

사용 설명서

Amazon CodeCatalyst



Amazon CodeCatalyst: 사용 설명서

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 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

.....	xxii
Amazon CodeCatalyst란?	1
CodeCatalyst로 할 수 있는 작업은 무엇인가요?	1
CodeCatalyst는 어떻게 시작할 수 있나요?	2
CodeCatalyst에 대해 자세히 알아보기	2
CodeCatalyst에서 마이그레이션하는 방법	3
리포지토리 마이그레이션	3
CodeCatalyst 리포지토리를 GitLab 리포지토리로 마이그레이션	3
CodeCatalyst 리포지토리를 GitHub 리포지토리로 마이그레이션	3
다른 리포지토리 공급자로 일반 마이그레이션	3
CodeCatalyst에서 데이터 추출	5
아티팩트 다운로드	5
문제 첨부 파일 다운로드	6
작업의 소스 코드 다운로드	6
CodeCatalyst에서 데이터 삭제	6
서비스 팀에 데이터를 대신 삭제해 주도록 요청	7
CodeCatalyst 스페이스 삭제	7
프로젝트 삭제	8
소스 리포지토리 삭제	8
사용자 지정 블루프린트 삭제	9
문제 첨부 파일 삭제	10
Amazon CodeCatalyst를 통해 액세스하는 개발 환경에서 파일 삭제	10
스페이스의 개발 환경 삭제	10
계정 연결 삭제	11
AWS 관리 콘솔의 CodeCatalyst 스페이스에서 계정 제거	11
보안 암호 삭제	12
팀 삭제	13
프로비저닝된 플릿 삭제	13
패키지 리포지토리 삭제	14
개념	15
AWS CodeCatalyst의 Builder ID 스페이스	16
CodeCatalyst에서 ID 페더레이션을 지원하는 스페이스	16
Projects	16
블루프린트	16

계정 연결	17
VPC 연결	17
AWS 빌더 ID	17
CodeCatalyst의 사용자 프로파일	18
소스 리포지토리	18
커밋	19
개발 환경	19
워크플로	19
작업	20
문제	20
개인 액세스 토큰(PAT)	20
개인 연결	20
역할	21
CodeCatalyst 설정 및 로그인	22
새 스페이스 및 개발 역할 생성(초대 없이 시작)	24
새 스페이스 및 IAM 역할 생성	25
초대 수락 및 AWS Builder ID 생성	30
초대 수락 및 AWS Builder ID 생성	31
AWS Builder ID로 로그인	32
신뢰할 수 있는 디바이스	33
SSO로 로그인	33
사용자의 모든 스페이스 및 프로젝트 보기	33
CodeCatalyst 프로파일 보기 및 관리	34
CodeCatalyst 프로파일 보기	35
다른 사용자의 CodeCatalyst 프로파일 보기	35
프로파일 업데이트	36
AWS Builder ID와 연결된 CodeCatalyst 암호 변경	37
CodeCatalyst AWS CLI 에서를 사용하도록 설정	37
시작하기 자습서	39
자습서: 최신 3계층 웹 애플리케이션 블루프린트로 프로젝트 만들기	40
사전 조건	42
1단계: 최신 3계층 웹 애플리케이션 프로젝트 생성	43
2단계: 프로젝트에 초대하기	44
3단계: 협업 및 작업 추적을 위한 문제 생성	45
4단계: 소스 리포지토리 보기	45
5단계: 테스트 브랜치를 사용하여 개발 환경 생성 및 빠른 코드 변경	46

6단계: 최신 애플리케이션을 빌드하는 워크플로 보기	48
7단계: 다른 사람에게 변경 사항을 검토하도록 요청	51
8단계: 문제 종료	54
리소스 정리	54
레퍼런스	55
자습서: 빈 프로젝트로 시작	57
사전 조건	58
빈 프로젝트 생성	58
소스 리포지토리 생성	58
코드 변경을 빌드, 테스트 및 배포하기 위한 워크플로 생성	60
프로젝트에 초대하기	60
작업을 공동 작업하고 추적하기 위한 문제 생성	60
자습서: 생성형 AI 기능 사용	61
사전 조건	63
Amazon Q를 사용하여 프로젝트를 생성하거나 기능을 추가할 때 블루프린트 선택	63
풀 요청을 생성할 때 자동 생성된 요약 추가	67
풀 요청의 코드 변경에 대해 남긴 주석 요약 생성	70
문제를 생성하고 Amazon Q에 할당	71
문제를 생성하고 Amazon Q에서 해당 이슈에 대한 작업 추천받기	77
리소스 정리	78
스페이스로 리소스 구성	79
스페이스 생성	81
스페이스 편집	83
스페이스 삭제	84
스페이스 내 사용자 및 리소스에 대한 활동 모니터링	85
연결된 AWS 리소스에 대한 액세스 허용 AWS 계정	86
스페이스 AWS 계정 에 추가	87
계정 연결에 IAM 역할 추가	90
배포 환경에 계정 연결 및 IAM 역할 추가	92
계정 연결 보기	93
계정 연결 삭제(CodeCatalyst)	93
스페이스에 대한 청구 계정 구성	94
연결된 계정에 대한 IAM 역할 구성	94
CodeCatalystWorkflowDevelopmentRole- <i>spaceName</i> 역할	95
AWSRoleForCodeCatalystSupport 역할	96
IAM 역할 생성 및 CodeCatalyst 신뢰 정책 사용	97

