaunchFailure 지표는 지난 24시간 동안 실패한 모든 커널 시작 요청을 제공합니다. 4XXError 및 5XXError 지표를 선택하여 어떤 종류의 커널 시작 실패가 발생했는지 알 수도 있습니다.

## 자체 호스팅 Jupyter Notebook 사용

Jupyter 또는 JupyterLab 노트북을 Amazon EC2 인스턴스 또는 자체 Amazon EKS 클러스터에서 자체 호스팅 Jupyter Notebook으로 호스팅하고 관리할 수 있습니다. 그런 다음 자체 호스팅 Jupyter Notebook으로 대화형 워크로드를 실행할 수 있습니다. 다음 섹션에서는 Amazon EKS 클러스터에 자체 호스팅 Jupyter Notebook을 설정하고 배포하는 프로세스를 안내합니다.

EKS 클러스터에 자체 호스팅 Jupyter Notebook 생성

- [보안 그룹 생성](#)
- [Amazon EMR on EKS 대화형 엔드포인트 생성](#)
- [대화형 엔드포인트의 게이트웨이 서버 URL 검색](#)
- [인증 토큰을 검색하여 대화형 엔드포인트에 연결](#)
- [예제: JupyterLab 노트북 배포](#)
- [자체 호스팅 Jupyter Notebook 삭제](#)

## 보안 그룹 생성

대화형 엔드포인트를 생성하고 자체 호스팅 Jupyter 또는 JupyterLab 노트북을 실행하려면 먼저 노트북과 대화형 엔드포인트 간 트래픽을 제어하는 보안 그룹을 생성해야 합니다. Amazon EC2 콘솔 또는 Amazon EC2 SDK를 사용하여 보안 그룹을 생성하려면 Amazon EC2 사용 설명서에서 [보안 그룹 생성](#)의 단계를 참조하세요. 노트북 서버를 배포하려는 VPC에서 보안 그룹을 생성해야 합니다.

이 안내서의 예제를 따르려면 Amazon EKS 클러스터와 동일한 VPC를 사용합니다. Amazon EKS 클러스터의 VPC와 다른 VPC에서 노트북을 호스팅하려면 두 VPC 사이에서 피어링 연결을 생성해야 할 수 있습니다. 두 VPC 사이에서 피어링 연결을 생성하는 단계는 Amazon VPC 시작 안내서의 [VPC 피어링 연결 생성](#)을 참조하세요.

다음 단계에서 [Amazon EMR on EKS 대화형 엔드포인트를 생성](#)하려면 보안 그룹의 ID가 필요합니다.

## Amazon EMR on EKS 대화형 엔드포인트 생성

노트북용 보안 그룹을 생성한 후에는 [가상 클러스터의 대화형 엔드포인트 생성](#)에 제공된 단계를 사용하여 대화형 엔드포인트를 생성합니다. [보안 그룹 생성](#)에서 노트북용으로 생성한 보안 그룹 ID를 제공해야 합니다.

다음 구성 재정의 설정에서 *your-notebook-security-group-id* 위치에 보안 ID를 삽입합니다.

```
--configuration-overrides '{
  "applicationConfiguration": [
    {
      "classification": "endpoint-configuration",
      "properties": {
        "notebook-security-group-id": "your-notebook-security-group-id"
      }
    }
  ],
  "monitoringConfiguration": {
    ...'
```

## 대화형 엔드포인트의 게이트웨이 서버 URL 검색

대화형 엔드포인트를 생성한 후 AWS CLI에서 `describe-managed-endpoint` 명령을 사용하여 게이트웨이 서버 URL을 검색합니다. 노트북을 엔드포인트에 연결하려면 이 URL이 필요합니다. 게이트웨이 서버 URL은 프라이빗 엔드포인트입니다.

```
aws emr-containers describe-managed-endpoint \
--region region \
--virtual-cluster-id virtualClusterId \
--id endpointId
```

처음에 엔드포인트 상태는 CREATING입니다. 몇 분 후 ACTIVE 상태로 전환됩니다. 엔드포인트 상태가 ACTIVE인 경우 사용할 준비가 된 것입니다.

aws emr-containers describe-managed-endpoint 명령이 활성 엔드포인트에서 반환하는 `serverUrl` 속성을 기록합니다. [자체 호스팅 Jupyter 또는 JupyterLab 노트북을 배포](#)할 때 노트북을 엔드포인트에 연결하려면 이 URL이 필요합니다.

## 인증 토큰을 검색하여 대화형 엔드포인트에 연결

Jupyter 또는 JupyterLab 노트북에서 대화형 엔드포인트에 연결하려면 `GetManagedEndpointSessionCredentials` API를 사용하여 세션 토큰을 생성해야 합니다. 토큰은 대화형 엔드포인트 서버에 연결하기 위한 인증 증명 역할을 합니다.

다음 명령은 아래 출력 예제를 통해 더 자세히 설명됩니다.

```
aws emr-containers get-managed-endpoint-session-credentials \
--endpoint-identifier endpointArn \
--virtual-cluster-identifier virtualClusterArn \
--execution-role-arn executionRoleArn \
--credential-type "TOKEN" \
--duration-in-seconds durationInSeconds \
--region region
```

### *endpointArn*

엔드포인트의 ARN. `describe-managed-endpoint` 직접 호출 결과에서 ARN을 찾을 수 있습니다.

### *virtualClusterArn*

가상 클러스터의 ARN.

### *executionRoleArn*

실행 역할의 ARN.

### *durationInSeconds*

토큰이 유효한 기간(초). 기본 기간은 15분(900)이고 최대 기간은 12시간(43200)입니다.

### *region*

엔드포인트와 동일한 리전.

출력은 다음 예제와 비슷합니다. [자체 호스팅 Jupyter 또는 JupyterLab 노트북을 배포](#)할 때 사용할 *session-token* 값을 기록합니다.

```
{
  "id": "credentialsId",
  "credentials": {
    "token": "session-token"
  },
  "expiresAt": "2022-07-05T17:49:38Z"
}
```

## 예제: JupyterLab 노트북 배포

위 단계를 완료했으면 이 예제 절차를 시도하여 대화형 엔드포인트가 있는 Amazon EKS 클러스터에 JupyterLab 노트북을 배포할 수 있습니다.

1. 네임스페이스를 생성하여 노트북 서버를 실행합니다.
2. 다음 콘텐츠를 포함하는 `notebook.yaml` 파일을 로컬로 생성합니다. 아래에서 파일 콘텐츠를 설명합니다.

```
apiVersion: v1
kind: Pod
metadata:
  name: jupyter-notebook
  namespace: namespace
spec:
  containers:
    - name: minimal-notebook
      image: jupyter/all-spark-notebook:lab-3.1.4 # open source image
      ports:
        - containerPort: 8888
      command: ["start-notebook.sh"]
      args: ["--LabApp.token=''"]
      env:
        - name: JUPYTER_ENABLE_LAB
          value: "yes"
        - name: KERNEL_LAUNCH_TIMEOUT
          value: "400"
        - name: JUPYTER_GATEWAY_URL
          value: "serverUrl"
        - name: JUPYTER_GATEWAY_VALIDATE_CERT
          value: "false"
        - name: JUPYTER_GATEWAY_AUTH_TOKEN
          value: "session-token"
```

Jupyter Notebook을 FarGate 전용 클러스터에 배포하는 경우 다음 예제와 같이 Jupyter 포드에 role 레이블을 지정합니다.

```
...
metadata:
  name: jupyter-notebook
  namespace: default
  labels:
    role: example-role-name-label
spec:
  ...
```

### ***namespace***

노트북을 배포할 Kubernetes 네임스페이스.

### ***serverUrl***

[대화형 엔드포인트의 게이트웨이 서버 URL 검색](#)에서 describe-managed-endpoint 명령이 반환한 serverUrl 속성.

### ***session-token***

[인증 토큰을 검색하여 대화형 엔드포인트에 연결](#)에서 get-managed-endpoint-session-credentials 명령이 반환한 session-token 속성.

## **KERNEL\_LAUNCH\_TIMEOUT**

커널이 RUNNING 상태가 될 때까지 대화형 엔드포인트가 기다리는 시간(초). 커널 시작 제한 시간을 적절한 값(최대 400초)으로 설정하여 커널 시작이 완료될 때까지 충분한 시간을 확보합니다.

## **KERNEL\_EXTRA\_SPARK\_OPTS**

선택적으로 Spark 커널에 대한 추가 Spark 구성을 전달할 수 있습니다. 다음 예제에서와 같이 Spark 구성 속성으로 값을 포함하는 이 환경 변수를 설정합니다.

```
- name: KERNEL_EXTRA_SPARK_OPTS
  value: "--conf spark.driver.cores=2
        --conf spark.driver.memory=2G
        --conf spark.executor.instances=2
        --conf spark.executor.cores=2
        --conf spark.executor.memory=2G"
```

```
--conf spark.dynamicAllocation.enabled=true
--conf spark.dynamicAllocation.shuffleTracking.enabled=true
--conf spark.dynamicAllocation.minExecutors=1
--conf spark.dynamicAllocation.maxExecutors=5
--conf spark.dynamicAllocation.initialExecutors=1
"
```

### 3. Amazon EKS 클러스터에 포드 사양을 배포합니다.

```
kubectl apply -f notebook.yaml -n namespace
```

그러면 Amazon EMR on EKS 대화형 엔드포인트에 연결된 최소 JupyterLab Notebook이 시작됩니다. 포드가 RUNNING 상태가 될 때까지 기다립니다. 다음 명령을 실행하여 상태를 확인할 수 있습니다.

```
kubectl get pod jupyter-notebook -n namespace
```

포드가 준비되면 get pod 명령은 다음과 비슷한 출력을 반환합니다.

| NAME             | READY | STATUS  | RESTARTS | AGE |
|------------------|-------|---------|----------|-----|
| jupyter-notebook | 1/1   | Running | 0        | 46s |

### 4. 노트북 보안 그룹을 노트북이 예약된 노드에 연결합니다.

- a. 먼저 describe pod 명령을 사용하여 jupyter-notebook 포드가 예약된 노드를 식별합니다.

```
kubectl describe pod jupyter-notebook -n namespace
```

- b. <https://console.aws.amazon.com/eks/home#/clusters>에서 Amazon EKS 콘솔을 엽니다.
- c. Amazon EKS 클러스터의 컴퓨팅 탭으로 이동하여 describe pod 명령으로 식별되는 노드를 선택합니다. 노드의 인스턴스 ID를 선택합니다.
- d. 작업 메뉴에서 보안 > 보안 그룹 변경을 선택하여 [보안 그룹 생성](#)에서 생성한 보안 그룹을 연결합니다.
- e. Jupyter 노트북 포드를 배포 [SecurityGroupPolicy](#)하는 경우 역할 레이블을 사용하여 Jupyter 노트북 포드에 적용할 AWS Fargate 생성합니다.

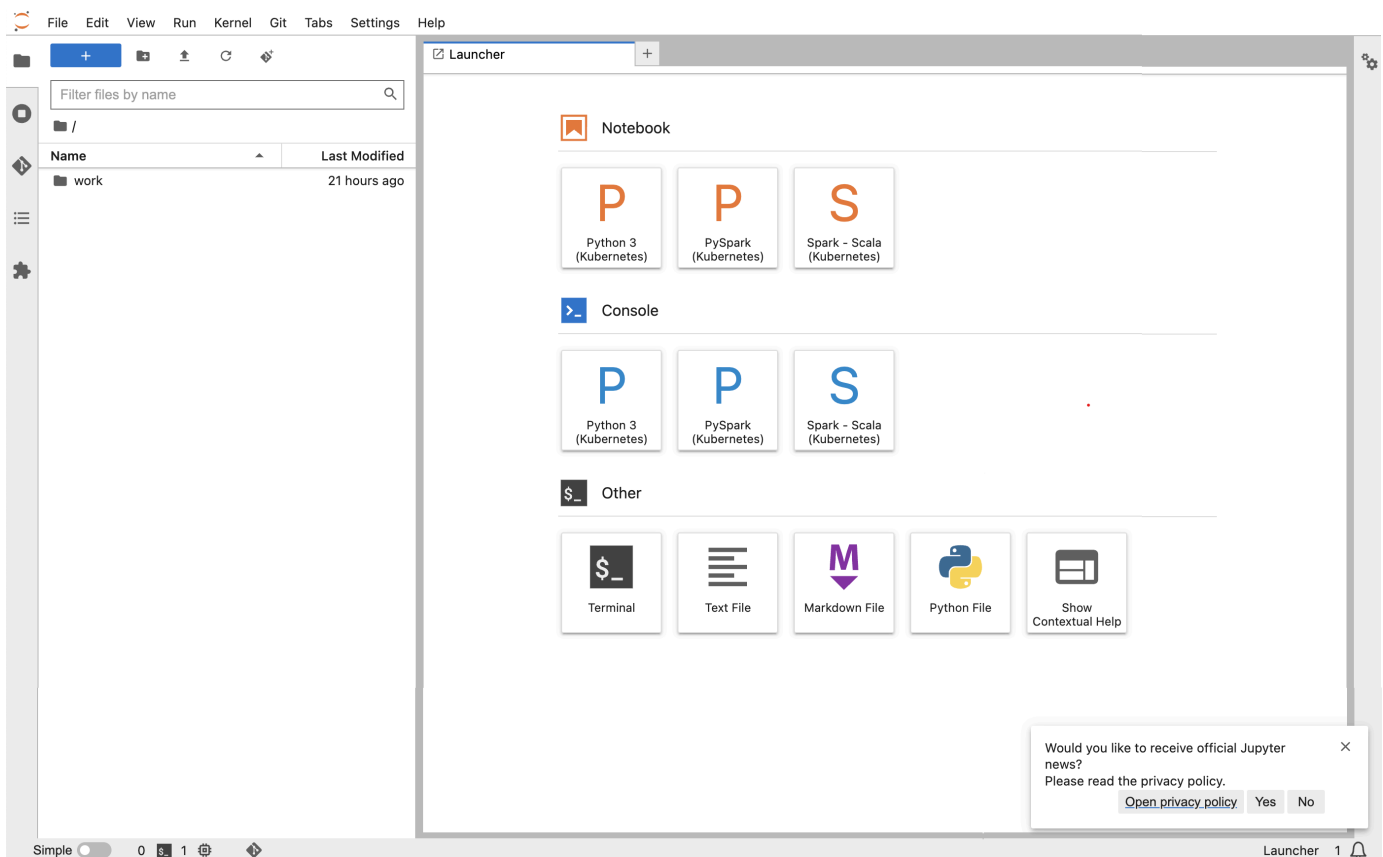
```
cat >my-security-group-policy.yaml <<EOF
apiVersion: vpcresources.k8s.aws/v1beta1
```

```
kind: SecurityGroupPolicy
metadata:
  name: example-security-group-policy-name
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: example-role-name-label
  securityGroups:
    groupIds:
      - your-notebook-security-group-id
EOF
```

5. 이제 JupyterLab 인터페이스에 로컬로 액세스할 수 있도록 포트 전달을 수행합니다.

```
kubectl port-forward jupyter-notebook 8888:8888 -n namespace
```

실행이 완료되면 로컬 브라우저로 이동하여 localhost:8888에서 JupyterLab 인터페이스를 확인합니다.



6. JupyterLab에서 새로운 Scala 노트북을 생성합니다. 다음은 Pi 값의 근사치를 구하기 위해 실행할 수 있는 샘플 코드 스니펫입니다.

```
import scala.math.random
import org.apache.spark.sql.SparkSession

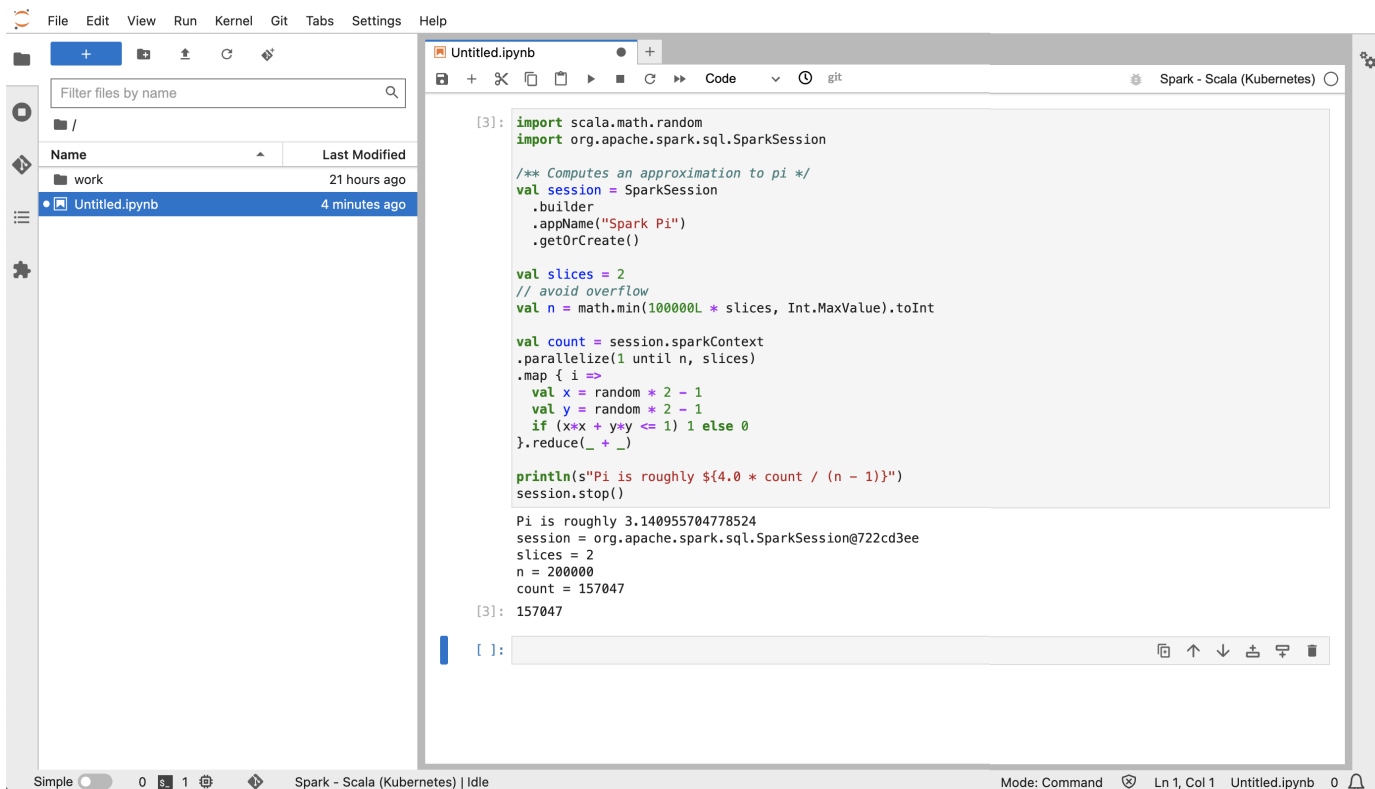
/** Computes an approximation to pi */
val session = SparkSession
    .builder
    .appName("Spark Pi")
    .getOrCreate()

val slices = 2
// avoid overflow
val n = math.min(100000L * slices, Int.MaxValue).toInt

val count = session.sparkContext
    .parallelize(1 until n, slices)
    .map { i =>
        val x = random * 2 - 1
        val y = random * 2 - 1
        if (x*x + y*y <= 1) 1 else 0
    }.reduce(_ + _)

println(s"Pi is roughly ${4.0 * count / (n - 1)}")
session.stop()
```





## 자체 호스팅 Jupyter Notebook 삭제

자체 호스팅 노트북을 삭제할 준비가 되면 대화형 엔드포인트와 보안 그룹도 삭제할 수 있습니다. 다음 순서대로 작업을 수행합니다.

1. 다음 명령을 사용하여 jupyter-notebook 포드를 삭제합니다.

```
kubectl delete pod jupyter-notebook -n namespace
```

2. 그런 다음 delete-managed-endpoint 명령을 사용하여 대화형 엔드포인트를 삭제합니다. 대화형 엔드포인트를 삭제하는 단계는 [대화형 엔드포인트 삭제](#) 섹션을 참조하세요. 처음에는 엔드포인트가 TERMINATING 상태가 됩니다. 모든 리소스가 정리되면 TERMINATED 상태로 전환됩니다.
3. [보안 그룹 생성](#)에서 생성한 노트북 보안 그룹을 다른 Jupyter Notebook 배포에 사용할 계획이 없다면 삭제할 수 있습니다. 자세한 내용은 Amazon EC2 사용 설명서에서 [보안 그룹 삭제](#)를 참조하세요.

## CLI 명령을 사용하여 대화형 엔드포인트에 대한 정보 가져오기

이 주제에서는 [create-managed-endpoint](#) 이외의 대화형 엔드포인트에서 지원되는 작업을 다룹니다.

### 대화형 엔드포인트 세부 정보 가져오기

대화형 엔드포인트를 생성한 후 describe-managed-endpoint AWS CLI 명령을 사용하여 세부 정보를 검색할 수 있습니다. *managed-endpoint-id*, *virtual-cluster-id*, *region*에 대한 고유한 값을 삽입합니다.

```
aws emr-containers describe-managed-endpoint --id managed-endpoint-id \
--virtual-cluster-id virtual-cluster-id --region region
```

출력은 지정된 엔드포인트(예: ARN, ID, 이름)를 포함하여 다음과 비슷합니다.

```
{
  "id": "as3ys2xxxxxxx",
  "name": "endpoint-name",
  "arn": "arn:aws:emr-containers:us-east-1:1828xxxxxxx:/virtualclusters/
1bhl6kwwyoxxxxxxxxxxxxxxxxxx/endpoints/as3ysxxxxxxxx",
  "virtualClusterId": "1bhl6kwwyoxxxxxxxxxxxxxxxxx",
  "type": "JUPYTER_ENTERPRISE_GATEWAY",
  "state": "ACTIVE",
  "releaseLabel": "emr-6.9.0-latest",
  "executionRoleArn": "arn:aws:iam::1828xxxxxxx:role/RoleName",
  "certificateAuthority": {
    "certificateArn": "arn:aws:acm:us-east-1:1828xxxxxxx:certificate/zzzzzzzz-
e59b-4ed0-aaaa-bbbbbbbbbbbb",
    "certificateData": "certificate-data"
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory": "8G"
        }
      }
    ],
    "monitoringConfiguration": {
      "persistentAppUI": "ENABLED",
```

```

        "cloudWatchMonitoringConfiguration": {
            "logGroupName": "log-group-name",
            "logStreamNamePrefix": "log-stream-name-prefix"
        },
        "s3MonitoringConfiguration": {
            "logUri": "s3-bucket-name"
        }
    },
    "serverUrl": "https://internal-k8s-namespace-ingressa-aaaaaaaaa-
zzzzzzzzzz.us-east-1.elb.amazonaws.com:18888 (https://internal-k8s-nspluto-
ingressa-51e860abbd-1620715833.us-east-1.elb.amazonaws.com:18888/)",
    "createdAt": "2022-09-19T12:37:49+00:00",
    "securityGroup": "sg-aaaaaaaaaaaaa",
    "subnetIds": [
        "subnet-111111111111",
        "subnet-222222222222",
        "subnet-333333333333"
    ],
    "stateDetails": "Endpoint created successfully. It took 3 Minutes 15 Seconds",
    "tags": {}
}

```

## 가상 클러스터에 연결된 모든 대화형 엔드포인트 나열

list-managed-endpoints AWS CLI 명령을 사용하여 지정된 가상 클러스터와 연결된 모든 대화형 엔드포인트 목록을 가져옵니다. virtual-cluster-id를 가상 클러스터 ID로 바꿉니다.

```
aws emr-containers list-managed-endpoints --virtual-cluster-id virtual-cluster-id
```

다음은 list-managed-endpoint 명령의 출력입니다.

```

{
  "endpoints": [{
    "id": "as3ys2xxxxxxx",
    "name": "endpoint-name",
    "arn": "arn:aws:emr-containers:us-east-1:1828xxxxxxx:/virtualclusters/
lbhl6kwwyoxxxxxxxxxxxxxxxxxx/endpoints/as3ysxxxxxxxx",
    "virtualClusterId": "lbhl6kwwyoxxxxxxxxxxxxxxxxx",
    "type": "JUPYTER_ENTERPRISE_GATEWAY",
    "state": "ACTIVE",
    "releaseLabel": "emr-6.9.0-latest",

```

```

    "executionRoleArn": "arn:aws:iam::1828xxxxxxx:role/RoleName",
    "certificateAuthority": {
      "certificateArn": "arn:aws:acm:us-east-1:1828xxxxxxx:certificate/zzzzzzzz-e59b-4ed0-aaaa-bbbbbbbbbbbb",
      "certificateData": "certificate-data"
    },
    "configurationOverrides": {
      "applicationConfiguration": [{
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.memory": "8G"
        }
      }],
      "monitoringConfiguration": {
        "persistentAppUI": "ENABLED",
        "cloudWatchMonitoringConfiguration": {
          "logGroupName": "log-group-name",
          "logStreamNamePrefix": "log-stream-name-prefix"
        },
        "s3MonitoringConfiguration": {
          "logUri": "s3-bucket-name"
        }
      }
    },
    "serverUrl": "https://internal-k8s-namespace-ingressa-aaaaaaaaa-zzzzzzzzz.us-east-1.elb.amazonaws.com:18888 (https://internal-k8s-nspluto-ingressa-51e860abbd-1620715833.us-east-1.elb.amazonaws.com:18888/)",
    "createdAt": "2022-09-19T12:37:49+00:00",
    "securityGroup": "sg-aaaaaaaaaaaaa",
    "subnetIds": [
      "subnet-11111111111",
      "subnet-22222222222",
      "subnet-33333333333"
    ],
    "stateDetails": "Endpoint created successfully. It took 3 Minutes 15 Seconds",
    "tags": {}
  }
}

```

## 대화형 엔드포인트 삭제

Amazon EMR on EKS 가상 클러스터와 연결된 대화형 엔드포인트를 삭제하려면 `delete-managed-endpoint` AWS CLI 명령을 사용합니다. 대화형 엔드포인트를 삭제하면 Amazon EMR on EKS는 해당 엔드포인트에 대해 생성된 기본 보안 그룹을 제거합니다.

명령에 다음 파라미터에 대한 값을 지정합니다.

- `--id`: 삭제하려는 대화형 엔드포인트의 식별자.
- `--virtual-cluster-id` - 삭제하려는 대화형 엔드포인트와 연결된 가상 클러스터의 식별자. 대화형 엔드포인트를 생성할 때 지정한 것과 동일한 가상 클러스터 ID입니다.

```
aws emr-containers delete-managed-endpoint --id managed-endpoint-id --virtual-cluster-id virtual-cluster-id
```

이 명령은 대화형 엔드포인트의 삭제를 확인하기 위해 다음과 비슷한 출력을 반환합니다.

```
{
  "id": "8gai4l4exxxxx",
  "virtualClusterId": "0b0qvauoy3ch1nqodxxxxxxx"
}
```

# Amazon EMR on EKS를 사용하여 Amazon S3 Express One Zone에 데이터 읽기, 쓰기 및 업로드

Amazon EMR 릴리스 7.2.0 이상에서는 작업 및 워크로드를 실행하는 경우 성능 개선을 위해 [Amazon S3 Express One Zone](#) 스토리지 클래스와 함께 Amazon EMR on EKS를 사용할 수 있습니다. S3 Express One Zone은 대부분의 지연 시간에 민감한 애플리케이션에서 일관되게 10밀리초 미만의 데이터 액세스를 지원하는 고성능 단일 영역 Amazon S3 스토리지 클래스입니다. 릴리스 시점에 S3 Express One Zone은 Amazon S3에서 지연 시간이 가장 낮고 성능은 가장 뛰어난 클라우드 객체 스토리지를 제공합니다.

## 사전 조건

Amazon EMR on EKS에서 S3 Express One Zone을 사용하려면 먼저 다음 사전 조건을 준수해야 합니다.

- [Amazon EMR on EKS 설정을 완료](#)했습니다.
- Amazon EMR on EKS를 설정한 후 [가상 클러스터를 생성](#)합니다.

## S3 Express One Zone 시작하기

S3 Express One Zone을 시작하려면 다음 단계를 수행합니다.

1. 작업 실행 역할에 필요한 CreateSession 권한을 추가합니다. S3 Express One Zone이 S3 객체에서 GET, LIST 또는 PUT과 같은 작업을 처음 수행하면 스토리지 클래스가 사용자를 대신하여 CreateSession을 직접 호출합니다. 다음은 CreateSession 권한을 부여하는 방법에 관한 예제입니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Resource": [
        "arn:aws:s3express:*:*:bucket/DOC-EXAMPLE-BUCKET"
      ]
    }
  ]
}
```

```

    ],
    "Action": [
        "s3express:CreateSession"
    ],
    "Sid": "AllowS3EXPRESSCreatesession"
  }
]
}

```

2. Apache Hadoop 커넥터 S3A를 사용하여 S3 Express 버킷에 액세스해야 하므로 s3a 스키마를 사용하여 커넥터를 사용하도록 Amazon S3 URI를 변경합니다. 스키마를 사용하지 않는 경우 s3 및 s3n 스키마에 대해 사용하는 파일 시스템 구현을 변경할 수 있습니다.

s3 체계를 변경하려면 다음 클러스터 구성을 지정하세요.

```

[
  {
    "Classification": "core-site",
    "Properties": {
      "fs.s3.impl": "org.apache.hadoop.fs.s3a.S3AFileSystem",
      "fs.AbstractFileSystem.s3.impl": "org.apache.hadoop.fs.s3a.S3A"
    }
  }
]

```

s3n 스키마를 변경하려면 다음 클러스터 구성을 지정합니다.

```

[
  {
    "Classification": "core-site",
    "Properties": {
      "fs.s3n.impl": "org.apache.hadoop.fs.s3a.S3AFileSystem",
      "fs.AbstractFileSystem.s3n.impl": "org.apache.hadoop.fs.s3a.S3A",
      "fs.s3a.endpoint.region": "us-west-2",
      "fs.s3a.change.detection.mode": "none",
      "fs.s3a.select.enabled": "false"
    }
  },
  {
    "Classification": "spark-defaults",
    "Properties": {

```

```
"spark.hadoop.fs.s3a.aws.credentials.provider":  
  "software.amazon.awssdk.auth.credentials.WebIdentityTokenFileCredentialsProvider",  
  "spark.sql.sources.fastS3PartitionDiscovery.enabled": "false"  
  }  
}  
]
```

3. spark-submit 구성에서 웹 자격 증명 제공업체를 사용합니다.

```
"spark.hadoop.fs.s3a.aws.credentials.provider=com.amazonaws.auth.WebIdentityTokenCredential
```



# 작업 모니터링

Amazon CloudWatch Events를 사용하여 Amazon EMR on EKS 가상 클러스터에서 실행하는 작업을 추적할 수 있습니다. 이벤트를 사용하여 가상 클러스터에서 실행하는 작업의 활동 및 상태를 추적할 수 있습니다. 다음 주제에서는 리소스 상태를 유지 관리하기 위해 모니터링을 효과적으로 구성하는 방법을 보여줍니다.

## 주제

- [Amazon CloudWatch Events를 사용하여 작업 모니터링](#)
- [CloudWatch Events를 통해 Amazon EMR on EKS 자동화](#)
- [예제: Lambda를 간접 호출하는 규칙 설정](#)
- [Amazon CloudWatch Events를 사용하여 재시도 정책을 포함하는 작업의 드라이버 포드 모니터링](#)

## Amazon CloudWatch Events를 사용하여 작업 모니터링

Amazon EMR on EKS에서는 작업 실행 상태가 변경될 때 이벤트를 생성합니다. 각 이벤트는 가상 클러스터 ID 및 영향을 받은 작업 실행 ID와 같은 이벤트에 대한 추가 세부 정보와 함께 이벤트가 생성된 날짜 및 시간과 같은 정보를 제공합니다.

이벤트를 사용하여 가상 클러스터에서 실행하는 작업의 활동 및 상태를 추적할 수 있습니다. 또한 Amazon CloudWatch Events를 사용하여 작업 실행 시 지정된 패턴과 일치하는 이벤트가 생성될 때 취할 조치를 정의할 수 있습니다. 이벤트는 작업 실행 수명 주기 동안 특정 발생을 모니터링하는 데 유용합니다. 예를 들어, 작업 실행 상태가 submitted에서 running으로 변경되는 시기를 모니터링할 수 있습니다. CloudWatch Events에 대한 자세한 내용은 [Amazon Amazon EventBridge 사용 설명서](#)를 참조하세요.

다음 테이블에는 Amazon EMR on EKS 이벤트와 함께 이벤트가 나타내는 상태 또는 상태 변경, 이벤트의 심각도 및 이벤트 메시지가 나와 있습니다. 각 이벤트는 자동으로 이벤트 스트림으로 전송되는 JSON 객체로 표시됩니다. JSON 객체에는 이벤트에 대한 추가 세부 정보가 포함되어 있습니다. CloudWatch Events를 사용한 이벤트 처리를 위한 규칙은 JSON 객체의 패턴과 일치하므로 JSON 객체는 이 규칙을 설정할 때 특히 중요합니다. 자세한 내용은 [Amazon EventBridge 사용 설명서](#)의 [Amazon EventBridge 이벤트 패턴](#) 및 Amazon EMR on EKS 이벤트를 참조하세요.

## 작업 실행 상태 변경 이벤트

| State     | 심각도  | 메시지                                                                                                                                                                    |
|-----------|------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SUBMITTED | INFO | UTC <i>Time</i> 에 <i>JobRunID</i> 작업 실행 ( <i>JobRunName</i> )이 가상 클러스터 <i>VirtualClusterId</i> 에 제출되었습니다.                                                              |
| RUNNING   | INFO | <i>Time</i> 에 가상 클러스터 <i>VirtualClusterId</i> 에서 <i>JobRunId</i> 작업 실행 ( <i>JobRunName</i> )의 실행이 시작되었습니다.                                                             |
| COMPLETED | INFO | 가상 클러스터 <i>VirtualClusterId</i> 에서 작업 실행 <i>jobRunId</i> ( <i>JobRunName</i> )이(<가) <i>Time</i> 에 완료되었습니다. 이 작업 실행이 <i>Time</i> 에 실행을 시작했고 완료되는 데 <i>Num</i> 분이 걸렸습니다. |
| 취소됨       | WARN | <i>Time</i> 에 가상 클러스터 <i>VirtualClusterId</i> 에서 <i>JobRunId</i> 작업 실행 ( <i>JobRunName</i> )에 대한 취소 요청이 성공했으며, 이제 작업 실행이 취소됩니다.                                        |
| FAILED    | 오류   | <i>Time</i> 에 가상 클러스터 <i>VirtualClusterId</i> 에서 <i>JobRunId</i> 작업 실행 ( <i>JobRunName</i> )이 완료되었습니다.                                                                 |

## CloudWatch Events를 통해 Amazon EMR on EKS 자동화

Amazon CloudWatch Events를 사용하여 AWS 서비스를 자동화하여 애플리케이션 가용성 문제 또는 리소스 변경과 같은 시스템 이벤트에 대응할 수 있습니다. AWS 서비스의 이벤트는 거의 실시간으로 CloudWatch Events로 전달됩니다. 원하는 이벤트만 표시하도록 간단한 규칙을 작성한 후 규칙과 일치하는 이벤트 발생 시 실행할 자동화 태스크를 지정할 수 있습니다. 자동으로 트리거할 수 있는 태스크는 다음과 같습니다.

- AWS Lambda 함수 호출
- Amazon EC2 Run Command 호출

- Amazon Kinesis Data Streams로 이벤트 릴레이
- AWS Step Functions 상태 시스템 활성화
- Amazon Simple Notification Service (SNS) 주제 또는 Amazon Simple Queue Service (SQS) 대기열 알림

다음은 Amazon EMR on EKS에서 CloudWatch Events를 사용하는 몇 가지 예입니다.

- 작업 실행에 성공하면 Lambda 함수 활성화
- 작업 실행에 실패하면 Amazon SNS 주제 알림

Amazon EMR on EKS는 SUBMITTED, RUNNING, CANCELLED, FAILED, COMPLETED 상태 변경에 대해 'detail-type:' 'EMR Job Run State Change'에 대한 CloudWatch Events를 생성합니다.

## 예제: Lambda를 간접 호출하는 규칙 설정

다음 단계를 사용하여 'EMR 작업 실행 상태 변경' 이벤트가 있을 때 Lambda를 간접 호출하는 CloudWatch 이벤트 규칙을 설정합니다.

```
aws events put-rule \  
--name cwe-test \  
--event-pattern '{"detail-type": ["EMR Job Run State Change"]}'
```

소유하고 있는 Lambda 함수를 새 대상으로 추가하고 다음과 같이 Lambda 함수를 간접 호출할 수 있는 권한을 CloudWatch Events에 부여합니다. **123456789012**를 계정 ID로 바꿉니다.

```
aws events put-targets \  
--rule cwe-test \  
--targets Id=1,Arn=arn:aws:lambda:us-east-1:123456789012:function:MyFunction
```

```
aws lambda add-permission \  
--function-name MyFunction \  
--statement-id MyId \  
--action 'lambda:InvokeFunction' \  
--principal events.amazonaws.com
```

 Note

순서가 잘못되었거나 누락되는 등 알림 이벤트의 순서 또는 존재 여부에 따라 프로그램을 작성할 수 없을 수 있습니다. 이벤트는 최선의 작업을 기반으로 발생합니다.

## Amazon CloudWatch Events를 사용하여 재시도 정책을 포함하는 작업의 드라이버 포드 모니터링

CloudWatch 이벤트를 사용하면 재시도 정책을 포함하는 작업에서 생성된 드라이버 포드를 모니터링할 수 있습니다. 자세한 내용은 이 안내서의 [재시도 정책으로 작업 모니터링](#) 섹션을 참조하세요.

## 가상 클러스터 관리

가상 클러스터는 Amazon EMR이 등록된 Kubernetes 네임스페이스입니다. 가상 클러스터를 생성, 설명, 나열 및 삭제할 수 있습니다. 가상 클러스터는 시스템의 추가 리소스를 소비하지 않습니다. 단일 가상 클러스터는 단일 Kubernetes 네임스페이스에 매핑됩니다. 이 관계를 감안하면 Kubernetes 네임스페이스를 모형화하는 것과 동일한 방식으로 가상 클러스터를 모형화하여 요구 사항을 충족할 수 있습니다. [Kubernetes Concepts Overview](#) 설명서에서 가능한 사용 사례를 참조하세요.

Amazon EMR을 Amazon EKS 클러스터의 Kubernetes 네임스페이스에 등록하려면 EKS 클러스터의 이름과 워크로드를 실행하기 위해 설정된 네임스페이스가 필요합니다. Amazon EMR에 등록된 이러한 클러스터는 물리적 컴퓨팅 또는 스토리지를 관리하지 않고 워크로드가 예약된 Kubernetes 네임스페이스를 가리키므로 가상 클러스터라고 합니다.

### Note

가상 클러스터를 생성하기 전에 먼저 [Amazon EMR on EKS 설정](#)에서 1~8단계를 완료해야 합니다.

### 주제

- [가상 클러스터 생성](#)
- [가상 클러스터 나열](#)
- [가상 클러스터 설명](#)
- [가상 클러스터 삭제](#)
- [가상 클러스터 상태](#)

## 가상 클러스터 생성

다음 명령을 실행하여 Amazon EMR을 EKS 클러스터의 네임스페이스에 등록하여 가상 클러스터를 생성합니다. **virtual\_cluster\_name**을 사용자가 제공한 가상 클러스터 이름으로 바꿉니다. **eks\_cluster-name**을 EKS 클러스터 이름으로 바꿉니다. **namespace\_name**을 네임스페이스 (Amazon EMR을 등록하려는 네임스페이스)로 바꿉니다.

```
aws emr-containers create-virtual-cluster \
--name virtual_cluster_name \
--container-provider '{
```

```

    "id": "eks_cluster_name",
    "type": "EKS",
    "info": {
        "eksInfo": {
            "namespace": "namespace_name"
        }
    }
}'

```

또는 다음 예제에서 볼 수 있듯이 가상 클러스터에 필요한 파라미터가 포함된 JSON 파일을 생성할 수 있습니다.

```

{
  "name": "virtual_cluster_name",
  "containerProvider": {
    "type": "EKS",
    "id": "eks_cluster_name",
    "info": {
      "eksInfo": {
        "namespace": "namespace_name"
      }
    }
  }
}

```

그런 다음, JSON 파일 경로와 함께 다음 `create-virtual-cluster` 명령을 실행합니다.

```

aws emr-containers create-virtual-cluster \
--cli-input-json file:///./create-virtual-cluster-request.json

```

#### Note

가상 클러스터가 생성되었는지 확인하려면 `list-virtual-clusters` 명령을 실행하거나 Amazon EMR 콘솔의 가상 클러스터 페이지로 이동하여 가상 클러스터의 상태를 확인합니다.

## 가상 클러스터 나열

가상 클러스터 상태를 보려면 다음 명령을 실행합니다.

```
aws emr-containers list-virtual-clusters
```

## 가상 클러스터 설명

다음 명령을 실행하여 네임스페이스, 상태, 등록 날짜 등 가상 클러스터에 대한 자세한 내용을 확인합니다. **123456**을 가상 클러스터 ID로 바꿉니다.

```
aws emr-containers describe-virtual-cluster --id 123456
```

## 가상 클러스터 삭제

다음 명령을 실행하여 가상 클러스터를 삭제합니다. **123456**을 가상 클러스터 ID로 바꿉니다.

```
aws emr-containers delete-virtual-cluster --id 123456
```

## 가상 클러스터 상태

다음 테이블에서는 가상 클러스터의 네 가지 가능한 상태를 설명합니다.

| State       | 설명                       |
|-------------|--------------------------|
| RUNNING     | 가상 클러스터가 RUNNING 상태입니다.  |
| TERMINATING | 요청한 가상 클러스터 종료가 진행 중입니다. |
| TERMINATED  | 요청한 종료가 완료되었습니다.         |
| ARRESTED    | 권한이 부족하여 요청한 종료에 실패했습니다. |

# Amazon EMR on EKS 자습서

이 섹션에서는 Amazon EMR on EKS 애플리케이션에서 작업할 때 일반적인 사용 사례를 설명합니다. 각 애플리케이션은 전문화되었으며, 고유한 단계를 수행하여 구성할 수 있습니다. 이 주제에서는 각 애플리케이션 사용에 대한 지침을 제공합니다.

## 주제

- [Amazon EMR on EKS에서 Delta Lake 사용](#)
- [Amazon EMR on EKS에서 Apache Iceberg 사용](#)
- [PyFlink 사용](#)
- [Flink에서 AWS Glue 사용](#)
- [Apache Flink에서 Apache Hudi 사용](#)
- [Amazon EMR on EKS에서 Apache Spark용 RAPIDS 액셀러레이터 사용](#)
- [Amazon EMR on EKS에서 Apache Spark용 Amazon Redshift 통합 사용](#)
- [Amazon EMR on EKS에서 Apache Spark의 사용자 지정 스케줄러로 Volcano 사용](#)
- [Amazon EMR on EKS에서 Apache Spark의 사용자 지정 스케줄러로 YuniKorn 사용](#)

## Amazon EMR on EKS에서 Delta Lake 사용

Delta Lake는 Lakehouse 아키텍처를 빌드하기 위한 오픈 소스 스토리지 프레임워크입니다. 다음 예제에서는 사용하도록 설정하는 방법을 보여줍니다.

Amazon EMR on EKS 애플리케이션에서 [Delta Lake](#)를 사용하는 방법

1. 애플리케이션 구성에서 Spark 작업을 제출하기 위해 작업 실행을 시작하는 경우 Delta Lake JAR 파일을 포함합니다.

```
--job-driver '{"sparkSubmitJobDriver" : {
    "sparkSubmitParameters" : "--jars local:///usr/share/aws/delta/lib/delta-core.jar,local:///usr/share/aws/delta/lib/delta-storage.jar,local:///usr/share/aws/delta/lib/delta-storage-s3-dynamodb.jar"}'}
```



**Note**

Amazon EMR 릴리스 7.0.0 이상은 Delta Lake 3.0(delta-core.jar 이름이 delta-spark.jar로 바뀜)을 사용합니다. Amazon EMR 릴리스 7.0.0 이상을 사용하는 경우 다음 예제와 같이 올바른 파일 이름을 사용해야 합니다.

```
--jars local:///usr/share/aws/delta/lib/delta-spark.jar
```

2. Delta Lake 추가 구성을 포함하고 AWS Glue 데이터 카탈로그를 메타스토어로 사용합니다.

```
--configuration-overrides '{
  "applicationConfiguration": [
    {
      "classification" : "spark-defaults",
      "properties" : {
        "spark.sql.extensions" : "io.delta.sql.DeltaSparkSessionExtension",

        "spark.sql.catalog.spark_catalog":"org.apache.spark.sql.delta.catalog.DeltaCatalog",
        "spark.hadoop.hive.metastore.client.factory.class":"com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClient"
      }
    }
  ]
}'
```

## Amazon EMR on EKS에서 Apache Iceberg 사용

Iceberg용 런타임 JAR에는 Spark 런타임 지원에 필요한 Iceberg 클래스가 포함되어 있습니다. 다음 절차에서는 Iceberg Spark 런타임을 사용하여 작업 실행을 시작하는 방법을 보여줍니다.

Amazon EMR on EKS 애플리케이션에서 Apache Iceberg를 사용하는 방법

1. 애플리케이션 구성에서 Spark 작업을 제출하기 위해 작업 실행을 시작하는 경우 Iceberg Spark 런타임 JAR 파일을 포함합니다.

```
--job-driver '{"sparkSubmitJobDriver" : {"sparkSubmitParameters" : "--jars
local:///usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar"}}'
```

2. Iceberg 추가 구성을 포함합니다.

```
--configuration-overrides '{
```

```

    "applicationConfiguration": [
      "classification" : "spark-defaults",
      "properties" : {
        "spark.sql.catalog.dev.warehouse" : "s3://amzn-s3-demo-bucket/EXAMPLE-
PREFIX/ ",
        "spark.sql.extensions ":"
org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions ",
        "spark.sql.catalog.dev" : "org.apache.iceberg.spark.SparkCatalog",
        "spark.sql.catalog.dev.catalog-impl" :
"org.apache.iceberg.aws.glue.GlueCatalog",
        "spark.sql.catalog.dev.io-impl": "org.apache.iceberg.aws.s3.S3FileIO"
      }
    ]
  }'

```

EMR의 Apache Iceberg 릴리스 버전에 대한 자세한 내용은 [Iceberg 릴리스 기록](#)을 참조하세요.

## 카탈로그 통합을 위한 Spark 세션 구성

### Iceberg AWS Glue 카탈로그 통합을 위한 Spark 세션 구성

이 샘플은 Iceberg를 AWS Glue 크롤러와 통합하는 방법을 보여줍니다.

```

spark-sql \
--conf spark.sql.catalog.rms = org.apache.iceberg.spark.SparkCatalog \
--conf spark.sql.catalog.rms.type = glue \
--conf spark.sql.catalog.rms.glue.id = glue RMS catalog ID \
--conf spark.sql.catalog.rms.glue.account-id = AWS account ID \

--conf spark.sql.extensions=
org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions

```

다음은 샘플 쿼리를 나타냅니다.

```
SELECT * FROM rms.rmsdb.table1
```

### Iceberg REST AWS Glue 카탈로그 통합을 위한 Spark 세션 구성

이 샘플은 Iceberg REST를 AWS Glue 크롤러와 통합하는 방법을 보여줍니다.

```
spark-sql \
```

```
--conf spark.sql.catalog.rms = org.apache.iceberg.spark.SparkCatalog \
--conf spark.sql.catalog.rms.type = rest \
--conf spark.sql.catalog.rms.warehouse = glue RMS catalog ID \
--conf spark.sql.catalog.rms.uri = glue endpoint URI/iceberg \
--conf spark.sql.catalog.rms.rest.sigv4-enabled = true \
--conf spark.sql.catalog.rms.rest.signing-name = glue \

--conf spark.sql.extensions=
    org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
```

다음은 샘플 쿼리를 나타냅니다.

```
SELECT * FROM rms.rmsdb.table1
```

이 구성은 Redshift Managed Storage에서만 작동합니다. Amazon S3용 FGAC는 지원되지 않습니다.

## PyFlink 사용

Amazon EMR on EKS 릴리스 6.15.0 이상에서는 PyFlink를 지원합니다. 이미 PyFlink 스크립트가 있는 경우 다음 중 하나를 수행할 수 있습니다.

- PyFlink 스크립트가 포함된 사용자 지정 이미지를 생성합니다.
- Amazon S3 위치에 스크립트 업로드

스크립트가 아직 없는 경우 다음 예제를 사용하여 PyFlink 작업을 시작할 수 있습니다. 이 예제에서는 S3에서 스크립트를 검색합니다. 이미지에 이미 포함된 스크립트와 함께 사용자 지정 이미지를 사용하는 경우 스크립트를 저장한 위치로 스크립트 경로를 업데이트해야 합니다. 스크립트가 S3 위치에 있는 경우 Amazon EMR on EKS에서 스크립트를 검색하여 Flink 컨테이너의 `/opt/flink/usr/lib/` 디렉터리 아래에 배치합니다.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: python-example
spec:
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "1"
  executionRoleArn: job-execution-role
  emrReleaseLabel: "emr-6.15.0-flink-latest"
```

```

jobManager:
  highAvailabilityEnabled: false
  replicas: 1
  resource:
    memory: "2048m"
    cpu: 1
taskManager:
  resource:
    memory: "2048m"
    cpu: 1
job:
  jarURI: s3://S3 bucket with your script/pyflink-script.py
  entryClass: "org.apache.flink.client.python.PythonDriver"
  args: ["-py", "/opt/flink/usrlib/pyflink-script.py"]
  parallelism: 1
  upgradeMode: stateless

```

## Flink에서 AWS Glue 사용

Amazon EMR on EKS with Apache Flink 릴리스 6.15.0 이상에서는 AWS Glue 데이터 카탈로그를 스트리밍 및 배치 SQL 워크플로의 메타데이터 스토어로 사용할 수 있습니다.

먼저 Flink SQL 카탈로그 역할을 default 하는 AWS Glue 데이터베이스를 생성해야 합니다. 이 Flink 카탈로그는 데이터베이스, 테이블, 파티션, 보기, 함수 및 기타 외부 시스템의 데이터에 액세스하는 데 필요한 기타 정보와 같은 메타데이터를 저장합니다.

```

aws glue create-database \
  --database-input "{\"Name\":\"default\"}"

```

AWS Glue 지원을 활성화하려면 FlinkDeployment 사양을 사용합니다. 이 예제 사양은 Python 스크립트를 사용하여 일부 Flink SQL 문을 빠르게 실행하여 AWS Glue 카탈로그와 상호 작용합니다.

```

apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: python-example
spec:
  flinkVersion: v1_17
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "1"
    aws.glue.enabled: "true"

```

```

executionRoleArn: job-execution-role-arn;
emrReleaseLabel: "emr-6.15.0-flink-latest"
jobManager:
  highAvailabilityEnabled: false
  replicas: 1
  resource:
    memory: "2048m"
    cpu: 1
taskManager:
  resource:
    memory: "2048m"
    cpu: 1
job:
  jarURI: s3://<S3_bucket_with_your_script/pyflink-glue-script.py
  entryClass: "org.apache.flink.client.python.PythonDriver"
  args: ["-py", "/opt/flink/usrlib/pyflink-glue-script.py"]
  parallelism: 1
  upgradeMode: stateless

```

다음은 PyFlink 스크립트에 대한 예제입니다.

```

import logging
import sys
from pyflink.datastream import StreamExecutionEnvironment
from pyflink.table import StreamTableEnvironment

def glue_demo():
    env = StreamExecutionEnvironment.get_execution_environment()
    t_env = StreamTableEnvironment.create(stream_execution_environment=env)
    t_env.execute_sql("""
        CREATE CATALOG glue_catalog WITH (
            'type' = 'hive',
            'default-database' = 'default',
            'hive-conf-dir' = '/glue/confs/hive/conf',
            'hadoop-conf-dir' = '/glue/confs/hadoop/conf'
        )
    """)
    t_env.execute_sql("""
        USE CATALOG glue_catalog;
    """)
    t_env.execute_sql("""
        DROP DATABASE IF EXISTS eks_flink_db CASCADE;
    """)

```

```

t_env.execute_sql("""
    CREATE DATABASE IF NOT EXISTS eks_flink_db WITH ('hive.database.location-
uri'= 's3a://S3-bucket-to-store-metadata/flink/flink-glue-for-hive/warehouse/');
    """)
t_env.execute_sql("""
    USE eks_flink_db;
    """)
t_env.execute_sql("""
    CREATE TABLE IF NOT EXISTS eksglueorders (
        order_number BIGINT,
        price          DECIMAL(32,2),
        buyer          ROW first_name STRING, last_name STRING,
        order_time      TIMESTAMP(3)
    ) WITH (
        'connector' = 'datagen'
    );
    """)
t_env.execute_sql("""
    CREATE TABLE IF NOT EXISTS eksdestglueorders (
        order_number BIGINT,
        price          DECIMAL(32,2),
        buyer          ROW first_name STRING, last_name STRING,
        order_time      TIMESTAMP(3)
    ) WITH (
        'connector' = 'filesystem',
        'path' = 's3://S3-bucket-to-store-metadata/flink/flink-glue-for-hive/
warehouse/eksdestglueorders',
        'format' = 'json'
    );
    """)
t_env.execute_sql("""
    CREATE TABLE IF NOT EXISTS print_table (
        order_number BIGINT,
        price          DECIMAL(32,2),
        buyer          ROW first_name STRING, last_name STRING,
        order_time      TIMESTAMP(3)
    ) WITH (
        'connector' = 'print'
    );
    """)
t_env.execute_sql("""
    EXECUTE STATEMENT SET
    BEGIN
    INSERT INTO eksdestglueorders SELECT * FROM eksglueorders LIMIT 10;

```

```

        INSERT INTO print_table SELECT * FROM eksdestglueorders;
    END;
    """)

if __name__ == '__main__':
    logging.basicConfig(stream=sys.stdout, level=logging.INFO, format="%(message)s")
    glue_demo()

```

## Apache Flink에서 Apache Hudi 사용

Apache Hudi는 삽입, 업데이트, 업서트 및 삭제와 같은 레코드 수준 작업을 통해 데이터 관리 및 데이터 파이프라인 개발을 간소화하는 데 사용할 수 있는 오픈 소스 데이터 관리 프레임워크입니다. Amazon S3에서 효율적인 데이터 관리와 결합하여 Hudi는 데이터를 실시간으로 수집 및 업데이트할 수 있도록 합니다. Hudi는 데이터세트에서 실행하는 모든 작업의 메타데이터를 유지 관리하므로 모든 작업은 원자적이고 일관되게 유지됩니다.

Apache Hudi는 Amazon EMR 릴리스 7.2.0 이상의 Apache Flink와 함께 Amazon EMR on EKS에서 사용할 수 있습니다. Apache Hudi 작업을 시작하고 제출하는 방법을 알아보려면 다음 단계를 참조하세요.

### Apache Hudi 작업 제출

Apache Hudi 작업을 제출하는 방법을 알아보려면 다음 단계를 참조하세요.

1. AWS 이름이 인 Glue 데이터베이스를 생성합니다default.

```
aws glue create-database --database-input "{\"Name\":\"default\"}"
```

2. [Flink Kubernetes Operator SQL Example](#)에 따라 flink-sql-runner.jar 파일을 빌드합니다.
3. 다음과 같은 Hudi SQL 스크립트를 생성합니다.

```

CREATE CATALOG hudi_glue_catalog WITH (
  'type' = 'hudi',
  'mode' = 'hms',
  'table.external' = 'true',
  'default-database' = 'default',
  'hive.conf.dir' = '/glue/confs/hive/conf/',
  'catalog.path' = 's3://<hudi-example-bucket>/FLINK_HUDI/warehouse/'
);

```

```

USE CATALOG hudi_glue_catalog;
CREATE DATABASE IF NOT EXISTS hudi_db;
use hudi_db;

CREATE TABLE IF NOT EXISTS hudi-flink-example-table(
    uuid VARCHAR(20),
    name VARCHAR(10),
    age INT,
    ts TIMESTAMP(3),
    `partition` VARCHAR(20)
)
PARTITIONED BY (`partition`)
WITH (
    'connector' = 'hudi',
    'path' = 's3://<hudi-example-bucket>/hudi-flink-example-table',
    'hive_sync.enable' = 'true',
    'hive_sync.mode' = 'glue',
    'hive_sync.table' = 'hudi-flink-example-table',
    'hive_sync.db' = 'hudi_db',
    'compaction.delta_commits' = '1',
    'hive_sync.partition_fields' = 'partition',
    'hive_sync.partition_extractor_class' =
    'org.apache.hudi.hive.MultiPartKeysValueExtractor',
    'table.type' = 'COPY_ON_WRITE'
);

EXECUTE STATEMENT SET
BEGIN

INSERT INTO hudi-flink-example-table VALUES
    ('id1','Alex',23,TIMESTAMP '1970-01-01 00:00:01','par1'),
    ('id2','Stephen',33,TIMESTAMP '1970-01-01 00:00:02','par1'),
    ('id3','Julian',53,TIMESTAMP '1970-01-01 00:00:03','par2'),
    ('id4','Fabian',31,TIMESTAMP '1970-01-01 00:00:04','par2'),
    ('id5','Sophia',18,TIMESTAMP '1970-01-01 00:00:05','par3'),
    ('id6','Emma',20,TIMESTAMP '1970-01-01 00:00:06','par3'),
    ('id7','Bob',44,TIMESTAMP '1970-01-01 00:00:07','par4'),
    ('id8','Han',56,TIMESTAMP '1970-01-01 00:00:08','par4');

END;

```

4. Hudi SQL 스크립트 및 flink-sql-runner.jar 파일을 S3 위치에 업로드합니다.
5. FlinkDeployments YAML 파일에서 hudi.enabled를 true로 설정합니다.



```
spec:
  flinkConfiguration:
    hudi.enabled: "true"
```

6. YAML 파일을 생성하여 구성을 실행합니다. 이 예제 파일 이름은 `hudi-write.yaml`입니다.

```
apiVersion: flink.apache.org/v1beta1
kind: FlinkDeployment
metadata:
  name: hudi-write-example
spec:
  flinkVersion: v1_18
  flinkConfiguration:
    taskmanager.numberOfTaskSlots: "2"
    hudi.enabled: "true"
  executionRoleArn: "<JobExecutionRole>"
  emrReleaseLabel: "emr-7.13.0-flink-latest"
  jobManager:
    highAvailabilityEnabled: false
    replicas: 1
    resource:
      memory: "2048m"
      cpu: 1
  taskManager:
    resource:
      memory: "2048m"
      cpu: 1
  job:
    jarURI: local:///opt/flink/usrlib/flink-sql-runner.jar
    args: ["/opt/flink/scripts/hudi-write.sql"]
    parallelism: 1
    upgradeMode: stateless
  podTemplate:
    spec:
      initContainers:
        - name: flink-sql-script-download
          args:
            - s3
            - cp
            - s3://<s3_location>/hudi-write.sql
            - /flink-scripts
      image: amazon/aws-cli:latest
```

```

    imagePullPolicy: Always
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
      - mountPath: /flink-scripts
        name: flink-scripts
- name: flink-sql-runner-download
  args:
    - s3
    - cp
    - s3://<s3_location>/flink-sql-runner.jar
    - /flink-artifacts
  image: amazon/aws-cli:latest
  imagePullPolicy: Always
  resources: {}
  terminationMessagePath: /dev/termination-log
  terminationMessagePolicy: File
  volumeMounts:
    - mountPath: /flink-artifacts
      name: flink-artifact
containers:
- name: flink-main-container
  volumeMounts:
    - mountPath: /opt/flink/scripts
      name: flink-scripts
    - mountPath: /opt/flink/usrlib
      name: flink-artifact
volumes:
- emptyDir: {}
  name: flink-scripts
- emptyDir: {}
  name: flink-artifact

```

## 7. [Flink Kubernetes 연산자](#)에 Flink Hudi 작업을 제출합니다.

```
kubectl apply -f hudi-write.yaml
```

# Amazon EMR on EKS에서 Apache Spark용 RAPIDS 액셀러레이터 사용

Amazon EMR on EKS에서는 Apache Spark용 Nvidia RAPIDS 액셀러레이터 작업을 실행할 수 있습니다. 이 자습서에서는 EC2 그래픽 처리 장치(GPU) 인스턴스 유형에서 RAPIDS를 사용하여 Spark 작업을 실행하는 방법을 다룹니다. 자습서에서는 다음 버전을 사용합니다.

- Amazon EMR on EKS 릴리스 버전 6.9.0 이상
- Apache Spark 3.x

[Apache Spark용 Nvidia RAPIDS 액셀러레이터](#) 플러그인을 사용하면 Amazon EC2 GPU 인스턴스 유형으로 Spark를 가속화할 수 있습니다. 이러한 기술을 함께 사용하면 코드를 변경하지 않고도 데이터 과학 파이프라인을 가속화할 수 있습니다. 따라서 데이터 처리 및 모델 학습에 필요한 실행 시간도 줄어듭니다. 더 짧은 시간에 더 많은 작업을 수행하면 인프라 비용을 절감할 수 있습니다.

시작하기 전에 다음 리소스가 준비되었는지 확인합니다.

- Amazon EMR on EKS 가상 클러스터
- GPU 지원 노드 그룹이 있는 Amazon EKS 클러스터

Amazon EKS 가상 클러스터는 Amazon EKS 클러스터의 Kubernetes 네임스페이스에 등록된 핸들로, Amazon EMR on EKS에서 관리됩니다. 이 핸들을 사용하면 Amazon EMR에서 Kubernetes 네임스페이스를 실행 중인 작업의 대상으로 사용할 수 있습니다. 가상 클러스터를 설정하는 방법에 대한 자세한 내용은 이 안내서에서 [Amazon EMR on EKS 설정](#) 섹션을 참조하세요.

GPU 인스턴스가 있는 노드 그룹을 포함하는 Amazon EKS 가상 클러스터를 구성해야 합니다. Nvidia 디바이스 플러그인으로 노드를 구성해야 합니다. 자세한 내용은 [관리형 노드 그룹](#)을 참조하세요.

GPU 지원 노드 그룹을 추가하도록 Amazon EKS 클러스터를 구성하려면 다음 절차를 수행합니다.

## GPU 지원 노드 그룹을 추가하는 방법

1. 다음 [create-nodegroup](#) 명령을 사용하여 GPU 지원 노드 그룹을 생성합니다. Amazon EKS 클러스터에 대한 올바른 파라미터를 대체해야 합니다. Spark RAPIDS를 지원하는 인스턴스 유형(예: P4, P3, G5 또는 G4dn)을 사용합니다.

```
aws eks create-nodegroup \
```

```
--cluster-name EKS_CLUSTER_NAME \
--nodegroup-name NODEGROUP_NAME \
--scaling-config minSize=0,maxSize=5,desiredSize=2 CHOOSE_APPROPRIATELY \
--ami-type AL2_x86_64_GPU \
--node-role NODE_ROLE \
--subnets SUBNETS_SPACE_DELIMITED \
--remote-access ec2SshKey= SSH_KEY \
--instance-types GPU_INSTANCE_TYPE \
--disk-size DISK_SIZE \
--region AWS_REGION
```

2. 클러스터의 각 노드에서 GPU 수를 생성하고 클러스터에서 GPU 지원 컨테이너를 실행하도록 클러스터에 Nvidia 디바이스 플러그인을 설치합니다. 다음 코드를 실행하여 플러그인을 설치합니다.

```
kubectl apply -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/v0.9.0/nvidia-device-plugin.yml
```

3. 클러스터의 각 노드에서 사용 가능한 GPU 수를 검증하려면 다음 명령을 실행합니다.

```
kubectl get nodes "-o=custom-columns=NAME:.metadata.name,GPU:.status.allocatable.nvidia\.com/gpu"
```

## Spark RAPIDS 작업을 실행하는 방법

1. Amazon EMR on EKS 클러스터에 Spark RAPIDS 작업을 제출합니다. 다음 코드에서는 작업을 시작하는 명령 예제를 보여줍니다. 작업을 처음 실행할 때 이미지를 다운로드하고 노드에 캐싱하는데 몇 분 정도 걸릴 수 있습니다.

```
aws emr-containers start-job-run \
--virtual-cluster-id VIRTUAL_CLUSTER_ID \
--execution-role-arn JOB_EXECUTION_ROLE \
--release-label emr-6.9.0-spark-rapids-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "local:///usr/lib/spark/examples/jars/spark-examples.jar", "entryPointArguments": ["10000"], "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi "}}' \
---configuration-overrides '{"applicationConfiguration": [{"classification": "spark-defaults", "properties": {"spark.executor.instances": "2", "spark.executor.memory": "2G"}}, {"monitoringConfiguration": {"cloudWatchMonitoringConfiguration": {"logGroupName": "LOG_GROUP_NAME", "s3MonitoringConfiguration": {"logUri": "LOG_GROUP_STREAM"}}}]'
```

2. Spark RAPIDS 액셀러레이터가 활성화되었는지 검증하려면 Spark 드라이버 로그를 확인합니다. 이 로그는 CloudWatch 또는 `start-job-run` 명령을 실행할 때 지정한 S3 위치에 저장됩니다. 다음 예제는 일반적으로 로그 줄의 모양을 보여줍니다.

```
22/11/15 00:12:44 INFO RapidsPluginUtils: RAPIDS Accelerator build:
{version=22.08.0-amzn-0, user=release, url=, date=2022-11-03T03:32:45Z, revision=,
cudf_version=22.08.0, branch=}
22/11/15 00:12:44 INFO RapidsPluginUtils: RAPIDS Accelerator JNI build:
{version=22.08.0, user=, url=https://github.com/NVIDIA/spark-rapids-jni.git,
date=2022-08-18T04:14:34Z, revision=a1b23cd_sample, branch=HEAD}
22/11/15 00:12:44 INFO RapidsPluginUtils: cudf build: {version=22.08.0,
user=, url=https://github.com/rapidsai/cudf.git, date=2022-08-18T04:14:34Z,
revision=a1b23ce_sample, branch=HEAD}
22/11/15 00:12:44 WARN RapidsPluginUtils: RAPIDS Accelerator 22.08.0-amzn-0 using
cudf 22.08.0.
22/11/15 00:12:44 WARN RapidsPluginUtils:
spark.rapids.sql.multiThreadedRead.numThreads is set to 20.
22/11/15 00:12:44 WARN RapidsPluginUtils: RAPIDS Accelerator is enabled, to disable
GPU support set `spark.rapids.sql.enabled` to false.
22/11/15 00:12:44 WARN RapidsPluginUtils: spark.rapids.sql.explain is set to
`NOT_ON_GPU`. Set it to 'NONE' to suppress the diagnostics logging about the query
placement on the GPU.
```

3. GPU에서 실행될 작업을 확인하려면 다음 단계를 수행하여 추가 로깅을 활성화합니다. 'spark.rapids.sql.explain : ALL' 구성을 참고합니다.

```
aws emr-containers start-job-run \
--virtual-cluster-id VIRTUAL_CLUSTER_ID \
--execution-role-arn JOB_EXECUTION_ROLE \
--release-label emr-6.9.0-spark-rapids-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "local:///usr/lib/
spark/examples/jars/spark-examples.jar","entryPointArguments": ["10000"],
"sparkSubmitParameters":"--class org.apache.spark.examples.SparkPi "}}' \
---configuration-overrides '{"applicationConfiguration":
[{"classification": "spark-defaults","properties":
{"spark.rapids.sql.explain":"ALL","spark.executor.instances":
"2","spark.executor.memory": "2G"}}],"monitoringConfiguration":
{"cloudWatchMonitoringConfiguration": {"logGroupName":
"LOG_GROUP_NAME"},"s3MonitoringConfiguration": {"logUri": "LOG_GROUP_STREAM"}}}'
```

이전 명령은 GPU를 사용하는 작업의 예제입니다. 출력은 아래 예제와 비슷합니다. 출력을 이해하는 데 도움이 필요하면 이 키를 참조하세요.

- \* - GPU에서 작동하는 작업 표시
- ! - GPU에서 실행할 수 없는 작업 표시
- @ - GPU에서 작동하지만 GPU에서 실행할 수 없는 계획에 있기 때문에 실행되지 않는 작업 표시

```

22/11/15 01:22:58 INFO GpuOverrides: Plan conversion to the GPU took 118.64 ms
22/11/15 01:22:58 INFO GpuOverrides: Plan conversion to the GPU took 4.20 ms
22/11/15 01:22:58 INFO GpuOverrides: GPU plan transition optimization took 8.37 ms
22/11/15 01:22:59 WARN GpuOverrides:
    *Exec <ProjectExec> will run on GPU
      *Expression <Alias> substring(cast(date#149 as string), 0, 7) AS month#310
will run on GPU
      *Expression <Substring> substring(cast(date#149 as string), 0, 7) will run
on GPU
      *Expression <Cast> cast(date#149 as string) will run on GPU
    *Exec <SortExec> will run on GPU
      *Expression <SortOrder> date#149 ASC NULLS FIRST will run on GPU
    *Exec <ShuffleExchangeExec> will run on GPU
      *Partitioning <RangePartitioning> will run on GPU
      *Expression <SortOrder> date#149 ASC NULLS FIRST will run on GPU
    *Exec <UnionExec> will run on GPU
      !Exec <ProjectExec> cannot run on GPU because not all expressions can
be replaced
        @Expression <AttributeReference> customerID#0 could run on GPU
        @Expression <Alias> Charge AS kind#126 could run on GPU
        @Expression <Literal> Charge could run on GPU
        @Expression <AttributeReference> value#129 could run on GPU
        @Expression <Alias> add_months(2022-11-15, cast(-(cast(_we0#142 as
bigint) + last_month#128L) as int)) AS date#149 could run on GPU
          ! <AddMonths> add_months(2022-11-15, cast(-
(cast(_we0#142 as bigint) + last_month#128L) as int)) cannot run
on GPU because GPU does not currently support the operator class
org.apache.spark.sql.catalyst.expressions.AddMonths
            @Expression <Literal> 2022-11-15 could run on GPU
            @Expression <Cast> cast(-(cast(_we0#142 as bigint) +
last_month#128L) as int) could run on GPU
            @Expression <UnaryMinus> -(cast(_we0#142 as bigint) +
last_month#128L) could run on GPU
            @Expression <Add> (cast(_we0#142 as bigint) +
last_month#128L) could run on GPU

```

```

@Expression <Cast> cast(_we0#142 as bigint) could run on
GPU

@Expression <AttributeReference> _we0#142 could run on
GPU

@Expression <AttributeReference> last_month#128L could run
on GPU

```

## Amazon EMR on EKS에서 Apache Spark용 Amazon Redshift 통합 사용

Amazon EMR 릴리스 6.9.0 이상에서 모든 릴리스 이미지에 [Apache Spark](#)와 Amazon Redshift 간 커넥터가 포함됩니다. 이 방법으로 Amazon EMR on EKS에서 Spark를 사용하여 Amazon Redshift에 저장된 데이터를 처리할 수 있습니다. 통합은 [spark-redshift 오픈 소스 커넥터](#)를 기반으로 합니다. Amazon EMR on EKS의 경우 [Apache Spark용 Amazon Redshift 통합](#)이 기본 통합으로 포함됩니다.

### 주제

- [Apache Spark용 Amazon Redshift 통합을 사용하여 Spark 애플리케이션 시작](#)
- [Apache Spark용 Amazon Redshift 통합으로 인증](#)
- [Amazon Redshift에서 읽고 쓰기](#)
- [Spark 커넥터 사용 시 고려 사항 및 제한 사항](#)

## Apache Spark용 Amazon Redshift 통합을 사용하여 Spark 애플리케이션 시작

통합을 사용하려면 필수 Spark Redshift 종속성을 Spark 작업과 함께 전달해야 합니다. Redshift 커넥터 관련 라이브러리를 포함하려면 `--jars`를 사용해야 합니다. `--jars` 옵션에서 지원하는 다른 파일 위치를 보려면 Apache Spark 설명서에서 [Advanced Dependency Management](#) 섹션을 참조하세요.

- `spark-redshift.jar`
- `spark-avro.jar`
- `RedshiftJDBC.jar`
- `minimal-json.jar`

Amazon EMR on EKS 릴리스 6.9.0 이상에서 Apache Spark용 Amazon Redshift 통합을 사용해 Spark 애플리케이션을 시작하려면 다음 예제 명령을 사용합니다. `--conf spark.jars` 옵션과 함께 나열된 경로는 JAR 파일의 기본 경로입니다.

```
aws emr-containers start-job-run \

--virtual-cluster-id cluster_id \
--execution-role-arn arn \
--release-label emr-6.9.0-latest\
--job-driver '{
    "sparkSubmitJobDriver": {
        "entryPoint": "s3://script_path",
        "sparkSubmitParameters":
            "--conf spark.kubernetes.file.upload.path=s3://upload_path
            --conf spark.jars=
                /usr/share/aws/redshift/jdbc/RedshiftJDBC.jar,
                /usr/share/aws/redshift/spark-redshift/lib/spark-redshift.jar,
                /usr/share/aws/redshift/spark-redshift/lib/spark-avro.jar,
                /usr/share/aws/redshift/spark-redshift/lib/minimal-json.jar"
    }
}'
```

## Apache Spark용 Amazon Redshift 통합으로 인증

다음 섹션에서는 Apache Spark와 통합하는 경우 Amazon Redshift를 사용한 인증 옵션을 보여줍니다. 이 섹션에서는 로그인 자격 증명을 검색하는 방법과 IAM 인증과 함께 JDBC 드라이버를 사용하는 방법에 대한 세부 정보를 설명합니다.

### AWS Secrets Manager 를 사용하여 자격 증명을 검색하고 Amazon Redshift에 연결

Secrets Manager에 보안 인증을 저장하여 Amazon Redshift에 안전하게 인증할 수 있습니다. Spark 작업에서 `GetSecretValue` API를 직접 호출하여 보안 인증을 가져올 수 있습니다.

```
from pyspark.sql import SQLContextimport boto3

sc = # existing SparkContext
sql_context = SQLContext(sc)

secretsmanager_client = boto3.client('secretsmanager',
    region_name=os.getenv('AWS_REGION'))
secret_manager_response = secretsmanager_client.get_secret_value(
```



```

    SecretId='string',
    VersionId='string',
    VersionStage='string'
)
username = # get username from secret_manager_response
password = # get password from secret_manager_response
url = "jdbc:redshift://redshifthost:5439/database?user=" + username + "&password="
      + password

# Access to Redshift cluster using Spark

```

## Amazon EMR on EKS 작업 실행 역할로 IAM 기반 인증 사용

Amazon EMR on EKS 릴리스 6.9.0부터 Amazon Redshift JDBC 드라이버 버전 2.1 이상이 환경에 함께 패키지로 포함됩니다. JDBC 드라이버 2.1 이상을 사용하면 JDBC URL을 지정하고 원시 사용자 이름과 암호는 포함하지 않을 수 있습니다. 대신 `jdbc:redshift:iam://` 스키마를 지정할 수 있습니다. 이를 통해 Amazon EMR on EKS 작업 실행 역할을 사용하여 보안 인증을 자동으로 가져오도록 JDBC 드라이버를 지시합니다.

자세한 내용은 Amazon Redshift 관리 안내서에서 [IAM 보안 인증을 사용하도록 JDBC 또는 ODBC 연결 구성](#)을 참조하세요.

다음 예제 URL은 `jdbc:redshift:iam://` 스키마를 사용합니다.

```

jdbc:redshift:iam://examplecluster.abc123xyz789.us-west-2.redshift.amazonaws.com:5439/
dev

```

제공된 조건을 충족하는 경우 작업 실행 역할에 다음 권한이 필요합니다.

| 권한                                              | 작업 실행 역할에 필요한 경우 조건                                               |
|-------------------------------------------------|-------------------------------------------------------------------|
| <code>redshift:GetClusterCredentials</code>     | JDBC 드라이버가 Amazon Redshift에서 보안 인증을 가져오는데 필요함                     |
| <code>redshift:DescribeCluster</code>           | Amazon Redshift 클러스터를 지정하고 엔드포인트 대신 JDBC URL에 AWS 리전을 지정하는 경우 필요함 |
| <code>redshift-serverless:GetCredentials</code> | JDBC 드라이버가 Amazon Redshift Serverless에서 보안 인증을 가져오는데 필요함          |

| 권한                               | 작업 실행 역할에 필요한 경우 조건                                                     |
|----------------------------------|-------------------------------------------------------------------------|
| redshift-serverless:GetWorkgroup | Amazon Redshift Serverless를 사용하고 있고 작업 그룹 이름 및 리전 측면에서 URL을 지정하는 경우 필요함 |

작업 실행 역할 정책에는 다음과 같은 권한이 있어야 합니다.

```
{
    "Effect": "Allow",
    "Action": [
        "redshift:GetClusterCredentials",
        "redshift:DescribeCluster",
        "redshift-serverless:GetCredentials",
        "redshift-serverless:GetWorkgroup"
    ],
    "Resource": [
        "arn:aws:redshift:AWS_REGION:ACCOUNT_ID:dbname:CLUSTER_NAME/DATABASE_NAME",
        "arn:aws:redshift:AWS_REGION:ACCOUNT_ID:dbuser:DATABASE_NAME/USER_NAME"
    ]
}
```

## JDBC 드라이버를 사용하여 Amazon Redshift에 인증

### JDBC URL에서 사용자 이름 및 암호 설정

Amazon Redshift 클러스터에 대한 Spark 작업을 인증하려면 JDBC URL에 Amazon Redshift 데이터베이스 이름과 암호를 지정할 수 있습니다.

#### Note

URL에 데이터베이스 보안 인증을 전달하면 URL에 액세스할 수 있는 모든 사용자도 보안 인증에 액세스할 수 있습니다. 이 방법은 안전한 옵션이 아니므로 일반적으로 권장되지 않습니다.

애플리케이션의 보안이 문제가 아닌 경우 다음 형식을 사용하여 JDBC URL에서 사용자 이름과 암호를 설정할 수 있습니다.

```
jdbc:redshift://redshifthost:5439/database?user=username&password=password
```

## Amazon Redshift에서 읽고 쓰기

다음 코드 예제는 데이터 소스 API와 SparkSQL을 통해 Amazon Redshift 데이터베이스에서 샘플 데이터를 읽고 쓰는 데 PySpark를 사용합니다.

### Data source API

PySpark를 사용하여 데이터 소스 API를 통해 Amazon Redshift 데이터베이스에서 샘플 데이터를 읽고 씁니다.

```
import boto3
from pyspark.sql import SQLContext

sc = # existing SparkContext
sql_context = SQLContext(sc)

url = "jdbc:redshift:iam://redshifthost:5439/database"
aws_iam_role_arn = "arn:aws:iam::accountID:role/roleName"

df = sql_context.read \
    .format("io.github.spark_redshift_community.spark.redshift") \
    .option("url", url) \
    .option("dbtable", "tableName") \
    .option("tempdir", "s3://path/for/temp/data") \
    .option("aws_iam_role", "aws_iam_role_arn") \
    .load()

df.write \
    .format("io.github.spark_redshift_community.spark.redshift") \
    .option("url", url) \
    .option("dbtable", "tableName_copy") \
    .option("tempdir", "s3://path/for/temp/data") \
    .option("aws_iam_role", "aws_iam_role_arn") \
    .mode("error") \
    .save()
```

### SparkSQL

PySpark를 사용하여 SparkSQL을 통해 Amazon Redshift 데이터베이스에서 샘플 데이터를 읽고 씁니다.

```
import boto3
import json
```

```

import sys
import os
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .enableHiveSupport() \
    .getOrCreate()

url = "jdbc:redshift:iam://redshifthost:5439/database"
aws_iam_role_arn = "arn:aws:iam::accountID:role/roleName"

bucket = "s3://path/for/temp/data"
tableName = "tableName" # Redshift table name

s = f"""CREATE TABLE IF NOT EXISTS {tableName} (country string, data string)
    USING io.github.spark_redshift_community.spark.redshift
    OPTIONS (dbtable '{tableName}', tempdir '{bucket}', url '{url}', aws_iam_role
    '{aws_iam_role_arn'} '); """

spark.sql(s)

columns = ["country" ,"data"]
data = [("test-country","test-data")]
df = spark.sparkContext.parallelize(data).toDF(columns)

# Insert data into table
df.write.insertInto(tableName, overwrite=False)
df = spark.sql(f"SELECT * FROM {tableName}")
df.show()

```

## Spark 커넥터 사용 시 고려 사항 및 제한 사항

Spark 커넥터는 자격 증명을 관리하고, 보안을 구성하고, 다른 AWS 서비스와 연결하는 다양한 방법을 지원합니다. 기능적이고 탄력적인 연결을 구성하려면 이 목록의 권장 사항을 숙지합니다.

- Amazon EMR의 Spark에서 Amazon Redshift로의 JDBC 연결을 위해 SSL을 활성화하는 것이 좋습니다.
- 모범 사례로 AWS Secrets Manager 에서 Amazon Redshift 클러스터의 보안 인증을 관리하는 것이 좋습니다. 예제 [AWS Secrets Manager 를 사용하여 Amazon Redshift에 연결하기 위한 자격 증명 검색을](#) 참조하세요.

- Amazon Redshift 인증 파라미터에 대해 `aws_iam_role` 파라미터를 사용하여 IAM 역할을 전달하는 것이 좋습니다.
- 현재 `tempformat` 파라미터는 Parquet 형식을 지원하지 않습니다.
- `tempdir` URI는 Amazon S3 위치를 가리킵니다. 이 임시 디렉터리는 자동으로 정리되지 않으므로, 추가 비용이 발생할 수 있습니다.
- Amazon Redshift에 대한 다음 권장 사항을 고려합니다.
  - Amazon Redshift 클러스터에 대한 퍼블릭 액세스를 차단하는 것이 좋습니다.
  - [Amazon Redshift 감사 로깅](#)을 켜는 것이 좋습니다.
  - [Amazon Redshift 저장 데이터 암호화](#)를 켜는 것이 좋습니다.
- Amazon S3에 대한 다음 권장 사항을 고려합니다.
  - [Amazon S3 버킷에 대한 퍼블릭 액세스를 차단](#)하는 것이 좋습니다.
  - [Amazon S3 서버 측 암호화](#)를 사용하여 사용할 S3 버킷을 암호화하는 것이 좋습니다.
  - [Amazon S3 수명 주기 정책](#)을 사용하여 S3 버킷에 대한 보존 규칙을 정의하는 것이 좋습니다.
  - Amazon EMR은 오픈 소스에서 이미지로 가져온 코드를 항상 확인합니다. 보안을 위해 Spark에서 Amazon S3로의 인증 방법으로 `tempdir` URI에서 AWS 액세스 키 인코딩을 지원하지 않습니다.

커넥터 사용 및 지원되는 파라미터에 대한 자세한 내용은 다음 리소스를 참조하세요.

- Amazon Redshift 관리 안내서의 [Apache Spark용 Amazon Redshift 통합](#)
- Github의 [spark-redshift community repository](#)

## Amazon EMR on EKS에서 Apache Spark의 사용자 지정 스케줄러로 Volcano 사용

Amazon EMR on EKS를 사용하면 Spark 운영자 또는 `spark-submit`을 사용하여 Kubernetes 사용자 지정 스케줄러로 Spark 작업을 실행할 수 있습니다. 이 자습서에서는 사용자 지정 대기열에서 Volcano 스케줄러를 통해 Spark 작업을 실행하는 방법을 다룹니다.

### 개요

[Volcano](#)는 대기열 예약, 공정 공유 예약, 리소스 예약과 같은 고급 기능을 통해 Spark 예약을 관리하는데 도움을 줄 수 있습니다. Volcano의 이점에 대한 자세한 내용은 Linux Foundation의 CNCF 블로그에서 [Why Spark chooses Volcano as built-in batch scheduler on Kubernetes](#)를 참조하세요.

## Volcano 설치 및 설정

1. 아키텍처 요구 사항에 따라 다음 kubectl 명령 중 하나를 선택하여 Volcano를 설치합니다.

```
# x86_64
kubectl apply -f https://raw.githubusercontent.com/volcano-sh/volcano/v1.5.1/
installer/volcano-development.yaml
# arm64:
kubectl apply -f https://raw.githubusercontent.com/volcano-sh/volcano/v1.5.1/
installer/volcano-development-arm64.yaml
```

2. 샘플 Volcano 대기열을 준비합니다. 대기열은 [PodGroups](#)의 모음입니다. 대기열은 FIFO를 따르며 리소스 분할의 기반이 됩니다.

```
cat << EOF > volcanoQ.yaml
apiVersion: scheduling.volcano.sh/v1beta1
kind: Queue
metadata:
  name: sparkqueue
spec:
  weight: 4
  reclaimable: false
  capability:
    cpu: 10
    memory: 20Gi
EOF

kubectl apply -f volcanoQ.yaml
```

3. Amazon S3에 샘플 PodGroup 매니페스트를 업로드합니다. PodGroup은 연결 관계가 강력한 포드 그룹입니다. 일반적으로 배치 예약에 PodGroup을 사용합니다. 이전 단계에서 정의한 대기열에 다음 샘플 PodGroup을 제출합니다.

```
cat << EOF > podGroup.yaml
apiVersion: scheduling.volcano.sh/v1beta1
kind: PodGroup
spec:
  # Set minMember to 1 to make a driver pod
  minMember: 1
  # Specify minResources to support resource reservation.
  # Consider the driver pod resource and executors pod resource.
  # The available resources should meet the minimum requirements of the Spark job
```

```
# to avoid a situation where drivers are scheduled, but they can't schedule
# sufficient executors to progress.
minResources:
  cpu: "1"
  memory: "1Gi"
# Specify the queue. This defines the resource queue that the job should be
submitted to.
queue: sparkqueue
EOF

aws s3 mv podGroup.yaml s3://bucket-name
```

Spark 운영자를 사용하여 Volcano 스케줄러에서 Spark 애플리케이션을 실행합니다.

1. 아직 실행하지 않았다면, 다음 단계를 완료하여 설정합니다.

- a. [Volcano 설치 및 설정](#)
- b. [Amazon EMR on EKS에 대한 Spark 운영자 구성](#)
- c. [Spark 운영자 설치](#)

helm install spark-operator-demo 명령을 실행할 때 다음 인수를 포함합니다.

```
--set batchScheduler.enable=true
--set webhook.enable=true
```

2. batchScheduler가 구성된 SparkApplication 정의 파일 spark-pi.yaml을 생성합니다.

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: spark-operator
spec:
  type: Scala
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
```

```

sparkVersion: "3.3.1"
batchScheduler: "volcano"    #Note: You must specify the batch scheduler name as
'volcano'
restartPolicy:
  type: Never
volumes:
  - name: "test-volume"
    hostPath:
      path: "/tmp"
      type: Directory
driver:
  cores: 1
  coreLimit: "1200m"
  memory: "512m"
  labels:
    version: 3.3.1
  serviceAccount: emr-containers-sa-spark
  volumeMounts:
    - name: "test-volume"
      mountPath: "/tmp"
executor:
  cores: 1
  instances: 1
  memory: "512m"
  labels:
    version: 3.3.1
  volumeMounts:
    - name: "test-volume"
      mountPath: "/tmp"

```

3. 다음 명령을 사용하여 Spark 애플리케이션을 제출합니다. 이렇게 하면 spark-pi라고 하는 SparkApplication 객체도 생성됩니다.

```
kubectl apply -f spark-pi.yaml
```

4. 다음 명령을 사용하여 SparkApplication 객체에 대한 이벤트를 확인합니다.

```
kubectl describe pods spark-pi-driver --namespace spark-operator
```

첫 번째 포드 이벤트는 Volcano에서 해당 포드를 예약했음을 보여줍니다.

| Type | Reason | Age | From | Message |
|------|--------|-----|------|---------|
|------|--------|-----|------|---------|



```

-----
Normal Scheduled 23s volcano
pi-driver to integration-worker2
-----
Successfully assigned default/spark-

```

## spark-submit을 사용하여 Volcano 스케줄러에서 Spark 애플리케이션 실행

1. 먼저, [Amazon EMR on EKS에서 spark-submit 설정](#) 섹션에 나온 단계를 완료합니다. Volcano 지원을 통해 spark-submit 배포를 구축해야 합니다. 자세한 내용은 Apache Spark 설명서에서 [Using Volcano as Customized Scheduler for Spark on Kubernetes](#)의 Build section을 참조하세요.
2. 다음과 같은 환경 변수의 값을 설정합니다.

```

export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon-EKS-cluster-endpoint

```

3. 다음 명령을 사용하여 Spark 애플리케이션을 제출합니다.

```

$SPARK_HOME/bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master $MASTER_URL \
  --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest \
  --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark \
  --deploy-mode cluster \
  --conf spark.kubernetes.namespace=spark-operator \
  --conf spark.kubernetes.scheduler.name=volcano \
  --conf spark.kubernetes.scheduler.volcano.podGroupTemplateFile=/path/to/podgroup-template.yaml \
  --conf
spark.kubernetes.driver.pod.featureSteps=org.apache.spark.deploy.k8s.features.VolcanoFeatu
\
  --conf
spark.kubernetes.executor.pod.featureSteps=org.apache.spark.deploy.k8s.features.VolcanoFea
\
  local:///usr/lib/spark/examples/jars/spark-examples.jar 20

```

4. 다음 명령을 사용하여 SparkApplication 객체에 대한 이벤트를 확인합니다.

```

kubectl describe pod spark-pi --namespace spark-operator

```

첫 번째 포드 이벤트는 Volcano에서 해당 포드를 예약했음을 보여줍니다.

| Type   | Reason    | Age  | From    | Message                                                              |
|--------|-----------|------|---------|----------------------------------------------------------------------|
| ----   | -----     | ---- | ----    | -----                                                                |
| Normal | Scheduled | 23s  | volcano | Successfully assigned default/spark-pi-driver to integration-worker2 |

## Amazon EMR on EKS에서 Apache Spark의 사용자 지정 스케줄러로 YuniKorn 사용

Amazon EMR on EKS를 사용하면 Spark 운영자 또는 spark-submit을 사용하여 Kubernetes 사용자 지정 스케줄러로 Spark 작업을 실행할 수 있습니다. 이 자습서에서는 사용자 지정 대기열 및 단체 예약 (gang scheduling)에서 YuniKorn 스케줄러를 통해 Spark 작업을 실행하는 방법을 다룹니다.

### 개요

[Apache YuniKorn](#)은 앱 인식 예약 기능으로 Spark 예약 관리를 지원하므로, 이를 통해 리소스 할당량 및 우선순위를 세밀하게 제어할 수 있습니다. 단체 예약 기능을 통해 YuniKorn은 앱에 대한 최소한의 리소스 요청을 충족할 수 있는 경우에만 앱을 예약합니다. 자세한 내용은 Apache YuniKorn 설명서 사이트에서 [What is gang scheduling](#)을 참조하세요.

### 클러스터를 생성하고 YuniKorn 설정

다음 단계에 따라 Amazon EKS 클러스터를 배포합니다. AWS 리전 (region) 및 가용 영역 (availabilityZones)을 변경할 수 있습니다.

1. Amazon EKS 클러스터를 정의합니다.

```
cat <<EOF >eks-cluster.yaml
---
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: emr-eks-cluster
  region: eu-west-1

vpc:
```

```

clusterEndpoints:
  publicAccess: true
  privateAccess: true

iam:
  withOIDC: true

nodeGroups:
  - name: spark-jobs
    labels: { app: spark }
    instanceType: m5.xlarge
    desiredCapacity: 2
    minSize: 2
    maxSize: 3
    availabilityZones: ["eu-west-1a"]
EOF

```

## 2. 클러스터를 생성합니다.

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

## 3. Spark 작업을 실행할 spark-job 네임스페이스를 생성합니다.

```
kubectl create namespace spark-job
```

## 4. 다음으로 Kubernetes 역할 및 역할 바인딩을 생성합니다. 이는 Spark 작업 실행에서 사용하는 서비스 계정에 필요합니다.

### a. Spark 작업에 대한 서비스 계정, 역할 및 역할 바인딩을 정의합니다.

```

cat <<EOF >emr-job-execution-rbac.yaml
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: spark-sa
  namespace: spark-job
automountServiceAccountToken: false
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: spark-role

```

```

namespace: spark-job
rules:
  - apiGroups: ["", "batch", "extensions"]
    resources: ["configmaps", "serviceaccounts", "events", "pods", "pods/
exec", "pods/log", "pods/
portforward", "secrets", "services", "persistentvolumeclaims"]
    verbs: ["create", "delete", "get", "list", "patch", "update", "watch"]
  ---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: spark-sa-rb
  namespace: spark-job
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: spark-role
subjects:
  - kind: ServiceAccount
    name: spark-sa
    namespace: spark-job
EOF

```

- b. 다음 명령을 사용하여 Kubernetes 역할 및 역할 바인딩 정의를 적용합니다.

```
kubectl apply -f emr-job-execution-rbac.yaml
```

## YuniKorn 설치 및 설정

1. 다음 kubectl 명령을 사용하여 YuniKorn 스케줄러를 배포하기 위해 네임스페이스(yunikorn)를 생성합니다.

```
kubectl create namespace unikorn
```

2. 스케줄러를 설치하려면 다음 Helm 명령을 실행합니다.

```
helm repo add unikorn https://apache.github.io/unikorn-release
```

```
helm repo update
```

```
helm install yunikorn yunikorn/yunikorn --namespace yunikorn
```

## Spark 운영자를 사용하여 YuniKorn 스케줄러에서 Spark 애플리케이션 실행

1. 아직 실행하지 않았다면, 다음 단계를 완료하여 설정합니다.

- a. [클러스터를 생성하고 YuniKorn 설정](#)
- b. [YuniKorn 설치 및 설정](#)
- c. [Amazon EMR on EKS에 대한 Spark 운영자 구성](#)
- d. [Spark 운영자 설치](#)

helm install spark-operator-demo 명령을 실행할 때 다음 인수를 포함합니다.

```
--set batchScheduler.enable=true
--set webhook.enable=true
```

2. SparkApplication 정의 파일 spark-pi.yaml을 생성합니다.

YuniKorn을 작업 스케줄러로 사용하려면 애플리케이션 정의에 특정 주석과 레이블을 추가해야 합니다. 주석과 레이블은 작업 대기열 및 사용하려는 예약 전략을 지정합니다.

다음 예제에서는 schedulingPolicyParameters 주석을 사용하여 애플리케이션에 대한 단체 예약을 설정합니다. 그런 다음, 이 예제에서는 작업 그룹(즉 작업 '단체')을 생성하여 작업 실행을 위해 포드를 예약하기 전에 사용할 수 있어야 하는 최소 용량을 지정합니다. 마지막으로, [클러스터를 생성하고 YuniKorn 설정](#) 섹션에 정의된 대로 작업 그룹 정의에서 "app": "spark" 레이블의 노드 그룹을 사용하도록 지정합니다.

```
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: spark-job
spec:
  type: Scala
  mode: cluster
  image: "895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
```

```
mainApplicationFile: "local:///usr/lib/spark/examples/jars/spark-examples.jar"
sparkVersion: "3.3.1"
restartPolicy:
  type: Never
volumes:
  - name: "test-volume"
    hostPath:
      path: "/tmp"
      type: Directory
driver:
  cores: 1
  coreLimit: "1200m"
  memory: "512m"
  labels:
    version: 3.3.1
  annotations:
    yunikorn.apache.org/schedulingPolicyParameters: "placeholderTimeoutSeconds=30
gangSchedulingStyle=Hard"
    yunikorn.apache.org/task-group-name: "spark-driver"
    yunikorn.apache.org/task-groups: |-
      [{
        "name": "spark-driver",
        "minMember": 1,
        "minResource": {
          "cpu": "1200m",
          "memory": "1Gi"
        },
        "nodeSelector": {
          "app": "spark"
        }
      },
      {
        "name": "spark-executor",
        "minMember": 1,
        "minResource": {
          "cpu": "1200m",
          "memory": "1Gi"
        },
        "nodeSelector": {
          "app": "spark"
        }
      }
    ]
  serviceAccount: spark-sa
volumeMounts:
```

```

    - name: "test-volume"
      mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.3.1
    annotations:
      yunikorn.apache.org/task-group-name: "spark-executor"
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"

```

3. 다음 명령을 사용하여 Spark 애플리케이션을 제출합니다. 이렇게 하면 spark-pi라고 하는 SparkApplication 객체도 생성됩니다.

```
kubectl apply -f spark-pi.yaml
```

4. 다음 명령을 사용하여 SparkApplication 객체에 대한 이벤트를 확인합니다.

```
kubectl describe sparkapplication spark-pi --namespace spark-job
```

첫 번째 포드 이벤트는 YuniKorn에서 해당 포드를 예약했음을 보여줍니다.

| Type   | Reason            | Age   | From     | Message                                                                                                       |
|--------|-------------------|-------|----------|---------------------------------------------------------------------------------------------------------------|
| Normal | Scheduling        | 3m12s | yunikorn | spark-operator/org-apache-spark-examples-sparkpi-2a777a88b98b8a95-driver is queued and waiting for allocation |
| Normal | GangScheduling    | 3m12s | yunikorn | Pod belongs to the taskGroup spark-driver, it will be scheduled as a gang member                              |
| Normal | Scheduled         | 3m10s | yunikorn | Successfully assigned spark                                                                                   |
| Normal | PodBindSuccessful | 3m10s | yunikorn | Pod spark-operator/                                                                                           |
| Normal | TaskCompleted     | 2m3s  | yunikorn | Task spark-operator/                                                                                          |
| Normal | Pulling           | 3m10s | kubelet  | Pulling                                                                                                       |

## spark-submit을 사용하여 YuniKorn 스케줄러에서 Spark 애플리케이션 실행

1. 먼저, [Amazon EMR on EKS에서 spark-submit 설정](#) 섹션에 나온 단계를 완료합니다.

## 2. 다음과 같은 환경 변수의 값을 설정합니다.

```
export SPARK_HOME=spark-home
export MASTER_URL=k8s://Amazon-EKS-cluster-endpoint
```

## 3. 다음 명령을 사용하여 Spark 애플리케이션을 제출합니다.

다음 예제에서는 `schedulingPolicyParameters` 주석을 사용하여 애플리케이션에 대한 단체 예약을 설정합니다. 그런 다음, 이 예제에서는 작업 그룹(즉 작업 '단체')을 생성하여 작업 실행을 위해 포드를 예약하기 전에 사용할 수 있어야 하는 최소 용량을 지정합니다. 마지막으로, [클러스터를 생성하고 YuniKorn 설정](#) 섹션에 정의된 대로 작업 그룹 정의에서 "app": "spark" 레이블의 노드 그룹을 사용하도록 지정합니다.

```
$SPARK_HOME/bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master $MASTER_URL \
  --conf spark.kubernetes.container.image=895885662937.dkr.ecr.us-west-2.amazonaws.com/spark/emr-6.10.0:latest \
  --conf spark.kubernetes.authenticate.driver.serviceAccountName=spark-sa \
  --deploy-mode cluster \
  --conf spark.kubernetes.namespace=spark-job \
  --conf spark.kubernetes.scheduler.name=yunikorn \
  --conf spark.kubernetes.driver.annotation.yunikorn.apache.org/
schedulingPolicyParameters="placeholderTimeoutSeconds=30 gangSchedulingStyle=Hard"
\
  --conf spark.kubernetes.driver.annotation.yunikorn.apache.org/task-group-
name="spark-driver" \
  --conf spark.kubernetes.executor.annotation.yunikorn.apache.org/task-group-
name="spark-executor" \
  --conf spark.kubernetes.driver.annotation.yunikorn.apache.org/task-groups='[{
    "name": "spark-driver",
    "minMember": 1,
    "minResource": {
      "cpu": "1200m",
      "memory": "1Gi"
    },
    "nodeSelector": {
      "app": "spark"
    }
  },
  {
    "name": "spark-executor",
```



```

        "minMember": 1,
        "minResource": {
            "cpu": "1200m",
            "memory": "1Gi"
        },
        "nodeSelector": {
            "app": "spark"
        }
    }]' \
local:///usr/lib/spark/examples/jars/spark-examples.jar 20

```

4. 다음 명령을 사용하여 SparkApplication 객체에 대한 이벤트를 확인합니다.

```
kubectl describe pod spark-driver-pod --namespace spark-job
```

첫 번째 포드 이벤트는 YuniKorn에서 해당 포드를 예약했음을 보여줍니다.

| Type   | Reason            | Age   | From     | Message                                                                                                       |
|--------|-------------------|-------|----------|---------------------------------------------------------------------------------------------------------------|
| ----   | -----             | ----  | ----     | -----                                                                                                         |
| Normal | Scheduling        | 3m12s | yunikorn | spark-operator/org-apache-spark-examples-sparkpi-2a777a88b98b8a95-driver is queued and waiting for allocation |
| Normal | GangScheduling    | 3m12s | yunikorn | Pod belongs to the taskGroup spark-driver, it will be scheduled as a gang member                              |
| Normal | Scheduled         | 3m10s | yunikorn | Successfully assigned spark                                                                                   |
| Normal | PodBindSuccessful | 3m10s | yunikorn | Pod spark-operator/                                                                                           |
| Normal | TaskCompleted     | 2m3s  | yunikorn | Task spark-operator/                                                                                          |
| Normal | Pulling           | 3m10s | kubelet  | Pulling                                                                                                       |

# Amazon EMR on EKS의 보안

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. [공동 책임 모델](#)은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드 보안 - AWS 는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. AWS 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사자는 [AWS 규정 준수 프로그램](#) 일환으로 보안의 효과를 정기적으로 테스트하고 확인합니다. Amazon EMR에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램 [AWS 제공 범위 내 서비스규정 준수 프로그램](#).
- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 귀하는 귀사의 데이터 민감도, 귀사의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 문서는 Amazon EMR on EKS를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목적에 맞게 Amazon EMR on EKS를 구성하는 방법을 보여줍니다. 또한 Amazon EMR on EKS 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법을 알아봅니다.

## 주제

- [Amazon EMR on EKS 보안 모범 사례](#)
- [데이터 보호](#)
- [자격 증명 및 액세스 관리](#)
- [세분화된 액세스 제어를 위해 AWS Lake Formation과 함께 Amazon EMR on EKS 사용](#)
- [로그 및 모니터링](#)
- [Amazon EMR on EKS로 Amazon S3 Access Grants 사용](#)
- [Amazon EMR on EKS에 대한 규정 준수 확인](#)
- [Amazon EMR on EKS의 복원성](#)
- [Amazon EMR on EKS의 인프라 보안](#)
- [구성 및 취약성 분석](#)
- [인터페이스 VPC 엔드포인트를 사용하여 Amazon EMR on EKS에 연결](#)

- [Amazon EMR on EKS에 대한 크로스 계정 액세스 설정](#)

## Amazon EMR on EKS 보안 모범 사례

Amazon EMR on EKS는 자체 보안 정책을 개발하고 구현할 때 고려해야 할 여러 보안 기능을 제공합니다. 다음 모범 사례는 일반적인 지침이며 완벽한 보안 솔루션을 나타내지는 않습니다. 이러한 모범 사례는 사용자의 환경에 적절하지 않거나 충분하지 않을 수 있으므로 규정이 아닌 참고용으로만 사용하세요.

### Note

보안 모범 사례는 [Amazon EMR on EKS 보안 모범 사례](#) 섹션을 참조하세요.

## 최소 권한의 원칙 적용

Amazon EMR on EKS는 실행 역할과 같은 IAM 역할을 사용하는 애플리케이션에 대한 세분화된 액세스 정책을 제공합니다. 이러한 실행 역할은 IAM 역할의 신뢰 정책을 통해 Kubernetes 서비스 계정에 매핑됩니다. Amazon EMR on EKS는 등록된 Amazon EKS 네임스페이스에서 사용자가 제공한 애플리케이션 코드를 실행하는 포드를 생성합니다. 애플리케이션 코드를 실행하는 작업 포드는 다른 AWS 서비스에 연결할 때 실행 역할을 수임합니다. 애플리케이션 조치와 로그 대상에 대한 액세스 등 작업에 필요한 최소한의 권한만 실행 역할에 부여하는 것이 좋습니다. 또한 애플리케이션 코드가 변경될 때마다 정기적으로 권한을 확인하기 위해 작업을 감사하는 것이 좋습니다.

## 엔드포인트에 대한 액세스 제어 목록

관리형 엔드포인트는 VPC에서 하나 이상의 프라이빗 서브넷을 사용하도록 구성된 EKS 클러스터에 대해서만 생성할 수 있습니다. 이 구성은 관리형 엔드포인트에서 생성한 로드 밸런서에 대한 액세스를 제한하므로 VPC에서만 액세스할 수 있습니다. 보안을 더욱 강화하기 위해 수신 트래픽을 선택한 IP 주소 세트로 제한할 수 있도록 이러한 로드 밸런서로 보안 그룹을 구성하는 것이 좋습니다.

## 사용자 지정 이미지에 대한 최신 보안 업데이트 받기

Amazon EMR on EKS에서 사용자 지정 이미지를 사용하려는 경우 이미지에 모든 바이너리와 라이브러리를 설치할 수 있습니다. 이미지에 추가하는 바이너리의 보안 패치는 사용자의 책임입니다. Amazon EMR on EKS 이미지는 최신 보안 패치를 통해 정기적으로 패치됩니다. 최신 이미지를 가져오려면 Amazon EMR 릴리스의 새 기본 이미지 버전이 있을 때마다 사용자 지정 이미지를 다시 빌드해야

합니다. 자세한 내용은 [Amazon EMR on EKS 릴리스](#) 및 [기본 이미지 URI 선택에 관한 세부 정보](#) 섹션을 참조하세요.

## 포드 보안 인증 액세스 제한

Kubernetes는 포드에 보안 인증을 할당하는 여러 방법을 지원합니다. 여러 보안 인증 제공업체를 프로비저닝하면 보안 모델이 더 복잡해질 수 있습니다. Amazon EMR on EKS는 등록된 EKS 네임스페이스에서 표준 보안 인증 제공업체로 [서비스 계정에 대한 IAM 역할\(IRSA\)](#)의 사용을 채택했습니다. [kube2iam](#), [kiam](#) 및 클러스터에서 실행되는 인스턴스의 EC2 인스턴스 프로파일 사용 등 다른 방법은 지원되지 않습니다.

## 신뢰할 수 없는 애플리케이션 코드 격리

Amazon EMR on EKS는 시스템 사용자가 제출한 애플리케이션 코드의 무결성을 검사하지 않습니다. 임의 코드를 실행하는 신뢰할 수 없는 테넌트가 작업을 제출하는 데 사용할 수 있는 다중 실행 역할로 구성된 다중 테넌트 가상 클러스터를 실행하는 경우 악성 애플리케이션이 권한을 에스컬레이션할 위험이 있습니다. 이 경우 비슷한 권한을 가진 실행 역할을 다른 가상 클러스터로 격리하는 방법을 고려합니다.

## 역할 기반 액세스 제어(RBAC) 권한

관리자는 Amazon EMR on EKS 관리형 네임스페이스에 대한 역할 기반 액세스 제어(RBAC)를 엄격하게 제어해야 합니다. 적어도 Amazon EMR on EKS 관리형 네임스페이스에서 작업 제출자에게 다음 권한을 부여해서는 안 됩니다.

- configmap을 수정할 Kubernetes RBAC 권한 - Amazon EMR on EKS는 Kubernetes configmap을 사용하여 관리형 서비스 계정 이름을 포함하는 관리형 포드 템플릿을 생성하기 때문입니다. 이 속성은 변경해서는 안 됩니다.
- Amazon EMR on EKS 포드로 실행할 Kubernetes RBAC 권한 - 관리형 SA 이름을 포함한 관리형 포드 템플릿에 액세스 권한을 부여하지 않도록 방지합니다. 이 속성은 변경해서는 안 됩니다. 또한 이 권한은 포드에 마운트된 JWT 토큰에 대한 액세스 권한을 부여하고, 이 권한을 통해 실행 역할 보안 인증을 검색하는 데 사용할 수 있습니다.
- 포드를 생성하기 위한 Kubernetes RBAC 권한 - 사용자가 사용자보다 더 많은 AWS 권한을 가진 IAM 역할에 매핑될 수 있는 Kubernetes ServiceAccount를 사용하여 포드를 생성하지 못하도록 합니다.
- 변형 웹후크를 배포할 Kubernetes RBAC 권한 - 사용자가 변형 웹후크를 사용하여 Amazon EMR on EKS에서 생성한 포드의 Kubernetes ServiceAccount 이름을 변경하지 못하도록 방지합니다.

- Kubernetes 보안 암호를 읽을 수 있는 Kubernetes RBAC 권한 - 사용자가 이러한 보안 암호에 저장된 기밀 데이터를 읽지 못하도록 방지합니다.

## 노드 그룹 IAM 역할 또는 인스턴스 프로파일 보안 인증에 대한 액세스를 제한합니다.

- 노드 그룹의 IAM 역할(들)에 최소 AWS 권한을 할당하는 것이 좋습니다. 이렇게 하면 EKS 워커 노드의 인스턴스 프로파일 보안 인증을 사용하여 실행할 수 있는 코드에 의한 권한 상승을 방지하는 데 도움이 됩니다.
- Amazon EMR on EKS 관리형 네임스페이스에서 실행되는 모든 포드의 인스턴스 프로파일 보안 인증에 대한 액세스를 완전히 차단하려면 EKS 노드에서 iptables 명령을 실행하는 것이 좋습니다. 자세한 내용은 [Amazon EC2 인스턴스 프로파일 보안 인증에 대한 액세스 제한](#)을 참조하세요. 서비스 계정 IAM 역할의 범위를 적절하게 지정하여 포드에 필요한 모든 권한을 부여하는 것이 중요합니다. 예를 들어 노드 IAM 역할에는 Amazon ECR에서 컨테이너 이미지를 가져올 수 있는 권한이 할당됩니다. 포드에 이러한 권한이 할당되지 않은 경우 포드는 Amazon ECR에서 컨테이너 이미지를 가져올 수 없습니다. VPC CNI 플러그인도 업데이트해야 합니다. 자세한 내용은 [안내: 서비스 계정에 IAM 역할을 사용하도록 Amazon VPC CNI 플러그인 업데이트](#)를 참조하세요.

## 데이터 보호

AWS [공동 책임 모델](#)은 Amazon EMR on EKS의 데이터 보호에 적용됩니다. 이 모델에 설명된 대로 AWS 는 모든 클라우드를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS . 이 인프라에서 호스팅되는 콘텐츠에 대한 통제를 유지하는 것은 사용자의 책임입니다. 이 콘텐츠에는 사용하는 AWS 서비스에 대한 보안 구성 및 관리 작업이 포함됩니다. 데이터 프라이버시에 대한 자세한 내용은 [데이터 프라이버시 FAQ](#)를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그 [의 AWS 공동 책임 모델 및 GDPR](#) 블로그 게시물을 참조하세요.

데이터 보호를 위해 AWS 계정 자격 증명을 보호하고 AWS Identity and Access Management(IAM)를 사용하여 개별 계정을 설정하는 것이 좋습니다. 이러한 방식에서는 각 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용합니다.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2 이상을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail.

- AWS 암호화 솔루션을 AWS 서비스 내의 모든 기본 보안 제어와 함께 사용합니다.
- Amazon S3에 저장된 개인 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용합니다.
- Amazon EMR on EKS 암호화 옵션을 사용해 유틸리티 및 전송 중 데이터를 암호화합니다.
- 명령줄 인터페이스 또는 API를 AWS 통해 액세스할 때 FIPS 140-2 검증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 [Federal Information Processing Standard\(FIPS\) 140-2](#) 섹션을 참조하세요.

명칭 필드와 같은 자유 형식 필드에 고객 계정 번호와 같은 중요 식별 정보를 절대 입력하지 마세요. 여기에는 Amazon EMR on EKS 또는 기타 AWS 서비스에서 콘솔 AWS CLI, API 또는 AWS SDKs를 사용하여 작업하는 경우가 포함됩니다. Amazon EMR on EKS 또는 기타 서비스에 입력하는 모든 데이터를 진단 로그에 포함할 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 보안 인증 정보를 URL에 포함시키지 마세요.

## 저장 시 암호화

데이터 암호화는 권한 없는 사용자가 클러스터 및 관련 데이터 스토리지 시스템에서 데이터를 읽는 것을 방지하는 데 도움이 됩니다. 여기에는 영구 미디어에 저장된 데이터인 유틸리티 데이터와 네트워크를 이동하는 동안 가로챌 수 있는 데이터인 전송 중 데이터가 포함됩니다.

데이터 암호화에는 키와 인증서가 필요합니다. 에서 관리하는 키, Amazon S3에서 관리하는 AWS Key Management Service 키, 제공하는 사용자 지정 공급자의 키 및 인증서를 비롯한 여러 옵션 중에서 선택할 수 있습니다. 를 키 공급자 AWS KMS 로 사용하는 경우 암호화 키의 저장 및 사용에 요금이 부과됩니다. 자세한 내용은 [AWS KMS 요금](#)을 참조하세요.

암호화 옵션을 지정하기 전에 사용할 키 및 인증서 관리 시스템을 결정합니다. 그런 다음 암호화 설정의 일부로 지정하는 사용자 지정 제공업체에 대한 키와 인증서를 생성합니다.

## Amazon S3에서 EMRFS 데이터에 대한 유틸리티 데이터 암호화

Amazon S3 암호화는 Amazon S3에서 읽고 쓰는 EMR 파일 시스템(EMRFS) 객체와 함께 작동합니다. 유틸리티 데이터 암호화를 활성화할 때 Amazon S3 서버 측 암호화(SSE) 또는 클라이언트 측 암호화(CSE)를 기본 암호화 모드로 지정합니다. 선택적으로 버킷별 암호화 재정의의 사용하여 개별 버킷에 서로 다른 암호화 방법을 지정할 수 있습니다. Amazon S3 암호화가 활성화되어 있는지 여부와 상관없이 전송 계층 보안(TLS)은 EMR 클러스터 노드 및 Amazon S3 간에 전송 중인 EMRFS 객체를 암호화합니다. Amazon S3 암호화에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서에서 [암호화를 사용하여 데이터 보호](#)를 참조하세요.

**Note**

를 사용하면 암호화 키의 저장 및 사용에 요금이 AWS KMS부과됩니다. 자세한 내용은 [AWS KMS 요금](#)을 참조하세요.

## Amazon S3 서버 측 암호화

Amazon S3 서버 측 암호화를 설정하면 Amazon S3에서 데이터를 디스크에 쓸 때 객체 수준에서 데이터를 암호화하고 데이터에 액세스할 때 데이터의 암호를 해독합니다. 자세한 내용은 Amazon Simple Storage Service 개발자 안내서에서 [서버 측 암호화를 사용하여 데이터 보호](#)를 참조하세요.

Amazon EMR on EKS에서 SSE를 지정할 때 다음 두 가지 키 관리 시스템 중에서 선택할 수 있습니다.

- SSE-S3 - Amazon S3에서는 자동으로 키를 관리합니다.
- SSE-KMS - AWS KMS key 를 사용하여 Amazon EMR on EKS에 적합한 정책을 설정합니다.

고객 제공 키를 사용하는 SSE(SSE-C)는 Amazon EMR on EKS에서 사용할 수 없습니다.

## Amazon S3 클라이언트 측 암호화

Amazon S3 클라이언트 측 암호화를 사용하면 클러스터의 EMRFS 클라이언트에서 Amazon S3 암호화 및 암호 해독이 수행됩니다. 객체는 Amazon S3에 업로드되기 전에 암호화되고 다운로드된 후 암호 해독됩니다. 지정하는 공급자는 클라이언트가 사용하는 암호화 키를 제공합니다. 클라이언트는 AWS KMS 에서 제공하는 키(CSE-KMS) 또는 클라이언트 측 루트 키(CSE-C)를 제공하는 사용자 지정 Java 클래스를 사용할 수 있습니다. 암호화 세부 사항은 지정된 공급자 및 암호 해독되거나 암호화되는 객체의 메타데이터에 따라 CSE-KMS 및 CSE-C 간에 약간 다릅니다. 이러한 차이에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서에서 [클라이언트 측 암호화를 사용하여 데이터 보호](#)를 참조하세요.

**Note**

Amazon S3 CSE는 Amazon S3와 교환하는 EMRFS 데이터만 암호화하며, 클러스터 인스턴스 볼륨에 있는 모든 데이터를 암호화하지는 않습니다. 뿐만 아니라, Hue에서 EMRFS가 사용되지 않으므로 Hue S3 파일 검색기를 사용하여 Amazon S3에 작성된 객체는 암호화되지 않습니다.

## 로컬 디스크 암호화

Apache Spark는 로컬 디스크에 기록된 임시 데이터 암호화를 지원합니다. 여기에는 셔플 파일, 셔플 유출, 캐싱 및 브로드캐스트 변수 모두에 대해 디스크에 저장된 데이터 블록이 포함됩니다. 또는 `saveAsHadoopFile` 또는 `saveAsTable` 같은 API를 사용하여 애플리케이션에서 생성된 출력 데이터를 암호화하는 방법은 다루지 않습니다. 또한 사용자가 명시적으로 생성한 임시 파일은 여기에 포함되지 않을 수 있습니다. 자세한 내용은 Spark 설명서에서 [Local Storage Encryption](#)을 참조하세요. Spark는 데이터가 메모리에 맞지 않을 때 실행기 프로세스가 로컬 디스크에 쓰는 중간 데이터와 같이 로컬 디스크에서 암호화된 데이터를 지원하지 않습니다. 디스크에서 지속되는 데이터의 범위는 작업 런타임으로 제한되며, 데이터를 암호화하는 데 사용되는 키는 Spark에서 작업이 실행될 때마다 동적으로 생성됩니다. Spark 작업이 종료되면 다른 프로세스에서 데이터를 해독할 수 없습니다.

드라이버 및 실행기 포드의 경우 마운트된 볼륨에서 지속되는 저장 데이터를 암호화합니다.

Kubernetes에는 [EBS](#), [EFS](#) 및 [FSx for Lustre](#)의 세 가지 AWS 기본 스토리지 옵션을 사용할 수 있습니다. 세 가지 모두 서비스 관리형 키 또는 AWS KMS key를 사용하여 저장 데이터 암호화를 제공합니다. 자세한 내용은 [EKS Best Practices Guide](#)를 참조하세요. 이 접근 방식을 사용하면 마운트된 볼륨에 유지되는 모든 데이터가 암호화됩니다.

## 키 관리

KMS 키를 자동으로 로테이션하도록 KMS를 구성할 수 있습니다. 이렇게 하면 1년에 한 번 키가 로테이션되고 이전 키는 무기한 저장되므로 데이터를 계속 해독할 수 있습니다. 자세한 내용은 [교체를 참조하세요 AWS KMS keys](#).

## 전송 중 암호화

전송 중 데이터 암호화와 함께 여러 가지 암호화 메커니즘이 활성화됩니다. 이러한 메커니즘은 오픈 소스 기능이고, 애플리케이션에 특정하며, Amazon EMR on EKS 릴리스에 따라 다를 수 있습니다. Amazon EMR on EKS에서 다음과 같은 애플리케이션별 암호화 기능을 활성화할 수 있습니다.

- Spark
  - Amazon EMR 버전 5.9.0 이상에서는 블록 전송 서비스, 외부 셔플 서비스 등 Spark 구성 요소 간에 이루어지는 내부 RPC 통신을 AES-256 암호로 암호화합니다. 그 이전 릴리스에서는 SASL과 DIGEST-MD5 암호를 사용하여 내부 RPC 통신을 암호화합니다.
  - Spark 기록 서버 및 HTTPS 지원 파일 서버 등 사용자 인터페이스와의 HTTP 프로토콜 통신은 Spark의 SSL 구성으로 암호화됩니다. 자세한 내용은 Spark 설명서의 [SSL 구성](#)을 읽어 보세요.

자세한 내용은 [Spark security settings](#)를 참조하세요.



- Amazon S3 버킷 IAM 정책에 [aws:SecureTransport](#) 조건을 사용하여 HTTPS(TLS)를 통해 암호화된 연결만 허용해야 합니다.
- JDBC 또는 ODBC 클라이언트로 스트리밍하는 쿼리 결과는 TLS를 사용하여 암호화됩니다.

## 자격 증명 및 액세스 관리

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 되는 AWS 서비스입니다. IAM 관리자는 Amazon EMR on EKS 리소스를 사용하도록 인증(로그인) 및 권한 부여된 사용자를 제어합니다. IAM은 추가 비용 없이 사용할 수 있는 AWS 서비스입니다.

### 주제

- [대상](#)
- [ID를 통한 인증](#)
- [정책을 사용하여 액세스 관리](#)
- [Amazon EMR on EKS에서 IAM을 사용하는 방법](#)
- [Amazon EMR on EKS에 대한 서비스 연결 역할 사용](#)
- [Amazon EMR on EKS 관리형 정책](#)
- [Amazon EMR on EKS에서 작업 실행 역할 사용](#)
- [Amazon EMR on EKS에 대한 자격 증명 기반 정책 예제](#)
- [태그 기반 액세스 제어를 위한 정책](#)
- [Amazon EMR on EKS 자격 증명 및 액세스 문제 해결](#)

## 대상

AWS Identity and Access Management (IAM)를 사용하는 방법은 역할에 따라 다릅니다.

- 서비스 사용자 - 기능에 액세스할 수 없는 경우 관리자에게 권한 요청([참조 Amazon EMR on EKS 자격 증명 및 액세스 문제 해결](#))
- 서비스 관리자 - 사용자 액세스 결정 및 권한 요청 제출([Amazon EMR on EKS에서 IAM을 사용하는 방법 참조](#))
- IAM 관리자 - 액세스를 관리하기 위한 정책 작성([Amazon EMR on EKS에 대한 자격 증명 기반 정책 예제 참조](#))

## ID를 통한 인증

인증은 자격 증명 자격 증명을 AWS 사용하여 로그인하는 방법입니다. AWS 계정 루트 사용자, IAM 사용자 또는 IAM 역할을 수임하여 인증해야 합니다.

AWS IAM Identity Center (IAM Identity Center), Single Sign-On 인증 또는 Google/Facebook 자격 증명과 같은 자격 증명 소스의 자격 증명을 사용하여 페더레이션 자격 증명으로 로그인할 수 있습니다. 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 [AWS 계정에 로그인하는 방법](#) 섹션을 참조하세요.

프로그래밍 방식 액세스를 위해서는 요청에 암호화 방식으로 서명할 수 있는 SDK 및 CLI를 AWS 제공합니다. 자세한 내용은 IAM 사용 설명서의 [API 요청용 AWS Signature Version 4](#) 섹션을 참조하세요.

### AWS 계정 루트 사용자

를 생성할 때 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 AWS 계정 theroot 사용자라는 하나의 로그인 자격 증명으로 AWS 계정시작합니다. 일상적인 태스크에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명이 필요한 작업은 IAM 사용 설명서의 [루트 사용자 자격 증명이 필요한 작업](#) 섹션을 참조하세요.

### 페더레이션 ID

가장 좋은 방법은 인간 사용자에게 자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 AWS 서비스 사용하여 액세스하도록 요구하는 것입니다.

페더레이션 자격 증명은 엔터프라이즈 디렉터리, 웹 자격 증명 공급자 또는 자격 증명 소스의 자격 증명을 AWS 서비스 사용하여 Directory Service 에 액세스하는 사용자입니다. 페더레이션 ID는 임시 자격 증명을 제공하는 역할을 수임합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center를 추천합니다. 자세한 정보는 AWS IAM Identity Center 사용 설명서의 [What is IAM Identity Center?](#)를 참조하세요.

### IAM 사용자 및 그룹

[IAM 사용자](#)는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가진 ID입니다. 장기 자격 증명에 있는 IAM 사용자 대신 임시 자격 증명을 사용하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 [자격 증명 공급자와의 페더레이션을 사용하여 임시 자격 증명을 AWS 사용하여 액세스하도록 인간 사용자에게 요구하기](#)를 참조하세요.

[IAM 그룹](#)은 IAM 사용자 모음을 지정하고 대규모 사용자 집합에 대한 관리 권한을 더 쉽게 만듭니다. 자세한 내용은 IAM 사용 설명서의 [IAM 사용자 사용 사례](#) 섹션을 참조하세요.

## IAM 역할

[IAM 역할](#)은 임시 자격 증명을 제공하는 특정 권한이 있는 자격 증명입니다. [사용자에서 IAM 역할\(콘솔\)로 전환하거나 또는 API 작업을 호출하여 역할을](#) 수임할 수 있습니다. AWS CLI AWS 자세한 내용은 IAM 사용 설명서의 [역할 수임 방법](#)을 참조하세요.

IAM 역할은 페더레이션 사용자 액세스, 임시 IAM 사용자 권한, 교차 계정 액세스, 교차 서비스 액세스 및 Amazon EC2에서 실행되는 애플리케이션에 유용합니다. 자세한 내용은 IAM 사용 설명서의 [교차 계정 리소스 액세스](#)를 참조하세요.

## 정책을 사용하여 액세스 관리

정책을 AWS 생성하고 자격 증명 또는 리소스에 연결하여 AWS 에서 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결될 때 권한을 정의합니다.는 보안 주체가 요청할 때 이러한 정책을 AWS 평가합니다. 대부분의 정책은 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서에 대한 자세한 내용은 IAM 사용 설명서의 [JSON 정책 개요](#) 섹션을 참조하세요.

정책을 사용하여 관리자는 어떤 보안 주체가 어떤 리소스에 대해 어떤 조건에서 작업을 수행할 수 있는지 정의하여 누가 무엇을 액세스할 수 있는지 지정합니다.

기본적으로 사용자 및 역할에는 어떠한 권한도 없습니다. IAM 관리자는 IAM 정책을 생성하고 사용자가 수임할 수 있는 역할에 추가합니다. IAM 정책은 작업을 수행하기 위해 사용하는 방법과 관계없이 작업에 대한 권한을 정의합니다.

## ID 기반 정책

ID 기반 정책은 ID(사용자, 사용자 그룹 또는 역할)에 연결하는 JSON 권한 정책 문서입니다. 이러한 정책은 자격 증명이 수행할 수 있는 작업, 대상 리소스 및 이에 관한 조건을 제어합니다. ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 [고객 관리형 정책으로 사용자 지정 IAM 권한 정의](#)를 참조하세요.

ID 기반 정책은 인라인 정책(단일 ID에 직접 포함) 또는 관리형 정책(여러 ID에 연결된 독립 실행형 정책)일 수 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 [관리형 정책 및 인라인 정책 중에서 선택](#) 섹션을 참조하세요.

## 리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 예를 들어 IAM 역할 신뢰 정책 및 Amazon S3 버킷 정책이 있습니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한

정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 리소스 기반 정책에서 [보안 주체를 지정](#)해야 합니다.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

## 기타 정책 유형

AWS 는 보다 일반적인 정책 유형에서 부여한 최대 권한을 설정할 수 있는 추가 정책 유형을 지원합니다.

- 권한 경계 - ID 기반 정책에서 IAM 엔터티에 부여할 수 있는 최대 권한을 설정합니다. 자세한 정보는 IAM 사용 설명서의 [IAM 엔터티의 권한 범위](#)를 참조하세요.
- 서비스 제어 정책(SCP) - AWS Organizations내 조직 또는 조직 단위에 대한 최대 권한을 지정합니다. 자세한 내용은 AWS Organizations 사용 설명서의 [서비스 제어 정책](#)을 참조하세요.
- 리소스 제어 정책(RCP) - 계정의 리소스에 사용할 수 있는 최대 권한을 설정합니다. 자세한 내용은 AWS Organizations 사용 설명서의 [리소스 제어 정책\(RCP\)](#)을 참조하세요.
- 세션 정책 - 역할 또는 페더레이션 사용자에게 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 자세한 내용은 IAM 사용 설명서의 [세션 정책](#)을 참조하세요.

## 여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 [정책 평가 로직](#)을 참조하세요.

## Amazon EMR on EKS에서 IAM을 사용하는 방법

IAM을 사용하여 Amazon EMR on EKS에 대한 액세스를 관리하기 전에 Amazon EMR on EKS에서 사용할 수 있는 IAM 기능을 알아봅니다.

Amazon EMR on EKS에서 사용할 수 있는 IAM 기능

| IAM 특성                      | Amazon EMR on EKS 지원 |
|-----------------------------|----------------------|
| <a href="#">자격 증명 기반 정책</a> | 예                    |
| <a href="#">리소스 기반 정책</a>   | 아니요                  |

| IAM 특성                       | Amazon EMR on EKS 지원 |
|------------------------------|----------------------|
| <a href="#">정책 작업</a>        | 예                    |
| <a href="#">정책 리소스</a>       | 예                    |
| <a href="#">정책 조건 키</a>      | 예                    |
| <a href="#">ACL</a>          | 아니요                  |
| <a href="#">ABAC(정책의 태그)</a> | 예                    |
| <a href="#">임시 보안 인증</a>     | 예                    |
| <a href="#">엔터티 권한</a>       | 예                    |
| <a href="#">서비스 역할</a>       | 아니요                  |
| <a href="#">서비스 연결 역할</a>    | 예                    |

Amazon EMR on EKS 및 기타 AWS 서비스에서 대부분의 IAM 기능을 사용하는 방법을 전체적으로 알아보려면 IAM 사용 설명서의 [AWS IAM으로 작업하는 서비스](#)를 참조하세요.

## Amazon EMR on EKS에 대한 자격 증명 기반 정책

ID 기반 정책 지원: 예

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 [고객 관리형 정책으로 사용자 지정 IAM 권한 정의](#)를 참조하세요.

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. JSON 정책에서 사용할 수 있는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 [IAM JSON 정책 요소 참조](#)를 참조하세요.

## Amazon EMR on EKS에 대한 자격 증명 기반 정책 예제

Amazon EMR on EKS 자격 증명 기반 정책 예제를 보려면 [Amazon EMR on EKS에 대한 자격 증명 기반 정책 예제](#) 섹션을 참조하세요.

## Amazon EMR on EKS 내 리소스 기반 정책

리소스 기반 정책 지원: 아니요

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 보안 주체가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 [보안 주체를 지정](#)해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는 이 포함될 수 있습니다 AWS 서비스.

교차 계정 액세스를 활성화하려는 경우, 전체 계정이나 다른 계정의 IAM 개체를 리소스 기반 정책의 보안 주체로 지정할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [IAM에서 교차 계정 리소스 액세스](#)를 참조하세요.

## Amazon EMR on EKS에 대한 정책 작업

정책 작업 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 Action요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

Amazon EMR on EKS 작업의 목록을 보려면 서비스 승인 참조에서 [Amazon EMR on EKS에 대한 작업, 리소스 및 조건 키](#)를 참조하세요.

Amazon EMR on EKS에서 정책 작업은 작업 앞에 다음 접두사를 사용합니다.

```
emr-containers
```

단일 문에서 여러 작업을 지정하려면 쉼표로 구분합니다.

```
"Action": [
    "emr-containers:action1",
    "emr-containers:action2"
]
```

Amazon EMR on EKS 자격 증명 기반 정책 예제를 보려면 [Amazon EMR on EKS에 대한 자격 증명 기반 정책 예제](#) 섹션을 참조하세요.

## Amazon EMR on EKS의 정책 리소스

정책 리소스 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 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. 모범 사례에 따라 [Amazon 리소스 이름\(ARN\)](#)을 사용하여 리소스를 지정합니다. 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(\*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"

```

Amazon EMR on EKS 리소스 유형 및 해당 ARN의 목록을 보려면 서비스 권한 부여 참조에서 [Amazon EMR on EKS에 의해 정의된 리소스](#)를 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업에 대해 알아보려면 [Amazon EMR on EKS에 대한 작업, 리소스 및 조건 키](#)를 참조하세요.

Amazon EMR on EKS 자격 증명 기반 정책 예제를 보려면 [Amazon EMR on EKS에 대한 자격 증명 기반 정책 예제](#) 섹션을 참조하세요.

## Amazon EMR on EKS에 대한 정책 조건 키

서비스별 정책 조건 키 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

Condition 요소는 정의된 기준에 따라 문이 실행되는 시기를 지정합니다. 같음(equals) 또는 미만(less than)과 같은 [조건 연산자](#)를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 [AWS 전역 조건 컨텍스트 키](#)를 참조하세요.

Amazon EMR on EKS 조건 키 목록을 보고 조건 키를 사용할 수 있는 작업 및 리소스에 대해 알아보려면 서비스 승인 참조에서 [Amazon EMR on EKS에 대한 작업, 리소스 및 조건 키](#)를 참조하세요.

Amazon EMR on EKS 자격 증명 기반 정책 예제를 보려면 [Amazon EMR on EKS에 대한 자격 증명 기반 정책 예제](#) 섹션을 참조하세요.

## Amazon EMR on EKS의 액세스 제어 목록(ACL)

ACL 지원: 아니요

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 EMR on EKS를 사용한 ABAC(속성 기반 액세스 제어)

ABAC 지원(정책의 태그) 예

속성 기반 액세스 제어(ABAC)는 태그라고 불리는 속성을 기반으로 권한을 정의하는 권한 부여 전략입니다. IAM 엔터티 및 AWS 리소스에 태그를 연결한 다음 보안 주체의 태그가 리소스의 태그와 일치할 때 작업을 허용하는 ABAC 정책을 설계할 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.

서비스가 모든 리소스 유형에 대해 세 가지 조건 키를 모두 지원하는 경우, 값은 서비스에 대해 예입니다. 서비스가 일부 리소스 유형에 대해서만 세 가지 조건 키를 모두 지원하는 경우, 값은 부분적입니다.

ABAC에 대한 자세한 내용은 IAM 사용 설명서의 [ABAC 권한 부여를 통한 권한 정의](#)를 참조하세요. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용 설명서의 [속성 기반 액세스 제어\(ABAC\) 사용](#)을 참조하세요.

## Amazon EMR on EKS에서 임시 보안 인증 사용

임시 자격 증명 지원: 예

임시 자격 증명은 AWS 리소스에 대한 단기 액세스를 제공하며 페더레이션을 사용하거나 역할을 전환할 때 자동으로 생성됩니다. 장기 액세스 키를 사용하는 대신 임시 자격 증명을 동적으로 생성하는 것이 AWS 좋습니다. 자세한 내용은 IAM 사용 설명서의 [IAM의 임시 보안 자격 증명](#) 및 [IAM으로 작업하는 AWS 서비스](#) 섹션을 참조하세요.

## Amazon EMR on EKS에 대한 교차 서비스 보안 주체 권한

전달 액세스 세션(FAS) 지원: 예

전달 액세스 세션(FAS)은 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스 함께 사용합니다. FAS 요청 시 정책 세부 정보는 [전달 액세스 세션](#)을 참조하세요.



## Amazon EMR on EKS에 대한 서비스 역할

|           |     |
|-----------|-----|
| 서비스 역할 지원 | 아니요 |
|-----------|-----|

## Amazon EMR on EKS에 대한 서비스 연결 역할

|              |   |
|--------------|---|
| 서비스 링크 역할 지원 | 예 |
|--------------|---|

서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 [IAM으로 작동하는AWS 서비스](#) 섹션을 참조하세요. 서비스 연결 역할 열에서 Yes가 포함된 서비스를 테이블에서 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예(Yes) 링크를 선택합니다.

## Amazon EMR on EKS에 대한 서비스 연결 역할 사용

Amazon EMR on EKS는 AWS Identity and Access Management (IAM) [서비스 연결 역할을](#) 사용합니다. 서비스 연결 역할은 Amazon EMR on EKS에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Amazon EMR on EKS에서 사전 정의하며 서비스가 사용자를 대신하여 다른 AWS 서비스를 호출하는 데 필요한 모든 권한을 포함합니다.

서비스 연결 역할을 사용하면 필요한 권한을 수동으로 추가할 필요가 없으므로 Amazon EMR on EKS를 더 쉽게 설정할 수 있습니다. Amazon EMR on EKS에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한, Amazon EMR on EKS만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔터티에 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 EMR on EKS 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 [IAM으로 작업하는AWS 서비스](#)를 참조해 서비스 연결 역할 열이 예(Yes)인 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예(Yes) 링크를 선택합니다.

## Amazon EMR on EKS에 대한 서비스 연결 역할 권한

Amazon EMR on EKS는 AWSServiceRoleForAmazonEMRContainers라는 서비스 연결 역할을 사용합니다.

AWSServiceRoleForAmazonEMRContainers 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- `emr-containers.amazonaws.com`

역할 권한 정책 `AmazonEMRContainersServiceRolePolicy`를 통해 Amazon EMR on EKS는 다음 정책 명령문에서 볼 수 있듯이 지정된 리소스에서 작업을 완료할 수 있습니다.

#### Note

관리형 정책 내용은 수시로 변경되므로 여기에 수록된 정책은 오래된 버전일 수 있습니다. AWS 관리형 정책 참조 가이드의 [AmazonEMRContainersServiceRolePolicy](#)에서 최신 정책 설명서를 확인하세요.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks:DescribeCluster",
        "eks:ListNodeGroups",
        "eks:DescribeNodeGroup",
        "ec2:DescribeRouteTables",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "elasticloadbalancing:DescribeInstanceHealth",
        "elasticloadbalancing:DescribeLoadBalancers",
        "elasticloadbalancing:DescribeTargetGroups",
        "elasticloadbalancing:DescribeTargetHealth",
        "eks:ListPodIdentityAssociations",
        "eks:DescribePodIdentityAssociation"
      ],
      "Resource": [
        "*"
      ],
      "Sid": "AllowEKSDescribecluster"
    },
    {
      "Effect": "Allow",
      "Action": [
```

```

        "acm:ImportCertificate",
        "acm:AddTagsToCertificate"
    ],
    "Resource": [
        "*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:RequestTag/emr-container:endpoint:managed-certificate": "true"
        }
    },
    "Sid": "AllowACMImportcertificate"
},
{
    "Effect": "Allow",
    "Action": [
        "acm:DeleteCertificate"
    ],
    "Resource": [
        "*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:ResourceTag/emr-container:endpoint:managed-certificate": "true"
        }
    },
    "Sid": "AllowACMDeletecertificate"
}
]
}

```

IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 연결 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 권한](#) 섹션을 참조하세요.

## Amazon EMR on EKS에 대한 서비스 연결 역할 생성

서비스 연결 역할은 수동으로 생성할 필요가 없습니다. 가상 클러스터를 생성할 때 Amazon EMR on EKS에서는 서비스 연결 역할을 자동으로 생성합니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 가상 클러스터를 생성할 때 Amazon EMR on EKS에서는 서비스 연결 역할을 자동으로 다시 생성합니다.

또한 IAM 콘솔을 사용해 Amazon EMR on EKS 사용 사례로 서비스 연결 역할을 생성할 수도 있습니다. AWS CLI 또는 AWS API에서 서비스 이름으로 `emr-containers.amazonaws.com` 서비스 연결 역할을 생성합니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 생성](#) 섹션을 참조하세요. 이 서비스 연결 역할을 삭제하면 동일한 프로세스를 사용하여 역할을 다시 생성할 수 있습니다.

## Amazon EMR on EKS에 대한 서비스 연결 역할 편집

Amazon EMR on EKS에서는 `AWSServiceRoleForAmazonEMRContainers` 서비스 연결 역할을 편집하도록 허용하지 않습니다. 서비스 연결 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 EMR on 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 역할에 대한 리소스를 먼저 정리해야 수동으로 삭제할 수 있습니다.

### Note

리소스를 삭제하려고 할 때 Amazon EMR on EKS 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

**AWSServiceRoleForAmazonEMRContainers**에 의해 사용되는 Amazon EMR on EKS 리소스를 삭제하는 방법

1. Amazon EMR 콘솔을 엽니다.
2. 가상 클러스터를 선택합니다.
3. Virtual Cluster 페이지에서 삭제를 선택합니다.
4. 계정의 다른 가상 클러스터에 대해 이 절차를 반복합니다.

IAM을 사용하여 수동으로 서비스 연결 역할을 삭제하려면 다음을 수행하세요.

IAM 콘솔 AWS CLI, 또는 AWS API를 사용하여 `AWSServiceRoleForAmazonEMRContainers` 서비스 연결 역할을 삭제합니다. 자세한 내용은 [IAM 사용 설명서](#)의 서비스 연결 역할 삭제를 참조하세요.

## Amazon EMR on EKS 서비스 연결 역할을 지원하는 리전

Amazon EMR on EKS는 서비스가 제공되는 모든 리전에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트 및 할당량](#) 단원을 참조하십시오.

## Amazon EMR on EKS 관리형 정책

2021년 3월 1일 이후 Amazon EMR on EKS의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다.

| 변경                                                                                                                                                                                                             | 설명                                                                                                                                                 | Date         |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------|--------------|
| AmazonEMRContainerServiceRolePolicy<br>- 클러스터의 EKS Pod Identity 연결을 나열하는 읽기 권한과 클러스터의 Pod Identity 연결에 대한 설명 정보를 반환하는 또 다른 읽기 권한을 추가했습니다. 자세한 내용은 <a href="#">AmazonEMRContainerServiceRolePolicy</a> 를 참조하세요. | 정책에 eks:ListPodIdentityAssociations, eks:DescribePodIdentityAssociation 의 권한이 추가되었습니다.                                                             | 2023년 2월 3일  |
| AmazonEMRContainerServiceRolePolicy<br>- Amazon EKS 노드 그룹을 설명 및 나열하고, 로드 밸런서 대상 그룹을 설명하며, 로드 밸런서 대상 상태를 설명하는 권한을 추가했습니다.                                                                                       | 정책에 eks:ListNodeGroups, eks:DescribeNodeGroup, elasticloadbalancing:DescribeTargetGroups, elasticloadbalancing:DescribeTargetHealth 의 권한이 추가되었습니다. | 2023년 3월 13일 |
| AmazonEMRContainerServiceRolePolicy<br>- 인증서를 가져오고 삭제할 수 있는 권한이 추가되                                                                                                                                            | 정책에 acm:ImportCertificate, acm:AddTagsToCertificate,                                                                                               | 2021년 12월 3일 |

| 변경                                   | 설명                                                  | Date        |
|--------------------------------------|-----------------------------------------------------|-------------|
| 있습니 AWS Certificate Manager.         | acm:DeleteCertificate 의 권한이 추가되었습니다.                |             |
| Amazon EMR on EKS가 변경 내용 추적을 시작했습니다. | Amazon EMR on EKS는 AWS 관리형 정책에 대한 변경 사항 추적을 시작했습니다. | 2021년 3월 1일 |

## Amazon EMR on EKS에서 작업 실행 역할 사용

StartJobRun 명령을 사용하여 EKS 클러스터에서 작업 실행을 제출하려면 먼저 가상 클러스터에서 사용할 작업 실행 역할을 온보딩해야 합니다. 자세한 설명은 [Amazon EMR on EKS 설정](#)에서 [작업 실행 역할 생성](#) 섹션을 참조하세요. Amazon EMR on EKS 워크숍의 [Create IAM Role for job execution](#) 섹션의 지침을 따를 수도 있습니다.

작업 실행 역할에 대한 신뢰 정책에 다음 권한이 포함되어야 합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::AWS_ACCOUNT_ID:oidc-provider/OIDC_PROVIDER"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "OIDC_PROVIDER:aud": "sts.amazonaws.com"
        },
        "StringLike": {
          "OIDC_PROVIDER:sub": "system:serviceaccount:NAMESPACE:emr-containers-sa-*-*AWS_ACCOUNT_ID-BASE36_ENCODED_ROLE_NAME"
        }
      }
    }
  ]
}
```

```
}
```

위 예제의 신뢰 정책은 `emr-containers-sa-*-`

`*-AWS_ACCOUNT_ID-BASE36_ENCODED_ROLE_NAME` 패턴과 일치하는 이름의 Amazon EMR 관리형 Kubernetes 서비스 계정에만 권한을 부여합니다. 이 패턴의 서비스 계정은 작업 제출 시 자동으로 생성되며 작업을 제출하는 네임스페이스로 범위가 지정됩니다. 이 신뢰 정책을 통해 이러한 서비스 계정이 실행 역할을 수임하고 실행 역할의 임시 보안 인증을 가져올 수 있습니다. 다른 Amazon EKS 클러스터 또는 동일한 EKS 클러스터 내 다른 네임스페이스의 서비스 계정은 실행 역할을 수임할 수 없습니다.

다음 명령을 실행하여 위에 제공된 형식으로 신뢰 정책을 자동 업데이트할 수 있습니다.

```
aws emr-containers update-role-trust-policy \
  --cluster-name cluster \
  --namespace namespace \
  --role-name iam_role_name_for_job_execution
```

## 실행 역할에 대한 액세스 제어

Amazon EKS 클러스터의 관리자는 Amazon EMR on EKS의 멀티 테넌트 가상 클러스터를 생성할 수 있으며, 여기에 IAM 관리자가 여러 실행 역할을 추가할 수 있습니다. 신뢰할 수 없는 테넌트는 이러한 실행 역할을 사용하여 임의 코드를 실행하는 작업을 제출할 수 있으므로, 이러한 실행 역할 중 하나 이상에 할당된 권한을 획득하는 코드를 실행할 수 없도록 테넌트를 제한할 수 있습니다. IAM 관리자는 선택적 Amazon 리소스 이름(ARN) 조건 키(`emr-containers:ExecutionRoleArn`)를 사용하여 IAM 자격 증명에 연결된 IAM 정책을 제한할 수 있습니다. 이 조건은 다음 권한 정책에서 볼 수 있듯이 가상 클러스터에 대한 권한이 있는 실행 역할 ARN 목록을 수락합니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:StartJobRun"
      ],
      "Resource": [
        "arn:aws:emr-containers:*:*:/virtualclusters/VIRTUAL_CLUSTER_ID"
      ]
    }
  ]
}
```

```

    ],
    "Condition": {
      "ArnEquals": {
        "emr-containers:ExecutionRoleArn": [
          "arn:aws:iam::*:role/execution_role_name_1",
          "arn:aws:iam::*:role/execution_role_name_2"
        ]
      }
    },
    "Sid": "AllowEMRCONTAINERSStartjobrun"
  }
]
}

```

특정 접두사로 시작하는 모든 실행 역할(예: MyRole)을 허용하려면 조건 연산자 ArnEquals를 ArnLike 연산자로 바꾸고 조건의 execution\_role\_arn 값을 와일드카드 \* 문자로 바꿀 수 있습니다. 예를 들어 arn:aws:iam::*AWS\_ACCOUNT\_ID*:role/MyRole\*입니다. 다른 모든 [ARN 조건 키](#)도 지원됩니다.

#### Note

Amazon EMR on EKS에서는 태그 또는 속성을 기반으로 실행 역할에 권한을 부여할 수 없습니다. Amazon EMR on EKS는 실행 역할에 대해 TBAC(태그 기반 액세스 제어) 또는 ABAC(속성 기반 액세스 제어)를 지원하지 않습니다.

## Amazon EMR on EKS에 대한 자격 증명 기반 정책 예제

기본적으로 사용자 및 역할은 Amazon EMR on EKS 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 [IAM 정책 생성\(콘솔\)](#)을 참조하세요.

각 리소스 유형에 대한 ARN 형식을 포함하여 Amazon EMR on EKS에서 정의한 작업 및 리소스 유형에 대한 자세한 내용은 서비스 인증 참조에서 [Amazon EMR on EKS에 대한 작업, 리소스 및 조건 키](#)를 참조하세요.

주제



- [정책 모범 사례](#)
- [Amazon EMR on EKS 콘솔 사용](#)
- [사용자가 자신의 고유한 권한을 볼 수 있도록 허용](#)

## 정책 모범 사례

자격 증명 기반 정책에 따라 계정에서 사용자가 Amazon EMR on EKS 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따르세요.

- AWS 관리형 정책을 시작하고 최소 권한으로 전환 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반적인 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 에서 사용할 수 있습니다 AWS 계정. 사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 추가로 줄이는 것이 좋습니다. 자세한 내용은 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 정책 조건을 작성할 수 있습니다. AWS 서비스와 같은 특징을 통해 사용되는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다 CloudFormation. 자세한 내용은 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 내용은 IAM 사용 설명서의 [IAM Access Analyzer에서 정책 검증](#)을 참조하세요.
- 다중 인증(MFA) 필요 -에서 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 AWS 계정킵니다. API 작업을 직접적으로 호출할 때 MFA가 필요하다면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 [MFA를 통한 보안 API 액세스](#)를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 [IAM의 보안 모범 사례](#)를 참조하세요.

## Amazon EMR on EKS 콘솔 사용

Amazon EMR on EKS 콘솔에 액세스하려면 최소 권한 세트가 있어야 합니다. 이러한 권한은 AWS 계정에서 Amazon EMR on EKS 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다. 최소 필수 권한보다 더 제한적인 ID 기반 정책을 생성하는 경우, 콘솔이 해당 정책에 연결된 엔티티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

AWS CLI 또는 AWS API만 호출하는 사용자에게는 최소 콘솔 권한을 허용할 필요가 없습니다. 대신, 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자와 역할이 여전히 Amazon EMR on EKS 콘솔을 사용할 수 있도록 하려면 Amazon EMR on EKS ConsoleAccess 또는 ReadOnly AWS 관리형 정책을 엔티티에 연결합니다. 자세한 내용은 IAM 사용 설명서의 [사용자에게 권한 추가](#)를 참조하세요.

### 사용자가 자신의 고유한 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 CLI 또는 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:ListGroupsWithUser",
        "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": "*"
}
]
}

```

## 태그 기반 액세스 제어를 위한 정책

자격 증명 기반 정책에서 조건을 사용하여 태그에 따라 가상 클러스터 및 작업 실행에 대한 액세스를 제어할 수 있습니다. 태그 지정에 대한 자세한 내용은 [Amazon EMR on EKS 리소스 태그 지정](#) 섹션을 참조하세요.

다음 예제에서는 Amazon EMR on EKS 조건 키와 함께 조건 연산자를 사용할 수 있는 다양한 시나리오와 방법을 보여줍니다. 이러한 IAM 정책 명령문은 데모용일 뿐이며 프로덕션 환경에서 사용해서는 안 됩니다. 다양한 방법으로 정책 명령문을 결합하여 요구 사항에 따라 권한을 부여하거나 거부할 수 있습니다. IAM 정책 계획 및 테스트에 대한 자세한 내용은 [IAM 사용 설명서](#)를 참조하세요.

### Important

태깅 작업에 대한 권한을 명시적으로 거부하는 것은 중요한 고려 사항입니다. 이렇게 하면 사용자가 리소스에 태그를 지정하지 못하므로, 부여할 의도가 없는 권한이 부여되지 않도록 방지할 수 있습니다. 리소스에 대한 태그 지정 작업이 거부되지 않는 경우 사용자는 태그를 수정하여 태그 기반 정책의 의도를 우회할 수 있습니다. 태그 지정 작업을 거부하는 정책의 예제는 [태그를 추가 또는 제거하는 액세스 거부](#) 섹션을 참조하세요.

아래의 예제에서는 Amazon EMR on EKS 가상 클러스터에서 허용되는 작업의 제어에 사용되는 자격 증명 기반 권한 정책을 보여줍니다.

### 특정 태그 값이 있는 리소스에서만 작업 허용

다음 정책 예제에서 StringEquals 조건 연산자는 dev를 department 태그의 값과 일치시키려고 시도합니다. department 태그가 가상 클러스터에 추가되지 않았거나 dev 값이 포함되지 않은 경우 정책이 적

용되지 않으며 이 정책에 따라 작업이 허용되지 않습니다. 작업을 허용하는 다른 정책 명령문이 없는 경우 사용자는 이 값과 함께 이 태그가 있는 가상 클러스터만 작업할 수 있습니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:DescribeVirtualCluster"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/department": "dev"
        }
      },
      "Sid": "AllowEMRCONTAINERDescribevirtualcluster"
    }
  ]
}
```

또한 조건 연산자를 사용하여 여러 태그 값을 지정할 수 있습니다. 예를 들어, department 태그에 dev 또는 test 값이 포함된 가상 클러스터에서 모든 작업을 허용하려면 이전 예제의 조건 블록을 다음으로 대체할 수 있습니다.

```
"Condition": {
  "StringEquals": {
    "aws:ResourceTag/department": ["dev", "test"]
  }
}
```

## 리소스 생성 시 태그 지정이 필요함

아래 예제에서는 가상 클러스터를 생성할 때 태그를 적용해야 합니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:CreateVirtualCluster"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/department": "dev"
        }
      },
      "Sid": "AllowEMRCONTAINERSCreatevirtualcluster"
    }
  ]
}
```

다음 정책 명령문에서는 클러스터에 department 태그가 있는 경우에만 사용자가 가상 클러스터를 생성할 수 있도록 허용합니다. 이 태그에는 어떤 값이든 포함될 수 있습니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-containers:CreateVirtualCluster"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
```

```

    "Null": {
      "aws:RequestTag/department": "false"
    },
    "Sid": "AllowEMRCONTAINERSCreatevirtualcluster"
  }
]
}

```

## 태그를 추가 또는 제거하는 액세스 거부

이 정책의 효과는 department 값이 포함된 dev 태그를 사용하여 태그가 지정된 가상 클러스터에서 태그를 추가하거나 제거할 수 있는 권한을 사용자에게 거부하는 것입니다.

## JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "emr-containers:TagResource",
        "emr-containers:UntagResource"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:ResourceTag/department": "dev"
        }
      },
      "Sid": "AllowEMRCONTAINERSTagresource"
    }
  ]
}

```

## Amazon EMR on EKS 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 Amazon EMR on EKS 및 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단하고 수정할 수 있습니다.

### 주제

- [Amazon EMR on EKS에서 작업을 수행할 권한이 없는 경우](#)
- [iam:PassRole을 수행하도록 인증되지 않음](#)
- [내 AWS 계정 외부의 사람이 내 Amazon EMR on EKS 리소스에 액세스하도록 허용하고 싶습니다.](#)

### Amazon EMR on EKS에서 작업을 수행할 권한이 없는 경우

에서 작업을 수행할 권한이 없다는 AWS Management Console 메시지가 표시되면 관리자에게 문의하여 도움을 받아야 합니다. 관리자는 사용자 이름과 암호를 제공한 사람입니다.

다음 예제 오류는 mateojackson 사용자가 콘솔을 사용하여 가상 *my-example-widget* 리소스에 대한 세부 정보를 보려고 하지만 가상 *emr-containers:RedWidget* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: emr-containers:RedWidget on resource: my-example-widget
```

이 경우 Mateo는 *my-example-widget* 작업을 사용하여 *emr-containers:RedWidget* 리소스에 액세스하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

### iam:PassRole을 수행하도록 인증되지 않음

*iam:PassRole* 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 Amazon EMR on EKS에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있습니다. 이렇게 하려면 역할을 서비스에 전달할 권한이 있어야 합니다.

다음 예제 오류는 marymajor라는 IAM 사용자가 콘솔을 사용하여 Amazon EMR on EKS에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 권한이 없습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

이 경우, Mary가 iam:PassRole작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 AWS 계정 외부의 사람이 내 Amazon EMR on EKS 리소스에 액세스하도록 허용하고 싶습니다.

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.

자세한 내용은 다음을 참조하세요.

- Amazon EMR on EKS에서 이러한 기능을 지원하는지 알아보려면 [Amazon EMR on EKS에서 IAM을 사용하는 방법](#) 섹션을 참조하세요.
- 소유 AWS 계정 한의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 [IAM 사용 설명서의 소유한 다른의 IAM 사용자에게 액세스 권한 제공을 참조 AWS 계정 하세요.](#)
- 타사에 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 [타사가 AWS 계정 소유한에 대한 액세스 권한 제공을](#) AWS 계정참조하세요.
- 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.

## 세분화된 액세스 제어를 위해 AWS Lake Formation과 함께 Amazon EMR on EKS 사용

Amazon EMR 릴리스 7.7 이상에서는 AWS Lake Formation을 활용하여 Amazon S3 버킷이 지원하는 AWS Glue Data Catalog 테이블에 세분화된 액세스 제어를 적용할 수 있습니다. 이 기능을 통해 Amazon EKS Spark 작업 내 EMR에서 읽기 쿼리에 대한 테이블, 행, 열 및 셀 수준의 액세스 제어를 구성할 수 있습니다.

주제

- [Amazon EMR on EKS가 AWS Lake Formation과 작동하는 방식](#)
- [Amazon EMR on EKS를 사용한 Lake Formation 활성화](#)

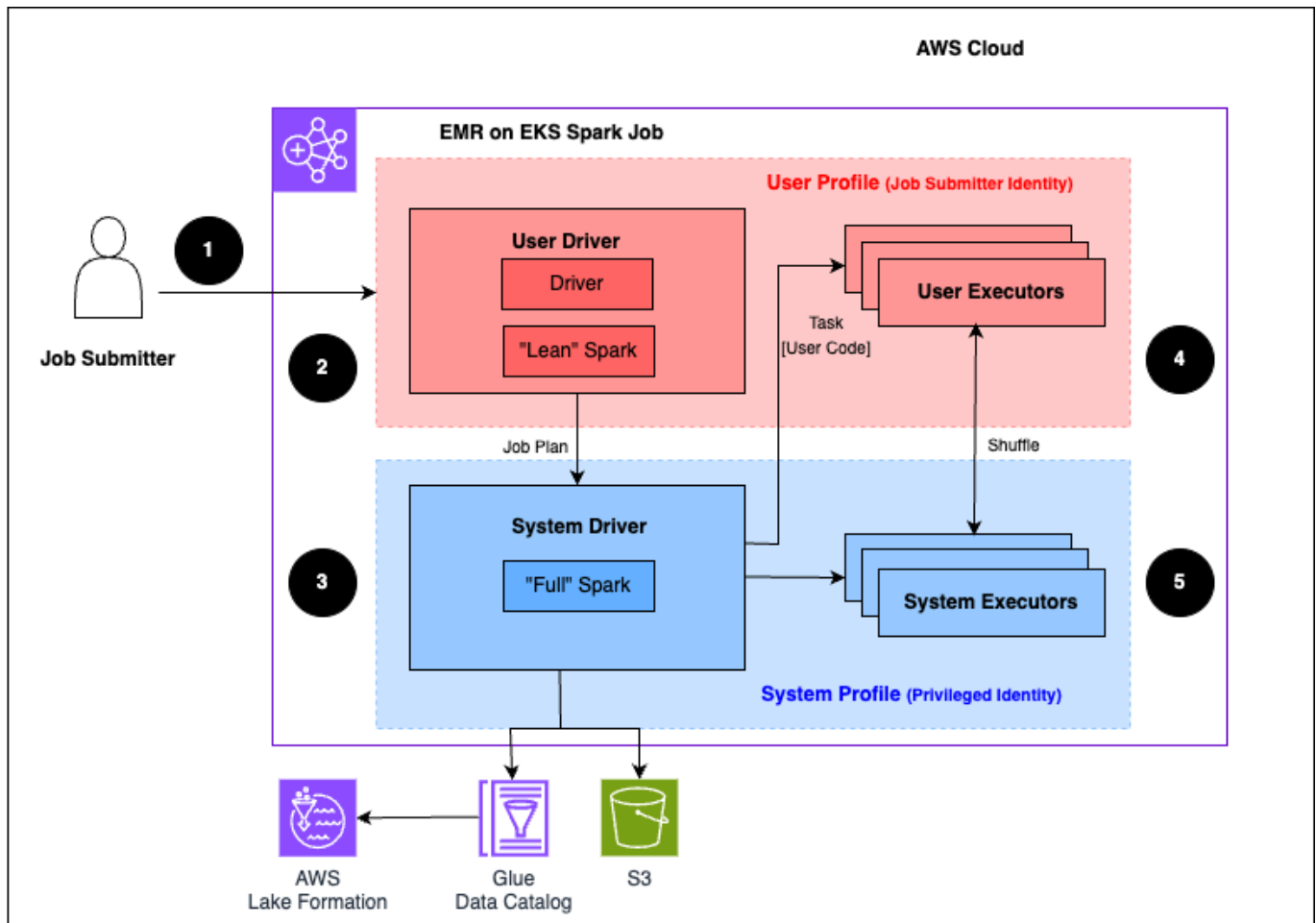


- [고려 사항 및 제한 사항](#)
- [문제 해결](#)

## Amazon EMR on EKS가 AWS Lake Formation과 작동하는 방식

Lake Formation과 함께 EKS의 Amazon EMR을 사용하면 각 Spark 작업에 대한 권한 계층을 적용하여 EKS의 Amazon EMR이 작업을 실행할 때 Lake Formation 권한 제어를 적용할 수 있습니다. Amazon EMR on EKS는 Spark 리소스 프로파일을 사용하여 작업을 효과적으로 실행하기 위해 두 개의 프로파일을 생성합니다. <https://spark.apache.org/docs/latest/api/java/org/apache/spark/resource/ResourceProfile.html> 사용자 프로파일은 사용자 제공 코드를 실행하는 반면, 시스템 프로파일은 Lake Formation 정책을 적용합니다. 각 Lake Formation 활성화 작업은 사용자 프로파일용과 시스템 프로파일용의 두 가지 Spark 드라이버를 사용합니다. 자세한 내용은 [AWS Lake Formation](#) 소개를 참조하세요.

다음 개요에서는 Amazon EMR on EKS가 Lake Formation 보안 정책에 따라 보호되는 데이터에 액세스하는 방법을 설명합니다.



다음 단계에서는 다음 프로세스에 대해 설명합니다.

1. 사용자가 AWS Lake Formation 지원 Amazon EMR on EKS 가상 클러스터에 Spark 작업을 제출합니다.
2. Amazon EMR on EKS 서비스는 사용자 드라이버를 설정하고 사용자 프로필에서 작업을 실행합니다. 사용자 드라이버는 작업을 시작하고, 실행기를 요청하며, Amazon S3 또는 Glue Data Catalog에 액세스할 수 없는 린 버전의 Spark를 실행합니다. 작업 계획의 빌드만을 수행합니다.
3. Amazon EMR on EKS 서비스는 시스템 드라이버라는 두 번째 드라이버를 설정하고 시스템 프로파일에서 실행합니다(권한 있는 ID로). Amazon EKS는 통신을 위해 두 드라이버 사이에서 암호화된 TLS 채널을 설정합니다. 사용자 드라이버는 채널을 사용하여 작업 계획을 시스템 드라이버로 전송합니다. 시스템 드라이버는 사용자가 제출한 코드를 실행하지 않습니다. 전체 Spark를 실행하고 Amazon S3 및 데이터 액세스를 위해 Data Catalog와 통신합니다. 실행기를 요청하고 작업 계획을 일련의 실행 단계로 컴파일합니다.
4. 그런 다음 Amazon EMR on EKS 서비스는 실행기에서 단계를 실행합니다. 모든 단계의 사용자 코드는 사용자 프로파일 실행기에서만 실행됩니다.
5. Lake Formation으로 보호되는 데이터 카탈로그 테이블에서 데이터를 읽는 단계 또는 보안 필터를 적용하는 단계는 시스템 실행기로 위임됩니다.

## Amazon EMR on EKS를 사용한 Lake Formation 활성화

Amazon EMR 릴리스 7.7 이상에서는 AWS Lake Formation을 활용하여 Amazon S3에서 지원하는 데이터 카탈로그 테이블에 세분화된 액세스 제어를 적용할 수 있습니다. 이 기능을 통해 Amazon EKS Spark 작업 내 EMR에서 읽기 쿼리에 대한 테이블, 행, 열 및 셀 수준의 액세스 제어를 구성할 수 있습니다.

이 섹션에서는 보안 구성을 생성하고 Amazon EMR과 함께 작동하도록 Lake Formation을 설정하는 방법을 다룹니다. 또한 Lake Formation용으로 생성한 보안 구성으로 가상 클러스터를 만드는 방법도 설명합니다. 이 섹션은 순서대로 완료되어야 합니다.

### 1단계: Lake Formation 기반 열, 행 또는 셀 수준 사용 권한 설정

먼저, Lake Formation으로 행 및 열 수준 권한을 적용하려면, Lake Formation의 데이터 레이크 관리자가 LakeFormationAuthorizedCaller 세션 태그를 설정해야 합니다. Lake Formation은 이 세션 태그를 사용하여 직접 호출자에게 권한을 부여하고 데이터 레이크에 대한 액세스를 제공합니다.

AWS Lake Formation 콘솔로 이동하여 사이드바의 관리 섹션에서 애플리케이션 통합 설정 옵션을 선택합니다. 그런 다음, 외부 엔진이 Lake Formation에 등록된 Amazon S3 위치의 데이터를 필터링하도록 허용 확인란을 선택합니다. Spark 작업이 실행될 AWS 계정 ID와 세션 태그 값을 추가합니다.

## Application integration settings [Learn more](#)

### Application integration settings

Use the options below to control which third-party engines are allowed to read and filter data in Amazon S3 locations registered with Lake Formation.

☒ Allow external engines to filter data in Amazon S3 locations registered with Lake Formation

Check this box to allow third-party engines to access data in Amazon S3 locations that are registered with Lake Formation.

#### Session tag values

Enter one or more strings that match the LakeFormationAuthorizedCaller session tag defined for third-party engines.

Clear all

EMR on EKS Engine X

Enter one or several string values separated by comma.

#### AWS account IDs

Enter the external AWS account IDs from where third-party engines are allowed to access locations registered with Lake Formation.

Clear all

012345678901 X

Account

Enter one or more AWS account IDs. Press enter after each ID.

☐ Allow external engines to access data in Amazon S3 locations with full table access

When you enable this option, Lake Formation will return credentials to the integrated application directly without IAM session tag validation.

Cancel

Save

여기에 전달된 LakeFormationAuthorizedCaller 세션 태그는 나중에 섹션 3에서 IAM 역할을 설정할 때 SecurityConfiguration에 전달됩니다.

## 2단계: EKS RBAC 권한 설정

두 번째로, 역할 기반 액세스 제어에 대한 권한을 설정합니다.

EKS 클러스터 권한을 Amazon EMR on EKS 서비스에 제공

Amazon EMR on EKS 서비스는 EKS 클러스터 역할 권한을 반드시 보유해야 합니다. 이를 통해 시스템 드라이버가 사용자 네임스페이스 내에서 사용자 실행기를 생성할 수 있도록 네임스페이스 간 권한을 생성할 수 있습니다.

### 클러스터 역할 생성

이 샘플은 리소스 모음의 권한을 정의합니다.

```
vim emr-containers-cluster-role.yaml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: emr-containers
rules:
  - apiGroups: [""]
    resources: ["namespaces"]
    verbs: ["get"]
  - apiGroups: [""]
    resources: ["serviceaccounts", "services", "configmaps", "events", "pods", "pods/
log"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["create", "patch", "delete", "watch"]
  - apiGroups: ["apps"]
    resources: ["statefulsets", "deployments"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete", "annotate",
"patch", "label"]
  - apiGroups: ["batch"]
    resources: ["jobs"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete", "annotate",
"patch", "label"]
  - apiGroups: ["extensions", "networking.k8s.io"]
    resources: ["ingresses"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete", "annotate",
"patch", "label"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["clusterroles", "clusterrolebindings", "roles", "rolebindings"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletecollection", "annotate", "patch", "label"]
  - apiGroups: [""]
    resources: ["persistentvolumeclaims"]
    verbs: ["get", "list", "watch", "describe", "create", "edit", "delete",
"deletecollection", "annotate", "patch", "label"]
  - apiGroups: ["kyverno.io"]
    resources: ["clusterpolicies"]
    verbs: ["create", "delete"]
---
```

```
kubectl apply -f emr-containers-cluster-role.yaml
```

## 클러스터 역할 바인딩 만들기

```
vim emr-containers-cluster-role-binding.yaml
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: emr-containers
subjects:
- kind: User
  name: emr-containers
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: emr-containers
  apiGroup: rbac.authorization.k8s.io
---
```

```
kubectl apply -f emr-containers-cluster-role-binding.yaml
```

## 네임스페이스 액세스 권한을 Amazon EMR on EKS 서비스에 제공

하나는 사용자 드라이버 및 실행기용이고 다른 하나는 시스템 드라이버 및 실행기용인 2개의 Kubernetes 네임스페이스를 생성하고, Amazon EMR on EKS 서비스 액세스를 활성화하여 사용자 및 시스템 네임스페이스 모두에서 작업을 제출할 수 있습니다. 각 네임스페이스에 대한 액세스를 제공하는 기존 가이드는 [aws-auth를 사용한 클러스터 액세스 사용 설정](#)에서 확인할 수 있습니다.

## 3단계: 사용자 및 시스템 프로필 구성 요소의 IAM 역할 설정

세 번째로, 특정 구성 요소의 역할을 설정합니다. Lake Formation 지원 Spark 작업에는 사용자와 시스템의 두 가지 구성 요소가 있습니다. 사용자 드라이버 및 실행기는 사용자 네임스페이스에서 실행되며, StartJobRun API로 전달되는 JobExecutionRole에 연결됩니다. 시스템 드라이버 및 실행기는 System 네임스페이스에서 실행되며 QueryEngine 역할에 연결됩니다.

### 쿼리 엔진 역할 구성

QueryEngine 역할은 시스템 스페이스 구성 요소에 연결되어 있으며 LakeFormationAuthorizedCaller 세션 태그가 있는 JobExecutionRole을 가정할 권한이 있습니다. 쿼리 엔진 역할의 IAM 권한 정책은 다음과 같습니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AssumeJobRoleWithSessionTagAccessForSystemDriver",
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole",
        "sts:TagSession"
      ],
      "Resource": [
        "arn:aws:iam::*:role/JobExecutionRole"
      ],
      "Condition": {
        "StringLike": {
          "aws:RequestTag/LakeFormationAuthorizedCaller": "EMR on EKS Engine"
        }
      }
    },
    {
      "Sid": "AssumeJobRoleWithSessionTagAccessForSystemExecutor",
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/JobExecutionRole"
      ]
    },
    {
      "Sid": "CreateCertificateAccessForTLS",
      "Effect": "Allow",
      "Action": [
        "emr-containers:CreateCertificate"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

}

Kubernetes 시스템 네임스페이스를 신뢰할 수 있도록 쿼리 엔진 역할의 신뢰 정책을 구성합니다.

```
aws emr-containers update-role-trust-policy \
  --cluster-name eks cluster \
  --namespace eks system namespace \
  --role-name query_engine_iam_role_name
```

자세한 내용은 [역할 신뢰 정책 업데이트](#)를 참조하세요.

## 작업 실행 역할 구성

Lake Formation 권한은 AWS Glue 데이터 카탈로그 리소스, Amazon S3 위치 및 해당 위치의 기본 데이터에 대한 액세스를 제어합니다. IAM 권한은 Lake Formation 및 AWS Glue APIs. Data Catalog의 테이블에 액세스할 수 있는 Lake Formation 권한이 있더라도(SELECT) glue:Get\* API 작업에 IAM 권한이 없으면 작업이 실패합니다.

JobExecutionRole의 IAM 권한 정책: JobExecution 역할에는 권한 정책에 정책 설명이 있어야 합니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GlueCatalogAccess",
      "Effect": "Allow",
      "Action": [
        "glue:Get*",
        "glue:Create*",
        "glue:Update*"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "LakeFormationAccess",
```

```

    "Effect": "Allow",
    "Action": [
        "lakeformation:GetDataAccess"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Sid": "CreateCertificateAccessForTLS",
    "Effect": "Allow",
    "Action": [
        "emr-containers:CreateCertificate"
    ],
    "Resource": [
        "*"
    ]
}
]
}

```

JobExecutionRole에 대한 IAM 신뢰 정책:

JSON

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "TrustQueryEngineRoleForSystemDriver",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::123456789012:role/QueryExecutionRole"
            },
            "Action": [
                "sts:AssumeRole",
                "sts:TagSession"
            ],
            "Condition": {
                "StringLike": {
                    "aws:RequestTag/LakeFormationAuthorizedCaller": "EMR on EKS Engine"
                }
            }
        }
    ]
}

```



```

    }
  },
  {
    "Sid": "TrustQueryEngineRoleForSystemExecutor",
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::123456789012:role/QueryEngineRole"
    },
    "Action": "sts:AssumeRole"
  }
]
}

```

Kubernetes 사용자 네임스페이스를 신뢰할 수 있도록 작업 실행 역할의 신뢰 정책을 구성합니다.

```

aws emr-containers update-role-trust-policy \
  --cluster-name eks cluster \
  --namespace eks User namespace \
  --role-name job_execution_role_name

```

자세한 내용은 [작업 실행 역할의 신뢰 정책 업데이트](#)를 참조하세요.

## 4단계: 보안 구성 설정

Lake Formation 지원 작업을 실행하려면 보안 구성을 생성합니다.

```

aws emr-containers create-security-configuration \
  --name 'security-configuration-name' \
  --security-configuration '{
    "authorizationConfiguration": {
      "lakeFormationConfiguration": {
        "authorizedSessionTagValue": "SessionTag configured in LakeFormation",
        "secureNamespaceInfo": {
          "clusterId": "eks-cluster-name",
          "namespace": "system-namespace-name"
        },
        "queryEngineRoleArn": "query-engine-IAM-role-ARN"
      }
    }
  }'

```

authorizedSessionTagValue 필드에 전달된 세션 태그가 Lake Formation을 승인할 수 있는지 확인합니다. [1단계: Lake Formation 기반 열, 행 또는 셀 수준 사용 권한 설정](#)의 Lake Formation에 구성된 값으로 값을 설정합니다.

## 5단계: 가상 클러스터 만들기

보안 구성을 사용하여 Amazon EMR on EKS 가상 클러스터를 생성합니다.

```
aws emr-containers create-virtual-cluster \
--name my-lf-enabled-vc \
--container-provider '{
  "id": "eks-cluster",
  "type": "EKS",
  "info": {
    "eksInfo": {
      "namespace": "user-namespace"
    }
  }
}' \
--security-configuration-id SecurityConfiguraionId
```

Lake Formation 권한 부여 구성이 가상 클러스터에서 실행되는 모든 작업에 적용되도록 이전 단계의 SecurityConfiguration ID가 전달되었는지 확인합니다. 자세한 내용은 [Amazon EMR에 Amazon EKS 클러스터 등록](#)을 참조하세요.

## 6단계: FGAC 지원 VirtualCluster에서 작업 제출

작업 제출 프로세스는 Lake Formation이 아닌 작업과 Lake Formation 작업 모두에서 동일합니다. 자세한 내용은 [StartJobRun을 사용한 작업 실행 제출](#)을 참조하세요.

시스템 드라이버의 Spark 드라이버, 실행기 및 이벤트 로그는 디버깅을 위해 AWS 서비스 계정의 S3 버킷에 저장됩니다. AWS 서비스 버킷에 저장된 모든 로그를 암호화하려면 작업 실행에서 고객 관리형 KMS 키를 구성하는 것이 좋습니다. 로그 암호화 활성화에 대한 자세한 내용은 [Amazon EMR on EKS 로그 암호화](#)를 참조하세요.

## 고려 사항 및 제한 사항

EKS에서 Amazon EMR과 함께 Lake Formation을 사용할 때는 다음 고려 사항과 제한 사항에 유의하세요.

- Amazon EMR on EKS는 Apache Hive, Apache Iceberg, Apache Hudi 및 Delta 테이블 형식에 대해서만 Lake Formation을 통한 세분화된 액세스 제어를 지원합니다. Apache Hive 형식으로는, Parquet, ORC 및 xSV가 포함됩니다.
- DynamicResourceAllocation은 기본적으로 활성화되어 있으며 사용자는 Lake Formation 작업의 DynamicResourceAllocation을 끌 수 없습니다. DRA `spark.dynamicAllocation.maxExecutors` 구성의 기본값은 무한이므로 워크로드에 따라 적절한 값을 구성하세요.
- 기본적으로 `spark.dynamicAllocation.preallocateExecutors`는 Amazon EMR Spark에서 활성화되므로 `spark.dynamicAllocation.initialExecutors` 및가 설정되지 않은 경우 과도한 컨테이너 이탈 `spark.dynamicAllocation.minExecutors`이 발생할 수 있습니다. 실행기 사전 할당을 관리하기 위한 권장 구성은의 [성능](#) 섹션을 참조하세요 [GitHub의 Amazon EMR on EKS 모범 사례 안내서에 대한 링크](#).
- Lake Formation 지원 작업은 시스템 드라이버 및 시스템 실행기에서 사용자 지정 EMR on EKS 이미지의 사용을 지원하지 않습니다.
- Spark 작업에서 Lake Formation만 사용할 수 있습니다.
- Lake Formation을 사용하는 EMR on EKS는 작업 전체에서 단일 Spark 세션만 지원합니다.
- Lake Formation을 사용하는 EMR on EKS는 리소스 링크를 통해 공유되는 교차 계정 테이블 쿼리만 지원합니다.
- 다음은 지원되지 않습니다.
  - 복원력 있는 분산 데이터세트(RDD)
  - Spark 스트리밍
  - Lake Formation에 부여된 권한으로 쓰기
  - 중첩된 열에 대한 액세스 제어
- EMR on EKS는 다음을 포함하여 시스템 드라이버의 완전한 격리를 저해할 수 있는 기능을 차단합니다.
  - UDT, HiveUDF 및 사용자 지정 클래스가 포함된 사용자 정의 함수
  - 사용자 지정 데이터 소스
  - Spark 확장, 커넥터 또는 메타스토어 ANALYZE TABLE 명령에 대한 추가 jar 제공
- 액세스 제어, EXPLAIN PLAN 및 DDL 작업(예: DESCRIBE TABLE)을 적용하려면 제한된 정보를 노출하지 않습니다.
- Amazon EMR on EKS는 Lake Formation 지원 작업의 시스템 드라이버 Spark 로그에 대한 액세스를 제한합니다. 시스템 드라이버는 더 많은 액세스 권한으로 실행되므로 시스템 드라이버가 생성하는 이벤트 및 로그에는 민감한 정보가 포함될 수 있습니다. 권한이 없는 사용자 또는 코드가 이 민감한

데이터에 액세스하지 못하도록 EMR on EKS는 시스템 드라이버 로그에 대한 액세스를 비활성화했습니다. 문제 해결은 AWS 지원팀에 문의하십시오.

- Lake Formation에 테이블 위치를 등록한 경우, 데이터 액세스 경로는 EMR on EKS 작업 실행 역할에 대한 IAM 권한에 관계없이 Lake Formation에 저장된 자격 증명을 통해 이루어집니다. 테이블 위치에 등록된 역할을 잘못 구성하면 테이블 위치에 대한 S3 IAM 권한이 있는 역할을 사용하는 제출된 작업이 실패합니다.
- Lake Formation 테이블에 쓰는 경우 Lake Formation에 부여된 권한이 아닌 IAM 권한을 사용합니다. 작업 실행 역할에 필요한 S3 권한이 있는 경우 이를 사용하여 쓰기 작업을 실행할 수 있습니다.

다음은 Apache Iceberg를 사용하는 경우 고려 사항 및 제한 사항입니다.

- Apache Iceberg는 세션 카탈로그에서만 사용할 수 있으며, 임의로 이름이 지정된 카탈로그에서는 사용할 수 없습니다.
- Lake Formation에 등록된 Iceberg 테이블은 메타데이터 테이블 history, metadata\_log\_entries, snapshots, files, manifests, refs만 지원합니다. Amazon EMR은 partitions, path, summaries와 같이 민감한 데이터를 포함할 수 있는 열을 숨깁니다. 이 제한 사항은 Lake Formation에 등록되지 않은 Iceberg 테이블에 적용되지 않습니다.
- Lake Formation에 등록하지 않은 테이블은 모든 Iceberg 저장 프로시저를 지원합니다. register\_table 및 migrate 절차는 어떤 테이블에서도 지원되지 않습니다.
- V1 대신 Iceberg DataFrameWriterV2를 사용하는 것이 좋습니다.

자세한 내용은 [Amazon EMR on EKS 개념 및 용어 이해](#) 및 [Amazon EMR on EKS에 대한 클러스터 액세스 활성화](#)를 참조하세요.

## 데이터 관리자에 대한 면책 조항

### Note

EMR on EKS의 IAM 역할에 Lake Formation 리소스에 대한 액세스 권한을 부여하는 경우 EMR 클러스터 관리자 또는 운영자가 신뢰할 수 있는 관리자인지 확인합니다. 이는 여러 조직 및 AWS 계정에서 공유되는 Lake Formation 리소스와 특히 관련이 있습니다.

## EKS 관리자의 책임

- System 네임스페이스는 보호되어야 합니다. 사용자, 리소스, 엔터티 또는 도구는 System 네임스페이스의 Kubernetes 리소스에 대한 Kubernetes RBAC 권한을 보유할 수 없습니다.
- EMR on EKS 서비스를 제외한 사용자, 리소스 또는 엔터티는 User 네임스페이스의 POD, CONFIG\_MAP 및 SECRET에 대한 CREATE 액세스에 액세스할 수 없습니다.
- System 드라이버 및 System 실행기에는 민감한 데이터가 포함됩니다. 따라서 System 네임스페이스의 Spark 이벤트, Spark 드라이버 로그 및 Spark 실행기 로그를 외부 로그 스토리지 시스템으로 전달해서는 안 됩니다.

## 문제 해결

### 로깅

EMR on EKS는 Spark 리소스 프로파일을 사용하여 작업 실행을 분할합니다. Amazon EMR on EKS는 사용자 프로파일을 사용하여 사용자가 제공한 코드를 실행하는 반면, 시스템 프로파일은 Lake Formation 정책을 적용합니다. 사용자 프로파일로 실행된 컨테이너의 로그에는 [MonitoringConfiguration](#)을 사용해 StartJobRun 요청을 구성하여 액세스할 수 있습니다.

### Spark 기록 서버

Spark 기록 서버에는 사용자 프로파일에서 생성된 모든 Spark 이벤트 및 시스템 드라이버에서 생성된 수정된 이벤트가 있습니다. 실행기 탭에서 사용자 및 시스템 드라이버의 모든 컨테이너를 볼 수 있습니다. 그러나 로그 링크는 사용자 프로파일에서만 사용할 수 있습니다.

### Lake Formation 권한이 부족하여 작업 실패

작업 실행 역할에 액세스 중인 테이블에서 SELECT 및 DESCRIBE를 실행할 수 있는 권한이 있는지 확인하세요.

### RDD 실행이 실패한 작업

EMR on EKS는 현재 Lake Formation 지원 작업에서 탄력적 분산 데이터세트(RDD) 작업을 지원하지 않습니다.

### Amazon S3의 데이터 파일에 액세스할 수 없음

Lake Formation에서 데이터 레이크의 위치를 등록했는지 확인합니다.

## 보안 검증 예외

EMR on EKS에서 보안 검증 오류를 탐지했습니다. AWS 지원 팀에 문의하여 도움을 받으세요.

## 계정 간에 AWS Glue 데이터 카탈로그 및 테이블 공유

여러 계정에서 데이터베이스와 테이블을 공유하고 Lake Formation을 계속 사용할 수 있습니다. 자세한 내용은 [Lake Formation의 교차 계정 데이터 공유](#) 및 [AWS Lake Formation을 사용하여 Glue 데이터 카탈로그와 테이블을 교차 계정과 공유하려면 어떻게 AWS 해야 하나요?](#)를 참조하세요.

## Iceberg 작업 발생 초기화 오류가 AWS 리전을 설정하지 않음

메시지는 다음과 같습니다.

```
25/02/25 13:33:19 ERROR SparkFGACEExceptionSanitizer: Client received error with id =
b921f9e6-f655-491f-b8bd-b2842cdc20c7,
reason = IllegalArgumentException, message = Cannot initialize
LakeFormationAwsClientFactory, please set client.region to a valid aws region
```

Spark 구성 `spark.sql.catalog.catalog_name.client.region`이 유효한 리전으로 설정되어 있는지 확인합니다.

## SparkUnsupportedOperationException이 발생하는 Iceberg 작업

메시지는 다음과 같습니다.

```
25/02/25 13:53:15 ERROR SparkFGACEExceptionSanitizer: Client received error with id =
921fef42-0800-448b-bef5-d283d1278ce0,
reason = SparkUnsupportedOperationException, message = Either glue.id or glue.account-
id is set with non-default account.
Cross account access with fine-grained access control is only supported with AWS
Resource Access Manager.
```

Spark 구성 `spark.sql.catalog.catalog_name.glue.account-id`이 유효한 계정 ID로 설정되어 있는지 확인합니다.

## MERGE 작업 중에 "403 액세스 거부됨" 오류가 발생하며 Iceberg 작업이 실패함

메시지는 다음과 같습니다.

```
software.amazon.awssdk.services.s3.model.S3Exception: Access Denied (Service: S3,
Status Code: 403,
```

```
...
at
software.amazon.awssdk.services.s3.DefaultS3Client.deleteObject(DefaultS3Client.java:3365)
at org.apache.iceberg.aws.s3.S3FileIO.deleteFile(S3FileIO.java:162)
at org.apache.iceberg.io.FileIO.deleteFile(FileIO.java:86)
at
org.apache.iceberg.io.RollingFileWriter.closeCurrentWriter(RollingFileWriter.java:129)
```

다음 속성을 추가하여 Spark에서 S3 삭제 작업을 비활성화합니다. `--conf spark.sql.catalog.s3-table-name.s3.delete-enabled=false`.

## 로깅 및 모니터링

인시던트를 탐지하고 인시던트가 발생했을 때 경고를 받고 그에 응답하려면 Amazon EMR on EKS에서 다음 옵션을 사용합니다.

- 를 사용하여 Amazon EMR on EKS 모니터링 AWS CloudTrail -는 Amazon EMR on EKS에서 사용자, 역할 또는 AWS 서비스가 수행한 작업에 대한 레코드를 [AWS CloudTrail](#) 제공합니다. 이 서비스는 Amazon EMR 콘솔에서 수행한 직접 호출과 Amazon EMR on EKS API 작업에 대한 코드 직접 호출을 이벤트로 캡처합니다. 이를 통해 Amazon EMR on EKS에서 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다. 자세한 내용은 [를 사용하여 Amazon EMR on EKS API 호출 로깅 AWS CloudTrail](#) 단원을 참조하십시오.
- Amazon EMR on EKS에서 CloudWatch Events 사용 - CloudWatch Events는 AWS 리소스의 변경 사항을 설명하는 시스템 이벤트의 스트림을 거의 실시간으로 제공합니다. 또한 CloudWatch Events는 이러한 운영 변경 발생 시 이를 인지하고 응답하며, 환경에 응답하기 위한 메시지를 전송하고, 함수를 활성화하고, 변경을 수행하고, 상태 정보를 기록하는 등 필요에 따라 교정 조치를 취합니다. Amazon EMR on EKS에서 CloudWatch Events를 사용하려면 CloudTrail을 통해 Amazon EMR on EKS API 직접 호출 시 트리거되는 규칙을 생성합니다. 자세한 내용은 [Amazon CloudWatch Events를 사용하여 작업 모니터링](#) 단원을 참조하십시오.

## 관리형 스토리지를 사용하여 Amazon EMR on EKS 로그 암호화

다음 섹션에서는 로그에 대한 암호화를 구성하는 방법을 보여줍니다.

### 암호화 활성화

자체 KMS 키를 사용하여 관리형 스토리지에서 로그를 암호화하려면 작업 실행을 제출할 때 다음 구성을 사용합니다.

```

"monitoringConfiguration": {
    "managedLogs": {
        "allowAWSToRetainLogs": "ENABLED",
        "encryptionKeyArn": "KMS key arn"
    },
    "persistentAppUI": "ENABLED"
}

```

allowAWSToRetainLogs 구성을 사용하면 AWS가 네이티브 FGAC를 사용하여 작업을 실행할 때 시스템 네임스페이스 로그를 유지할 수 있습니다. persistentAppUI 구성을 통해 Spark UI AWS를 생성하는 데 사용되는 이벤트 로그를 저장할 수 있습니다. encryptionKeyArn은 AWS에 저장된 로그를 암호화하는 데 사용할 KMS 키 ARN을 지정하는 데 사용됩니다.

## 로그 암호화에 필요한 권한

작업을 제출하거나 Spark UI를 보는 사용자는 암호화 키에 대해 kms:DescribeKey, kms:GenerateDataKey 및 kms:Decrypt 작업을 허용해야 합니다. 이러한 권한은 키의 유효성을 확인하고 사용자에게 KMS 키로 암호화된 로그를 읽고 쓰는 데 필요한 권한이 있는지 확인하기 위한 용도로 사용됩니다. 작업을 제출하는 사용자에게 필요한 키 권한이 없는 경우 Amazon EMR on EKS는 작업 실행 제출을 거부합니다.

StartJobRun을 호출하는 데 사용되는 역할에 대한 IAM 정책 예제

## JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "emr-containers:StartJobRun"
      ],
      "Resource": [
        "*"
      ],
      "Effect": "Allow",
      "Sid": "AllowEMRCONTAINERSStartjobrun"
    },
    {
      "Action": [
        "kms:DescribeKey",

```



```

        "kms:Decrypt",
        "kms:GenerateDataKey"
    ],
    "Resource": [
        "arn:aws:kms:*:*:key/key-id"
    ],
    "Effect": "Allow",
    "Sid": "AllowKMSDescribekey"
}
]
}

```

또한 `persistentappui.elasticmapreduce.amazonaws.com` 및 `elasticmapreduce.amazonaws.com` 서비스 보안 주체가 `kms:GenerateDataKey` 및 `kms:Decrypt`를 허용하도록 KMS 키를 구성해야 합니다. 이 기능을 통해 EMR이 KMS 키로 암호화된 로그를 읽고 관리형 스토리지에 쓸 수 있습니다.

## KMS 키 정책 예제

### JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:DescribeKey"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringLike": {
          "kms:viaService": "emr-containers.*.amazonaws.com"
        }
      },
      "Sid": "AllowKMSDescribekey"
    },
    {
      "Effect": "Allow",

```

```

    "Action": [
      "kms:Decrypt",
      "kms:GenerateDataKey"
    ],
    "Resource": [
      "*"
    ],
    "Condition": {
      "StringLike": {
        "kms:viaService": "emr-containers.*.amazonaws.com",
        "kms:EncryptionContext:aws:emr-containers:virtualClusterId": "virtual
cluster id"
      }
    },
    "Sid": "AllowKMSDecryptGenerate"
  },
  {
    "Effect": "Allow",
    "Action": [
      "kms:Decrypt",
      "kms:GenerateDataKey"
    ],
    "Resource": [
      "*"
    ],
    "Condition": {
      "StringLike": {
        "kms:EncryptionContext:aws:emr-containers:virtualClusterId": "virtual
cluster id"
      },
      "ArnLike": {
        "aws:SourceArn": "arn:aws:emr-containers:*:*:/
virtualclusters/virtual_cluster_id"
      }
    },
    "Sid": "AllowKMSDecryptService"
  }
]
}

```

보안을 위해 `kms:viaService`, `kms:EncryptionContext` 및 `aws:SourceArn` 조건을 추가하는 것이 좋습니다. 이러한 조건은 키가 Amazon EMR on EKS에서만 사용되며 특정 가상 클러스터에서 실행되는 작업에서 생성된 로그에만 사용되도록 합니다.

## 를 사용하여 Amazon EMR on EKS API 호출 로그 AWS CloudTrail

Amazon EMR on EKS는 Amazon EMR on EKS에서 사용자, 역할 또는 AWS CloudTrail서비스가 수행한 작업에 대한 레코드를 제공하는 AWS 서비스와 통합됩니다. CloudTrail은 Amazon EMR on EKS에 대한 모든 API 직접 호출을 이벤트로 캡처합니다. 캡처되는 직접 호출에는 Amazon EMR on EKS 콘솔로부터의 직접 호출과 Amazon EMR on EKS API 작업에 대한 코드 직접 호출이 포함됩니다. 추적을 생성하면 Amazon EMR on EKS 이벤트를 포함한 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 전송할 수 있습니다. 추적을 구성하지 않은 경우에도 이벤트 기록에서 CloudTrail 콘솔의 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 Amazon EMR on EKS에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 내용은 [AWS CloudTrail 사용 설명서](#)를 참조하세요.

## CloudTrail에서 Amazon EMR on EKS 정보

AWS 계정을 생성할 때 계정에서 CloudTrail이 활성화됩니다. Amazon EMR on EKS에서 활동이 발생하면 해당 활동이 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최근 이벤트를 보고 검색하고 다운로드할 수 있습니다. 자세한 내용은 [CloudTrail 이벤트 기록을 사용하여 이벤트 보기](#)를 참조하세요.

Amazon EMR on EKS 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 기본적으로 콘솔에서 추적을 생성하면 추적이 모든 AWS 리전에 적용됩니다. 추적은 AWS 파티션의 모든 리전에서 이벤트를 로깅하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 조치를 취하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- [추적 생성 개요](#)
- [CloudTrail 지원 서비스 및 통합](#)
- [CloudTrail에 대한 Amazon SNS 알림 구성](#)
- [여러 리전에서 CloudTrail 로그 파일 받기](#) 및 [여러 계정에서 CloudTrail 로그 파일 받기](#)

모든 Amazon EMR on EKS 작업은 CloudTrail에 의해 로깅되며 [Amazon EMR on EKS API 참조](#)에 설명되어 있습니다. 예를 들어 `CreateVirtualCluster`, `StartJobRun`, `ListJobRuns` 작업을 직접 호출하면 CloudTrail 로그 파일에 항목이 생성됩니다.

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 관한 정보가 포함됩니다. ID 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청이 루트 또는 AWS Identity and Access Management (IAM) 사용자 자격 증명으로 이루어졌는지 여부입니다.
- 역할 또는 페더레이션 사용자의 임시 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 요청이 다른 AWS 서비스에서 이루어졌는지 여부입니다.

자세한 내용은 [CloudTrail user Identity element](#)를 참조하세요.

## Amazon EMR on EKS 로그 파일 항목 이해

트레일이란 지정한 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 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출의 주문 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음은 [ListJobRuns](#) 작업을 보여주는 CloudTrail 로그 항목이 나타낸 예시입니다.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:admin",
    "arn": "arn:aws:sts::012345678910:assumed-role/Admin/admin",
    "accountId": "012345678910",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::012345678910:role/Admin",
        "accountId": "012345678910",
        "userName": "Admin"
      },
      "webIdFederationData": {},
      "attributes": {
```

```

        "mfaAuthenticated": "false",
        "creationDate": "2020-11-04T21:49:36Z"
    }
},
"eventTime": "2020-11-04T21:52:58Z",
"eventSource": "emr-containers.amazonaws.com",
"eventName": "ListJobRuns",
"awsRegion": "us-east-1",
"sourceIPAddress": "203.0.113.1",
"userAgent": "aws-cli/1.11.167 Python/2.7.10 Darwin/16.7.0 botocore/1.7.25",
"requestParameters": {
    "virtualClusterId": "1K48XXXXXXHCB"
},
"responseElements": null,
"requestID": "890b8639-e51f-11e7-b038-EXAMPLE",
"eventID": "874f89fa-70fc-4798-bc00-EXAMPLE",
"readOnly": true,
"eventType": "AwsApiCall",
"recipientAccountId": "012345678910"
}

```

## Amazon EMR on EKS로 Amazon S3 Access Grants 사용

### Amazon EMR on EKS에 대한 S3 Access Grants 개요

Amazon EMR 릴리스 6.15.0 이상을 사용하면 Amazon S3 Access Grants에서 Amazon EMR on EKS의 Amazon S3 데이터에 대한 액세스를 강화하기 위해 사용 가능한 확장 가능한 액세스 제어 솔루션이 제공됩니다. S3 데이터에 대한 권한 구성이 복잡하거나 대규모인 경우 Access Grants를 사용하여 사용자, 역할 및 애플리케이션을 위한 S3 데이터 권한을 확장할 수 있습니다.

S3 Access Grants를 사용하여 Amazon S3 데이터에 대한 액세스를 Amazon EMR on EKS 클러스터에 액세스할 수 있는 자격 증명에 연결된 런타임 역할 또는 IAM 역할에 의해 부여된 권한 이상으로 강화할 수 있습니다.

자세한 내용은 Amazon EMR 관리 가이드의 [Amazon EMR을 위한 S3 Access Grants를 사용한 액세스 관리](#) 및 Amazon Simple Storage Service 사용 설명서의 [S3 Access Grants를 사용한 액세스 관리](#)를 참조하세요.

이 페이지에서는 S3 Access Grants 통합을 사용하여 Amazon EMR on EKS에서 Spark 작업을 실행하는 데 필요한 요건을 설명합니다. Amazon EMR on EKS에서 S3 Access Grants를 사용하려면 작업에

대한 실행 역할의 추가 IAM 정책 설명과 StartJobRun API에 대한 추가 재정의의 구성이 필요합니다. 기타 Amazon EMR 배포를 이용하여 S3 Access Grants을 설정하는 단계를 확인하려면 다음 설명서를 참조하세요.

- [Amazon EMR을 통해 S3 Access Grants 사용](#)
- [EMR Serverless를 통해 S3 Access Grants 사용](#)

## 데이터 관리를 위해 S3 Access Grants를 사용하여 Amazon EMR on EKS 클러스터 시작

Amazon EMR on EKS에서 S3 Access Grants를 활성화하여 Spark 작업을 시작할 수 있습니다. 애플리케이션에서 S3 데이터에 대한 요청이 발생할 경우 Amazon S3에서는 특정 버킷, 접두사 또는 객체로 범위가 지정된 임시 보안 인증을 제공합니다.

1. Amazon EMR on EKS 클러스터에 대한 작업 실행 역할을 설정합니다. Spark 작업 `s3:GetDataAccess` 및 `s3:GetAccessGrantsInstanceForPrefix`를 실행하는 데 필요한 필수 IAM 권한을 포함합니다.

```
{
  "Effect": "Allow",
  "Action": [
    "s3:GetDataAccess",
    "s3:GetAccessGrantsInstanceForPrefix"
  ],
  "Resource": [
    //LIST ALL INSTANCE ARNS THAT THE ROLE IS ALLOWED TO QUERY
    "arn:aws_partition:s3:Region:account-id1:access-grants/default",
    "arn:aws_partition:s3:Region:account-id2:access-grants/default"
  ]
}
```

### Note

S3에 직접 액세스할 수 있는 추가 권한을 지닌 작업 실행을 위한 IAM 역할을 지정하면 사용자가 S3 Access Grants에서 정의한 권한에 상관없이 데이터에 액세스하는 것이 가능할 수 있습니다.

- 다음 예시에 나타난 것처럼 Amazon EMR 릴리스 라벨이 6.15 이상이고 `emrfs-site` 분류가 있는 Amazon EMR on EKS 클러스터에 작업을 제출하세요. *red text*의 값을 사용 시나리오에 적합한 값으로 바꾸세요.

```
{
  "name": "myjob",
  "virtualClusterId": "123456",
  "executionRoleArn": "iam_role_name_for_job_execution",
  "releaseLabel": "emr-7.13.0-latest",
  "jobDriver": {
    "sparkSubmitJobDriver": {
      "entryPoint": "entryPoint_location",
      "entryPointArguments": ["argument1", "argument2"],
      "sparkSubmitParameters": "--class main_class"
    }
  },
  "configurationOverrides": {
    "applicationConfiguration": [
      {
        "classification": "emrfs-site",
        "properties": {
          "fs.s3.s3AccessGrants.enabled": "true",
          "fs.s3.s3AccessGrants.fallbackToIAM": "false"
        }
      }
    ]
  }
}
```

## Amazon EMR on EKS에서의 S3 Access Grants 고려 사항

Amazon EMR on EKS에서 Amazon S3 Access Grants를 사용하는 경우의 중요 지원, 호환성 및 동작 정보를 확인하려면 Amazon EMR 관리 가이드의 [Amazon EMR에서의 S3 Access Grants 고려 사항](#)을 참조하세요.

## Amazon EMR on EKS에 대한 규정 준수 확인

타사 감사자는 여러 규정 준수 프로그램의 일환으로 Amazon EMR on EKS의 보안 및 AWS 규정 준수를 평가합니다. 여기에는 SOC, PCI, FedRAMP, HIPAA 등이 포함됩니다.

## Amazon EMR on EKS의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전은 물리적으로 분리되고 격리된 여러 가용 영역을 제공하며, 이 가용 영역은 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹과 연결됩니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 [AWS 글로벌 인프라](#)를 참조하세요.

AWS 글로벌 인프라 외에도 Amazon EMR on EKS는 EMRFS를 통해 Amazon S3와의 통합을 제공하여 데이터 복원성과 백업 요구 사항을 지원합니다.

## Amazon EMR on EKS의 인프라 보안

관리형 서비스인 Amazon EMR은 AWS 글로벌 네트워크 보안으로 보호됩니다. AWS 보안 서비스 및 가용 인프라를 AWS 보호하는 방법에 대한 자세한 내용은 [AWS 클라우드 보안을](#) 참조하세요. 인프라 보안 모범 사례를 사용하여 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 [인프라 보호](#)를 참조하세요 AWS .

AWS 에서 게시한 API 호출을 사용하여 네트워크를 통해 Amazon EMR에 액세스합니다. 클라이언트는 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.

## 구성 및 취약성 분석

AWS 는 게스트 운영 체제(OS) 및 데이터베이스 패치, 방화벽 구성, 재해 복구와 같은 기본 보안 작업을 처리합니다. 적합한 제3자가 이 절차를 검토하고 인증하였습니다. 자세한 내용은 다음 리소스를 참조하세요.

- [Amazon EMR on EKS에 대한 규정 준수 확인](#)
- [공동 책임 모델](#)
- [Amazon Web Services: 보안 프로세스의 개요](#)(백서)



# 인터페이스 VPC 엔드포인트를 사용하여 Amazon EMR on EKS에 연결

인터넷을 통해 연결하는 대신 Virtual Private Cloud(VPC)의 [인터페이스 VPC 엔드포인트\(AWS PrivateLink\)](#)를 통해 Amazon EMR on EKS에 직접 연결할 수 있습니다. 인터페이스 VPC 엔드포인트를 사용하는 경우 VPC와 Amazon EMR on EKS 간의 통신은 전적으로 AWS 네트워크 내에서 수행됩니다. 각 VPC 엔드포인트는 하나 이상의 [탄력적 네트워크 인터페이스\(ENI\)](#) 및 VPC 서브넷의 프라이빗 IP 주소로 표현됩니다.

인터페이스 VPC 엔드포인트는 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결 없이 VPC를 Amazon EMR on EKS에 직접 연결합니다. VPC에 있는 인스턴스는 퍼블릭 IP 주소가 없어도 Amazon EMR on EKS API와 통신할 수 있습니다.

AWS Management Console 또는 AWS Command Line Interface (AWS CLI) 명령을 사용하여 인터페이스 VPC 엔드포인트를 생성하여 Amazon EMR on EKS에 연결할 수 있습니다. 자세한 내용은 [인터페이스 엔드포인트 생성](#)을 참조하세요.

인터페이스 VPC 엔드포인트를 생성한 후 엔드포인트에 대해 프라이빗 DNS 호스트 이름을 활성화하는 경우 기본 Amazon EMR on EKS 엔드포인트는 VPC 엔드포인트로 확인됩니다. Amazon EMR on EKS에 대한 기본 서비스 이름 엔드포인트의 형식은 다음과 같습니다.

```
emr-containers.Region.amazonaws.com
```

프라이빗 DNS 호스트 이름을 활성화하지 않은 경우, Amazon VPC는 다음 형식으로 사용할 수 있는 DNS 엔드포인트를 제공합니다.

```
VPC_Endpoint_ID.emr-containers.Region.vpce.amazonaws.com
```

자세한 내용은 Amazon [VPC 사용 설명서의 인터페이스 VPC 엔드포인트\(AWS PrivateLink\)](#)를 참조하세요. Amazon EMR on EKS는 VPC 내부에 있는 모든 [API 작업](#)에 대한 직접 호출을 지원합니다.

VPC 엔드포인트 정책을 VPC 엔드포인트에 연결하여 IAM 보안 주체에 대한 액세스를 제어할 수 있습니다. 보안 그룹을 VPC 엔드포인트와 연결하여 IP 주소 범위와 같은 네트워크 트래픽의 소스와 대상을 기반으로 인바운드 및 아웃바운드 액세스를 제어할 수도 있습니다. 자세한 정보는 [VPC 종단점을 통해 서비스에 대한 액세스 제어](#)를 참조하십시오.

## Amazon EMR on EKS에 대한 VPC 엔드포인트 정책 생성

Amazon EMR on EKS에 대한 Amazon VPC 엔드포인트 정책을 생성하여 다음을 지정할 수 있습니다.

- 작업을 수행할 수 있거나 수행할 수 없는 보안 주체
- 수행할 수 있는 작업
- 작업을 수행할 수 있는 리소스

자세한 내용은 Amazon VPC 사용 설명서에서 [VPC 엔드포인트를 사용하여 서비스에 대한 액세스 제](#)  
[어](#)를 참조하세요.

Example 지정된 AWS 계정의 모든 액세스를 거부하는 VPC 엔드포인트 정책

다음 VPC 엔드포인트 정책은 엔드포인트를 사용하여 리소스에 대한 모든 액세스를 AWS 계정  
**123456789012**에 거부합니다.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "*",
      "Principal": {
        "AWS": [
          "123456789012"
        ]
      }
    }
  ]
}
```

Example 지정된 IAM 보안 주체(사용자)에 대해서만 VPC 액세스를 허용하는 VPC 엔드포인트 정책

다음 VPC 엔드포인트 정책은 AWS 계정 **123456789012**의 IAM 사용자 **###**에 대한 전체 액세스만 허용합니다. 다른 모든 IAM 보안 주체는 엔드포인트를 사용하는 액세스가 거부됩니다.

```
{
  "Statement": [
```

```
{
  "Action": "*",
  "Effect": "Allow",
  "Resource": "*",
  "Principal": {
    "AWS": [
      "arn:aws:iam::123456789012:user/lijuan"
    ]
  }
}
```

Example읽기 전용 Amazon EMR on EKS 작업을 허용하는 VPC 엔드포인트 정책

다음 VPC 엔드포인트 정책은 AWS 계정 **123456789012**만 지정된 Amazon EMR on EKS 작업을 수행하도록 허용합니다.

지정된 작업은 Amazon EMR on EKS에 대한 읽기 전용 액세스 권한을 제공합니다. VPC의 다른 모든 작업은 지정된 계정에 대해 거부됩니다. 다른 모든 계정의 액세스는 거부됩니다. Amazon EMR on EKS 작업 목록은 [Amazon EMR on EKS에 대한 작업, 리소스 및 조건 키](#)를 참조하세요.

```
{
  "Statement": [
    {
      "Action": [
        "emr-containers:DescribeJobRun",
        "emr-containers:DescribeVirtualCluster",
        "emr-containers:ListJobRuns",
        "emr-containers:ListTagsForResource",
        "emr-containers:ListVirtualClusters"
      ],
      "Effect": "Allow",
      "Resource": "*",
      "Principal": {
        "AWS": [
          "123456789012"
        ]
      }
    }
  ]
}
```

## Example 지정된 가상 클러스터에 대한 액세스를 거부하는 VPC 엔드포인트 정책

다음 VPC 엔드포인트 정책은 모든 계정 및 보안 주체에 대한 전체 액세스를 허용하지만 클러스터 ID **A1B2CD34EF5G**를 사용하여 가상 클러스터에서 수행된 작업에 대한 AWS 계정 **123456789012**의 액세스를 거부합니다. 가상 클러스터에 대한 리소스 수준 권한을 지원하지 않는 다른 Amazon EMR on EKS 작업은 여전히 허용됩니다. Amazon EMR on EKS 작업 및 해당 리소스 유형 목록은 AWS Identity and Access Management 사용 설명서에서 [Amazon EMR on EKS에 대한 작업, 리소스 및 조건 키](#)를 참조하세요.

```
{
  "Statement": [
    {
      "Action": "*",
      "Effect": "Allow",
      "Resource": "*",
      "Principal": "*"
    },
    {
      "Action": "*",
      "Effect": "Deny",
      "Resource": "arn:aws:emr-containers:us-west-2:123456789012:/virtualclusters/A1B2CD34EF5G",
      "Principal": {
        "AWS": [
          "123456789012"
        ]
      }
    }
  ]
}
```

## Amazon EMR on EKS에 대한 크로스 계정 액세스 설정

Amazon EMR on EKS에 대한 크로스 계정 액세스를 설정할 수 있습니다. 교차 계정 액세스를 사용하면 한 AWS 계정의 사용자가 Amazon EMR on EKS 작업을 실행하고 다른 AWS 계정에 속한 기본 데이터에 액세스할 수 있습니다.

### 사전 조건

Amazon EMR on EKS에 대한 교차 계정 액세스를 설정하려면 다음 AWS 계정에 로그인한 상태에서 작업을 완료합니다.

- AccountA - Amazon EMR을 EKS 클러스터의 네임스페이스에 등록하여 Amazon EMR on EKS 가상 클러스터를 생성한 AWS 계정입니다.
- AccountB - Amazon EMR on EKS 작업이 액세스하도록 하려는 Amazon S3 버킷 또는 DynamoDB 테이블이 포함된 AWS 계정입니다.

교차 AWS 계정 액세스를 설정하기 전에 계정에서 다음을 준비해야 합니다.

- 작업을 실행하려는 AccountA의 Amazon EMR on EKS 가상 클러스터.
- 가상 클러스터에서 작업을 실행하는 데 필요한 권한이 있는 AccountA의 작업 실행 역할. 자세한 내용은 [작업 실행 역할 생성](#) 및 [Amazon EMR on EKS에서 작업 실행 역할 사용](#) 섹션을 참조하세요.

## 크로스 계정 Amazon S3 버킷 또는 DynamoDB 테이블에 액세스하는 방법

Amazon EMR on EKS에서 크로스 계정 액세스를 설정하려면 다음 단계를 수행합니다.

1. AccountB에서 Amazon S3 버킷(cross-account-bucket)을 생성합니다. 자세한 내용은 [버킷 생성](#) 섹션을 참조하세요. DynamoDB에 대한 크로스 계정 액세스를 원하는 경우 AccountB에서 DynamoDB 테이블을 생성할 수도 있습니다. 자세한 내용은 [DynamoDB 테이블 생성](#)을 참조하세요.
2. AccountB에서 cross-account-bucket에 액세스할 수 있는 Cross-Account-Role-B IAM 역할을 생성합니다.
  1. IAM 콘솔에 로그인합니다.
  2. 역할을 선택하고 새 역할(Cross-Account-Role-B)을 생성합니다. IAM 역할 생성에 대한 자세한 내용은 IAM 사용 설명서에서 [IAM 역할 생성](#)을 참조하세요.
  3. 다음 정책 명령문에서 볼 수 있듯이 cross-account-bucket S3 버킷에 액세스할 수 있는 Cross-Account-Role-B에 대한 권한을 지정하는 IAM 정책을 생성합니다. 그런 다음, IAM 정책을 Cross-Account-Role-B에 연결합니다. 자세한 내용은 IAM 사용 설명서에서 [새 정책 생성](#)을 참조하세요.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```

    "Action": [
      "s3:*"
    ],
    "Resource": [
      "arn:aws:s3:::cross-account-bucket",
      "arn:aws:s3:::cross-account-bucket/*"
    ],
    "Sid": "AllowS3"
  }
]
}

```

DynamoDB 액세스가 필요한 경우 크로스 계정 DynamoDB 테이블에 액세스할 권한을 지정하는 IAM 정책을 생성합니다. 그런 다음, IAM 정책을 Cross-Account-Role-B에 연결합니다. 자세한 내용은 IAM 사용 설명서에서 [DynamoDB 테이블 생성](#)을 참조하세요.

다음은 DynamoDB 테이블(CrossAccountTable)에 액세스하기 위한 정책입니다.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:*"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:*:table/CrossAccountTable"
      ],
      "Sid": "AllowDYNAMODB"
    }
  ]
}

```

### 3. Cross-Account-Role-B 역할에 대한 신뢰 관계를 편집합니다.

1. 역할에 대한 신뢰 관계를 구성하려면 2단계에서 생성한 역할(Cross-Account-Role-B)에 대해 IAM 콘솔에서 신뢰 관계 탭을 선택합니다.
2. 신뢰 관계 편집을 선택합니다.

3. 다음 정책 문서를 추가합니다. 그러면 AccountA의 Job-Execution-Role-A에서 이 Cross-Account-Role-B 역할을 수임할 수 있습니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSTSAssumerole",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/Job-Execution-Role-A"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

4. STS 역할 수임을 통해 AccountA의 Job-Execution-Role-A에 Cross-Account-Role-B를 수임할 권한을 부여합니다.

1. AWS 계정의 IAM 콘솔에서 AccountA를 선택합니다. Job-Execution-Role-A.
2. 다음 정책 명령을 Job-Execution-Role-A에 추가하여 Cross-Account-Role-B 역할에서 AssumeRole 작업을 허용합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": [
        "arn:aws:iam::*:role/Cross-Account-Role-B"
      ],
      "Sid": "AllowSTSAssumerole"
    }
  ]
}
```

}

5. Amazon S3 액세스의 경우 Amazon EMR on EKS에 작업을 제출하는 동안 다음 spark-submit 파라미터(spark conf)를 설정합니다.

 Note

기본적으로 EMRFS는 작업 실행 역할을 사용하여 작업에서 S3 버킷에 액세스합니다. 그러나 customAWSCredentialsProvider가 AssumeRoleAWSCredentialsProvider로 설정된 경우 EMRFS는 Amazon S3 액세스에 대해 Job-Execution-Role-A 대신, 사용자가 ASSUME\_ROLE\_CREDENTIALS\_ROLE\_ARN에서 지정하는 해당 역할을 사용합니다.

- --conf  
spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRoleAWSCredentialsProvider
- --conf  
spark.kubernetes.driverEnv.ASSUME\_ROLE\_CREDENTIALS\_ROLE\_ARN=arn:aws:iam::**AccountA**:role/**Cross-Account-Role-B** \
- --conf  
spark.executorEnv.ASSUME\_ROLE\_CREDENTIALS\_ROLE\_ARN=arn:aws:iam::**AccountB**:role/**Cross-Account-Role-B** \

 Note

작업 spark 구성에서 실행기 및 드라이버 env 모두에 대해 ASSUME\_ROLE\_CREDENTIALS\_ROLE\_ARN을 설정해야 합니다.

DynamoDB 크로스 계정 액세스에 대해 --conf  
spark.dynamodb.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRoleAWSCredentialsProvider로 설정해야 합니다.

6. 다음 예제에서 볼 수 있듯이 크로스 계정 액세스를 사용하여 Amazon EMR on EKS 작업을 실행합니다.

```
aws emr-containers start-job-run \
--virtual-cluster-id 123456 \
```



```
--name myjob \
--execution-role-arn execution-role-arn \
--release-label emr-6.2.0-latest \
--job-driver '{"sparkSubmitJobDriver": {"entryPoint": "entryPoint_location",
"entryPointArguments": ["arguments_list"], "sparkSubmitParameters": "--class
<main_class> --conf spark.executor.instances=2 --conf spark.executor.memory=2G
--conf spark.executor.cores=2 --conf spark.driver.cores=1 --conf
spark.hadoop.fs.s3.customAWSCredentialsProvider=com.amazonaws.emr.AssumeRoleAWSCredentials
--conf
spark.kubernetes.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::AccountB:role/
Cross-Account-Role-B --conf
spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN=arn:aws:iam::AccountB:role/
Cross-Account-Role-B"}} ' \
--configuration-overrides '{"applicationConfiguration": [{"classification":
"spark-defaults", "properties": {"spark.driver.memory": "2G"}}],
"monitoringConfiguration": {"cloudWatchMonitoringConfiguration":
{"logGroupName": "log_group_name", "logStreamNamePrefix": "log_stream_prefix"},
"persistentAppUI": "ENABLED", "s3MonitoringConfiguration": {"logUri": "s3://
my_s3_log_location" }}}
```

# Amazon EMR on EKS 리소스 태그 지정

Amazon EMR on EKS 리소스 관리를 지원하려면 태그를 사용하여 각 리소스에 고유한 메타데이터를 할당할 수 있습니다. 이 주제에서는 태그 함수에 대한 개요를 제공하고 태그를 생성하는 방법을 보여줍니다.

## 주제

- [태그 기본 사항](#)
- [리소스에 태그 지정](#)
- [태그 제한](#)
- [AWS CLI 및 Amazon EMR on EKS API를 사용하여 태그 작업](#)

## 태그 기본 사항

태그는 AWS 리소스에 할당하는 레이블입니다. 각 태그는 사용자가 정의하는 키와 선택적 값으로 구성됩니다.

태그를 사용하면 용도, 소유자 또는 환경과 같은 속성별로 AWS 리소스를 분류할 수 있습니다. 동일한 유형의 리소스가 많은 경우 할당한 태그에 따라 특정 리소스를 빠르게 식별할 수 있습니다. 예를 들어, Amazon EMR on EKS 클러스터에 태그 세트를 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.

태그가 리소스에 자동으로 할당되는 것은 아닙니다. 태그를 추가한 후에는 언제든지 태그 키와 값을 편집하거나 리소스에서 태그를 제거할 수 있습니다. 리소스를 삭제하면 리소스에 대한 모든 태그도 삭제됩니다.

태그는 Amazon EMR on EKS에는 아무런 의미가 없으며 엄격하게 문자열로 해석됩니다.

태그 값에 빈 문자열은 지정할 수 있지만, null은 지정할 수 없습니다. 태그 키는 빈 문자열일 수 없습니다. 해당 리소스에 대해 키가 기존 태그와 동일한 태그를 추가하는 경우 새 값이 이전 값을 덮어씁니다.

AWS Identity and Access Management (IAM)를 사용하는 경우 태그 관리 권한이 있는 AWS 계정의 사용자를 제어할 수 있습니다.

태그 기반 액세스 제어 정책 예제는 [태그 기반 액세스 제어를 위한 정책](#) 섹션을 참조하세요.

## 리소스에 태그 지정

활성 상태인 새로운 가상 클러스터 또는 기존 가상 클러스터와 작업 실행에 태그를 지정할 수 있습니다. 작업 실행의 활성 상태로, PENDING, SUBMITTED, RUNNING, CANCEL\_PENDING이 있습니다. 가상 클러스터의 활성 상태로, RUNNING, TERMINATING, ARRESTED가 있습니다. 자세한 내용은 [작업 실행 상태](#) 및 [가상 클러스터 상태](#) 섹션을 참조하세요.

가상 클러스터가 종료되면 태그가 정리되고 더 이상 태그에 액세스할 수 없습니다.

Amazon EMR on EKS API AWS CLI, 또는 AWS SDK를 사용하는 경우 관련 API 작업의 태그 파라미터를 사용하여 새 리소스에 태그를 적용할 수 있습니다. TagResource API 작업을 사용하여 기존 리소스에 태그를 적용할 수도 있습니다.

일부 리소스 생성 작업을 사용해서 리소스 생성 시 리소스의 태그를 지정할 수 있습니다. 이 경우 리소스를 생성하는 동안 태그를 적용할 수 없는 경우 리소스를 생성하지 못합니다. 이 매커니즘에서는 생성 중에 태그를 지정하려는 리소스가 지정된 태그와 함께 생성되거나 전혀 생성되지 않습니다. 생성 시 리소스에 태그를 지정하면 리소스 생성 후 사용자 지정 태깅 스크립트를 실행할 필요가 없습니다.

다음 테이블에서는 태그를 지정할 수 있는 Amazon EMR on EKS 리소스를 설명합니다.

| Resource | 태그 지원 | 태그 전달 지원                                                | 생성 시 태그 지정 지원(Amazon EMR on EKS API AWS CLI 및 AWS SDK) | 생성을 위한 API(생성 중에 태그를 추가할 수 있음) |
|----------|-------|---------------------------------------------------------|--------------------------------------------------------|--------------------------------|
| 가상 클러스터  | 예     | 아니요. 가상 클러스터와 관련된 태그는 해당 가상 클러스터에 제출된 작업 실행에 전파되지 않습니다. | 예                                                      | CreateVirtualCluster           |
| 작업 실행    | 예     | 아니요                                                     | 예                                                      | StartJobRun                    |

## 태그 제한

태그에 적용되는 기본 제한은 다음과 같습니다.

- 리소스당 최대 태그 수 - 50개
- 각 리소스에 대해 각 태그 키는 고유하며 하나의 값만 가질 수 있습니다.
- 최대 키 길이 - UTF-8 형식의 유니코드 문자 128자
- 최대 값 길이 - UTF-8 형식의 유니코드 문자 256자
- 여러 AWS 서비스 및 리소스에서 태그 지정 스키마를 사용하는 경우 다른 서비스에서는 허용되는 문자에 제한이 있을 수 있습니다. 일반적으로 허용되는 문자는 UTF-8로 표시할 수 있는 문자, 숫자 및 공백과 특수 문자 + - = . \_ : / @입니다.
- 태그 키와 값은 대소문자를 구분합니다.
- 태그 값에 빈 문자열은 지정할 수 있지만, null은 지정할 수 없습니다. 태그 키는 빈 문자열일 수 없습니다.
- 키 또는 값에 aws:, AWS: 또는 이러한 접두사의 대문자 또는 소문자 조합을 사용하지 않습니다. 이는 AWS 전용입니다.

## AWS CLI 및 Amazon EMR on EKS API를 사용하여 태그 작업

다음 AWS CLI 명령 또는 Amazon EMR on EKS API 작업을 사용하여 리소스에 대한 태그를 추가, 업데이트, 나열 및 삭제합니다.

| Task                    | AWS CLI                                | API 작업                              |
|-------------------------|----------------------------------------|-------------------------------------|
| 하나 이상의 태그를 추가하거나 덮어씁니다. | <a href="#">tag-resource</a>           | <a href="#">TagResource</a>         |
| 리소스에 대한 태그 나열           | <a href="#">list-tags-for-resource</a> | <a href="#">ListTagsForResource</a> |
| 하나 이상의 태그를 삭제합니다.       | <a href="#">untag-resource</a>         | <a href="#">UntagResource</a>       |

다음 예제는 AWS CLI를 사용하여 리소스에 태그를 지정하거나 태그를 제거하는 방법을 보여줍니다.

### 예제 1: 기존 클러스터에 태그 지정

다음 명령은 기존 가상 클러스터에 태그를 지정합니다.

```
aws emr-containers tag-resource --resource-arn resource_ARN --tags team=devs
```

## 예제 2: 기존 클러스터에서 태그 제거

다음 명령은 기존 가상 클러스터에서 태그를 삭제합니다.

```
aws emr-containers untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

## 예제 3: 리소스의 태그 나열

다음 명령은 기존 리소스와 연결된 태그를 나열합니다.

```
aws emr-containers list-tags-for-resource --resource-arn resource_ARN
```

# Amazon EMR on EKS에서 문제 해결

이 섹션에서는 Amazon EMR on EKS의 문제를 해결하는 방법을 설명합니다. Amazon EMR과 관련된 일반적인 문제를 해결하는 방법에 대한 자세한 내용은 Amazon EMR 관리 안내서에서 [클러스터 문제 해결](#)을 참조하세요.

## 주제

- [PersistentVolumeClaims\(PVC\)를 사용하는 작업 문제 해결](#)
- [Amazon EMR on EKS 수직 자동 조정 문제 해결](#)
- [Amazon EMR on EKS Spark 운영자 문제 해결](#)

## PersistentVolumeClaims(PVC)를 사용하는 작업 문제 해결

작업의 PersistentVolumeClaims(PVC)를 생성, 나열 또는 삭제해야 하는 경우 기본 Kubernetes 역할 `emr-containers`를 추가하지 않으면 작업 제출 시 작업에 실패합니다. 이 권한이 없으면 `emr-container` 역할은 Spark 드라이버 또는 Spark 클라이언트에 필요한 역할을 생성할 수 없습니다. 오류 메시지에서 알 수 있듯이 Spark 드라이버 또는 클라이언트 역할에 권한을 추가하는 것만으로는 충분하지 않습니다. `emr-containers` 기본 역할에도 필수 권한이 포함되어야 합니다. 이 섹션에서는 `emr-containers` 기본 역할에 필수 권한을 추가하는 방법을 설명합니다.

## Verification(확인)

`emr-containers` 역할에 필수 권한이 있는지 확인하려면 `NAMESPACE` 변수를 자체 값으로 설정한 후 다음 명령을 실행합니다.

```
export NAMESPACE=YOUR_VALUE
kubectl describe role emr-containers -n ${NAMESPACE}
```

또한 Spark 및 클라이언트 역할에 필수 권한이 있는지 확인하려면 다음 명령을 실행합니다.

```
kubectl describe role emr-containers-role-spark-driver -n ${NAMESPACE}
kubectl describe role emr-containers-role-spark-client -n ${NAMESPACE}
```

권한이 없는 경우 다음과 같이 패치를 진행합니다.

## 패치

1. 권한이 없는 작업이 현재 실행 중인 경우 해당 작업을 중지합니다.
2. 다음과 같이 RBAC\_Patch.py라는 파일을 생성합니다.

```
import os
import subprocess as sp
import tempfile as temp
import json
import argparse
import uuid

def delete_if_exists(dictionary: dict, key: str):
    if dictionary.get(key, None) is not None:
        del dictionary[key]

def doTerminalCmd(cmd):
    with temp.TemporaryFile() as f:
        process = sp.Popen(cmd, stdout=f, stderr=f)
        process.wait()
        f.seek(0)
        msg = f.read().decode()
    return msg

def patchRole(roleName, namespace, extraRules, skipConfirmation=False):
    cmd = f"kubectl get role {roleName} -n {namespace} --output json".split(" ")
    msg = doTerminalCmd(cmd)
    if "(NotFound)" in msg and "Error" in msg:
        print(msg)
        return False
    role = json.loads(msg)
    rules = role["rules"]
    rulesToAssign = extraRules[::]
    passedRules = []
    for rule in rules:
        apiGroups = set(rule["apiGroups"])
        resources = set(rule["resources"])
        verbs = set(rule["verbs"])
        for extraRule in extraRules:
            passes = 0
            apiGroupsExtra = set(extraRule["apiGroups"])
            resourcesExtra = set(extraRule["resources"])
            verbsExtra = set(extraRule["verbs"])
```

```

        passes += len(apiGroupsExtra.intersection(apiGroups)) >=
len(apiGroupsExtra)
        passes += len(resourcesExtra.intersection(resources)) >=
len(resourcesExtra)
        passes += len(verbsExtra.intersection(verbs)) >= len(verbsExtra)
        if passes >= 3:
            if extraRule not in passedRules:
                passedRules.append(extraRule)
                if extraRule in rulesToAssign:
                    rulesToAssign.remove(extraRule)
            break
    prompt_text = "Apply Changes?"
    if len(rulesToAssign) == 0:
        print(f"The role {roleName} seems to already have the necessary
permissions!")
        prompt_text = "Proceed anyways?"
    for ruleToAssign in rulesToAssign:
        role["rules"].append(ruleToAssign)
    delete_if_exists(role, "creationTimestamp")
    delete_if_exists(role, "resourceVersion")
    delete_if_exists(role, "uid")
    new_role = json.dumps(role, indent=3)
    uid = uuid.uuid4()
    filename = f"Role-{roleName}-New_Permissions-{uid}-TemporaryFile.json"
    try:
        with open(filename, "w+") as f:
            f.write(new_role)
            f.flush()
        prompt = "y"
        if not skipConfirmation:
            prompt = input(
                doTerminalCmd(f"kubectl diff -f {filename}".split(" ")) +
f"\n{prompt_text} y/n: "
                ).lower().strip()
            while prompt != "y" and prompt != "n":
                prompt = input("Please make a valid selection. y/n:
").lower().strip()
            if prompt == "y":
                print(doTerminalCmd(f"kubectl apply -f {filename}".split(" ")))
    except Exception as e:
        print(e)
    os.remove(f"./{filename}")

if __name__ == '__main__':

```



```

parser = argparse.ArgumentParser()
parser.add_argument("-n", "--namespace",
                    help="Namespace of the Role. By default its the
VirtualCluster's namespace",
                    required=True,
                    dest="namespace"
                    )

parser.add_argument("-p", "--no-prompt",
                    help="Applies the patches without asking first",
                    dest="no_prompt",
                    default=False,
                    action="store_true"
                    )
args = parser.parse_args()

emrRoleRules = [
    {
        "apiGroups": [""],
        "resources": ["persistentvolumeclaims"],
        "verbs": ["list", "create", "delete", "patch"]
    }
]

driverRoleRules = [
    {
        "apiGroups": [""],
        "resources": ["persistentvolumeclaims"],
        "verbs": ["list", "create", "delete", "patch", "deletcollection"]
    },
    {
        "apiGroups": [""],
        "resources": ["services"],
        "verbs": ["get", "list", "describe", "create", "delete", "watch",
"deletcollection"]
    },
    {
        "apiGroups": [""],
        "resources": ["configmaps", "pods"],
        "verbs": ["deletcollection"]
    }
]

```

```

clientRoleRules = [
    {
        "apiGroups": [""],
        "resources": ["persistentvolumeclaims"],
        "verbs": ["list", "create", "delete", "patch"]
    }
]

patchRole("emr-containers", args.namespace, emrRoleRules, args.no_prompt)
patchRole("emr-containers-role-spark-driver", args.namespace, driverRoleRules,
args.no_prompt)
patchRole("emr-containers-role-spark-client", args.namespace, clientRoleRules,
args.no_prompt)

```

3. Python 스크립트를 실행합니다.

```
python3 RBAC_Patch.py -n ${NAMESPACE}
```

4. 새 권한과 이전 권한 간의 kubectl 차이가 표시됩니다. y를 눌러 역할을 패치합니다.

5. 다음과 같이 추가 권한이 있는 세 가지 역할을 확인합니다.

```
kubectl describe role -n ${NAMESPACE}
```

6. Python 스크립트를 실행합니다.

```
python3 RBAC_Patch.py -n ${NAMESPACE}
```

7. 명령을 실행하면 새 권한과 이전 권한 사이에 kubectl 차이가 표시됩니다. y를 눌러 역할을 패치합니다.

8. 추가 권한이 있는 세 가지 역할을 확인합니다.

```
kubectl describe role -n ${NAMESPACE}
```

9. 작업을 다시 제출합니다.

## 수동 패치

애플리케이션에 필요한 권한이 PVC 규칙 이외의 다른 항목에 적용되는 경우 필요에 따라 Amazon EMR 가상 클러스터에 대한 Kubernetes 권한을 수동으로 추가할 수 있습니다.

**Note**

emr-containers 역할은 기본 역할입니다. 즉, 기본 드라이버 또는 클라이언트 역할을 변경하려면 먼저 필요한 모든 권한을 제공해야 합니다.

1. 아래 명령을 실행하여 현재 권한을 yaml 파일로 다운로드합니다.

```
kubectl get role -n ${NAMESPACE} emr-containers -o yaml >> emr-containers-role-patch.yaml
kubectl get role -n ${NAMESPACE} emr-containers-role-spark-driver -o yaml >> driver-role-patch.yaml
kubectl get role -n ${NAMESPACE} emr-containers-role-spark-client -o yaml >> client-role-patch.yaml
```

2. 애플리케이션에 필요한 권한에 따라 각 파일을 편집하고 다음과 같은 추가 규칙을 추가합니다.

- emr-containers-role-patch.yaml

```
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - list
  - create
  - delete
  - patch
```

- driver-role-patch.yaml

```
- apiGroups:
  - ""
  resources:
  - persistentvolumeclaims
  verbs:
  - list
  - create
  - delete
  - patch
  - deletecollection
- apiGroups:
  - ""
```

```
resources:
- services
verbs:
- get
- list
- describe
- create
- delete
- watch
- deletecollection
- apiGroups:
- ""
resources:
- configmaps
- pods
verbs:
- deletecollection
```

- client-role-patch.yaml

```
- apiGroups:
- ""
resources:
- persistentvolumeclaims
verbs:
- list
- create
- delete
- patch
```

3. 다음 속성을 해당 값과 함께 제거합니다. 업데이트를 적용하는 데 필요합니다.

- creationTimestamp
- resourceVersion
- uid

4. 마지막으로 패치를 실행합니다.

```
kubectl apply -f emr-containers-role-patch.yaml
kubectl apply -f driver-role-patch.yaml
kubectl apply -f client-role-patch.yaml
```

## Amazon EMR on EKS 수직 자동 조정 문제 해결

Operator Lifecycle Manager를 사용하여 Amazon EKS 클러스터에서 Amazon EMR on EKS 수직 자동 조정 운영자를 설정할 때 문제가 발생하는 경우 다음 섹션을 참조하세요. 설치 완료 단계를 포함한 자세한 내용은 [Amazon EMR Spark 작업에 수직 자동 조정 사용](#) 섹션을 참조하세요.

### 403 금지됨 오류

[Amazon EKS 클러스터에 Operator Lifecycle Manager\(OLM\) 설치](#)의 단계를 따라 olm status 명령을 실행했지만 아래와 같은 403 Forbidden 오류가 반환되었다면 운영자용 Amazon ECR 리포지토리에 대한 인증 토큰을 얻지 못했을 수 있습니다.

이 문제를 해결하려면 [Amazon EMR on EKS 수직 자동 조정 운영자 설치](#)의 단계를 반복하여 토큰을 확보합니다. 그런 다음, 설치를 다시 시도합니다.

```
Error: FATA[0002] Failed to run bundle: pull bundle image: error pulling image IMAGE.
error resolving name : unexpected status code [manifests latest]: 403 Forbidden
```

### Kubernetes 네임스페이스를 찾을 수 없음

Amazon EKS 클러스터에서 [Amazon EMR on EKS 수직 자동 조정 운영자를 설정](#)할 때 다음과 같이 namespaces not found 오류가 발생할 수 있습니다.

```
FATA[0020] Failed to run bundle: create catalog: error creating catalog source:
namespaces "NAME" not found.
```

지정한 네임스페이스가 없는 경우 OLM은 수직 자동 조정 운영자를 설치하지 않습니다. 이 문제를 해결하려면 다음 명령을 사용하여 네임스페이스를 생성합니다. 그런 다음, 설치를 다시 시도합니다.

```
kubectl create namespace NAME
```

### Docker 보안 인증을 저장하는 중 오류가 발생했습니다.

[수직 자동 조정을 설정](#)하려면 Amazon EMR on EKS 수직 자동 조정 관련 도커 이미지를 인증하고 가져와야 합니다. 이때 Docker가 실행되지 않는 경우 다음과 같은 오류가 발생할 수 있습니다.

```
aws ecr get-login-password \
--region $REGION | docker login \
```

```
--username AWS \
--password-stdin $ACCOUNT_ID.dkr.ecr.$REGION.amazonaws.com
```

```
Error saving credentials: error storing credentials - err: exit status 1
out: 'Post "http://ipc/registry/credstore-updated": dial unix backend.sock: connect: no
such file or directory'
```

이 문제를 해결하려면 Docker가 실행 중인지 확인하거나 Docker Desktop을 엽니다. 그런 다음 보안 인증을 다시 저장해 보세요.

## Amazon EMR on EKS Spark 운영자 문제 해결

Amazon EMR on EKS Spark 운영자와 관련된 문제가 발생하는 경우 다음 섹션을 참조하세요. 설치 완료 단계를 포함한 자세한 내용은 [Spark 운영자에서 Spark 작업 실행](#) 섹션을 참조하세요.

### 차트 Helm 설치 중 오류 발생

[Spark 운영자 설치](#)의 단계를 따랐지만 차트 Helm을 설치하거나 확인하려고 할 때 아래와 같은 INSTALLATION FAILED 오류가 반환되었다면 운영자용 Amazon ECR 리포지토리에 대한 인증 토큰을 얻지 못했을 수 있습니다.

이 문제를 해결하려면 [Spark 운영자 설치](#)의 단계를 반복하여 Helm 클라이언트를 Amazon ECR 레지스트리에 인증합니다. 그런 다음 설치 단계를 다시 시도합니다.

```
Error: INSTALLATION FAILED: Kubernetes cluster unreachable: the server has asked for
the client to provide credentials
```

### UnsupportedFileSystemException: No FileSystem for scheme "s3"

'main' 스레드에서 다음과 같은 예외가 발생할 수 있습니다.

```
org.apache.hadoop.fs.UnsupportedFileSystemException: No FileSystem for scheme "s3"
```

이 경우 SparkApplication 사양에 다음 예외를 추가합니다.

```
hadoopConf:
  # EMRFS filesystem
  fs.s3.customAWSCredentialsProvider:
    com.amazonaws.auth.WebIdentityTokenCredentialsProvider
```

```

fs.s3.impl: com.amazon.ws.emr.hadoop.fs.EmrFileSystem
fs.AbstractFileSystem.s3.impl: org.apache.hadoop.fs.s3.EMRFSDelegate
fs.s3.buffer.dir: /mnt/s3
fs.s3.getObject.initialSocketTimeoutMilliseconds: "2000"
mapreduce.fileoutputcommitter.algorithm.version.emr_internal_use_only.EmrFileSystem:
"2"
mapreduce.fileoutputcommitter.cleanup-
failures.ignored.emr_internal_use_only.EmrFileSystem: "true"
sparkConf:
  # Required for EMR Runtime
  spark.driver.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
  spark.driver.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native
  spark.executor.extraClassPath: /usr/lib/hadoop-lzo/lib/*:/usr/lib/hadoop/hadoop-
aws.jar:/usr/share/aws/aws-java-sdk/*:/usr/share/aws/emr/emrfs/conf:/usr/share/aws/
emr/emrfs/lib/*:/usr/share/aws/emr/emrfs/auxlib/*:/usr/share/aws/emr/security/conf:/
usr/share/aws/emr/security/lib/*:/usr/share/aws/hmclient/lib/aws-glue-datacatalog-
spark-client.jar:/usr/share/java/Hive-JSON-Serde/hive-openx-serde.jar:/usr/share/aws/
sagemaker-spark-sdk/lib/sagemaker-spark-sdk.jar:/home/hadoop/extrajars/*
  spark.executor.extraLibraryPath: /usr/lib/hadoop/lib/native:/usr/lib/hadoop-lzo/lib/
native:/docker/usr/lib/hadoop/lib/native:/docker/usr/lib/hadoop-lzo/lib/native

```

# Amazon EMR on EKS 서비스 엔드포인트 및 할당량

다음은 Amazon EMR on EKS에 대한 서비스 엔드포인트 및 서비스 할당량입니다. AWS 서비스에 프로그래밍 방식으로 연결하려면 엔드포인트를 사용합니다. 일부 AWS 서비스는 표준 AWS 엔드포인트 외에도 선택한 리전에서 FIPS 엔드포인트를 제공합니다. 자세한 내용은 [AWS 서비스 엔드포인트](#)를 참조하세요. Service Quotas(제한이라고도 함)는 AWS 계정의 최대 서비스 리소스 또는 작업 수입니다. 자세한 내용은 [AWS Service Quotas](#)를 참조하세요.

## 서비스 엔드포인트

| AWS 리전 이름       | 코드             | 엔드포인트                                       | 프로토콜  |
|-----------------|----------------|---------------------------------------------|-------|
| 미국 동부(버지니아 북부)  | us-east-1      | emr-containers.us-east-1.amazonaws.com      | HTTPS |
| 미국 동부(오하이오)     | us-east-2      | emr-containers.us-east-2.amazonaws.com      | HTTPS |
| 미국 서부(캘리포니아 북부) | us-west-1      | emr-containers.us-west-1.amazonaws.com      | HTTPS |
| 미국 서부(오리건)      | us-west-2      | emr-containers.us-west-2.amazonaws.com      | HTTPS |
| 아시아 태평양(도쿄)     | ap-northeast-1 | emr-containers.ap-northeast-1.amazonaws.com | HTTPS |
| 아시아 태평양(서울)     | ap-northeast-2 | emr-containers.ap-northeast-2.amazonaws.com | HTTPS |
| 아시아 태평양(오사카)    | ap-northeast-3 | emr-containers.ap-northeast-3.amazonaws.com | HTTPS |
| 아시아 태평양(뭄바이)    | ap-south-1     | emr-containers.ap-south-1.amazonaws.com     | HTTPS |
| 아시아 태평양(하이데라바드) | ap-south-2     | emr-containers.ap-south-2.amazonaws.com     | HTTPS |



| AWS 리전 이름     | 코드             | 엔드포인트                                          | 프로토콜  |
|---------------|----------------|------------------------------------------------|-------|
| 아시아 태평양(싱가포르) | ap-southeast-1 | emr-containers.ap-southeast-1.amazonaws.com    | HTTPS |
| 아시아 태평양(시드니)  | ap-southeast-2 | emr-containers.ap-southeast-2.amazonaws.com    | HTTPS |
| 아시아 태평양(자카르타) | ap-southeast-3 | emr-containers.ap-southeast-3.amazonaws.com    | HTTPS |
| 아시아 태평양(홍콩)   | ap-east-1      | emr-containers.ap-east-1.amazonaws.com         | HTTPS |
| 아프리카(케이프타운)   | af-south-1     | emr-containers.af-south-1.amazonaws.com        | HTTPS |
| 캐나다(중부)       | ca-central-1   | emr-containers.ca-central-1.amazonaws.com      | HTTPS |
| 중국(닝샤)        | cn-northwest-1 | emr-containers.cn-northwest-1.amazonaws.com.cn | HTTPS |
| 중국(베이징)       | cn-north-1     | emr-containers.cn-north-1.amazonaws.com.cn     | HTTPS |
| 유럽(프랑크푸르트)    | eu-central-1   | emr-containers.eu-central-1.amazonaws.com      | HTTPS |
| 유럽(취리히)       | eu-central-2   | emr-containers.eu-central-2.amazonaws.com      | HTTPS |
| 유럽(아일랜드)      | eu-west-1      | emr-containers.eu-west-1.amazonaws.com         | HTTPS |
| 유럽(런던)        | eu-west-2      | emr-containers.eu-west-2.amazonaws.com         | HTTPS |

| AWS 리전 이름             | 코드            | 엔드포인트                                      | 프로토콜  |
|-----------------------|---------------|--------------------------------------------|-------|
| 유럽(파리)                | eu-west-3     | emr-containers.eu-west-3.amazonaws.com     | HTTPS |
| 유럽(스톡홀름)              | eu-north-1    | emr-containers.eu-north-1.amazonaws.com    | HTTPS |
| 유럽(밀라노)               | eu-south-1    | emr-containers.eu-south-1.amazonaws.com    | HTTPS |
| 유럽(스페인)               | eu-south-2    | emr-containers.eu-south-2.amazonaws.com    | HTTPS |
| 이스라엘(텔아비브)            | il-central-1  | emr-containers.il-central-1.amazonaws.com  | HTTPS |
| 남아메리카(상파울루)           | sa-east-1     | emr-containers.sa-east-1.amazonaws.com     | HTTPS |
| 중동(UAE)               | me-central-1  | emr-containers.me-central-1.amazonaws.com  | HTTPS |
| Middle East (Bahrain) | me-south-1    | emr-containers.me-south-1.amazonaws.com    | HTTPS |
| AWS GovCloud(미국 동부)   | us-gov-east-1 | emr-containers.us-gov-east-1.amazonaws.com | HTTPS |
| AWS GovCloud(미국 서부)   | us-gov-west-1 | emr-containers.us-gov-west-1.amazonaws.com | HTTPS |

## Service Quotas

Amazon EMR on EKS는 리전별로 각 AWS 계정에 대해 다음과 같은 API 요청을 제한합니다. 조절이 적용되는 방식에 대한 자세한 내용은 Amazon EC2 API 레퍼런스에서 [API 요청 조절](#)을 참조하세요. 다음 가이드에 따라 AWS 계정에 대한 API 제한 할당량 증가를 요청할 수 있습니다.

| API 작업                                   | 버킷 최대 용량 | 초당 버킷 다시 채우기 속도 |
|------------------------------------------|----------|-----------------|
| CancelJobRun                             | 25       | 1               |
| CreateJobTemplate                        | 25       | 1               |
| CreateManagedEndpoint                    | 25       | 1               |
| CreateSecurityConfiguration              | 25       | 1               |
| CreateVirtualCluster                     | 25       | 1               |
| DeleteJobTemplate                        | 25       | 1               |
| DeleteManagedEndpoint                    | 25       | 1               |
| DeleteVirtualCluster                     | 25       | 1               |
| DescribeJobRun                           | 100      | 20              |
| DescribeJobTemplate                      | 25       | 1               |
| DescribeManagedEndpoint                  | 100      | 5               |
| DescribeSecurityConfiguration            | 25       | 1               |
| DescribeVirtualCluster                   | 100      | 5               |
| GetManagedEndpoint<br>SessionCredentials | 25       | 1               |
| ListJobRuns                              | 100      | 5               |
| ListJobTemplates                         | 25       | 1               |
| ListManagedEndpoints                     | 25       | 1               |
| ListSecurityConfigurations               | 25       | 1               |
| ListVirtualClusters                      | 100      | 5               |
| StartJobRun                              | 25       | 1               |

| API 작업                                         | 버킷 최대 용량 | 초당 버킷 다시 채우기 속도 |
|------------------------------------------------|----------|-----------------|
| Throttle quota for all EMR on EKS API requests | 200      | 20              |

AWS 계정을 생성할 때 리전별로 AWS 리소스에 기본 할당량(한도라고도 함)을 설정합니다. 리소스 할당량을 초과하려고 하면 요청이 실패합니다. 이 경우 리소스 사용량을 줄이거나 할당량 증가를 요청할 수 있습니다.

Service Quotas 콘솔은 AWS 서비스에 대한 할당량을 보고 관리하고 사용하는 많은 리소스에 대한 할당량 증가를 요청할 수 있는 중앙 위치입니다. 여기에 제공된 할당량 정보를 사용하여 AWS 인프라를 관리합니다. 실제로 필요할 시점보다 미리 할당량 증가를 요청하도록 계획하세요.

## 할당량 보기 및 할당량 증가 요청

Service Quotas 콘솔을 사용하면 각 리전의 현재 서비스 할당량을 볼 수 있습니다. 자세한 지침은 [Service Quotas 사용 설명서](#)의 Service Quotas 보기를 참조하세요.

AWS 관리 콘솔 또는 AWS CLI를 사용하여 할당량 증가를 요청할 수 있습니다. 단계는 [할당량 증가 요청](#)에 자세히 설명되어 있습니다.

# Amazon EMR on EKS 릴리스

Amazon EMR 릴리스는 빅 데이터 에코시스템의 오픈 소스 애플리케이션입니다. 각 릴리스는 클러스터를 작업을 실행할 때 Amazon EMR on EKS를 배포 및 구성하기 위해 선택한 여러 빅 데이터 애플리케이션, 구성 요소 및 기능으로 구성됩니다.

Amazon EMR 릴리스 5.32.0 및 6.2.0부터 Amazon EMR on EKS를 배포할 수 있습니다. 이 배포 옵션은 이전 Amazon EMR 릴리스 버전에서 사용할 수 없습니다. 작업을 제출할 때 지원되는 릴리스 버전을 지정해야 합니다.

Amazon EMR on EKS는 특정 릴리스 날짜와 함께 `emr-x.x.x-latest` 또는 `emr-x.x.x-yyyyymmdd` 양식의 릴리스 레이블을 사용합니다. 예: `emr-7.13.0-latest` 또는 `emr-7.13.0-20210129`. `-latest` 접미사를 사용하면 Amazon EMR 버전에서 항상 최신 보안 업데이트를 포함하도록 보장할 수 있습니다.

## Note

Amazon EMR on EKS와 EC2에서 실행되는 Amazon EMR 간의 비교는 AWS 웹 사이트의 [Amazon EMR FAQs](#)를 참조하세요.

## 주제

- [Amazon EMR on EKS 7.13.0 릴리스](#)
- [Amazon EMR on EKS 7.12.0 릴리스](#)
- [Amazon EMR on EKS 7.11.0 릴리스](#)
- [Amazon EMR on EKS 7.10.0 릴리스](#)
- [Amazon EMR on EKS 7.9.0 릴리스](#)
- [Amazon EMR on EKS 7.8.0 릴리스](#)
- [Amazon EMR on EKS 7.7.0 릴리스](#)
- [Amazon EMR on EKS 7.6.0 릴리스](#)
- [Amazon EMR on EKS 7.5.0 릴리스](#)
- [Amazon EMR on EKS 7.4.0 릴리스](#)
- [Amazon EMR on EKS 7.3.0 릴리스](#)
- [Amazon EMR on EKS 7.2.0 릴리스](#)

- [Amazon EMR on EKS 7.1.0 릴리스](#)
- [Amazon EMR on EKS 7.0.0 릴리스](#)
- [Amazon EMR on EKS 6.15.0 릴리스](#)
- [Amazon EMR on EKS 6.14.0 릴리스](#)
- [Amazon EMR on EKS 6.13.0 릴리스](#)
- [Amazon EMR on EKS 6.12.0 릴리스](#)
- [Amazon EMR on EKS 6.11.0 릴리스](#)
- [Amazon EMR on EKS 6.10.0 릴리스](#)
- [Amazon EMR on EKS 6.9.0 릴리스](#)
- [Amazon EMR on EKS 6.8.0 릴리스](#)
- [Amazon EMR on EKS 6.7.0 릴리스](#)
- [Amazon EMR on EKS 6.6.0 릴리스](#)
- [Amazon EMR on EKS 6.5.0 릴리스](#)
- [Amazon EMR on EKS 6.4.0 릴리스](#)
- [Amazon EMR on EKS 6.3.0 릴리스](#)
- [Amazon EMR on EKS 6.2.0 릴리스](#)
- [Amazon EMR on EKS 5.36.0 릴리스](#)
- [Amazon EMR on EKS 5.35.0 릴리스](#)
- [Amazon EMR on EKS 5.34.0 릴리스](#)
- [Amazon EMR on EKS 5.33.0 릴리스](#)
- [Amazon EMR on EKS 5.32.0 릴리스](#)

## Amazon EMR on EKS 7.13.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 7.13.0 릴리스에 대한 일반적인 자세한 내용은 [Amazon EMR 릴리스 안내서의 Amazon EMR 7.13.0](#)을 참조하세요.

### Amazon EMR on EKS 7.13 릴리스

Amazon EMR on EKS에는 다음 Amazon EMR 7.13.0 릴리스를 사용할 수 있습니다. 관련 컨테이너 이미지 태그와 같은 세부 정보를 보려면 특정 emr-7.13.0-XXXX 릴리스를 선택합니다.

## Flink releases

Flink 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.13.0 릴리스를 사용할 수 있습니다.

- [emr-7.13.0-flink-latest](#)
- [emr-7.13.0-flink-20260410](#)

## Spark releases

Spark 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.13.0 릴리스를 사용할 수 있습니다.

- [emr-7.13.0-최신](#)
- [emr-7.13.0-20260410](#)
- emr-7.13.0-spark-rapids-latest
- emr-7.13.0-spark-rapids-20260410
- emr-7.13.0-java11-latest
- emr-7.13.0-java11-20260410
- emr-7.13.0-java8-latest
- emr-7.13.0-java8-20260410
- emr-7.13.0-spark-rapids-java8-latest
- emr-7.13.0-spark-rapids-java8-20260410
- notebook-spark/emr-7.13.0-latest
- notebook-spark/emr-7.13.0-20260410
- notebook-spark/emr-7.13.0-spark-rapids-latest
- notebook-spark/emr-7.13.0-spark-rapids-20260410
- notebook-spark/emr-7.13.0-java11-latest
- notebook-spark/emr-7.13.0-java11-20260410
- notebook-spark/emr-7.13.0-java8-latest
- notebook-spark/emr-7.13.0-java8-20260410
- notebook-spark/emr-7.13.0-spark-rapids-java8-latest
- notebook-spark/emr-7.13.0-spark-rapids-java8-20260410
- notebook-python/emr-7.13.0-latest

- notebook-python/emr-7.13.0-20260410
- notebook-python/emr-7.13.0-spark-rapids-latest
- notebook-python/emr-7.13.0-spark-rapids-20260410
- notebook-python/emr-7.13.0-java11-latest
- notebook-python/emr-7.13.0-java11-20260410
- notebook-python/emr-7.13.0-java8-latest
- notebook-python/emr-7.13.0-java8-20260410
- notebook-python/emr-7.13.0-spark-rapids-java8-latest
- notebook-python/emr-7.13.0-spark-rapids-java8-20260410
- livy/emr-7.13.0-latest
- livy/emr-7.13.0-20260410
- livy/emr-7.13.0-java11-latest
- livy/emr-7.13.0-java11-20260410
- livy/emr-7.13.0-java8-latest
- livy/emr-7.13.0-java8-20260410

## 릴리스 노트

Amazon EMR on EKS 7.13.0 릴리스 정보:

- 지원되는 애플리케이션 - AWS SDK for Java 2.42.12 and 1.12.797, Apache Spark 3.5.6-amzn-2, Apache Hudi 1.0.2-amzn-2, Apache Iceberg 1.10.0-amzn-1, Delta 3.3.2-amzn-2, Apache Spark RAPIDS 25.04.0-amzn-0, Apache Flink 1.20.0-amzn-7
- 지원되는 구성 요소 -emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류        | 설명                                  |
|-----------|-------------------------------------|
| core-site | Hadoop core-site.xml 파일에서 값을 변경합니다. |



| 분류                | 설명                                       |
|-------------------|------------------------------------------|
| emrfs-site        | EMRFS 설정을 변경합니다.                         |
| spark-metrics     | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults    | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env         | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site   | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j2      | Spark log4j2.properties 파일에서 값을 변경합니다.   |
| emr-job-submitter | <a href="#">작업 제출자 포드</a> 구성.            |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 변경 사항 및 기능

Amazon EMR on EKS의 7.13.0 릴리스에는 다음 기능이 포함되어 있습니다.

- PySpark 및 Spark 워크로드의 Python 3.11 기본값 - Python 3.11은 이제 PySpark 및 Spark 워크로드의 기본 Python 버전입니다. Python 3.9는 다른 모든 애플리케이션의 기본값으로 유지됩니다. Python 3.9와 3.11이 모두 릴리스에 포함되어 있습니다.

### emr-7.13.0-최신

릴리스 정보: `emr-7.13.0-latest`는 현재 `emr-7.13.0-20260410`를 가리킵니다.

리전: `emr-7.13.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.13.0:latest`

### emr-7.13.0-20260410

릴리스 정보: `emr-7.13.0-20260410`는 2026년 4월에 릴리스되었습니다. Amazon EMR 7.13.0(Spark)의 최초 릴리스입니다.

리전: `emr-7.13.0-20260410`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.13.0-20260410`

### emr-7.13.0-flink-latest

릴리스 정보: `emr-7.13.0-flink-latest`는 현재 `emr-7.13.0-flink-20260410`를 가리킵니다.

리전: `emr-7.13.0-flink-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.13.0-flink:latest`

### emr-7.13.0-flink-20260410

릴리스 정보: `emr-7.13.0-flink-20260410`는 2026년 4월에 릴리스되었습니다. Amazon EMR 7.13.0(Flink)의 최초 릴리스입니다.

리전: `emr-7.13.0-flink-20260410`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.13.0-flink:20260410`

## Amazon EMR on EKS 7.12.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 7.12.0 릴리스에 대한 일반적인 자세한 내용은 [Amazon EMR 릴리스 안내서의 Amazon EMR 7.12.0](#)을 참조하세요.

### Amazon EMR on EKS 7.12 릴리스

Amazon EMR on EKS에는 다음 Amazon EMR 7.12.0 릴리스를 사용할 수 있습니다. 관련 컨테이너 이미지 태그와 같은 세부 정보를 보려면 특정 `emr-7.12.0-XXXX` 릴리스를 선택합니다.

#### Flink releases

Flink 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.12.0 릴리스를 사용할 수 있습니다.

- [emr-7.12.0-flink-latest](#)
- [emr-7.12.0-flink-20251111](#)

#### Spark releases

Spark 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.12.0 릴리스를 사용할 수 있습니다.

- [emr-7.12.0-최신](#)
- [emr-7.12.0-20251111](#)
- `emr-7.12.0-spark-rapids-latest`
- `emr-7.12.0-spark-rapids-20251111`
- `emr-7.12.0-java11-latest`
- `emr-7.12.0-java11-20251111`
- `emr-7.12.0-java8-latest`

- emr-7.12.0-java8-20251111
- emr-7.12.0-spark-rapids-java8-latest
- emr-7.12.0-spark-rapids-java8-20251111
- notebook-spark/emr-7.12.0-latest
- notebook-spark/emr-7.12.0-20251111
- notebook-spark/emr-7.12.0-spark-rapids-latest
- notebook-spark/emr-7.12.0-spark-rapids-20251111
- notebook-spark/emr-7.12.0-java11-latest
- notebook-spark/emr-7.12.0-java11-20251111
- notebook-spark/emr-7.12.0-java8-latest
- notebook-spark/emr-7.12.0-java8-20251111
- notebook-spark/emr-7.12.0-spark-rapids-java8-latest
- notebook-spark/emr-7.12.0-spark-rapids-java8-20251111
- notebook-python/emr-7.12.0-latest
- notebook-python/emr-7.12.0-20251111
- notebook-python/emr-7.12.0-spark-rapids-latest
- notebook-python/emr-7.12.0-spark-rapids-20251111
- notebook-python/emr-7.12.0-java11-latest
- notebook-python/emr-7.12.0-java11-20251111
- notebook-python/emr-7.12.0-java8-latest
- notebook-python/emr-7.12.0-java8-20251111
- notebook-python/emr-7.12.0-spark-rapids-java8-latest
- notebook-python/emr-7.12.0-spark-rapids-java8-20251111
- livy/emr-7.12.0-latest
- livy/emr-7.12.0-20251111
- livy/emr-7.12.0-java11-latest
- livy/emr-7.12.0-java11-20251111
- livy/emr-7.12.0-java8-latest
- livy/emr-7.12.0-java8-20251111

## 릴리스 노트

Amazon EMR on EKS 7.12.0 릴리스 정보:

- 지원되는 애플리케이션 - AWS SDK for Java 2.35.5 and 1.12.792, Apache Spark 3.5.6-amzn-1, Apache Hudi 1.0.2-amzn-1, Apache Iceberg 1.10.0-amzn-0, Delta 3.3.2-amzn-1, Apache Spark RAPIDS 25.04.0-amzn-0, Apache Flink 1.20.0-amzn-6
- 지원되는 구성 요소 -emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                | 설명                                       |
|-------------------|------------------------------------------|
| core-site         | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site        | EMRFS 설정을 변경합니다.                         |
| spark-metrics     | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults    | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env         | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site   | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j2      | Spark log4j2.properties 파일에서 값을 변경합니다.   |
| emr-job-submitter | <a href="#">작업 제출자 포트</a> 구성.            |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 변경 사항 및 기능

Amazon EMR on EKS의 7.12.0 릴리스에는 다음 기능이 포함되어 있습니다.

- Iceberg 구체화된 뷰 - EMR 7.12.0부터 EMR Spark는 Iceberg 구체화된 뷰(MV)의 생성 및 관리를 지원합니다.
- Hudi 전체 테이블 액세스 - EMR 7.12.0부터 EMR은 이제 Lake Formation에 정의된 정책을 기반으로 Apache Spark의 Apache Hudi에 대한 전체 테이블 액세스(FTA) 제어를 지원합니다. 이 기능을 사용하면 작업 역할에 전체 테이블 액세스 권한이 있을 때 Lake Formation 등록 테이블의 Amazon EMR Spark 작업에서 읽기 및 쓰기 작업을 수행할 수 있습니다.
- Iceberg 버전 업그레이드 - EMR 7.12.0은 Apache Iceberg 버전 1.10을 지원합니다.
- Livy 대화형 워크로드 로깅 - EMR 7.12.0부터 EMR은 Livy Spark 작업 실패에 대한 문제 해결을 개선하기 위해 주요 시스템 구성 요소에 대한 광범위한 로깅을 지원합니다. 이 기능은 EMR 서비스에 추가 Livy 및 SecretAgent 로그에 대한 액세스 권한을 제공하여 문제 해결을 간소화합니다.

## emr-7.12.0-최신

릴리스 정보: emr-7.12.0-latest는 현재 emr-7.12.0-20251111를 가리킵니다.

리전: emr-7.12.0-latest는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.12.0:latest`

## `emr-7.12.0-20251111`

릴리스 정보: `emr-7.12.0-20251111`는 2025년 11월에 릴리스되었습니다. Amazon EMR 7.12.0(Spark)의 최초 릴리스입니다.

리전: `emr-emr-7.12.0-20251111`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.12.0-20251111`

## `emr-7.12.0-flink-latest`

릴리스 정보: `emr-7.12.0-flink-latest`는 현재 `emr-7.12.0-flink-20251111`을 가리킵니다.

리전: `emr-7.12.0-flink-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.12.0-flink:latest`

## `emr-7.12.0-flink-20251111`

릴리스 정보: `emr-7.12.0-flink-20251111`는 2025년 11월에 릴리스되었습니다. Amazon EMR 7.12.0(Flink)의 최초 릴리스입니다.

리전: `emr-7.12.0-flink-20251111`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.12.0-flink:20251111`

## Amazon EMR on EKS 7.11.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR과 일반적으로 Amazon EMR 7.11.0 릴리스에 대한 자세한 내용은 [Amazon EMR 릴리스 안내서의 Amazon EMR 7.11.0](#)을 참조하세요.

## Amazon EMR on EKS 7.11 릴리스

Amazon EMR on EKS에는 다음 Amazon EMR 7.11.0 릴리스를 사용할 수 있습니다. 관련 컨테이너 이미지 태그와 같은 세부 정보를 보려면 특정 `emr-7.11.0-XXXX` 릴리스를 선택합니다.

## Flink releases

Flink 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.11.0 릴리스를 사용할 수 있습니다.

- [emr-7.11.0-flink-latest](#)
- [emr-7.11.0-flink-20251020](#)

## Spark releases

Spark 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.11.0 릴리스를 사용할 수 있습니다.

- [emr-7.11.0-최신](#)
- [emr-7.11.0-20251020](#)
- emr-7.11.0-spark-rapids-latest
- emr-7.11.0-spark-rapids-20251020
- emr-7.11.0-java11-latest
- emr-7.11.0-java11-20251020
- emr-7.11.0-java8-latest
- emr-7.11.0-java8-20251020
- emr-7.11.0-spark-rapids-java8-latest
- emr-7.11.0-spark-rapids-java8-20251020
- notebook-spark/emr-7.11.0-latest
- notebook-spark/emr-7.11.0-20251020
- notebook-spark/emr-7.11.0-spark-rapids-latest
- notebook-spark/emr-7.11.0-spark-rapids-20251020
- notebook-spark/emr-7.11.0-java11-latest
- notebook-spark/emr-7.11.0-java11-20251020
- notebook-spark/emr-7.11.0-java8-latest
- notebook-spark/emr-7.11.0-java8-20251020
- notebook-spark/emr-7.11.0-spark-rapids-java8-latest
- notebook-spark/emr-7.11.0-spark-rapids-java8-20251020
- notebook-python/emr-7.11.0-latest



- notebook-python/emr-7.11.0-20251020
- notebook-python/emr-7.11.0-spark-rapids-latest
- notebook-python/emr-7.11.0-spark-rapids-20251020
- notebook-python/emr-7.11.0-java11-latest
- notebook-python/emr-7.11.0-java11-20251020
- notebook-python/emr-7.11.0-java8-latest
- notebook-python/emr-7.11.0-java8-20251020
- notebook-python/emr-7.11.0-spark-rapids-java8-latest
- notebook-python/emr-7.11.0-spark-rapids-java8-20251020
- livy/emr-7.11.0-latest
- livy/emr-7.11.0-20251020
- livy/emr-7.11.0-java11-latest
- livy/emr-7.11.0-java11-20251020
- livy/emr-7.11.0-java8-latest
- livy/emr-7.11.0-java8-20251020

## 릴리스 노트

Amazon EMR on EKS 7.11.0 릴리스 정보:

- 지원되는 애플리케이션 - AWS SDK for Java 2.35.5 and 1.12.792, Apache Spark 3.5.6-amzn-0, Apache Hudi 1.0.2-amzn-0, Apache Iceberg 1.9.1-amzn-0, Delta 3.3.2-amzn-0, Apache Spark RAPIDS 25.04.0-amzn-0, Apache Flink 1.20.0-amzn-5
- 지원되는 구성 요소 -emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류        | 설명                                  |
|-----------|-------------------------------------|
| core-site | Hadoop core-site.xml 파일에서 값을 변경합니다. |

| 분류                | 설명                                       |
|-------------------|------------------------------------------|
| emrfs-site        | EMRFS 설정을 변경합니다.                         |
| spark-metrics     | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults    | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env         | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site   | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j2      | Spark log4j2.properties 파일에서 값을 변경합니다.   |
| emr-job-submitter | <a href="#">작업 제출자 포드</a> 구성.            |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 변경 사항 및 기능

Amazon EMR on EKS의 7.11.0 릴리스에는 다음과 같은 변경 사항이 포함되어 있습니다.

Amazon EMR on EKS는 이제 다음을 포함한 관리형 Livy 솔루션을 통해 SageMaker Unified Studio와의 통합을 지원합니다.

- 관리형 세션: SageMaker Unified Studio를 통해 EKS 클러스터에서 Spark 세션을 실행하기 위한 사용자 지정 HTTPS 엔드포인트를 제공하는 새로운 대화형 세션 리소스 유형
- Lake Formation 통합: 두 가지 모드로 데이터 액세스 제어 지원 a) 세분화된 액세스 제어 b) 전체 테이블 액세스(호환성 모드)
- Identity Management: 유연한 인증 옵션 a) IAM 역할 기반 액세스 제어 b) 기반 액세스 제어.
- 통합이 포함된 사용자 백그라운드 세션: 사용자가 SageMaker Unified Studio에서 로그인한 후에도 장기 실행 Spark 워크로드가 계속 실행되도록 지원하여 최대 90일의 세션 지원

### emr-7.11.0-최신

릴리스 정보: `emr-7.11.0-latest`는 현재 `emr-7.11.0-20251020`를 가리킵니다.

리전: `emr-7.11.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.11.0:latest`

### emr-7.11.0-20251020

릴리스 정보: `emr-7.11.0-20251020`는 2025년 11월에 릴리스되었습니다. Amazon EMR 7.11.0(Spark)의 최초 릴리스입니다.

리전: `emr-emr-7.11.0-20251020`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.11.0-20251020`

### emr-7.11.0-flink-latest

릴리스 정보: `emr-7.11.0-flink-latest`는 현재 `emr-7.11.0-flink-20251020`를 가리킵니다.

리전: `emr-7.11.0-flink-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.11.0-flink:latest`

## emr-7.11.0-flink-20251020

릴리스 정보: `7.11.0-flink-20251020`는 2025년 11월에 릴리스되었습니다. Amazon EMR 7.11.0(Flink)의 최초 릴리스입니다.

리전: `emr-7.11.0-flink-20251020`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.11.0-flink:20251020`

## Amazon EMR on EKS 7.10.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 7.10.0 릴리스에 대한 일반적인 자세한 내용은 [Amazon EMR 릴리스 안내서의 Amazon EMR 7.10.0](#)을 참조하세요.

## Amazon EMR on EKS 7.10 릴리스

Amazon EMR on EKS에는 다음 Amazon EMR 7.10.0 릴리스를 사용할 수 있습니다. 관련 컨테이너 이미지 태그와 같은 세부 정보를 보려면 특정 `emr-7.10.0-XXXX` 릴리스를 선택합니다.

### Flink releases

Flink 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.10.0 릴리스를 사용할 수 있습니다.

- [emr-7.10.0-flink-latest](#)
- [emr-7.10.0-flink-20250801](#)

### Spark releases

Spark 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.10.0 릴리스를 사용할 수 있습니다.

- [emr-7.10.0-최신](#)
- [emr-7.10.0-20250801](#)
- `emr-7.10.0-spark-rapids-latest`
- `emr-7.10.0-spark-rapids-20250801`

- emr-7.10.0-java11-latest
- emr-7.10.0-java11-20250801
- emr-7.10.0-java8-latest
- emr-7.10.0-java8-20250801
- emr-7.10.0-spark-rapids-java8-latest
- emr-7.10.0-spark-rapids-java8-20250801
- notebook-spark/emr-7.10.0-latest
- notebook-spark/emr-7.10.0-20250801
- notebook-spark/emr-7.10.0-spark-rapids-latest
- notebook-spark/emr-7.10.0-spark-rapids-20250801
- notebook-spark/emr-7.10.0-java11-latest
- notebook-spark/emr-7.10.0-java11-20250801
- notebook-spark/emr-7.10.0-java8-latest
- notebook-spark/emr-7.10.0-java8-20250801
- notebook-spark/emr-7.10.0-spark-rapids-java8-latest
- notebook-spark/emr-7.10.0-spark-rapids-java8-20250801
- notebook-python/emr-7.10.0-latest
- notebook-python/emr-7.10.0-20250801
- notebook-python/emr-7.10.0-spark-rapids-latest
- notebook-python/emr-7.10.0-spark-rapids-20250801
- notebook-python/emr-7.10.0-java11-latest
- notebook-python/emr-7.10.0-java11-20250801
- notebook-python/emr-7.10.0-java8-latest
- notebook-python/emr-7.10.0-java8-20250801
- notebook-python/emr-7.10.0-spark-rapids-java8-latest
- notebook-python/emr-7.10.0-spark-rapids-java8-20250801
- livy/emr-7.10.0-latest
- livy/emr-7.10.0-20250801
- livy/emr-7.10.0-java11-latest
- livy/emr-7.10.0-java11-20250801

- livy/emr-7.10.0-java8-latest
- livy/emr-7.10.0-java8-20250801

## 릴리스 노트

Amazon EMR on EKS 7.10.0 릴리스 정보:

- 지원되는 애플리케이션 - AWS SDK for Java 2.31.48 and 1.12.782, Apache Spark 3.5.5-amzn-1, Apache Hudi 0.15.0-amzn-7, Apache Iceberg 1.8.1-amzn-0, Delta 3.3.0-amzn-2, Apache Spark RAPIDS 25.04.0-amzn-0, Apache Flink 1.20.0-amzn-4, Flink Kubernetes Operator 1.10.0-amzn-4
- 지원되는 구성 요소 -emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                | 설명                                       |
|-------------------|------------------------------------------|
| core-site         | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site        | EMRFS 설정을 변경합니다.                         |
| spark-metrics     | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults    | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env         | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site   | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j2      | Spark log4j2.properties 파일에서 값을 변경합니다.   |
| emr-job-submitter | <a href="#">작업 제출자 포드</a> 구성.            |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 변경 사항 및 기능

Amazon EMR on EKS 7.10.0 릴리스에는 다음과 같은 기능이 포함되어 있습니다.

- S3A 파일 시스템 - 7.10.0 릴리스부터 S3A 파일 시스템은 EMRFS를 기본 EMR S3 커넥터로 대체했습니다. 자세한 내용은 [EMR 파일 시스템\(EMRFS\)](#)을 참조하세요.

## emr-7.10.0-최신

릴리스 정보: emr-7.10.0-latest는 현재 emr-7.10.0-20250801를 가리킵니다.

리전: emr-7.10.0-latest는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-7.10.0:latest

## emr-7.10.0-20250801

릴리스 정보: emr-7.10.0-20250801은 2025년 2월에 릴리스되었습니다. Amazon EMR 7.10.0(Spark)의 최초 릴리스입니다.

리전: `emr-emr-7.10.0-20250801`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.10.0-20250801`

## emr-7.10.0-flink-latest

릴리스 정보: `emr-7.10.0-flink-latest`는 현재 `emr-7.10.0-flink-20250801`을 가리킵니다.

리전: `emr-7.10.0-flink-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.10.0-flink:latest`

## emr-7.10.0-flink-20250801

릴리스 정보: `7.10.0-flink-20250801`은 2025년 2월에 릴리스되었습니다. Amazon EMR 7.10.0(Flink)의 최초 릴리스입니다.

리전: `emr-7.10.0-flink-20250801`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.10.0-flink:20250801`

## Amazon EMR on EKS 7.9.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 7.9.0 릴리스에 대한 일반적인 자세한 내용은 [Amazon EMR 릴리스 안내서의 Amazon EMR 7.9.0](#)을 참조하세요.

## Amazon EMR on EKS 7.9 릴리스

Amazon EMR on EKS에는 다음 Amazon EMR 7.9.0 릴리스를 사용할 수 있습니다. 관련 컨테이너 이미지 태그와 같은 세부 정보를 보려면 특정 `emr-7.9.0-XXXX` 릴리스를 선택합니다.

### Flink releases

Flink 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.9.0 릴리스를 사용할 수 있습니다.

- [emr-7.9.0-flink-latest](#)



- [emr-7.9.0-flink-20250425](#)

## Spark releases

Spark 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.9.0 릴리스를 사용할 수 있습니다.

- [emr-7.9.0-최신](#)
- [emr-7.9.0-20250425](#)
- emr-7.9.0-spark-rapids-latest
- emr-7.9.0-spark-rapids-20250425
- emr-7.9.0-java11-latest
- emr-7.9.0-java11-20250425
- emr-7.9.0-java8-latest
- emr-7.9.0-java8-20250425
- emr-7.9.0-spark-rapids-java8-latest
- emr-7.9.0-spark-rapids-java8-20250425
- notebook-spark/emr-7.9.0-latest
- notebook-spark/emr-7.9.0-20250425
- notebook-spark/emr-7.9.0-spark-rapids-latest
- notebook-spark/emr-7.9.0-spark-rapids-20250425
- notebook-spark/emr-7.9.0-java11-latest
- notebook-spark/emr-7.9.0-java11-20250425
- notebook-spark/emr-7.9.0-java8-latest
- notebook-spark/emr-7.9.0-java8-20250425
- notebook-spark/emr-7.9.0-spark-rapids-java8-latest
- notebook-spark/emr-7.9.0-spark-rapids-java8-20250425
- notebook-python/emr-7.9.0-latest
- notebook-python/emr-7.9.0-20250425
- notebook-python/emr-7.9.0-spark-rapids-latest
- notebook-python/emr-7.9.0-spark-rapids-20250425
- notebook-python/emr-7.9.0-java11-latest

- notebook-python/emr-7.9.0-java11-20250425
- notebook-python/emr-7.9.0-java8-latest
- notebook-python/emr-7.9.0-java8-20250425
- notebook-python/emr-7.9.0-spark-rapids-java8-latest
- notebook-python/emr-7.9.0-spark-rapids-java8-20250425
- livy/emr-7.9.0-latest
- livy/emr-7.9.0-20250425
- livy/emr-7.9.0-java11-latest
- livy/emr-7.9.0-java11-20250425
- livy/emr-7.9.0-java8-latest
- livy/emr-7.9.0-java8-20250425

## 릴리스 노트

### Amazon EMR on EKS 7.9.0 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 2.31.16 and 1.12.782, Apache Spark 3.5.5, Apache Hudi 0.15.0-amzn-6, Apache Iceberg 1.7.1-amzn-2, Delta 3.3.0-amzn-1, Apache Spark RAPIDS 25.02.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.20.0-amzn-3, Flink Operator 1.10.0-amzn-3
- 지원되는 구성 요소 -emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류            | 설명                                      |
|---------------|-----------------------------------------|
| core-site     | Hadoop core-site.xml 파일에서 값을 변경합니다.     |
| emrfs-site    | EMRFS 설정을 변경합니다.                        |
| spark-metrics | Spark metrics.properties 파일에서 값을 변경합니다. |

| 분류                | 설명                                       |
|-------------------|------------------------------------------|
| spark-defaults    | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env         | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site   | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j2      | Spark log4j2.properties 파일에서 값을 변경합니다.   |
| emr-job-submitter | <a href="#">작업 제출자 포트</a> 구성.            |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 변경 사항

Amazon EMR on EKS 7.9.0 릴리스에는 다음과 같은 변경 사항이 포함되어 있습니다.

- 이 릴리스에는 변경 사항이 없습니다.

## emr-7.9.0-최신

릴리스 정보: `emr-7.9.0-latest`는 현재 `emr-7.9.0-20250425`를 가리킵니다.

리전: `emr-7.9.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.9.0:latest`

## emr-7.9.0-20250425

릴리스 정보: `emr-7.9.0-20250425`은 2025년 2월에 릴리스되었습니다. Amazon EMR 7.9.0(Spark)의 최초 릴리스입니다.

리전: `emr-emr-7.9.0-20250425`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.9.0-20250425`

## emr-7.9.0-flink-latest

릴리스 정보: `emr-7.9.0-flink-latest`는 현재 `emr-7.9.0-flink-20250425`을 가리킵니다.

리전: `emr-7.9.0-flink-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.9.0-flink:latest`

## emr-7.9.0-flink-20250425

릴리스 정보: `7.9.0-flink-20250425`은 2025년 2월에 릴리스되었습니다. Amazon EMR 7.9.0(Flink)의 최초 릴리스입니다.

리전: `emr-7.9.0-flink-20250425`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.9.0-flink:20250425`

# Amazon EMR on EKS 7.8.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 7.8.0 릴리스에 대한 일반적인 자세한 내용은 [Amazon EMR 릴리스 안내서의 Amazon EMR 7.8.0](#)을 참조하세요.

## Amazon EMR on EKS 7.8 릴리스

Amazon EMR on EKS에는 다음 Amazon EMR 7.8.0 릴리스를 사용할 수 있습니다. 관련 컨테이너 이미지 태그와 같은 세부 정보를 보려면 특정 emr-7.8.0-XXXX 릴리스를 선택합니다.

### Flink releases

Flink 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.8.0 릴리스를 사용할 수 있습니다.

- [emr-7.8.0-flink-latest](#)
- [emr-7.8.0-flink-20250228](#)

### Spark releases

Spark 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.8.0 릴리스를 사용할 수 있습니다.

- [emr-7.8.0-최신](#)
- [emr-7.8.0-20250228](#)
- emr-7.8.0-spark-rapids-latest
- emr-7.8.0-spark-rapids-20250228
- emr-7.8.0-java11-latest
- emr-7.8.0-java11-20250228
- emr-7.8.0-java8-latest
- emr-7.8.0-java8-20250228
- emr-7.8.0-spark-rapids-java8-latest
- emr-7.8.0-spark-rapids-java8-20250228
- notebook-spark/emr-7.8.0-latest
- notebook-spark/emr-7.8.0-20250228

- notebook-spark/emr-7.8.0-spark-rapids-latest
- notebook-spark/emr-7.8.0-spark-rapids-20250228
- notebook-spark/emr-7.8.0-java11-latest
- notebook-spark/emr-7.8.0-java11-20250228
- notebook-spark/emr-7.8.0-java8-latest
- notebook-spark/emr-7.8.0-java8-20250228
- notebook-spark/emr-7.8.0-spark-rapids-java8-latest
- notebook-spark/emr-7.8.0-spark-rapids-java8-20250228
- notebook-python/emr-7.8.0-latest
- notebook-python/emr-7.8.0-20250228
- notebook-python/emr-7.8.0-spark-rapids-latest
- notebook-python/emr-7.8.0-spark-rapids-20250228
- notebook-python/emr-7.8.0-java11-latest
- notebook-python/emr-7.8.0-java11-20250228
- notebook-python/emr-7.8.0-java8-latest
- notebook-python/emr-7.8.0-java8-20250228
- notebook-python/emr-7.8.0-spark-rapids-java8-latest
- notebook-python/emr-7.8.0-spark-rapids-java8-20250228
- livy/emr-7.8.0-latest
- livy/emr-7.8.0-20250228
- livy/emr-7.8.0-java11-latest
- livy/emr-7.8.0-java11-20250228
- livy/emr-7.8.0-java8-latest
- livy/emr-7.8.0-java8-20250228

## 릴리스 노트

### Amazon EMR on EKS 7.8.0 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 2.29.52 and 1.12.780, Apache Spark 3.5.4, Apache Hudi 0.15.0-amzn-5, Apache Iceberg 1.7.1-amzn-1, Delta 3.3.0-amzn-0, Apache Spark RAPIDS

24.12.0-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.20.0-amzn-2, Flink Operator 1.10.0-amzn-2

- 지원되는 구성 요소 -emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                | 설명                                       |
|-------------------|------------------------------------------|
| core-site         | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site        | EMRFS 설정을 변경합니다.                         |
| spark-metrics     | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults    | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env         | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site   | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j2      | Spark log4j2.properties 파일에서 값을 변경합니다.   |
| emr-job-submitter | <a href="#">작업 제출자 포트</a> 구성.            |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 변경 사항

Amazon EMR on EKS 7.8.0 릴리스에는 다음과 같은 변경 사항이 포함되어 있습니다.

- 다음을 포함한 네이티브 FGAC 기능:
  - Iceberg는 세분화된 액세스 제어(FGAC) 가상 클러스터에서 지연되지 않는 형성 테이블에서 작업을 수행하는 작업을 실행할 수 있도록 지원합니다. (IAM에 대한 대체가 있습니다.)
  - S3 테이블 지원
- Spark 연결

## emr-7.8.0-최신

릴리스 정보: emr-7.8.0-latest는 현재 emr-7.8.0-20250228를 가리킵니다.

리전: emr-7.8.0-latest는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-7.8.0:latest

## emr-7.8.0-20250228

릴리스 정보: emr-7.8.0-20250228은 2025년 2월에 릴리스되었습니다. Amazon EMR 7.8.0(Spark)의 최초 릴리스입니다.



리전: `emr-emr-7.8.0-20250228`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.8.0-20250228`

## emr-7.8.0-flink-latest

릴리스 정보: `emr-7.8.0-flink-latest`는 현재 `emr-7.8.0-flink-20250228`을 가리킵니다.

리전: `emr-7.8.0-flink-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.8.0-flink:latest`

## emr-7.8.0-flink-20250228

릴리스 정보: `7.8.0-flink-20250228`은 2025년 2월에 릴리스되었습니다. Amazon EMR 7.8.0(Flink)의 최초 릴리스입니다.

리전: `emr-7.8.0-flink-20250228`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.8.0-flink:20250228`

## Amazon EMR on EKS 7.7.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 7.7.0 릴리스에 대한 일반적인 자세한 내용은 [Amazon EMR 릴리스 안내서의 Amazon EMR 7.7.0](#)을 참조하세요.

## Amazon EMR on EKS 7.7 릴리스

Amazon EMR on EKS에는 다음 Amazon EMR 7.7.0 릴리스를 사용할 수 있습니다. 관련 컨테이너 이미지 태그와 같은 세부 정보를 보려면 특정 `emr-7.7.0-XXXX` 릴리스를 선택합니다.

### Flink releases

Flink 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.7.0 릴리스를 사용할 수 있습니다.

- [emr-7.7.0-flink-latest](#)

- [emr-7.7.0-flink-20250131](#)

## Spark releases

Spark 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.7.0 릴리스를 사용할 수 있습니다.

- [emr-7.7.0-최신](#)
- [emr-7.7.0-20250131](#)
- emr-7.7.0-spark-rapids-latest
- emr-7.7.0-spark-rapids-20250131
- emr-7.7.0-java11-latest
- emr-7.7.0-java11-20250131
- emr-7.7.0-java8-latest
- emr-7.7.0-java8-20250131
- emr-7.7.0-spark-rapids-java8-latest
- emr-7.7.0-spark-rapids-java8-20250131
- notebook-spark/emr-7.7.0-latest
- notebook-spark/emr-7.7.0-20250131
- notebook-spark/emr-7.7.0-spark-rapids-latest
- notebook-spark/emr-7.7.0-spark-rapids-20250131
- notebook-spark/emr-7.7.0-java11-latest
- notebook-spark/emr-7.7.0-java11-20250131
- notebook-spark/emr-7.7.0-java8-latest
- notebook-spark/emr-7.7.0-java8-20250131
- notebook-spark/emr-7.7.0-spark-rapids-java8-latest
- notebook-spark/emr-7.7.0-spark-rapids-java8-20250131
- notebook-python/emr-7.7.0-latest
- notebook-python/emr-7.7.0-20250131
- notebook-python/emr-7.7.0-spark-rapids-latest
- notebook-python/emr-7.7.0-spark-rapids-20250131
- notebook-python/emr-7.7.0-java11-latest

- notebook-python/emr-7.7.0-java11-20250131
- notebook-python/emr-7.7.0-java8-latest
- notebook-python/emr-7.7.0-java8-20250131
- notebook-python/emr-7.7.0-spark-rapids-java8-latest
- notebook-python/emr-7.7.0-spark-rapids-java8-20250131
- livy/emr-7.7.0-latest
- livy/emr-7.7.0-20250131
- livy/emr-7.7.0-java11-latest
- livy/emr-7.7.0-java11-20250131
- livy/emr-7.7.0-java8-latest
- livy/emr-7.7.0-java8-20250131

## 릴리스 노트

### Amazon EMR on EKS 7.7.0 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 2.29.25 and 1.12.779, Apache Spark 3.5.3-amzn-0, Apache Hudi 0.15.0-amzn-3, Apache Iceberg 1.6.1-amzn-2, Delta 3.2.1-amzn-1, Apache Spark RAPIDS 24.10.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.20.0-amzn-0, Flink Operator 1.10.0-amzn-0
- 지원되는 구성 요소 -emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류            | 설명                                      |
|---------------|-----------------------------------------|
| core-site     | Hadoop core-site.xml 파일에서 값을 변경합니다.     |
| emrfs-site    | EMRFS 설정을 변경합니다.                        |
| spark-metrics | Spark metrics.properties 파일에서 값을 변경합니다. |

| 분류                | 설명                                       |
|-------------------|------------------------------------------|
| spark-defaults    | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env         | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site   | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j2      | Spark log4j2.properties 파일에서 값을 변경합니다.   |
| emr-job-submitter | <a href="#">작업 제출자 포트</a> 구성.            |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 변경 사항

Amazon EMR on EKS 7.7.0 릴리스에는 다음과 같은 변경 사항이 포함되어 있습니다.

- EMR 7.7.0부터 사용 중인 Iceberg 버전이 더 이상 Java 8을 지원하지 않습니다. 또한 Iceberg는 Java 8 이미지 emr-7.7.0-java8-latest 및 emr-7.7.0-spark-rapids-java8-latest에서 제외됩니다.

## emr-7.7.0-최신

릴리스 정보: `emr-7.7.0-latest`는 현재 `emr-7.7.0-20250131`를 가리킵니다.

리전: `emr-7.7.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.7.0:latest`

## emr-7.7.0-20250131

릴리스 정보: `emr-7.7.0-20250131`은 2025년 2월에 릴리스되었습니다. Amazon EMR 7.7.0(Spark)의 최초 릴리스입니다.

리전: `emr-emr-7.7.0-20250131`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.7.0-20250131`

## emr-7.7.0-flink-latest

릴리스 정보: `emr-7.7.0-flink-latest`는 현재 `emr-7.7.0-flink-20250131`을 가리킵니다.

리전: `emr-7.7.0-flink-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.7.0-flink:latest`

## emr-7.7.0-flink-20250131

릴리스 정보: `7.7.0-flink-20250131`은 2025년 2월에 릴리스되었습니다. Amazon EMR 7.7.0(Flink)의 최초 릴리스입니다.

리전: `emr-7.7.0-flink-20250131`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.7.0-flink:20250131`

# Amazon EMR on EKS 7.6.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 7.6.0 릴리스에 대한 일반적인 자세한 내용은 [Amazon EMR 릴리스 안내서의 Amazon EMR 7.6.0](#)을 참조하세요.

## Amazon EMR on EKS 7.6 릴리스

Amazon EMR on EKS에는 다음 Amazon EMR 7.6.0 릴리스를 사용할 수 있습니다. 관련 컨테이너 이미지 태그와 같은 세부 정보를 보려면 특정 emr-7.6.0-XXXX 릴리스를 선택합니다.

### Flink releases

Flink 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.6.0 릴리스를 사용할 수 있습니다.

- [emr-7.6.0-flink-latest](#)
- [emr-7.6.0-flink-20241213](#)

### Spark releases

Spark 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.6.0 릴리스를 사용할 수 있습니다.

- [emr-7.6.0-최신](#)
- [emr-7.6.0-20241213](#)
- emr-7.6.0-spark-rapids-latest
- emr-7.6.0-spark-rapids-20241213
- emr-7.6.0-java11-latest
- emr-7.6.0-java11-20241213
- emr-7.6.0-java8-latest
- emr-7.6.0-java8-20241213
- emr-7.6.0-spark-rapids-java8-latest
- emr-7.6.0-spark-rapids-java8-20241213
- notebook-spark/emr-7.6.0-latest
- notebook-spark/emr-7.6.0-20241213

- notebook-spark/emr-7.6.0-spark-rapids-latest
- notebook-spark/emr-7.6.0-spark-rapids-20241213
- notebook-spark/emr-7.6.0-java11-latest
- notebook-spark/emr-7.6.0-java11-20241213
- notebook-spark/emr-7.6.0-java8-latest
- notebook-spark/emr-7.6.0-java8-20241213
- notebook-spark/emr-7.6.0-spark-rapids-java8-latest
- notebook-spark/emr-7.6.0-spark-rapids-java8-20241213
- notebook-python/emr-7.6.0-latest
- notebook-python/emr-7.6.0-20241213
- notebook-python/emr-7.6.0-spark-rapids-latest
- notebook-python/emr-7.6.0-spark-rapids-20241213
- notebook-python/emr-7.6.0-java11-latest
- notebook-python/emr-7.6.0-java11-20241213
- notebook-python/emr-7.6.0-java8-latest
- notebook-python/emr-7.6.0-java8-20241213
- notebook-python/emr-7.6.0-spark-rapids-java8-latest
- notebook-python/emr-7.6.0-spark-rapids-java8-20241213
- livy/emr-7.6.0-latest
- livy/emr-7.6.0-20241213
- livy/emr-7.6.0-java11-latest
- livy/emr-7.6.0-java11-20241213
- livy/emr-7.6.0-java8-latest
- livy/emr-7.6.0-java8-20241213

## 릴리스 노트

### Amazon EMR on EKS 7.6.0 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 2.29.25 and 1.12.779, Apache Spark 3.5.3-amzn-0, Apache Hudi 0.15.0-amzn-3, Apache Iceberg 1.6.1-amzn-2, Delta 3.2.1-amzn-1, Apache Spark

RAPIDS 24.10.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.20.0-amzn-0, Flink Operator 1.10.0-amzn-0

- 지원되는 구성 요소 - aws-sagemaker-spark-sdk, emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                | 설명                                       |
|-------------------|------------------------------------------|
| core-site         | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site        | EMRFS 설정을 변경합니다.                         |
| spark-metrics     | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults    | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env         | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site   | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j2      | Spark log4j2.properties 파일에서 값을 변경합니다.   |
| emr-job-submitter | <a href="#">작업 제출자 포트</a> 구성.            |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:



| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 주목할 만한 기능

Amazon EMR on EKS 7.6.0 릴리스에는 다음과 같은 기능이 포함되어 있습니다.

- Apache Spark 운영자에 대한 구성 모니터링 지원 - 구성을 모니터링하면 Amazon S3 또는 Amazon CloudWatch에 대한 Spark 애플리케이션 및 운영자 로그의 로그 아카이빙을 쉽게 설정할 수 있습니다. 사용자는 하나 또는 둘 다를 선택할 수 있습니다. 이 작업을 수행하면 Spark 운영자 포드, 드라이버 및 실행기 포드에 로그 에이전트 사이드카가 추가되고 이후에 이러한 구성 요소의 로그가 구성된 싱크에 전달됩니다. 자세한 내용은 [모니터링 구성을 사용하여 Spark Kubernetes 운영자 및 Spark 작업 모니터링](#)을 참조하세요.

## 변경 사항

Amazon EMR on EKS 7.6.0 릴리스에는 다음과 같은 변경 사항이 포함되어 있습니다.

- 이 릴리스에는 변경 사항이 없습니다.

## emr-7.6.0-최신

릴리스 정보: emr-7.6.0-latest는 현재 emr-7.6.0-20241213를 가리킵니다.

리전: `emr-7.6.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.6.0:latest`

## emr-7.6.0-20241213

릴리스 정보: `7.6.0-20241213` 버전은 2024년 1월에 릴리스되었습니다. Amazon EMR 7.6.0(Spark)의 최초 릴리스입니다.

리전: `emr-7.6.0-20241213`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.6.0:20241213`

## emr-7.6.0-flink-latest

릴리스 정보: `emr-7.6.0-flink-latest`는 현재 `emr-7.6.0-flink-20241213`을 가리킵니다.

리전: `emr-7.6.0-flink-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.6.0-flink:latest`

## emr-7.6.0-flink-20241213

릴리스 정보: `7.6.0-flink-20241213` 버전은 2024년 1월에 릴리스되었습니다. Amazon EMR 7.6.0(Flink)의 최초 릴리스입니다.

리전: `emr-7.6.0-flink-20241213`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.6.0-flink:20241213`

## Amazon EMR on EKS 7.5.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 7.5.0 릴리스에 대한 일반적인 자세한 내용은 [Amazon EMR 릴리스 안내서의 Amazon EMR 7.5.0](#)을 참조하세요.

## Amazon EMR on EKS 7.5 릴리스

Amazon EMR on EKS에는 다음 Amazon EMR 7.5.0 릴리스를 사용할 수 있습니다. 관련 컨테이너 이미지 태그와 같은 세부 정보를 보려면 특정 `emr-7.5.0-XXXX` 릴리스를 선택합니다.

### 릴리스 노트

#### Amazon EMR on EKS 7.5.0 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 2.28.8 and 1.12.772, Apache Spark 3.5.2-amzn-1, Apache Hudi 0.15.0-amzn-1, Apache Iceberg 1.6.1-amzn-0, Delta 3.2.0-amzn-1, Apache Spark RAPIDS 24.08.1-amzn-1, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.19.1-amzn-1, Flink Operator 1.9.0-amzn-0
- 지원되는 구성 요소 - `aws-sagemaker-spark-sdk`, `emr-ddb`, `emr-goodies`, `emr-s3-select`, `emrfs`, `hadoop-client`, `hudi`, `hudi-spark`, `iceberg`, `spark-kubernetes`.

## Amazon EMR on EKS 7.4.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 7.4.0 릴리스에 대한 일반적인 자세한 내용은 [Amazon EMR 릴리스 안내서의 Amazon EMR 7.4.0](#)을 참조하세요.

## Amazon EMR on EKS 7.4 릴리스

Amazon EMR on EKS에는 다음 Amazon EMR 7.4.0 릴리스를 사용할 수 있습니다. 관련 컨테이너 이미지 태그와 같은 세부 정보를 보려면 특정 `emr-7.4.0-XXXX` 릴리스를 선택합니다.

### 릴리스 노트

#### Amazon EMR on EKS 7.4.0 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 2.25.70 and 1.12.772, Apache Spark 3.5.2-amzn-0, Apache Hudi 0.15.0-amzn-1, Apache Iceberg 1.6.1-amzn-0, Delta 3.2.0-amzn-1, Apache Spark RAPIDS 24.08.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.19.1-amzn-0, Flink Operator 1.9.0-amzn-1
- 지원되는 구성 요소 - `aws-sagemaker-spark-sdk`, `emr-ddb`, `emr-goodies`, `emr-s3-select`, `emrfs`, `hadoop-client`, `hudi`, `hudi-spark`, `iceberg`, `spark-kubernetes`.

# Amazon EMR on EKS 7.3.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 7.3.0 릴리스에 대한 일반적인 세부 정보는 Amazon EMR 릴리스 안내서에서 [Amazon EMR 7.3.0](#)을 참조하세요.

## Amazon EMR on EKS 7.3 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 7.3.0 릴리스를 사용할 수 있습니다. 특정 emr-7.3.0-XXXX 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

### Flink releases

Flink 애플리케이션을 실행하는 경우 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.3.0 릴리스를 사용할 수 있습니다.

- [emr-7.3.0-flink-latest](#)
- [emr-7.3.0-flink-29240920](#)

### Spark releases

Spark 애플리케이션을 실행하는 경우 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.3.0 릴리스를 사용할 수 있습니다.

- [emr-7.3.0-latest](#)
- [emr-7.3.0-20240920](#)
- emr-7.3.0-spark-rapids-latest
- emr-7.3.0-spark-rapids-29240920
- emr-7.3.0-java11-latest
- emr-7.3.0-java11-29240920
- emr-7.3.0-java8-latest
- emr-7.3.0-java8-29240920
- emr-7.3.0-spark-rapids-java8-latest
- emr-7.3.0-spark-rapids-java8-29240920
- notebook-spark/emr-7.3.0-latest
- notebook-spark/emr-7.3.0-20240920

- notebook-spark/emr-7.3.0-spark-rapids-latest
- notebook-spark/emr-7.3.0-spark-rapids-29240920
- notebook-spark/emr-7.3.0-java11-latest
- notebook-spark/emr-7.3.0-java11-29240920
- notebook-spark/emr-7.3.0-java8-latest
- notebook-spark/emr-7.3.0-java8-29240920
- notebook-spark/emr-7.3.0-spark-rapids-java8-latest
- notebook-spark/emr-7.3.0-spark-rapids-java8-29240920
- notebook-python/emr-7.3.0-latest
- notebook-python/emr-7.3.0-20240920
- notebook-python/emr-7.3.0-spark-rapids-latest
- notebook-python/emr-7.3.0-spark-rapids-29240920
- notebook-python/emr-7.3.0-java11-latest
- notebook-python/emr-7.3.0-java11-29240920
- notebook-python/emr-7.3.0-java8-latest
- notebook-python/emr-7.3.0-java8-29240920
- notebook-python/emr-7.3.0-spark-rapids-java8-latest
- notebook-python/emr-7.3.0-spark-rapids-java8-29240920
- livy/emr-7.3.0-latest
- livy/emr-7.3.0-20240920
- livy/emr-7.3.0-java11-latest
- livy/emr-7.3.0-java11-29240920
- livy/emr-7.3.0-java8-latest
- livy/emr-7.3.0-java8-29240920

## 릴리스 노트

### Amazon EMR on EKS 7.3.0 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 2.25.70 and 1.12.747, Apache Spark 3.5.1-amzn-1, Apache Hudi 0.15.0-amzn-0, Apache Iceberg 1.5.2-amzn-0, Delta 3.2.0-amzn-0, Apache Spark

RAPIDS 24.06.1-amzn-0, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.18.1-amzn-2, Flink Operator 1.9.0-amzn-0

- 지원되는 구성 요소 - aws-sagemaker-spark-sdk, emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                | 설명                                       |
|-------------------|------------------------------------------|
| core-site         | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site        | EMRFS 설정을 변경합니다.                         |
| spark-metrics     | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults    | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env         | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site   | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j2      | Spark log4j2.properties 파일에서 값을 변경합니다.   |
| emr-job-submitter | <a href="#">작업 제출자 포트</a> 구성.            |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 주목할 만한 기능

Amazon EMR on EKS 7.3.0 릴리스에 다음과 같은 기능이 포함되어 있습니다.

- 애플리케이션 업그레이드 - 이제 Amazon EMR on EKS에는 [Flink 연산자](#) 1.9.0이 포함됩니다. 다른 기능 외에도 이제 Flink Kubernetes에서는 오토스케일러에 대한 CPU 및 메모리 할당량을 설정할 수 있습니다.
- Apache Flink에 대한 Apache Iceberg 지원 - Apache Iceberg는 오픈 소스 고성능 형식의 대규모 분석 테이블입니다. Amazon EMR 7.3.0부터 Amazon EMR on EKS에서 Apache Flink를 실행하는 경우 Apache Iceberg 테이블을 사용할 수 있습니다. 자세한 내용은 Amazon EMR on EKS의 [Amazon EMR on EKS에서 Apache Iceberg 사용](#)을 참조하세요.
- Apache Flink에 대한 Delta Lake 지원 - Delta Lake는 일반적으로 Amazon S3에 구축되는 레이크 하우스 아키텍처를 위한 스토리지 계층 프레임워크입니다. Amazon EMR 7.3.0 이상에서는 Amazon EMR on EKS에서 Apache Flink를 실행하는 경우 Delta 테이블을 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS에서 Delta Lake 사용](#)을 참조하세요.

## 변경 사항

Amazon EMR on EKS 7.3.0 릴리스에 다음과 같은 변경 사항이 포함되어 있습니다.

- Amazon EMR on EKS 7.3.0 이상에서 이제 Apache Flink는 기본적으로 Java 17 런타임을 사용합니다.

## emr-7.3.0-latest

릴리스 정보: `emr-7.3.0-latest`는 현재 `emr-7.3.0-20240920`를 가리킵니다.

리전: `emr-7.3.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.3.0:latest`

## emr-7.3.0-20240920

릴리스 정보: `7.3.0-20240920`은 2023년 12월에 출시되었습니다. Amazon EMR 7.3.0(Spark)의 최초 릴리스입니다.

리전: `emr-7.3.0-20240920`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.3.0:29240920`

## emr-7.3.0-flink-latest

릴리스 정보: `emr-7.3.0-flink-latest`는 현재 `emr-7.3.0-flink-29240920`를 가리킵니다.

리전: `emr-7.3.0-flink-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.3.0-flink:latest`

## emr-7.3.0-flink-29240920

릴리스 정보: `7.3.0-flink-29240920`은 2023년 12월에 출시되었습니다. Amazon EMR 7.3.0(Flink)의 최초 릴리스입니다.

리전: `emr-7.3.0-flink-29240920`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.3.0-flink:29240920`



# Amazon EMR on EKS 7.2.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 7.2.0 릴리스에 대한 일반적인 세부 정보는 Amazon EMR 릴리스 안내서에서 [Amazon EMR 7.2.0](#)을 참조하세요.

## Amazon EMR on EKS 7.2 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 7.2.0 릴리스를 사용할 수 있습니다. 특정 emr-7.2.0-XXXX 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

### Flink releases

Flink 애플리케이션을 실행하는 경우 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.2.0 릴리스를 사용할 수 있습니다.

- [emr-7.2.0-flink-latest](#)
- [emr-7.2.0-flink-20240610](#)

### Spark releases

Spark 애플리케이션을 실행하는 경우 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.2.0 릴리스를 사용할 수 있습니다.

- [emr-7.2.0-latest](#)
- [emr-7.2.0-20240610](#)
- emr-7.2.0-spark-rapids-latest
- emr-7.2.0-spark-rapids-20240610
- emr-7.2.0-java11-latest
- emr-7.2.0-java11-20240610
- emr-7.2.0-java8-latest
- emr-7.2.0-java8-20240610
- emr-7.2.0-spark-rapids-java8-latest
- emr-7.2.0-spark-rapids-java8-20240610
- notebook-spark/emr-7.2.0-latest
- notebook-spark/emr-7.2.0-20240610

- notebook-spark/emr-7.2.0-spark-rapids-latest
- notebook-spark/emr-7.2.0-spark-rapids-20240610
- notebook-spark/emr-7.2.0-java11-latest
- notebook-spark/emr-7.2.0-java11-20240610
- notebook-spark/emr-7.2.0-java8-latest
- notebook-spark/emr-7.2.0-java8-20240610
- notebook-spark/emr-7.2.0-spark-rapids-java8-latest
- notebook-spark/emr-7.2.0-spark-rapids-java8-20240610
- notebook-python/emr-7.2.0-latest
- notebook-python/emr-7.2.0-20240610
- notebook-python/emr-7.2.0-spark-rapids-latest
- notebook-python/emr-7.2.0-spark-rapids-20240610
- notebook-python/emr-7.2.0-java11-latest
- notebook-python/emr-7.2.0-java11-20240610
- notebook-python/emr-7.2.0-java8-latest
- notebook-python/emr-7.2.0-java8-20240610
- notebook-python/emr-7.2.0-spark-rapids-java8-latest
- notebook-python/emr-7.2.0-spark-rapids-java8-20240610
- livy/emr-7.2.0-latest
- livy/emr-7.2.0-20240610
- livy/emr-7.2.0-java11-latest
- livy/emr-7.2.0-java11-20240610
- livy/emr-7.2.0-java8-latest
- livy/emr-7.2.0-java8-20240610

## 릴리스 노트

### Amazon EMR on EKS 7.2.0 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 2.23.18 and 1.12.705, Apache Spark 3.5.1-amzn-1, Apache Hudi 0.14.1-amzn-0, Apache Iceberg 1.5.0-amzn-0, Delta 3.1.0, Apache Spark RAPIDS

24.02.0-amzn-1, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.18.1-amzn-0, Flink Operator 1.8.0-amzn-1

- 지원되는 구성 요소 - aws-sagemaker-spark-sdk, emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                | 설명                                       |
|-------------------|------------------------------------------|
| core-site         | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site        | EMRFS 설정을 변경합니다.                         |
| spark-metrics     | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults    | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env         | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site   | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j2      | Spark log4j2.properties 파일에서 값을 변경합니다.   |
| emr-job-submitter | <a href="#">작업 제출자 포트</a> 구성.            |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 주목할 만한 기능

Amazon EMR on EKS 7.2.0 릴리스에 다음과 같은 기능이 포함되어 있습니다.

- 애플리케이션 업그레이드 – Amazon EMR on EKS 7.2.0 애플리케이션 업그레이드는 Spark 3.5.1, Flink 1.18.1과 [Flink Operator](#) 1.8.0을 포함합니다.
- [Flink 업데이트를 위한 오토스케일러](#) - 7.2.0 릴리스는 오픈 소스 구성 job.autoscaler.restart.time-tracking.enabled를 사용하여 시간 추정 조정 기능을 활성화하므로 더 이상 재시작 시간에 경험에 기반한 값을 수동으로 할당하지 않아도 됩니다. 7.1.0 이하를 실행하는 경우에도 Amazon EMR 오토스케일링을 사용할 수 있습니다.
- [Amazon EMR on EKS에서 Apache Hudi 통합 Apache Flink](#) - 이 릴리스에서는 Apache Hudi 및 Apache Flink 간 통합을 추가합니다. 이를 통해 Flink Kubernetes 연산자를 사용하여 Hudi 작업을 실행할 수 있습니다. Hudi에서는 데이터 관리 및 데이터 파이프라인 개발을 간소화하는 데 사용할 수 있는 레코드 수준 작업을 사용할 수 있습니다.
- [Amazon EMR on EKS와 Amazon S3 Express One Zone 통합](#) - 7.2.0 이상에서 Amazon EMR on EKS를 사용하여 S3 Express One Zone에 데이터를 업로드할 수 있습니다. S3 Express One Zone은 대부분의 지연 시간에 민감한 애플리케이션에서 일관되게 10밀리초 미만의 데이터 액세스를 지원하는 고성능 단일 영역 Amazon S3 스토리지 클래스입니다. 릴리스 시점에 S3 Express One Zone은 Amazon S3에서 지연 시간이 가장 낮고 성능은 가장 뛰어난 클라우드 객체 스토리지를 제공합니다.

- [Spark 운영자에서 기본 구성에 대한 지원](#) - 이제 Amazon EKS의 Spark 운영자는 7.2.0 이상에 대해 Amazon EMR on EKS의 작업 실행 시작 모델과 동일한 기본 구성을 지원합니다. 즉, Amazon S3 및 EMRFS와 같은 기능에는 더 이상 yaml 파일의 수동 구성이 필요하지 않습니다.

## emr-7.2.0-latest

릴리스 정보: `emr-7.2.0-latest`는 현재 `emr-7.2.0-20240610`를 가리킵니다.

리전: `emr-7.2.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.2.0:latest`

## emr-7.2.0-20240610

릴리스 정보: `7.2.0-20240610`은 2023년 12월에 출시되었습니다. Amazon EMR 7.2.0(Spark)의 최초 릴리스입니다.

리전: `emr-7.2.0-20240610`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.2.0:20240610`

## emr-7.2.0-flink-latest

릴리스 정보: `emr-7.2.0-flink-latest`는 현재 `emr-7.2.0-flink-20240610`를 가리킵니다.

리전: `emr-7.2.0-flink-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.2.0-flink:latest`

## emr-7.2.0-flink-20240610

릴리스 정보: `7.2.0-flink-20240610`은 2023년 12월에 출시되었습니다. Amazon EMR 7.2.0(Flink)의 최초 릴리스입니다.

리전: `emr-7.2.0-flink-20240610`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.2.0-flink:20240610`

# Amazon EMR on EKS 7.1.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 7.1.0 릴리스에 대한 일반적인 세부 정보는 Amazon EMR 릴리스 안내서에서 [Amazon EMR 7.1.0](#)을 참조하세요.

## Amazon EMR on EKS 7.1 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 7.1.0 릴리스를 사용할 수 있습니다. 특정 emr-7.1.0-XXXX 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

### Flink releases

Flink 애플리케이션을 실행하는 경우 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.1.0 릴리스를 사용할 수 있습니다.

- [emr-7.1.0-flink-latest](#)
- [emr-7.1.0-flink-20240321](#)

### Spark releases

Spark 애플리케이션을 실행하는 경우 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.1.0 릴리스를 사용할 수 있습니다.

- [emr-7.1.0-latest](#)
- [emr-7.1.0-20240321](#)
- emr-7.1.0-spark-rapids-latest
- emr-7.1.0-spark-rapids-20240321
- emr-7.1.0-java11-latest
- emr-7.1.0-java11-20240321
- emr-7.1.0-java8-latest
- emr-7.1.0-java8-20240321
- emr-7.1.0-spark-rapids-java8-latest
- emr-7.1.0-spark-rapids-java8-20240321
- notebook-spark/emr-7.1.0-latest
- notebook-spark/emr-7.1.0-20240321

- notebook-spark/emr-7.1.0-spark-rapids-latest
- notebook-spark/emr-7.1.0-spark-rapids-20240321
- notebook-spark/emr-7.1.0-java11-latest
- notebook-spark/emr-7.1.0-java11-20240321
- notebook-spark/emr-7.1.0-java8-latest
- notebook-spark/emr-7.1.0-java8-20240321
- notebook-spark/emr-7.1.0-spark-rapids-java8-latest
- notebook-spark/emr-7.1.0-spark-rapids-java8-20240321
- notebook-python/emr-7.1.0-latest
- notebook-python/emr-7.1.0-20240321
- notebook-python/emr-7.1.0-spark-rapids-latest
- notebook-python/emr-7.1.0-spark-rapids-20240321
- notebook-python/emr-7.1.0-java11-latest
- notebook-python/emr-7.1.0-java11-20240321
- notebook-python/emr-7.1.0-java8-latest
- notebook-python/emr-7.1.0-java8-20240321
- notebook-python/emr-7.1.0-spark-rapids-java8-latest
- notebook-python/emr-7.1.0-spark-rapids-java8-20240321
- livy/emr-7.1.0-latest
- livy/emr-7.1.0-20240321
- livy/emr-7.1.0-java11-latest
- livy/emr-7.1.0-java11-20240321
- livy/emr-7.1.0-java8-latest
- livy/emr-7.1.0-java8-20240321

## 릴리스 노트

### Amazon EMR on EKS 7.1.0 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 2.23.18 and 1.12.656, Apache Spark 3.5.0-amzn-1, Apache Hudi 0.14.1-amzn-0, Apache Iceberg 1.4.3-amzn-0, Delta 3.0.0, Apache Spark RAPIDS

23.10.0-amzn-1, Jupyter Enterprise Gateway 2.6.0, Apache Flink 1.18.1-amzn-0, Flink Operator 1.6.1-amzn-1

- 지원되는 구성 요소 - aws-sagemaker-spark-sdk, emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                | 설명                                       |
|-------------------|------------------------------------------|
| core-site         | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site        | EMRFS 설정을 변경합니다.                         |
| spark-metrics     | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults    | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env         | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site   | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j2      | Spark log4j2.properties 파일에서 값을 변경합니다.   |
| emr-job-submitter | <a href="#">작업 제출자 포트</a> 구성.            |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:



| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 주목할 만한 기능

Amazon EMR on EKS 7.1.0 릴리스에 다음과 같은 기능이 포함되어 있습니다.

- [Amazon EMR on EKS에 대한 Apache Livy 지원](#) - Amazon EMR on EKS 릴리스 7.1.0 이상에서는 Amazon EKS 클러스터에서 Apache Livy를 사용하여 Spark 작업 또는 Spark 코드 조각을 제출하도록 Apache Livy REST 인터페이스를 생성할 수 있습니다. 이를 통해 Amazon EMR 최적화 Spark 런타임, SSL 지원 Livy 엔드포인트 및 프로그래밍 방식의 설정 환경과 같은 Amazon EMR on EKS 이점을 계속 활용하면서 동기식 및 비동기식으로 결과를 검색할 수 있습니다.

## emr-7.1.0-latest

릴리스 정보: emr-7.1.0-latest는 현재 emr-7.1.0-20240321를 가리킵니다.

리전: emr-7.1.0-latest는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-7.1.0:latest

## emr-7.1.0-20240321

릴리스 정보: 7.1.0-20240321은 2023년 12월에 출시되었습니다. Amazon EMR 7.1.0(Spark)의 최초 릴리스입니다.

리전: `emr-7.1.0-20240321`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.1.0:20240321`

## emr-7.1.0-flink-latest

릴리스 정보: `emr-7.1.0-flink-latest`는 현재 `emr-7.1.0-flink-20240321`를 가리킵니다.

리전: `emr-7.1.0-flink-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.1.0-flink:latest`

## emr-7.1.0-flink-20240321

릴리스 정보: `7.1.0-flink-20240321`은 2023년 12월에 출시되었습니다. Amazon EMR 7.1.0(Flink)의 최초 릴리스입니다.

리전: `emr-7.1.0-flink-20240321`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.1.0-flink:20240321`

# Amazon EMR on EKS 7.0.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 7.0.0 릴리스에 대한 일반적인 세부 정보는 Amazon EMR 릴리스 안내서에서 [Amazon EMR 7.0.0](#)을 참조하세요.

## Amazon EMR on EKS 7.0 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 7.0.0 릴리스를 사용할 수 있습니다. 특정 `emr-7.0.0-XXXX` 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

### Flink releases

Flink 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.0.0 릴리스를 사용할 수 있습니다.

- [emr-7.0.0-flink-latest](#)
- [emr-7.0.0-flink-2024321](#)

- [emr-7.0.0-flink-20231211](#)

## Spark releases

Spark 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 7.0.0 릴리스를 사용할 수 있습니다.

- [emr-7.0.0-latest](#)
- [emr-7.0.0-20231211](#)
- emr-7.0.0-spark-rapids-latest
- emr-7.0.0-spark-rapids-20231211
- emr-7.0.0-java11-latest
- emr-7.0.0-java11-20231211
- emr-7.0.0-java8-latest
- emr-7.0.0-java8-20231211
- emr-7.0.0-spark-rapids-java8-latest
- emr-7.0.0-spark-rapids-java8-20231211
- notebook-spark/emr-7.0.0-latest
- notebook-spark/emr-7.0.0-20231211
- notebook-spark/emr-7.0.0-spark-rapids-latest
- notebook-spark/emr-7.0.0-spark-rapids-20231211
- notebook-spark/emr-7.0.0-java11-latest
- notebook-spark/emr-7.0.0-java11-20231211
- notebook-spark/emr-7.0.0-java8-latest
- notebook-spark/emr-7.0.0-java8-20231211
- notebook-spark/emr-7.0.0-spark-rapids-java8-latest
- notebook-spark/emr-7.0.0-spark-rapids-java8-20231211
- notebook-python/emr-7.0.0-latest
- notebook-python/emr-7.0.0-20231211
- notebook-python/emr-7.0.0-spark-rapids-latest
- notebook-python/emr-7.0.0-spark-rapids-20231211
- notebook-python/emr-7.0.0-java11-latest

- notebook-python/emr-7.0.0-java11-20231211
- notebook-python/emr-7.0.0-java8-latest
- notebook-python/emr-7.0.0-java8-20231211
- notebook-python/emr-7.0.0-spark-rapids-java8-latest
- notebook-python/emr-7.0.0-spark-rapids-java8-20231211

## 릴리스 노트

### Amazon EMR on EKS 7.0.0 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 2.20.160-amzn-0 and 1.12.595, Apache Spark 3.5.0-amzn-0, Apache Flink 1.18.0-amzn-0, Flink Operator 1.6.1, Apache Hudi 0.14.0-amzn-1, Apache Iceberg 1.4.2-amzn-0, Delta 3.0.0, Apache Spark RAPIDS 23.10.0-amzn-0, Jupyter Enterprise Gateway 2.6.0
- 지원되는 구성 요소 - aws-sagemaker-spark-sdk, emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류              | 설명                                       |
|-----------------|------------------------------------------|
| core-site       | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site      | EMRFS 설정을 변경합니다.                         |
| spark-metrics   | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults  | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env       | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site | Spark hive-site.xml 파일에서 값을 변경합니다.       |

| 분류                | 설명                                     |
|-------------------|----------------------------------------|
| spark-log4j       | Spark log4j2.properties 파일에서 값을 변경합니다. |
| emr-job-submitter | <a href="#">작업 제출자 포트</a> 구성.          |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 주목할 만한 기능

Amazon EMR on EKS 7.0 릴리스에 다음과 같은 기능이 포함되어 있습니다.

- 애플리케이션 업그레이드 – Amazon EMR on EKS 7.0.0 애플리케이션 업그레이드는 Spark 3.5, Flink 1.18과 [Flink Operator](#) 1.6.1을 포함합니다.
- Flink Autoscaler 파라미터 자동 튜닝 – Flink Autoscaler에서 조정 계산을 위해 사용하는 기본 파라미터는 특정 작업의 최적값이 아닐 수 있습니다. Amazon EMR on EKS 7.0.0은 작업에 맞게 조정된 최적의 파라미터를 계산하기 위해 수집된 특정 지표의 과거 추세를 사용합니다.

## 변경 사항

Amazon EMR on EKS 7.0 릴리스에 다음과 같은 변경 사항이 포함되어 있습니다.

- Amazon Linux 2023 – Amazon EMR on EKS 7.0.0 이상에서 모든 컨테이너 이미지는 Amazon Linux 2023을 기반으로 합니다.
- Java 17을 기본 런타임으로 사용하는 Spark – Amazon EMR on EKS 7.0.0 Spark는 Java 17을 기본 런타임으로 사용합니다. 필요한 경우 [Amazon EMR on EKS 7.0 릴리스](#) 목록에 기재된 대로 해당 릴리스 레이블이 포함된 Java 8 또는 Java 11을 사용하도록 전환할 수 있습니다.

## emr-7.0.0-latest

릴리스 정보: `emr-7.0.0-latest`는 현재 `emr-7.0.0-2024321`를 가리킵니다.

리전: `emr-7.0.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.0.0:latest`

## emr-7.0.0-2024321

릴리스 정보: `7.0.0-2024321` 버전은 2024년 3월 11일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-7.0.0-2024321`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.0.0:2024321`

## emr-7.0.0-20231211

릴리스 정보: `7.0.0-20231211`은 2023년 12월에 출시되었습니다. Amazon EMR 7.0.0(Spark)의 최초 릴리스입니다.

리전: `emr-7.0.0-20231211`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.0.0:20231211`

## emr-7.0.0-flink-latest

릴리스 정보: `emr-7.0.0-flink-latest`는 현재 `emr-7.0.0-flink-2024321`를 가리킵니다.

리전: `emr-7.0.0-flink-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.0.0-flink:latest`

## emr-7.0.0-flink-2024321

릴리스 정보: 7.0.0-flink-2024321 버전은 2024년 3월 11일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: emr-7.0.0-flink-2024321는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.0.0-flink:2024321`

## emr-7.0.0-flink-20231211

릴리스 정보: 7.0.0-flink-20231211은 2023년 12월에 출시되었습니다. Amazon EMR 7.0.0(Flink)의 최초 릴리스입니다.

리전: emr-7.0.0-flink-20231211는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-7.0.0-flink:20231211`

## Amazon EMR on EKS 6.15.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 6.15.0 릴리스에 대한 일반적인 세부 정보는 Amazon EMR 릴리스 안내서에서 [Amazon EMR 6.15.0](#)을 참조하세요.

## Amazon EMR on EKS 6.15 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 6.15.0 릴리스를 사용할 수 있습니다. 특정 emr-6.15.0-XXXX 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

### Flink releases

Flink 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 6.15.0 릴리스를 사용할 수 있습니다.

- [emr-6.15.0-flink-latest](#)

- [emr-6.15.0-flink-20240105](#)
- [emr-6.15.0-flink-20231109](#)

## Spark releases

Spark 애플리케이션을 실행할 때 Amazon EMR on EKS에 대해 다음 Amazon EMR 6.15.0 릴리스를 사용할 수 있습니다.

- [emr-6.15.0-latest](#)
- [emr-6.15.0-20231109](#)
- emr-6.15.0-spark-rapids-latest
- emr-6.15.0-spark-rapids-20231109
- emr-6.15.0-java11-latest
- emr-6.15.0-java11-20231109
- emr-6.15.0-java17-latest
- emr-6.15.0-java17-20231109
- emr-6.15.0-java17-al2023-latest
- emr-6.15.0-java17-al2023-20231109
- emr-6.15.0-spark-rapids-java17-latest
- emr-6.15.0-spark-rapids-java17-20231109
- emr-6.15.0-spark-rapids-java17-al2023-latest
- emr-6.15.0-spark-rapids-java17-al2023-20231109
- notebook-spark/emr-6.15.0-latest
- notebook-spark/emr-6.15.0-20231109
- notebook-spark/emr-6.15.0-spark-rapids-latest
- notebook-spark/emr-6.15.0-spark-rapids-20231109
- notebook-spark/emr-6.15.0-java11-latest
- notebook-spark/emr-6.15.0-java11-20231109
- notebook-spark/emr-6.15.0-java17-latest
- notebook-spark/emr-6.15.0-java17-20231109
- notebook-spark/emr-6.15.0-java17-al2023-latest



- notebook-spark/emr-6.15.0-java17-al2023-20231109
- notebook-python/emr-6.15.0-latest
- notebook-python/emr-6.15.0-20231109
- notebook-python/emr-6.15.0-spark-rapids-latest
- notebook-python/emr-6.15.0-spark-rapids-20231109
- notebook-python/emr-6.15.0-java11-latest
- notebook-python/emr-6.15.0-java11-20231109
- notebook-python/emr-6.15.0-java17-latest
- notebook-python/emr-6.15.0-java17-20231109
- notebook-python/emr-6.15.0-java17-al2023-latest
- notebook-python/emr-6.15.0-java17-al2023-20231109

## 릴리스 노트

### Amazon EMR on EKS 6.15.0 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 1.12.569, Apache Spark 3.4.1-amzn-2, Apache Flink 1.17.1-amzn-1, Apache Hudi 0.14.0-amzn-0, Apache Iceberg 1.4.0-amzn-0, Delta 2.4.0, Apache Spark RAPIDS 23.08.01-amzn-0, Jupyter Enterprise Gateway 2.6.0
- 지원되는 구성 요소 - aws-sagemaker-spark-sdk, emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류            | 설명                                      |
|---------------|-----------------------------------------|
| core-site     | Hadoop core-site.xml 파일에서 값을 변경합니다.     |
| emrfs-site    | EMRFS 설정을 변경합니다.                        |
| spark-metrics | Spark metrics.properties 파일에서 값을 변경합니다. |

| 분류                | 설명                                       |
|-------------------|------------------------------------------|
| spark-defaults    | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env         | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site   | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j       | Spark log4j2.properties 파일에서 값을 변경합니다.   |
| emr-job-submitter | <a href="#">작업 제출자 포트</a> 구성.            |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 주목할 만한 기능

Amazon EMR on EKS 6.15 릴리스에 다음과 같은 기능이 포함되어 있습니다.

- [Amazon EMR on EKS\(Apache Flink 포함\)](#) - Amazon EMR on EKS 6.15.0을 사용하면 해당 Apache Flink 기반 애플리케이션을 동일한 Amazon EKS 클러스터에 있는 다른 유형의 애플리케이션과 함께 실행할 수 있습니다. 이를 통해 리소스 활용도를 높이고 인프라 관리를 간소화할 수 있습니다. Flink

애플리케이션에서 정상 서비스 해제를 통해 스팟 인스턴스를 활용하고 Amazon EBS를 통한 세분화된 복구 및 태스크 로컬 복구를 통해 다시 시작 시간을 단축할 수 있습니다. 접근성 및 모니터링 기능에는 Amazon S3에 저장된 jar로 Flink 애플리케이션을 시작하는 기능, AWS Glue 데이터 카탈로그에 대한 액세스, Amazon S3 및 Amazon CloudWatch와의 통합 모니터링, 컨테이너 로그 교체가 포함됩니다.

## emr-6.15.0-latest

릴리스 정보: `emr-6.15.0-latest`는 현재 `emr-6.15.0-20240105`를 가리킵니다.

리전: `emr-6.15.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.15.0:latest`

## emr-6.15.0-20240105

릴리스 정보: `6.15.0-20240105` 버전은 2024년 1월 17일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.15.0-20240105`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.15.0:20240105`

## emr-6.15.0-20231109

릴리스 정보: `6.15.0-20231109`는 2023년 11월 17일에 출시되었습니다. Amazon EMR 6.15.0의 최초 릴리스입니다.

리전: `emr-6.15.0-20231109`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.15.0:20231109`

## emr-6.15.0-flink-latest

릴리스 정보: `emr-6.15.0-flink-latest`는 현재 `emr-6.15.0-flink-20240105`를 가리킵니다.

리전: `emr-6.15.0-flink-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.15.0-flink:latest`

## emr-6.15.0-flink-20240105

릴리스 정보: `6.15.0-flink-20240105` 버전은 2024년 1월 17일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.15.0-flink-20240105`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.15.0-flink:20240105`

## emr-6.15.0-flink-20231109

릴리스 정보: `6.15.0-flink-20231109`는 2023년 11월 17일에 출시되었습니다. Amazon EMR 6.15.0의 최초 릴리스입니다.

리전: `emr-6.15.0-flink-20231109`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.15.0-flink:20231109`

## Amazon EMR on EKS 6.14.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 6.14.0 릴리스에 대한 일반적인 세부 정보는 Amazon EMR 릴리스 안내서에서 [Amazon EMR 6.14.0](#)을 참조하세요.

## Amazon EMR on EKS 6.14 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 6.14.0 릴리스를 사용할 수 있습니다. 특정 `emr-6.14.0-XXXX` 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-6.14.0-latest](#)
- [emr-6.14.0-20231005](#)

- emr-6.14.0-spark-rapids-latest
- emr-6.14.0-spark-rapids-20231005
- emr-6.14.0-java11-latest
- emr-6.14.0-java11-20231005
- emr-6.14.0-java17-latest
- emr-6.14.0-java17-20231005
- emr-6.14.0-java17-al2023-latest
- emr-6.14.0-java17-al2023-20231005
- emr-6.14.0-spark-rapids-java17-latest
- emr-6.14.0-spark-rapids-java17-20231005
- emr-6.14.0-spark-rapids-java17-al2023-latest
- emr-6.14.0-spark-rapids-java17-al2023-20231005
- notebook-spark/emr-6.14.0-latest
- notebook-spark/emr-6.14.0-20231005
- notebook-spark/emr-6.14.0-spark-rapids-latest
- notebook-spark/emr-6.14.0-spark-rapids-20231005
- notebook-spark/emr-6.14.0-java11-latest
- notebook-spark/emr-6.14.0-java11-20231005
- notebook-spark/emr-6.14.0-java17-latest
- notebook-spark/emr-6.14.0-java17-20231005
- notebook-spark/emr-6.14.0-java17-al2023-latest
- notebook-spark/emr-6.14.0-java17-al2023-20231005
- notebook-python/emr-6.14.0-latest
- notebook-python/emr-6.14.0-20231005
- notebook-python/emr-6.14.0-spark-rapids-latest
- notebook-python/emr-6.14.0-spark-rapids-20231005
- notebook-python/emr-6.14.0-java11-latest
- notebook-python/emr-6.14.0-java11-20231005
- notebook-python/emr-6.14.0-java17-latest
- notebook-python/emr-6.14.0-java17-20231005

- notebook-python/emr-6.14.0-java17-al2023-latest
- notebook-python/emr-6.14.0-java17-al2023-20231005

## 릴리스 노트

### Amazon EMR on EKS 6.14.0 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 1.12.543, Apache Spark 3.4.1-amzn-1, Apache Hudi 0.13.1-amzn-2, Apache Iceberg 1.3.0-amzn-0, Delta 2.4.0, Apache Spark RAPIDS 23.06.0-amzn-2, Jupyter Enterprise Gateway 2.7.0
- 지원되는 구성 요소 - aws-sagemaker-spark-sdk, emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                | 설명                                       |
|-------------------|------------------------------------------|
| core-site         | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site        | EMRFS 설정을 변경합니다.                         |
| spark-metrics     | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults    | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env         | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site   | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j       | Spark log4j2.properties 파일에서 값을 변경합니다.   |
| emr-job-submitter | <a href="#">작업 제출자 포드</a> 구성.            |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                       | 설명                                                                                          |
|--------------------------|---------------------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway <code>jupyter_enterprise_gateway_config.py</code> 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                                         |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 `spark-hive-site.xml`과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 주목할 만한 기능

Amazon EMR on EKS 6.14 릴리스에 다음과 같은 기능이 포함되어 있습니다.

- [Apache Livy](#) 지원 - 이제 Amazon EMR on EKS는 `spark-submit`을 통해 Apache Livy를 지원합니다.

### emr-6.14.0-latest

릴리스 정보: `emr-6.14.0-latest`는 현재 `emr-6.14.0-20231005`를 가리킵니다.

리전: `emr-6.14.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.14.0:latest`

### emr-6.14.0-20231005

릴리스 정보: `6.14.0-20231005`는 2023년 10월 17일에 출시되었습니다. Amazon EMR 6.14.0의 최초 릴리스입니다.

리전: `emr-6.14.0-20231005`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.14.0:20231005`

## Amazon EMR on EKS 6.13.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 6.13.0 릴리스에 대한 일반적인 세부 정보는 Amazon EMR 릴리스 안내서에서 [Amazon EMR 6.13.0](#)을 참조하세요.

### Amazon EMR on EKS 6.13 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 6.13.0 릴리스를 사용할 수 있습니다. 특정 `emr-6.13.0-XXXX` 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-6.13.0-latest](#)
- [emr-6.13.0-20230814](#)
- `emr-6.13.0-spark-rapids-latest`
- `emr-6.13.0-spark-rapids-20230814`
- `emr-6.13.0-java11-latest`
- `emr-6.13.0-java11-20230814`
- `emr-6.13.0-java17-latest`
- `emr-6.13.0-java17-20230814`
- `emr-6.13.0-java17-al2023-latest`
- `emr-6.13.0-java17-al2023-20230814`
- `emr-6.13.0-spark-rapids-java17-latest`
- `emr-6.13.0-spark-rapids-java17-20230814`
- `emr-6.13.0-spark-rapids-java17-al2023-latest`
- `emr-6.13.0-spark-rapids-java17-al2023-20230814`
- `notebook-spark/emr-6.13.0-latest`
- `notebook-spark/emr-6.13.0-20230814`
- `notebook-spark/emr-6.13.0-spark-rapids-latest`
- `notebook-spark/emr-6.13.0-spark-rapids-20230814`



- notebook-spark/emr-6.13.0-java11-latest
- notebook-spark/emr-6.13.0-java11-20230814
- notebook-spark/emr-6.13.0-java17-latest
- notebook-spark/emr-6.13.0-java17-20230814
- notebook-spark/emr-6.13.0-java17-al2023-latest
- notebook-spark/emr-6.13.0-java17-al2023-20230814
- notebook-python/emr-6.13.0-latest
- notebook-python/emr-6.13.0-20230814
- notebook-python/emr-6.13.0-spark-rapids-latest
- notebook-python/emr-6.13.0-spark-rapids-20230814
- notebook-python/emr-6.13.0-java11-latest
- notebook-python/emr-6.13.0-java11-20230814
- notebook-python/emr-6.13.0-java17-latest
- notebook-python/emr-6.13.0-java17-20230814
- notebook-python/emr-6.13.0-java17-al2023-latest
- notebook-python/emr-6.13.0-java17-al2023-20230814

## 릴리스 노트

### Amazon EMR on EKS 6.13.0 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 1.12.513, Apache Spark 3.4.1-amzn-0, Apache Hudi 0.13.1-amzn-0, Apache Iceberg 1.3.0-amzn-0, Delta 2.4.0, Apache Spark RAPIDS 23.06.0-amzn-1, Jupyter Enterprise Gateway 2.6.0.amzn
- 지원되는 구성 요소 - aws-sagemaker-spark-sdk, emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                | 설명                                       |
|-------------------|------------------------------------------|
| core-site         | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site        | EMRFS 설정을 변경합니다.                         |
| spark-metrics     | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults    | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env         | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site   | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j       | Spark log4j2.properties 파일에서 값을 변경합니다.   |
| emr-job-submitter | <a href="#">작업 제출자 포트</a> 구성.            |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 `spark-hive-site.xml`과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 주목할 만한 기능

Amazon EMR on EKS 6.13 릴리스에 다음과 같은 기능이 포함되어 있습니다.

- Amazon Linux 2023 - Amazon EMR on EKS 6.13 이상에서 Java 17 런타임과 함께 AL2023 운영 체제로 Spark(AL2023 포함)를 실행할 수 있습니다. 이를 위해 이름에 `a12023`을 포함하는 릴리스 레이블을 사용합니다. 예를 들어 `emr-6.13.0-java17-a12023-latest`입니다. 프로덕션 워크로드를 AL2023 및 Java 17로 이동하기 전에 성능 테스트를 검증하고 실행하는 것이 좋습니다.
- [Apache Flink를 포함하는 Amazon EMR on EKS](#)(퍼블릭 평가판) - Amazon EMR on EKS 릴리스 6.13 이상에서는 Apache Flink를 지원합니다(퍼블릭 평가판에서 사용 가능). 이번 출시를 통해 동일한 Amazon EKS 클러스터에서 다른 유형의 애플리케이션과 함께 Apache Flink 기반 애플리케이션을 실행할 수 있습니다. 이를 통해 리소스 활용도를 높이고 인프라 관리를 간소화할 수 있습니다. 이미 Amazon EKS에서 빅 데이터 프레임워크를 실행하는 경우 이제 Amazon EMR에서 프로비저닝 및 관리를 자동화할 수 있습니다.

### emr-6.13.0-latest

릴리스 정보: `emr-6.13.0-latest`는 현재 `emr-6.13.0-20230814`를 가리킵니다.

리전: `emr-6.13.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.13.0:latest`

### emr-6.13.0-20230814

릴리스 정보: `6.13.0-20230814`는 2023년 9월 7일에 출시되었습니다. Amazon EMR 6.13.0의 최초 릴리스입니다.

리전: `emr-6.13.0-20230814`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.13.0:20230814`

# Amazon EMR on EKS 6.12.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 6.12.0 릴리스에 대한 일반적인 세부 정보는 Amazon EMR 릴리스 안내서에서 [Amazon EMR 6.12.0](#)을 참조하세요.

## Amazon EMR on EKS 6.12 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 6.12.0 릴리스를 사용할 수 있습니다. 특정 emr-6.12.0-XXXX 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-6.12.0-latest](#)
- [emr-6.12.0-20240321](#)
- [emr-6.12.0-20230701](#)
- emr-6.12.0-spark-rapids-latest
- emr-6.12.0-spark-rapids-20230701
- emr-6.12.0-java11-latest
- emr-6.12.0-java11-20230701
- emr-6.12.0-java17-latest
- emr-6.12.0-java17-20230701
- emr-6.12.0-spark-rapids-java17-latest
- emr-6.12.0-spark-rapids-java17-20230701
- notebook-spark/emr-6.12.0-latest
- notebook-spark/emr-6.12.0-20230701
- notebook-spark/emr-6.12.0-spark-rapids-latest
- notebook-spark/emr-6.12.0-spark-rapids-20230701
- notebook-python/emr-6.12.0-latest
- notebook-python/emr-6.12.0-20230701
- notebook-python/emr-6.12.0-spark-rapids-latest
- notebook-python/emr-6.12.0-spark-rapids-20230701

## 릴리스 노트

### Amazon EMR on EKS 6.12.0 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 1.12.490, Apache Spark 3.4.0-amzn-0, Apache Hudi 0.13.1-amzn-0, Apache Iceberg 1.3.0-amzn-0, Delta 2.4.0, Apache Spark RAPIDS 23.06.0-amzn-0, Jupyter Enterprise Gateway 2.6.0
- 지원되는 구성 요소 - aws-sagemaker-spark-sdk, emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                | 설명                                       |
|-------------------|------------------------------------------|
| core-site         | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site        | EMRFS 설정을 변경합니다.                         |
| spark-metrics     | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults    | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env         | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site   | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j       | Spark log4j2.properties 파일에서 값을 변경합니다.   |
| emr-job-submitter | <a href="#">작업 제출자 포트</a> 구성.            |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 주목할 만한 기능

Amazon EMR on EKS 6.12 릴리스에 다음과 같은 기능이 포함되어 있습니다.

- Java 17 - Amazon EMR on EKS 6.12 이상에서 Java 17 런타임과 함께 Spark를 시작할 수 있습니다. 이를 수행하려면 emr-6.12.0-java17-latest를 릴리스 레이블로 전달합니다. 프로덕션 워크로드를 이전 버전의 Java 이미지에서 Java 17로 이동하기 전에 성능 테스트를 검증하고 실행하는 것이 좋습니다.

## emr-6.12.0-latest

릴리스 정보: emr-6.12.0-latest는 현재 emr-6.12.0-20240321를 가리킵니다.

리전: emr-6.12.0-latest는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-6.12.0:latest

## emr-6.12.0-20240321

릴리스 정보: 6.12.0-20240321 버전은 2024년 3월 11일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.12.0-20240321`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.12.0:20240321`

## emr-6.12.0-20230701

릴리스 정보: `6.12.0-20230701`은 2023년 7월 1일에 출시되었습니다. Amazon EMR 6.12.0의 최초 릴리스입니다.

리전: `emr-6.12.0-20230701`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.12.0:20230701`

## Amazon EMR on EKS 6.11.0 릴리스

이 페이지에서는 Amazon EMR on EKS 배포에 특정한 Amazon EMR의 신규 기능 및 업데이트된 기능을 설명합니다. Amazon EC2에서 실행되는 Amazon EMR 및 Amazon EMR 6.11.0 릴리스에 대한 일반적인 세부 정보는 Amazon EMR 릴리스 안내서에서 [Amazon EMR 6.11.0](#)을 참조하세요.

## Amazon EMR on EKS 6.11 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 6.11.0 릴리스를 사용할 수 있습니다. 특정 `emr-6.11.0-XXXX` 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-6.11.0-latest](#)
- [emr-6.11.0-20230905](#)
- [emr-6.11.0-20230509](#)
- `emr-6.11.0-spark-rapids-latest`
- `emr-6.11.0-spark-rapids-20230509`
- `emr-6.11.0-java11-latest`
- `emr-6.11.0-java11-20230509`
- `notebook-spark/emr-6.11.0-latest`
- `notebook-spark/emr-6.11.0-20230509`
- `notebook-python/emr-6.11.0-latest`

- notebook-python/emr-6.11.0-20230509

## 릴리스 노트

### Amazon EMR on EKS 6.11.0 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 1.12.446, Apache Spark 3.3.2-amzn-0, Apache Hudi 0.13.0-amzn-0, Apache Iceberg 1.2.0-amzn-0, Delta 2.2.0, Apache Spark RAPIDS 23.02.0-amzn-0, Jupyter Enterprise Gateway 2.6.0
- 지원되는 구성 요소 - aws-sagemaker-spark-sdk, emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류              | 설명                                       |
|-----------------|------------------------------------------|
| core-site       | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site      | EMRFS 설정을 변경합니다.                         |
| spark-metrics   | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults  | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env       | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j     | Spark log4j.properties 파일에서 값을 변경합니다.    |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:



| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 주목할 만한 기능

Amazon EMR on EKS 6.11 릴리스에 다음과 같은 기능이 포함되어 있습니다.

- [Amazon ECR 퍼블릭 갤러리의 Amazon EMR on EKS 기본 이미지](#) - [사용자 지정 이미지](#) 기능을 사용하는 경우 기본 이미지에서 Amazon EMR on EKS와 상호 작용하는 데 필요한 jar, 구성 및 라이브러리를 제공합니다. 이제 [Amazon ECR 퍼블릭 갤러리](#)에서 기본 이미지를 찾을 수 있습니다.
- [Spark 컨테이너 로그 로테이션](#) - Amazon EMR on EKS 6.11에서는 Spark 컨테이너 로그 로테이션을 지원합니다. StartJobRun API의 MonitoringConfiguration작업 내에서 containerLogRotationConfiguration을 통해 기능을 활성화할 수 있습니다. Amazon EMR on EKS에서 Spark 드라이버 및 실행기 포트에 보관할 로그 파일의 수와 크기를 지정하도록 rotationSize 및 maxFilestoKeep을 구성할 수 있습니다. 자세한 내용은 [Spark 컨테이너 로그 로테이션 사용](#) 단원을 참조하십시오.
- Spark 운영자 및 spark-submit에서 Volcano 지원 - Amazon EMR on EKS 6.11에서는 [Spark 운영자](#) 및 [spark-submit](#)에서 Volcano를 Kubernetes 사용자 지정 스케줄러로 사용하여 Spark 작업 실행을 지원합니다. 단체 예약, 대기열 관리, 선점, 공정 공유 예약과 같은 기능을 사용하여 높은 예약 처리량 및 최적화된 용량을 지원할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS에서 Apache Spark의 사용자 지정 스케줄러로 Volcano 사용](#) 단원을 참조하십시오.

## emr-6.11.0-latest

릴리스 정보: `emr-6.11.0-latest`는 현재 `emr-20230905`를 가리킵니다.

리전: `emr-6.11.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.11.0:latest`

## emr-6.11.0-20230905

릴리스 정보: `6.11.0-20230905`는 2023년 9월 29일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.11.0-20230509`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.11.0:20230509`

## emr-6.11.0-20230509

릴리스 정보: `6.11.0-20230509`는 2023년 5월 9일에 출시되었습니다. Amazon EMR 6.11.0의 최초 릴리스입니다.

리전: `emr-6.11.0-20230509`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.11.0:20230509`

## Amazon EMR on EKS 6.10.0 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 6.10.0 릴리스를 사용할 수 있습니다. 특정 `emr-6.10.0-XXXX` 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-6.10.0-latest](#)
- [emr-6.10.0-20230905](#)
- [emr-6.10.0-20230624](#)
- [emr-6.10.0-20230421](#)
- [emr-6.10.0-20230403](#)
- [emr-6.10.0-20230220](#)

- emr-6.10.0-spark-rapids-latest
- emr-6.10.0-spark-rapids-20230624
- emr-6.10.0-spark-rapids-20230220
- emr-6.10.0-java11-latest
- emr-6.10.0-java11-20230624
- emr-6.10.0-java11-20230220
- notebook-spark/emr-6.10.0-latest
- notebook-spark/emr-6.10.0-20230624
- notebook-spark/emr-6.10.0-20230220
- notebook-python/emr-6.10.0-latest
- notebook-python/emr-6.10.0-20230624
- notebook-python/emr-6.10.0-20230220

### Amazon EMR 6.10.0용 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 1.12.397, Spark 3.3.1-amzn-0, Hudi 0.12.2-amzn-0, Iceberg 1.1.0-amzn-0, Delta 2.2.0.
- 지원되는 구성 요소 - aws-sagemaker-spark-sdk, emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류:

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류             | 설명                                       |
|----------------|------------------------------------------|
| core-site      | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site     | EMRFS 설정을 변경합니다.                         |
| spark-metrics  | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults | Spark spark-defaults.conf 파일에서 값을 변경합니다. |

| 분류              | 설명                                    |
|-----------------|---------------------------------------|
| spark-env       | Spark 환경에서 값을 변경합니다.                  |
| spark-hive-site | Spark hive-site.xml 파일에서 값을 변경합니다.    |
| spark-log4j     | Spark log4j.properties 파일에서 값을 변경합니다. |

특별히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 주목할 만한 기능

- Spark 운영자 - Amazon EMR on EKS 6.10.0 이상에서 Apache Spark용 Kubernetes 운영자 또는 Spark 운영자를 사용하여 자체 Amazon EKS 클러스터에서 Amazon EMR 릴리스 런타임과 함께 Spark 애플리케이션을 배포하고 관리할 수 있습니다. 자세한 내용은 [Spark 운영자에서 Spark 작업 실행](#) 단원을 참조하십시오.
- Java 11 - Amazon EMR on EKS 6.10 이상에서 Java 11 런타임과 함께 Spark를 시작할 수 있습니다. 이를 수행하려면 emr-6.10.0-java11-latest를 릴리스 레이블로 전달합니다. 프로덕션 워크로드를 Java 8 이미지에서 Java 11 이미지로 이동하기 전에 성능 테스트를 검증하고 실행하는 것이 좋습니다.

- Apache Spark용 Amazon Redshift 통합과 관련하여 Amazon EMR on EKS 6.10.0에서는 `minimal-json.jar`에서 종속성을 제거하고, 필요한 `spark-redshift` 관련 jar(`spark-redshift.jar`, `spark-avro.jar`, `RedshiftJDBC.jar`)을 Spark의 실행기 클래스 경로에 자동으로 추가합니다.

## 변경

- 이제 Parquet, ORC 및 텍스트 기반 형식(CSV 및 JSON 포함)에 대해 EMRFS S3에 최적화된 커미터가 기본적으로 활성화됩니다.

## emr-6.10.0-latest

릴리스 정보: `emr-6.10.0-latest`는 현재 `emr-6.10.0-20230905`를 가리킵니다.

리전: `emr-6.10.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.10.0:latest`

## emr-6.10.0-20230905

릴리스 정보: `6.10.0-20230905`는 2023년 9월 29일에 출시되었습니다. 이전 릴리스와 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.10.0-20230905`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.10.0:20230905`

## emr-6.10.0-20230624

릴리스 정보: `6.10.0-20230624`는 2023년 7월 7일에 출시되었습니다. 이전 릴리스와 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.10.0-20230624`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.10.0:20230624`

## emr-6.10.0-20230421

릴리스 정보: 6.10.0-20230421은 2023년 4월 28일에 출시되었습니다. 이전 릴리스와 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: emr-6.10.0-20230421는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-6.10.0:20230421

## emr-6.10.0-20230403

릴리스 정보: 6.10.0-20230403은 2023년 4월 12일에 출시되었습니다. 이전 릴리스와 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: emr-6.10.0-20230403는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-6.10.0:20230403

## emr-6.10.0-20230220

릴리스 정보: emr-6.10.0-20230220은 2023년 2월 20일에 출시되었습니다. Amazon EMR 6.10.0의 최초 릴리스입니다.

리전: emr-6.10.0-20230220는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-6.10.0:20230220

## Amazon EMR on EKS 6.9.0 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 6.9.0 릴리스를 사용할 수 있습니다. 특정 emr-6.9.0-XXXX 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-6.9.0-latest](#)
- [emr-6.9.0-20230905](#)
- [emr-6.9.0-20230624](#)
- [emr-6.9.0-20221108](#)

- emr-6.9.0-spark-rapids-latest
- emr-6.9.0-spark-rapids-20230624
- emr-6.9.0-spark-rapids-20221108
- notebook-spark/emr-6.9.0-latest
- notebook-spark/emr-6.9.0-20230624
- notebook-spark/emr-6.9.0-20221108
- notebook-python/emr-6.9.0-latest
- notebook-python/emr-6.9.0-20230624
- notebook-python/emr-6.9.0-20221108

### Amazon EMR 6.9.0용 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 1.12.331, Spark 3.3.0-amzn-1, Hudi 0.12.1-amzn-0, Iceberg 0.14.1-amzn-0, Delta 2.1.0.
- 지원되는 구성 요소 - aws-sagemaker-spark-sdk, emr-ddb, emr-goodies, emr-s3-select, emrfs, hadoop-client, hudi, hudi-spark, iceberg, spark-kubernetes.
- 지원되는 구성 분류:

[StartJobRun](#) 및 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류              | 설명                                       |
|-----------------|------------------------------------------|
| core-site       | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site      | EMRFS 설정을 변경합니다.                         |
| spark-metrics   | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults  | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env       | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site | Spark hive-site.xml 파일에서 값을 변경합니다.       |

| 분류          | 설명                                    |
|-------------|---------------------------------------|
| spark-log4j | Spark log4j.properties 파일에서 값을 변경합니다. |

특히 [CreateManagedEndpoint](#) API와 함께 사용하는 경우:

| 분류                       | 설명                                                                             |
|--------------------------|--------------------------------------------------------------------------------|
| jeg-config               | Jupyter Enterprise Gateway jupyter_enterprise_gateway_config.py 파일에서 값을 변경합니다. |
| jupyter-kernel-overrides | Jupyter 커널 사양 파일에서 커널 이미지 값을 변경합니다.                                            |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 주목할 만한 기능

- Apache Spark용 Nvidia RAPIDS 액셀러레이터 - Amazon EMR on EKS는 EC2 그래픽 처리 장치 (GPU) 인스턴스 유형을 사용하여 Spark를 가속화합니다. Spark 이미지를 RAPIDS 액셀러레이터와 함께 사용하려면 릴리스 레이블을 emr-6.9.0-spark-rapids-latest로 지정합니다. 자세한 내용은 [설명서 페이지](#)를 참조하세요.
- Spark-Redshift 커넥터 - Apache Spark용 Amazon Redshift 통합은 Amazon EMR 릴리스 6.9.0 이상에 포함되어 있습니다. 이전의 오픈 소스 도구였던, 이 기본 통합은 Spark 커넥터로, Amazon Redshift와 Amazon Redshift Serverless에서 데이터를 읽고 쓰는 Apache Spark 애플리케이션을 빌드할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS에서 Apache Spark용 Amazon Redshift 통합 사용](#) 단원을 참조하십시오.
- Delta Lake - [Delta Lake](#)는 트랜잭션 일관성, 일관된 데이터 세트 정의, 스키마 진화 변경 및 데이터 변형 지원과 같은 기능을 포함하는 데이터 레이크를 구축할 수 있는 오픈 소스 스토리지 형식입니다. 자세한 내용은 [Delta Lake 사용](#)을 참조하세요.



- PySpark 파라미터 수정 - 대화형 엔드포인트는 이제 EMR Studio Jupyter Notebook의 PySpark 세션과 관련된 Spark 파라미터 수정을 지원합니다. 자세한 내용은 [PySpark 세션 파라미터 수정](#)을 참조하세요.

## 해결된 문제

- Amazon EMR 버전 6.6.0, 6.7.0 및 6.8.0에서 Spark와 함께 DynamoDB 커넥터를 사용하는 경우 입력 분할이 비어 있지 않은 데이터를 참조하더라도 테이블에서 모든 읽기는 빈 결과를 반환합니다. Amazon EMR 릴리스 6.9.0에서는 이 문제가 수정되었습니다.
- Amazon EMR on EKS 6.8.0에서는 [Apache Spark](#)를 사용하여 생성된 Parquet 파일 메타데이터에서 빌드 해시를 잘못 채웁니다. 이 문제로 인해 Amazon EMR on EKS 6.8.0에서 생성한 Parquet 파일의 메타데이터 버전 문자열을 구문 분석하는 도구에서 문제가 발생할 수 있습니다.

## 알려진 문제

- Apache Spark용 Amazon Redshift 통합을 사용하고 Parquet 형식의 time, timetz, timestamp 또는 timestamptz(마이크로초 정밀도)를 사용하는 경우 커넥터는 시간 값을 가장 가까운 밀리초 값으로 반올림합니다. 해결 방법으로, 텍스트 언로드 형식 unload\_s3\_format 파라미터를 사용합니다.

## emr-6.9.0-latest

릴리스 정보: emr-6.9.0-latest는 현재 emr-6.9.0-20230905를 가리킵니다.

리전: emr-6.9.0-latest는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-6.9.0:latest

## emr-6.9.0-20230905

릴리스 정보: emr-6.9.0-20230905. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: emr-6.9.0-20230905는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-6.9.0:20230905

## emr-6.9.0-20230624

릴리스 정보: `emr-6.9.0-20230624`는 2023년 7월 7일에 출시되었습니다.

리전: `emr-6.9.0-20230624`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.9.0:20230624`

## emr-6.9.0-20221108

릴리스 정보: `emr-6.9.0-20221108`은 2022년 12월 8일에 출시되었습니다. Amazon EMR 6.9.0의 최초 릴리스입니다.

리전: `emr-6.9.0-20221108`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.9.0:20221108`

## Amazon EMR on EKS 6.8.0 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 6.8.0 릴리스를 사용할 수 있습니다. 특정 `emr-6.8.0-XXXX` 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-6.8.0-latest](#)
- [emr-6.8.0-20230905](#)
- [emr-6.8.0-20230624](#)
- [emr-6.8.0-20221219](#)
- [emr-6.8.0-20220802](#)

### Amazon EMR 6.8.0용 릴리스 정보

- 지원되는 애플리케이션 - AWS SDK for Java 1.12.170, Spark 3.3.0-amzn-0, Hudi 0.11.1-amzn-0, Iceberg 0.14.0-amzn-0.
- 지원되는 구성 요소 - `aws-sagemaker-spark-sdk`, `emr-ddb`, `emr-goodies`, `emr-s3-select`, `emrfs`, `hadoop-client`, `hudi`, `hudi-spark`, `iceberg`, `spark-kubernetes`.
- 지원되는 구성 분류:

| 분류              | 설명                                       |
|-----------------|------------------------------------------|
| core-site       | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site      | EMRFS 설정을 변경합니다.                         |
| spark-metrics   | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults  | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env       | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j     | Spark log4j.properties 파일에서 값을 변경합니다.    |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 주목할 만한 기능

- Spark3.3.0 - Amazon EMR on EKS 6.8에는 Spark 드라이버 실행기 포트에 별도의 노드 선택기 레이블 사용을 지원하는 Spark 3.3.0이 포함되어 있습니다. 이 새 레이블을 사용하면 포트 템플릿을 사용하지 않고도 StartJobRun API에서 드라이버 및 실행기 포트의 노드 유형을 개별적으로 정의할 수 있습니다.
  - 드라이버 노드 선택기 속성: spark.kubernetes.driver.node.selector.[labelKey]
  - 실행기 노드 선택기 속성: spark.kubernetes.executor.node.selector.[labelKey]
- 개선된 작업 실패 메시지 - 이 릴리스에서는 사용자 코드로 인한 작업 실패를 추적하기 위해 spark.stage.extraDetailsOnFetchFailures.enabled 및 spark.stage.extraDetailsOnFetchFailures.maxFailuresToInclude 구성을 도입합니다.

다. 이러한 세부 정보는 셔플 가져오기 실패로 인해 스테이지가 중단된 경우 드라이버 로그에 표시되는 실패 메시지를 개선하는 데 사용됩니다.

| 속성 이름                                                       | 기본값   | 의미                                                                                                                                                                                                                                                                                     | 이후 버전   |
|-------------------------------------------------------------|-------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| <code>spark.stage.extraDetailsOnFetchFailure.enabled</code> | false | <p>true로 설정하면 셔플 가져오기 실패로 인해 스테이지가 중단되는 경우 드라이버 로그에 표시되는 작업 실패 메시지를 개선하는 데 이 속성을 사용합니다. 기본적으로 사용자 코드로 인한 실패한 마지막 5개 작업을 추적하며, 실패 오류 메시지는 드라이버 로그에 추가됩니다.</p> <p>추적할 사용자 예외를 포함한 작업 실패 횟수를 늘리려면 <code>spark.stage.extraDetailsOnFetchFailure.maxFailuresToInclude</code> 구성을 참조하세요.</p> | emr-6.8 |

| 속성 이름                                                        | 기본값 | 의미                                                                                                                                                                                                               | 이후 버전   |
|--------------------------------------------------------------|-----|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| spark.stage.extraDetailsOnFetchFailures.maxFailuresToInclude | 5   | <p>스테이지 및 시도당 추적할 작업 실패 횟수. 셔플 가져오기 실패로 인해 스테이지가 중단되는 경우 드라이버 로그에 표시되는 사용자 예외를 포함한 작업 실패 메시지를 개선하는 데 이 속성을 사용합니다.</p> <p>이 속성은 Config spark.stage.extraDetailsOnFetchFailures.enabled를 true로 설정한 경우에만 작동합니다.</p> | emr-6.8 |

자세한 내용은 [Apache Spark configuration 설명서](#)를 참조하세요.

#### 알려진 문제

- Amazon EMR on EKS 6.8.0에서는 [Apache Spark](#)를 사용하여 생성된 Parquet 파일 메타데이터에서 빌드 해시를 잘못 채웁니다. 이 문제로 인해 Amazon EMR on EKS 6.8.0에서 생성한 Parquet 파일의 메타데이터 버전 문자열을 구문 분석하는 도구에서 문제가 발생할 수 있습니다. Parquet 메타데이터에서 버전 문자열을 구문 분석하고 빌드 해시를 사용하는 고객은 다른 Amazon EMR 버전으로 전환하고 파일을 다시 작성해야 합니다.

#### 해결된 문제

- pySpark 커널에 대한 커널 기능 중단 - 노트북에서 셀을 실행하여 트리거되는 진행 중인 대화형 워크로드는 Interrupt Kernel 기능을 사용하여 중지할 수 있습니다. 이 기능이 pySpark 커널에서 작동하도록 수정 사항이 도입되었습니다. [Changes for handling interrupts for PySpark Kubernetes Kernel #1115](#)에서 오픈 소스로도 사용할 수 있습니다.

## emr-6.8.0-latest

릴리스 정보: `emr-6.8.0-latest`는 현재 `emr-6.8.0-20230624`를 가리킵니다.

리전: `emr-6.8.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.8.0:latest`

## emr-6.8.0-20230905

릴리스 정보: `emr-6.8.0-20230905`는 2023년 9월 29일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.8.0-20230905`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.8.0:20230905`

## emr-6.8.0-20230624

릴리스 정보: `emr-6.8.0-20230624`는 2023년 7월 7일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.8.0-20230624`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.8.0:20230624`

## emr-6.8.0-20221219

릴리스 정보: `emr-6.8.0-20221219`는 2023년 1월 19일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.8.0-20221219`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.8.0:20221219`

## emr-6.8.0-20220802

릴리스 정보: `emr-6.8.0-20220802`는 2022년 9월 27일에 출시되었습니다. Amazon EMR 6.8.0의 최초 릴리스입니다.

리전: `emr-6.8.0-20220802`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.8.0:20220802`

## Amazon EMR on EKS 6.7.0 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 6.7.0 릴리스를 사용할 수 있습니다. 특정 `emr-6.7.0-XXXX` 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-6.7.0-latest](#)
- [emr-6.7.0-20240321](#)
- [emr-6.7.0-20230624](#)
- [emr-6.7.0-20221219](#)
- [emr-6.7.0-20220630](#)

### Amazon EMR 6.7.0용 릴리스 정보

- 지원되는 애플리케이션 - Spark 3.2.1-amzn-0, Jupyter Enterprise Gateway 2.6, Hudi 0.11-amzn-0, Iceberg 0.13.1.
- 지원되는 구성 요소 - `aws-hm-client`(Glue 커넥터), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- JEG 2.6으로 업그레이드하면 이제 커널 관리가 비동기식이므로 커널 실행이 진행 중일 때 JEG는 트랜잭션을 차단하지 않습니다. 이를 통해 다음과 같은 기능을 제공함으로써 사용자 경험이 크게 개선됩니다.
  - 다른 커널 시작이 진행 중일 때 현재 실행 중인 노트북에서 명령을 실행하는 기능
  - 이미 실행 중인 커널에 영향을 주지 않고 여러 커널을 동시에 시작하는 기능
- 지원되는 구성 분류:

| 분류                      | 설명                                               |
|-------------------------|--------------------------------------------------|
| <code>core-site</code>  | Hadoop <code>core-site.xml</code> 파일에서 값을 변경합니다. |
| <code>emrfs-site</code> | EMRFS 설정을 변경합니다.                                 |

| 분류              | 설명                                       |
|-----------------|------------------------------------------|
| spark-metrics   | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults  | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env       | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j     | Spark log4j.properties 파일에서 값을 변경합니다.    |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하십시오.

## 해결된 문제

- Amazon EMR on EKS 6.7에서는 Apache Spark의 포트 템플릿 기능을 대화형 엔드포인트와 함께 사용할 때 6.6에서 발생하는 문제를 수정합니다. 이 문제는 Amazon EMR on EKS 릴리스 6.4, 6.5, 6.6에서 발생했습니다. 이제 포트 템플릿을 사용하여 대화형 엔드포인트로 대화형 분석을 실행할 때 Spark 드라이버 및 실행기 포트 시작 방식을 정의할 수 있습니다.
- 이전 Amazon EMR on EKS 릴리스에서 Jupyter Enterprise Gateway는 커널이 시작 중일 때 트랜잭션을 차단했고, 이로 인해 현재 실행 중인 노트북 세션의 실행을 방해했습니다. 이제 다른 커널 시작이 진행 중일 때 현재 실행 중인 노트북에서 명령을 실행할 수 있습니다. 또한 이미 실행 중인 커널과의 연결을 끊지 않고도 여러 커널을 동시에 시작할 수 있습니다.

## emr-6.7.0-latest

릴리스 정보: emr-6.7.0-latest는 현재 emr-6.7.0-20240321를 가리킵니다.

리전: emr-6.7.0-latest는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.



컨테이너 이미지 태그: `emr-6.7.0:latest`

## emr-6.7.0-20240321

릴리스 정보: `emr-6.7.0-20240321` 버전은 2024년 3월 11일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.7.0-20240321`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.7.0:20240321`

## emr-6.7.0-20230624

릴리스 정보: `emr-6.7.0-20230624`는 2023년 7월 7일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.7.0-20230624`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.7.0:20230624`

## emr-6.7.0-20221219

릴리스 정보: `emr-6.7.0-20221219`는 2023년 1월 19일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.7.0-20221219`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.7.0:20221219`

## emr-6.7.0-20220630

릴리스 정보: `emr-6.7.0-20220630`는 2022년 7월 12일에 출시되었습니다. Amazon EMR 6.7.0의 최초 릴리스입니다.

리전: `emr-6.7.0-20220630`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.7.0:20220630`

## Amazon EMR on EKS 6.6.0 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 6.6.0 릴리스를 사용할 수 있습니다. 특정 `emr-6.6.0-XXXX` 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-6.6.0-latest](#)
- [emr-6.6.0-20240321](#)
- [emr-6.6.0-20230624](#)
- [emr-6.6.0-20221219](#)
- [emr-6.6.0-20220411](#)

### Amazon EMR 6.6.0용 릴리스 정보

- 지원되는 애플리케이션 - Spark 3.2.0-amzn-0, Jupyter Enterprise Gateway(엔드포인트, 퍼블릭 평가판), Hudi 0.10.1-amzn-0, Iceberg 0.13.1.
- 지원되는 구성 요소 - `aws-hm-client`(Glue 커넥터), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- 지원되는 구성 분류:

| 분류                           | 설명                                                    |
|------------------------------|-------------------------------------------------------|
| <code>core-site</code>       | Hadoop <code>core-site.xml</code> 파일에서 값을 변경합니다.      |
| <code>emrfs-site</code>      | EMRFS 설정을 변경합니다.                                      |
| <code>spark-metrics</code>   | Spark <code>metrics.properties</code> 파일에서 값을 변경합니다.  |
| <code>spark-defaults</code>  | Spark <code>spark-defaults.conf</code> 파일에서 값을 변경합니다. |
| <code>spark-env</code>       | Spark 환경에서 값을 변경합니다.                                  |
| <code>spark-hive-site</code> | Spark <code>hive-site.xml</code> 파일에서 값을 변경합니다.       |

| 분류          | 설명                                    |
|-------------|---------------------------------------|
| spark-log4j | Spark log4j.properties 파일에서 값을 변경합니다. |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하십시오.

### 알려진 문제

- 대화형 엔드포인트를 포함하는 Spark 포트 템플릿 기능은 Amazon EMR on EKS 릴리스 6.4, 6.5, 6.6에서 작동하지 않습니다.

### 해결된 문제

- 대화형 엔드포인트 로그는 Cloudwatch 및 S3에 업로드됩니다.

## emr-6.6.0-latest

릴리스 정보: `emr-6.6.0-latest`는 현재 `emr-6.6.0-20240321`를 가리킵니다.

리전: `emr-6.6.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.6.0:latest`

## emr-6.6.0-20240321

릴리스 정보: `emr-6.6.0-20240321` 버전은 2024년 3월 11일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.6.0-20240321`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.6.0:20240321`

## emr-6.6.0-20230624

릴리스 정보: emr-6.6.0-20230624는 2023년 1월 27일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: emr-6.6.0-20230624는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-6.6.0:20230624

## emr-6.6.0-20221219

릴리스 정보: emr-6.6.0-20221219는 2023년 1월 27일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: emr-6.6.0-20221219는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-6.6.0:20221219

## emr-6.6.0-20220411

릴리스 정보: emr-6.6.0-20220411은 2022년 5월 20일에 출시되었습니다. Amazon EMR 6.6.0의 최초 릴리스입니다.

리전: emr-6.6.0-20220411는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-6.6.0:20220411

## Amazon EMR on EKS 6.5.0 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 6.5.0 릴리스를 사용할 수 있습니다. 특정 emr-6.5.0-XXXX 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-6.5.0-latest](#)
- [emr-6.5.0-20240321](#)
- [emr-6.5.0-20221219](#)

- [emr-6.5.0-20220802](#)
- [emr-6.5.0-20211119](#)

## Amazon EMR 6.5.0용 릴리스 정보

- 지원되는 애플리케이션 - Spark 3.1.2-amzn-1, Jupyter Enterprise Gateway(엔드포인트, 퍼블릭 평가판).
- 지원되는 구성 요소 - aws-hm-client(Glue 커넥터), aws-sagemaker-spark-sdk, emr-s3-select, emrfs, emr-ddb, hudi-spark.
- 지원되는 구성 분류:

| 분류              | 설명                                       |
|-----------------|------------------------------------------|
| core-site       | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site      | EMRFS 설정을 변경합니다.                         |
| spark-metrics   | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults  | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env       | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j     | Spark log4j.properties 파일에서 값을 변경합니다.    |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하십시오.

## 알려진 문제

- 대화형 엔드포인트를 포함하는 Spark 포트 템플릿 기능은 Amazon EMR on EKS 릴리스 6.4 및 6.5에서 작동하지 않습니다.

## emr-6.5.0-latest

릴리스 정보: `emr-6.5.0-latest`는 현재 `emr-6.5.0-20240321`를 가리킵니다.

리전: `emr-6.5.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.5.0:latest`

## emr-6.5.0-20240321

릴리스 정보: `emr-6.5.0-20240321` 버전은 2024년 3월 11일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.5.0-20240321`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.5.0:20240321`

## emr-6.5.0-20221219

릴리스 정보: `emr-6.5.0-20221219`는 2023년 1월 19일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.5.0-20221219`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.5.0:20221219`

## emr-6.5.0-20220802

릴리스 정보: `emr-6.5.0-20220802`는 2022년 8월 24일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지를 포함한 새 버전입니다.

리전: `emr-6.5.0-20220802`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.5.0:20220802`

## emr-6.5.0-20211119

릴리스 정보: `emr-6.5.0-20211119`는 2022년 1월 20일에 출시되었습니다. Amazon EMR 6.5.0의 최초 릴리스입니다.

리전: `emr-6.5.0-20211119`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.5.0:20211119`

## Amazon EMR on EKS 6.4.0 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 6.4.0 릴리스를 사용할 수 있습니다. 특정 `emr-6.4.0-XXXX` 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-6.4.0-latest](#)
- [emr-6.4.0-20240321](#)
- [emr-6.4.0-20221219](#)
- [emr-6.4.0-20210830](#)

### Amazon EMR 6.4.0용 릴리스 정보

- 지원되는 애플리케이션 - Spark 3.1.2-amzn-0, Jupyter Enterprise Gateway(엔드포인트, 퍼블릭 평가판).
- 지원되는 구성 요소 - `aws-hm-client`(Glue 커넥터), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- 지원되는 구성 분류:

| 분류                      | 설명                                               |
|-------------------------|--------------------------------------------------|
| <code>core-site</code>  | Hadoop <code>core-site.xml</code> 파일에서 값을 변경합니다. |
| <code>emrfs-site</code> | EMRFS 설정을 변경합니다.                                 |

| 분류              | 설명                                       |
|-----------------|------------------------------------------|
| spark-metrics   | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults  | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env       | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j     | Spark log4j.properties 파일에서 값을 변경합니다.    |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하십시오.

## 알려진 문제

- 대화형 엔드포인트를 포함하는 Spark 포트 템플릿 기능은 Amazon EMR on EKS 릴리스 6.4에서 작동하지 않습니다.

## emr-6.4.0-latest

릴리스 정보: emr-6.4.0-latest는 현재 emr-6.4.0-20240321를 가리킵니다.

리전: emr-6.4.0-latest는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-6.4.0:latest

## emr-6.4.0-20240321

릴리스 정보: emr-6.4.0-20240321 버전은 2024년 3월 11일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.



리전: `emr-6.4.0-20240321`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.4.0:20240321`

## emr-6.4.0-20221219

릴리스 정보: `emr-6.4.0-20221219`는 2023년 1월 27일에 출시되었습니다. 이전 버전과 비교하여 최근에 추가된 Amazon Linux 패키지를 포함하도록 이 버전을 새로 고쳤습니다.

리전: `emr-6.4.0-20221219`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.4.0:20221219`

## emr-6.4.0-20210830

릴리스 정보: `emr-6.4.0-20210830`은 2021년 12월 9일에 출시되었습니다. Amazon EMR 6.4.0의 최초 릴리스입니다.

리전: `emr-6.4.0-20210830`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.4.0:20210830`

## Amazon EMR on EKS 6.3.0 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 6.3.0 릴리스를 사용할 수 있습니다. 특정 `emr-6.3.0-XXXX` 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-6.3.0-latest](#)
- [emr-6.3.0-20240321](#)
- [emr-6.3.0-20220802](#)
- [emr-6.3.0-20211008](#)
- [emr-6.3.0-20210802](#)
- [emr-6.3.0-20210429](#)

Amazon EMR 6.3.0용 릴리스 정보

- 새로운 기능 - 6.x 릴리스 시리즈의 Amazon EMR 6.3.0부터 Amazon EMR on EKS는 Spark의 포트 템플릿 기능을 지원합니다. 또한 Amazon EMR on EKS에 대한 Spark 이벤트 로그 로테이션 기능을 켤 수도 있습니다. 자세한 내용은 [포트 템플릿 사용](#) 및 [Spark 이벤트 로그 로테이션 사용](#) 섹션을 참조하세요.
- 지원되는 애플리케이션 - Spark 3.1.1-amzn-0, Jupyter Enterprise Gateway(엔드포인트, 퍼블릭 평가판).
- 지원되는 구성 요소 - aws-hm-client(Glue 커넥터), aws-sagemaker-spark-sdk, emr-s3-select, emrfs, emr-ddb, hudi-spark.
- 지원되는 구성 분류:

| 분류              | 설명                                       |
|-----------------|------------------------------------------|
| core-site       | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site      | EMRFS 설정을 변경합니다.                         |
| spark-metrics   | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults  | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env       | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j     | Spark log4j.properties 파일에서 값을 변경합니다.    |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하십시오.

## emr-6.3.0-latest

릴리스 정보: emr-6.3.0-latest는 현재 emr-6.3.0-20240321를 가리킵니다.

리전: `emr-6.3.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.3.0:latest`

## `emr-6.3.0-20240321`

릴리스 정보: `emr-6.3.0-20240321` 버전은 2024년 3월 11일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.3.0-20240321`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.3.0:20240321`

## `emr-6.3.0-20220802`

릴리스 정보: `emr-6.3.0-20220802`는 2022년 9월 27일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지를 포함한 새 버전입니다.

리전: `emr-6.3.0-20220802`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.3.0:20220802`

## `emr-6.3.0-20211008`

릴리스 정보: `emr-6.3.0-20211008`은 2021년 12월 9일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 문제 수정 사항 및 보안 업데이트를 포함합니다.

리전: `emr-6.3.0-20211008`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.3.0:20211008`

## `emr-6.3.0-20210802`

릴리스 정보: `emr-6.3.0-20210802`는 2021년 8월 2일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 문제 수정 사항 및 보안 업데이트를 포함합니다.

리전: `emr-6.3.0-20210802`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.3.0:20210802`

## emr-6.3.0-20210429

릴리스 정보: `emr-6.3.0-20210429`는 2021년 4월 29일에 출시되었습니다. Amazon EMR 6.3.0의 최초 릴리스입니다.

리전: `emr-6.3.0-20210429`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.3.0:20210429`

## Amazon EMR on EKS 6.2.0 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 6.2.0 릴리스를 사용할 수 있습니다. 특정 `emr-6.2.0-XXXX` 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-6.2.0-latest](#)
- [emr-6.2.0-20240321](#)
- [emr-6.2.0-20220802](#)
- [emr-6.2.0-20211008](#)
- [emr-6.2.0-20210802](#)
- [emr-6.2.0-20210615](#)
- [emr-6.2.0-20210129](#)
- [emr-6.2.0-20201218](#)
- [emr-6.2.0-20201201](#)

### Amazon EMR 6.2.0용 릴리스 정보

- 지원되는 애플리케이션 - Spark 3.0.1-amzn-0, Jupyter Enterprise Gateway(엔드포인트, 퍼블릭 평가판).
- 지원되는 구성 요소 - `aws-hm-client`(Glue 커넥터), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- 지원되는 구성 분류:

| 분류              | 설명                                       |
|-----------------|------------------------------------------|
| core-site       | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site      | EMRFS 설정을 변경합니다.                         |
| spark-metrics   | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults  | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env       | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j     | Spark log4j.properties 파일에서 값을 변경합니다.    |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하십시오.

## emr-6.2.0-latest

릴리스 정보: emr-6.2.0-latest는 현재 emr-6.2.0-20240321를 가리킵니다.

리전: emr-6.2.0-latest는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-6.2.0:20240321

## emr-6.2.0-20240321

릴리스 정보: emr-6.2.0-20240321 버전은 2024년 3월 11일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-6.2.0-20240321`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.2.0:20240321`

## emr-6.2.0-20220802

릴리스 정보: `emr-6.2.0-20220802`는 2022년 9월 27일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지를 포함한 새 버전입니다.

리전: `emr-6.2.0-20220802`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-6.2.0:20220802`

## emr-6.2.0-20211008

릴리스 정보: `emr-6.2.0-20211008`은 2021년 12월 9일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 문제 수정 사항 및 보안 업데이트를 포함합니다.

리전: `emr-6.2.0-20211008`은 미국 동부(버지니아 북부), 미국 서부(오레곤), 아시아 태평양(도쿄), 유럽(아일랜드), 남아메리카(상파울루) 리전에서 사용할 수 있습니다.

컨테이너 이미지 태그: `emr-6.2.0:20211008`

## emr-6.2.0-20210802

릴리스 정보: `emr-6.2.0-20210802`는 2021년 8월 2일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 문제 수정 사항 및 보안 업데이트를 포함합니다.

리전: `emr-6.2.0-20210802`은 미국 동부(버지니아 북부), 미국 서부(오레곤), 아시아 태평양(도쿄), 유럽(아일랜드), 남아메리카(상파울루) 리전에서 사용할 수 있습니다.

컨테이너 이미지 태그: `emr-6.2.0:20210802`

## emr-6.2.0-20210615

릴리스 정보: `emr-6.2.0-20210615`는 2021년 6월 15일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 문제 수정 사항 및 보안 업데이트를 포함합니다.

리전: `emr-6.2.0-20210615`은 미국 동부(버지니아 북부), 미국 서부(오레곤), 아시아 태평양(도쿄), 유럽(아일랜드), 남아메리카(상파울루) 리전에서 사용할 수 있습니다.

컨테이너 이미지 태그: `emr-6.2.0:20210615`

## emr-6.2.0-20210129

릴리스 정보: `emr-6.2.0-20210129`는 2021년 1월 29일에 출시되었습니다.

`emr-6.2.0-20210129`와 비교하여 이 버전은 문제 수정 사항 및 보안 업데이트를 포함합니다.

리전: `emr-6.2.0-20210129`는 미국 동부(버지니아 북부), 미국 서부(오레곤), 아시아 태평양(도쿄), 유럽(아일랜드), 남아메리카(상파울루) 리전에서 사용할 수 있습니다.

컨테이너 이미지 태그: `emr-6.2.0-20210129`

## emr-6.2.0-20201218

릴리스 정보: `emr-6.2.0-20201218`는 2020년 12월 18일에 출시되었습니다.

`emr-6.2.0-20201218`와 비교하여 이 버전은 문제 수정 사항 및 보안 업데이트를 포함합니다.

리전: `emr-6.2.0-20201218`는 미국 동부(버지니아 북부), 미국 서부(오레곤), 아시아 태평양(도쿄), 유럽(아일랜드), 남아메리카(상파울루) 리전에서 사용할 수 있습니다.

컨테이너 이미지 태그: `emr-6.2.0-20201218`

## emr-6.2.0-20201201

릴리스 정보: `emr-6.2.0-20201201`는 2020년 12월 1일에 출시되었습니다. Amazon EMR 6.2.0의 최초 릴리스입니다.

리전: `emr-6.2.0-20201201`는 미국 동부(버지니아 북부), 미국 서부(오레곤), 아시아 태평양(도쿄), 유럽(아일랜드), 남아메리카(상파울루) 리전에서 사용할 수 있습니다.

컨테이너 이미지 태그: `emr-6.2.0-20201201`

## Amazon EMR on EKS 5.36.0 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 5.36.0 릴리스를 사용할 수 있습니다. 특정 `emr-5.36.0-XXXX` 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-5.36.0-latest](#)
- [emr-5.36.0-20240321](#)

- [emr-5.36.0-20221219](#)
- [emr-5.36.0-20220620](#)
- [emr-5.36.0-20220525](#)

## Amazon EMR 5.36.0용 릴리스 정보

- log4j2 보안 문제를 수정했습니다.
- 지원되는 애플리케이션 - Spark 2.4.8-amzn-2, Jupyter Enterprise Gateway(엔드포인트, 퍼블릭 평가판, Scala 커널은 지원되지 않음), livy-0.7.1, fluentd-4.0.0.
- 지원되는 구성 요소 - aws-hm-client, aws-sagemaker-spark-sdk, emr-ddb, emr-goodies, emr-kinesis, kerberos-server.
- 지원되는 구성 분류:

| 분류              | 설명                                       |
|-----------------|------------------------------------------|
| core-site       | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site      | EMRFS 설정을 변경합니다.                         |
| spark-metrics   | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults  | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env       | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j     | Spark log4j.properties 파일에서 값을 변경합니다.    |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하십시오.



## emr-5.36.0-latest

릴리스 정보: `emr-5.36.0-latest`는 현재 `emr-5.36.0-20240321`를 가리킵니다.

리전: `emr-5.36.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.36.0:latest`

## emr-5.36.0-20240321

릴리스 정보: `emr-5.36.0-20240321` 버전은 2024년 3월 11일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-5.36.0-20240321`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.36.0:20240321`

## emr-5.36.0-20221219

릴리스 정보: `emr-5.36.0-20221219`는 2023년 1월 27일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지를 포함한 새 버전입니다.

리전: `emr-5.36.0-20221219`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.36.0:20221219`

## emr-5.36.0-20220620

릴리스 정보: `emr-5.36.0-20220620`은 2022년 7월 27일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지를 포함한 새 버전입니다.

리전: `emr-5.36.0-20220620`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.36.0:20220620`

## emr-5.36.0-20220525

릴리스 정보: `emr-5.36.0-20220525`는 2022년 6월 16일에 출시되었습니다. Amazon EMR 5.36.0의 최초 릴리스입니다.

리전: `emr-5.36.0-20220525`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.36.0:20220525`

## Amazon EMR on EKS 5.35.0 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 5.35.0 릴리스를 사용할 수 있습니다. 특정 `emr-5.35.0-XXXX` 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-5.35.0-latest](#)
- [emr-5.35.0-20240321](#)
- [emr-5.35.0-20221219](#)
- [emr-5.35.0-20220802](#)
- [emr-5.35.0-20220307](#)

### Amazon EMR 5.35.0용 릴리스 정보

- log4j2 보안 문제를 수정했습니다.
- 지원되는 애플리케이션 - Spark 2.4.8-amzn-1, Hudi 0.9.0-amzn-2, Jupyter Enterprise Gateway(엔드포인트, 퍼블릭 평가판, Scala 커널은 지원되지 않음).
- 지원되는 구성 요소 - aws-hm-client(Glue 커넥터), aws-sagemaker-spark-sdk, emr-s3-select, emrfs, emr-ddb, hudi-spark.
- 지원되는 구성 분류:

| 분류         | 설명                                  |
|------------|-------------------------------------|
| core-site  | Hadoop core-site.xml 파일에서 값을 변경합니다. |
| emrfs-site | EMRFS 설정을 변경합니다.                    |

| 분류              | 설명                                       |
|-----------------|------------------------------------------|
| spark-metrics   | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults  | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env       | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j     | Spark log4j.properties 파일에서 값을 변경합니다.    |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하십시오.

## emr-5.35.0-latest

릴리스 정보: emr-5.35.0-latest는 현재 emr-5.35.0-20240321를 가리킵니다.

리전: emr-5.35.0-latest는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-5.35.0:latest

## emr-5.35.0-20240321

릴리스 정보: emr-5.35.0-20240321 버전은 2024년 3월 11일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: emr-5.35.0-20240321는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-5.35.0:20240321

## emr-5.35.0-20221219

릴리스 정보: `emr-5.35.0-20221219`는 2023년 1월 27일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지를 포함한 새 버전입니다.

리전: `emr-5.35.0-20221219`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.35.0:20221219`

## emr-5.35.0-20220802

릴리스 정보: `emr-5.35.0-20220802`는 2022년 9월 27일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지를 포함한 새 버전입니다.

리전: `emr-5.35.0-20220802`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.35.0:20220802`

## emr-5.35.0-20220307

릴리스 정보: `emr-5.35.0-20220307`은 2022년 5월 30일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지를 포함한 새 버전입니다.

리전: `emr-5.35.0-20220307`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.35.0:20220307`

## Amazon EMR on EKS 5.34.0 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 5.34.0 릴리스를 사용할 수 있습니다. 특정 `emr-5.34.0-XXXX` 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-5.34.0-latest](#)
- [emr-5.34.0-20240321](#)
- [emr-5.34.0-20220802](#)

## Amazon EMR 5.34.0용 릴리스 정보

- 지원되는 애플리케이션 - Spark 2.4.8-amzn-0, Jupyter Enterprise Gateway(엔드포인트, 퍼블릭 평가판, Scala 커널은 지원되지 않음).
- 지원되는 구성 요소 - aws-hm-client(Glue 커넥터), aws-sagemaker-spark-sdk, emr-s3-select, emrfs, emr-ddb, hudi-spark.
- 지원되는 구성 분류:

| 분류              | 설명                                       |
|-----------------|------------------------------------------|
| core-site       | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site      | EMRFS 설정을 변경합니다.                         |
| spark-metrics   | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults  | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env       | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j     | Spark log4j.properties 파일에서 값을 변경합니다.    |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하십시오.

## emr-5.34.0-latest

릴리스 정보: emr-5.34.0-latest는 현재 emr-5.34.0-20220802를 가리킵니다.

리전: emr-5.34.0-latest는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.34.0:latest`

## emr-5.34.0-20240321

릴리스 정보: `emr-5.34.0-20240321` 버전은 2024년 3월 11일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-5.34.0-20240321`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.34.0:20240321`

## emr-5.34.0-20220802

릴리스 정보: `emr-5.34.0-20220802`는 2022년 8월 24일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지를 포함한 새 버전입니다.

리전: `emr-5.34.0-20220802`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.34.0:20220802`

## emr-5.34.0-20211208

릴리스 정보: `emr-5.34.0-20211208`는 2022년 1월 20일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지를 포함한 새 버전입니다.

리전: `emr-5.34.0-20211208`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.34.0:20211208`

## Amazon EMR on EKS 5.33.0 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 5.33.0 릴리스를 사용할 수 있습니다. 특정 `emr-5.33.0-XXXX` 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-5.33.0-latest](#)
- [emr-5.33.0-20240321](#)

- [emr-5.33.0-20221219](#)
- [emr-5.33.0-20220802](#)
- [emr-5.33.0-20211008](#)
- [emr-5.33.0-20210802](#)
- [emr-5.33.0-20210615](#)
- [emr-5.33.0-20210323](#)

## Amazon EMR 5.33.0용 릴리스 정보

- 새로운 기능 - 5.x 릴리스 시리즈의 Amazon EMR 5.33.0부터 Amazon EMR on EKS는 Spark의 포드 템플릿 기능을 지원합니다. 자세한 내용은 [포드 템플릿 사용](#) 단원을 참조하십시오.
- 지원되는 애플리케이션 - Spark 2.4.7-amzn-1, Jupyter Enterprise Gateway(엔드포인트, 퍼블릭 평가판, Scala 커널은 지원되지 않음).
- 지원되는 구성 요소 - aws-hm-client(Glue 커넥터), aws-sagemaker-spark-sdk, emr-s3-select, emrfs, emr-ddb, hudi-spark.
- 지원되는 구성 분류:

| 분류              | 설명                                       |
|-----------------|------------------------------------------|
| core-site       | Hadoop core-site.xml 파일에서 값을 변경합니다.      |
| emrfs-site      | EMRFS 설정을 변경합니다.                         |
| spark-metrics   | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults  | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env       | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j     | Spark log4j.properties 파일에서 값을 변경합니다.    |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 `spark-hive-site.xml`과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하십시오.

## emr-5.33.0-latest

릴리스 정보: `emr-5.33.0-latest`는 현재 `emr-5.33.0-20240321`를 가리킵니다.

리전: `emr-5.33.0-latest`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.33.0:latest`

## emr-5.33.0-20240321

릴리스 정보: `emr-5.33.0-20240321` 버전은 2024년 3월 11일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-5.33.0-20240321`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.33.0:20240321`

## emr-5.33.0-20221219

릴리스 정보: `emr-5.33.0-20221219`는 2023년 1월 19일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: `emr-5.33.0-20221219`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.33.0:20221219`

## emr-5.33.0-20220802

릴리스 정보: `emr-5.33.0-20220802`는 2022년 8월 24일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지를 포함한 새 버전입니다.



리전: `emr-5.33.0-20220802`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.33.0:20220802`

## `emr-5.33.0-20211008`

릴리스 정보: `emr-5.33.0-20211008`은 2021년 12월 9일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 문제 수정 사항 및 보안 업데이트를 포함합니다.

리전: `emr-5.33.0-20211008`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.33.0:20211008`

## `emr-5.33.0-20210802`

릴리스 정보: `emr-5.33.0-20210802`는 2021년 8월 2일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 문제 수정 사항 및 보안 업데이트를 포함합니다.

리전: `emr-5.33.0-20210802`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.33.0:20210802`

## `emr-5.33.0-20210615`

릴리스 정보: `emr-5.33.0-20210615`는 2021년 6월 15일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 문제 수정 사항 및 보안 업데이트를 포함합니다.

리전: `emr-5.33.0-20210615`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.33.0:20210615`

## `emr-5.33.0-20210323`

릴리스 정보: `emr-5.33.0-20210323`은 2021년 3월 23일에 출시되었습니다. Amazon EMR 5.33.0의 최초 릴리스입니다.

리전: `emr-5.33.0-20210323`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.33.0-20210323`

## Amazon EMR on EKS 5.32.0 릴리스

Amazon EMR on EKS에 대해 다음 Amazon EMR 5.32.0 릴리스를 사용할 수 있습니다. 특정 `emr-5.32.0-XXXX` 릴리스를 선택하여 관련 컨테이너 이미지 태그와 같은 세부 정보를 확인합니다.

- [emr-5.32.0-latest](#)
- [emr-5.32.0-20240321](#)
- [emr-5.32.0-20220802](#)
- [emr-5.32.0-20211008](#)
- [emr-5.32.0-20210802](#)
- [emr-5.32.0-20210615](#)
- [emr-5.32.0-20210129](#)
- [emr-5.32.0-20201218](#)
- [emr-5.32.0-20201201](#)

### Amazon EMR 5.32.0용 릴리스 정보

- 지원되는 애플리케이션 - Spark 2.4.7-amzn-0, Jupyter Enterprise Gateway(엔드포인트, 퍼블릭 평가판, Scala 커널은 지원되지 않음).
- 지원되는 구성 요소 - `aws-hm-client`(Glue 커넥터), `aws-sagemaker-spark-sdk`, `emr-s3-select`, `emrfs`, `emr-ddb`, `hudi-spark`.
- 지원되는 구성 분류:

| 분류                      | 설명                                               |
|-------------------------|--------------------------------------------------|
| <code>core-site</code>  | Hadoop <code>core-site.xml</code> 파일에서 값을 변경합니다. |
| <code>emrfs-site</code> | EMRFS 설정을 변경합니다.                                 |

| 분류              | 설명                                       |
|-----------------|------------------------------------------|
| spark-metrics   | Spark metrics.properties 파일에서 값을 변경합니다.  |
| spark-defaults  | Spark spark-defaults.conf 파일에서 값을 변경합니다. |
| spark-env       | Spark 환경에서 값을 변경합니다.                     |
| spark-hive-site | Spark hive-site.xml 파일에서 값을 변경합니다.       |
| spark-log4j     | Spark log4j.properties 파일에서 값을 변경합니다.    |

구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 이는 종종 spark-hive-site.xml과 같이 애플리케이션의 구성 XML 파일에 해당합니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하십시오.

## emr-5.32.0-latest

릴리스 정보: emr-5.32.0-latest는 현재 emr-5.32.0-20240321를 가리킵니다.

리전: emr-5.32.0-latest는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-5.32.0:latest

## emr-5.32.0-20240321

릴리스 정보: emr-5.32.0-20240321 버전은 2024년 3월 11일에 출시되었습니다. 이전 릴리스와 비교하여 이 릴리스는 최근 업데이트된 Amazon Linux 패키지 및 중요 수정 사항을 포함한 새 버전입니다.

리전: emr-5.32.0-20240321는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: emr-5.32.0:20240321

## emr-5.32.0-20220802

릴리스 정보: `emr-5.32.0-20220802`는 2022년 8월 24일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 최근 업데이트된 Amazon Linux 패키지를 포함한 새 버전입니다.

리전: `emr-5.32.0-20220802`는 Amazon EMR on EKS에서 지원하는 모든 리전에서 사용할 수 있습니다. 자세한 내용은 [Amazon EMR on EKS 서비스 엔드포인트](#)를 참조하세요.

컨테이너 이미지 태그: `emr-5.32.0:20220802`

## emr-5.32.0-20211008

릴리스 정보: `emr-5.32.0-20211008`은 2021년 12월 9일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 문제 수정 사항 및 보안 업데이트를 포함합니다.

리전: `emr-5.32.0-20211008`은 미국 동부(버지니아 북부), 미국 서부(오레곤), 아시아 태평양(도쿄), 유럽(아일랜드), 남아메리카(상파울루) 리전에서 사용할 수 있습니다.

컨테이너 이미지 태그: `emr-5.32.0:20211008`

## emr-5.32.0-20210802

릴리스 정보: `emr-5.32.0-20210802`는 2021년 8월 2일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 문제 수정 사항 및 보안 업데이트를 포함합니다.

리전: `emr-5.32.0-20210802`은 미국 동부(버지니아 북부), 미국 서부(오레곤), 아시아 태평양(도쿄), 유럽(아일랜드), 남아메리카(상파울루) 리전에서 사용할 수 있습니다.

컨테이너 이미지 태그: `emr-5.32.0:20210802`

## emr-5.32.0-20210615

릴리스 정보: `emr-5.32.0-20210615`는 2021년 6월 15일에 출시되었습니다. 이전 버전과 비교하여 이 버전은 문제 수정 사항 및 보안 업데이트를 포함합니다.

리전: `emr-5.32.0-20210615`은 미국 동부(버지니아 북부), 미국 서부(오레곤), 아시아 태평양(도쿄), 유럽(아일랜드), 남아메리카(상파울루) 리전에서 사용할 수 있습니다.

컨테이너 이미지 태그: `emr-5.32.0:20210615`

## emr-5.32.0-20210129

릴리스 정보: emr-5.32.0-20210129는 2021년 1월 29일에 출시되었습니다.

emr-5.32.0-20201218과 비교하여 이 버전은 문제 수정 사항 및 보안 업데이트를 포함합니다.

리전: emr-5.32.0-20210129는 미국 동부(버지니아 북부), 미국 서부(오레곤), 아시아 태평양(도쿄), 유럽(아일랜드), 남아메리카(상파울루) 리전에서 사용할 수 있습니다.

컨테이너 이미지 태그: emr-5.32.0-20210129

## emr-5.32.0-20201218

릴리스 정보: 5.32.0-20201218은 2020년 12월 18일에 출시되었습니다. 5.32.0-20201201과 비교하여 이 버전은 문제 수정 사항 및 보안 업데이트를 포함합니다.

리전: emr-5.32.0-20201218은 미국 동부(버지니아 북부), 미국 서부(오레곤), 아시아 태평양(도쿄), 유럽(아일랜드), 남아메리카(상파울루) 리전에서 사용할 수 있습니다.

컨테이너 이미지 태그: emr-5.32.0-20201218

## emr-5.32.0-20201201

릴리스 정보: 5.32.0-20201201은 2020년 12월 1일에 출시되었습니다. Amazon EMR 5.32.0의 최초 릴리스입니다.

리전: 5.32.0-20201201은 미국 동부(버지니아 북부), 미국 서부(오리건), 아시아 태평양(도쿄), 유럽(아일랜드), 남아메리카(상파울루) 리전에서 사용할 수 있습니다.

컨테이너 이미지 태그: emr-5.32.0-20201201

## 문서 이력

다음 테이블에서는 Amazon EMR on EKS의 최신 릴리스 이후 이 설명서에서 변경된 중요 사항을 설명합니다. 이 설명서의 업데이트에 대한 자세한 내용을 보려면 RSS 피드를 구독하면 됩니다.

| 변경                   | 설명                                                                                          | Date          |
|----------------------|---------------------------------------------------------------------------------------------|---------------|
| EMR 7.13.0 릴리스 정보 추가 | PySpark 및 애플리케이션 업그레이드에 대한 Python 3.11 기본값이 포함된 EMR 7.13.0 릴리스 정보 추가                        | 2026년 4월      |
| EMR 7.12.0 릴리스 정보 추가 | Iceberg 구체화된 뷰 및 Hudi 전체 테이블 액세스와 함께 EMR 7.12.0 릴리스 정보 추가                                   | 2025년 11월     |
| EMR 7.11.0 릴리스 정보 추가 | SageMaker Unified Studio 통합과 함께 EMR 7.11.0 릴리스 정보 추가                                        | 2025년 11월     |
| 업데이트 내용              | <a href="#">Amazon EMR on EKS 관리형 정책</a> – AmazonEMRContainersServiceRolePolicy 에 대한 추가 권한. | 2025년 2월 3일   |
| 새로운 릴리스              | <a href="#">Amazon EMR on EKS 7.6.0 릴리스</a>                                                 | 2025년 1월 10일  |
| 새로운 릴리스              | <a href="#">Amazon EMR on EKS 7.5.0 릴리스</a>                                                 | 2024년 11월 21일 |
| 새로운 릴리스              | <a href="#">Amazon EMR on EKS 7.4.0 릴리스</a>                                                 | 2024년 11월 13일 |
| 새로운 릴리스              | <a href="#">Amazon EMR on EKS 7.3.0 릴리스</a>                                                 | 2024년 10월 16일 |
| 새로운 릴리스              | <a href="#">Amazon EMR on EKS 7.2.0 릴리스</a>                                                 | 2024년 7월 25일  |
| 새로운 릴리스              | <a href="#">Amazon EMR on EKS 7.1.0 릴리스</a>                                                 | 2024년 4월 17일  |
| 새로운 릴리스              | <a href="#">Amazon EMR on EKS 7.0.0 릴리스</a>                                                 | 2023년 12월 22일 |
| 새로운 릴리스              | <a href="#">Amazon EMR on EKS 6.15.0 릴리스</a>                                                | 2023년 11월 17일 |
| 새로운 릴리스              | <a href="#">Amazon EMR on EKS 6.14.0 릴리스</a>                                                | 2023년 10월 17일 |

| 변경      | 설명                                                                                                                       | Date         |
|---------|--------------------------------------------------------------------------------------------------------------------------|--------------|
| 업데이트 내용 | '관리형 엔드포인트'의 이름을 <a href="#">대화형 엔드포인트</a> 로 바꾸고, <a href="#">대화형 엔드포인트 정식 출시</a>                                        | 2023년 9월 29일 |
| 새로운 릴리스 | <a href="#">Amazon EMR on EKS 6.13.0 릴리스</a> 및 <a href="#">Amazon EMR on EKS를 사용하여 Flink 작업 실행</a> 에 대한 퍼블릭 평가판 문서       | 2023년 9월 12일 |
| 새로운 릴리스 | <a href="#">Amazon EMR on EKS 6.12.0 릴리스</a>                                                                             | 2023년 7월 21일 |
| 새로운 내용  | <a href="#">Amazon EMR on EKS에서 Apache Spark의 사용자 지정 스케줄러로 Volcano 사용</a> 이 추가되었습니다.                                     | 2023년 6월 13일 |
| 새로운 내용  | <a href="#">Amazon EMR on EKS에서 Apache Spark의 사용자 지정 스케줄러로 Volcano 사용</a> 이 추가되었습니다.                                     | 2023년 6월 13일 |
| 새로운 내용  | <a href="#">Spark 컨테이너 로그 로테이션 사용</a> 이 추가되었습니다.                                                                         | 2023년 6월 12일 |
| 업데이트 내용 | Amazon ECR Public Gallery에서 기본 이미지 정보를 찾을 수 있도록 <a href="#">사용자 지정 이미지 문서</a> 가 업데이트되었습니다.                               | 2023년 6월 8일  |
| 새로운 릴리스 | <a href="#">Amazon EMR on EKS 6.11.0 릴리스</a>                                                                             | 2023년 6월 8일  |
| 새로운 내용  | <a href="#">Spark 운영자에서 Spark 작업 실행</a> 를 추가했고 <a href="#">Amazon EMR on EKS를 사용하여 Spark 작업 실행</a> 아래 작업 실행 섹션을 재구성했습니다. | 2023년 6월 5일  |
| 새로운 내용  | <a href="#">Amazon EMR Spark 작업에 수직 자동 조정 사용</a> 및 <a href="#">자체 호스팅 Jupyter Notebook 사용</a> 섹션 2개를 추가함                 | 2023년 5월 4일  |

| 변경         | 설명                                        | Date         |
|------------|-------------------------------------------|--------------|
| 문서 기록 페이지  | Amazon EMR on EKS에 대한 문서 기록 페이지를 생성했습니다.  | 2023년 3월 13일 |
| 관리형 정책 페이지 | Amazon EMR on EKS에 대한 관리형 정책 페이지를 생성했습니다. | 2023년 3월 13일 |



기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.