사용자에게 스페이스 권한 부여	98
스페이스에서 멤버 보기	99
사용자를 직접 스페이스에 초대	100
스페이스 초대 취소	101
스페이스 멤버의 역할 변경	102
스페이스 멤버 제거	103
스페이스 관리자 역할이 있는 사용자의 역할 제거 또는 변경	103
팀을 사용하여 스페이스 액세스 허용	105
팀 생성	105
팀 보기	107
팀에 스페이스 역할 부여	108
스페이스 수준에서 팀에 프로젝트 역할 부여	108
팀에 직접 사용자 추가	109
팀에서 사용자 직접 제거	110
팀에 SSO 그룹 추가	111
팀 삭제	111
시스템 리소스에 대한 스페이스 액세스 허용	112
시스템 리소스에 대한 스페이스 액세스 보기	113
시스템 리소스에 대한 스페이스 액세스 비활성화	113
시스템 리소스에 대한 스페이스 액세스 활성화	114
스페이스의 개발 환경 관리	114
스페이스의 개발 환경 보기	115
스페이스의 개발 환경 편집	116
스페이스의 개발 환경 중지	117
스페이스의 개발 환경 삭제	117
스페이스 할당량	118
프로젝트 작업 구성	120
프로젝트 생성	121
빈 프로젝트 생성	121
연결된 타사 리포지토리를 사용하여 프로젝트 생성	122
블루프린트를 사용하여 프로젝트 생성	127
Amazon Q를 사용하여 프로젝트를 생성하거나 블루프린트를 사용하여 기능을 추가할 때의 모범 사례	129
프로젝트에서 블루프린트를 사용하는 모범 사례	131
생성된 프로젝트에 리소스 및 작업 추가	131
프로젝트 목록 가져오기	132

프로젝트 작업 및 개발 환경 보기	132
한 스페이스에서 모든 프로젝트 보기	133
프로젝트 설정 보기	133
CodeCatalyst에서 다른 프로젝트로 변경	134
프로젝트 삭제	134
사용자에게 프로젝트 권한 부여	134
멤버 및 해당 프로젝트 역할 목록 가져오기	135
프로젝트에 사용자 초대	136
초대 취소	137
프로젝트에서 사용자 제거	137
프로젝트 초대 수락 또는 거부	138
팀을 통해 프로젝트 액세스 허용	138
프로젝트에 팀 추가	139
팀에 프로젝트 역할 부여	139
팀의 프로젝트 역할 제거	140
시스템 리소스에 대한 프로젝트 액세스 허용	140
시스템 리소스에 대한 프로젝트 액세스 보기	141
시스템 리소스에 대한 프로젝트 액세스 비활성화	142
시스템 리소스에 대한 프로젝트 액세스 활성화	143
프로젝트 할당량	143
알림 전송	144
알림은 어떻게 작동합니까?	144
스택 알림 시작하기	146
Slack 및 이메일 알림 전송	149
블루프린트를 사용하여 프로젝트 설정	154
블루프린트를 사용하여 프로젝트 생성	155
프로젝트에 블루프린트를 추가하여 리소스 통합	155
프로젝트에서 블루프린트 연결 해제	158
프로젝트의 블루프린트 버전 변경	159
프로젝트의 블루프린트에 대한 설명 편집	161
블루프린트 사용자로서 수명 주기 관리 작업	162
기존 프로젝트에서 수명 주기 관리 사용	162
프로젝트의 여러 블루프린트에서 수명 주기 관리 사용	162
수명 주기 풀 요청의 충돌 처리	163
수명 주기 관리 변경 사항 옵트아웃	163
프로젝트에서 블루프린트의 수명 주기 관리 재정의	163

블루프린트를 사용하여 포괄적인 프로젝트 생성	164
사용 수 있는 블루프린트	164
프로젝트 블루프린트 정보 찾기	168
사용자 지정 블루프린트로 프로젝트 표준화	168
사용자 지정 블루프린트 개념	169
사용자 지정 블루프린트 시작하기	173
자습서: React 애플리케이션 생성 및 업데이트	178
소스 리포지토리를 사용자 지정 블루프린트로 변환	185
블루프린트 작성자로서 수명 주기 관리 작업	186
프로젝트 요구 사항을 충족하기 위한 사용자 지정 블루프린트 개발	192
사용자 지정 블루프린트를 스페이스에 게시	230
사용자 지정 블루프린트에 대한 게시 권한 설정	234
스페이스 블루프린트 카탈로그에 사용자 지정 블루프린트 추가	235
사용자 지정 블루프린트의 카탈로그 버전 변경	236
사용자 지정 블루프린트의 세부 정보, 버전 및 프로젝트 보기	236
스페이스 블루프린트 카탈로그에서 사용자 지정 블루프린트 제거	237
게시된 사용자 지정 블루프린트 또는 버전 삭제	238
종속성 추가, 불일치 처리, 도구 및 구성 요소 업그레이드	240
기여	242
블루프린트 할당량	242
소스 리포지토리를 사용하여 코드 저장 및 협업	244
프라이빗 리포지토리 개념	245
Projects	16
소스 리포지토리	246
개발 환경	19
개인 액세스 토큰(PAT)	20
브랜치	247
기본 브랜치	247
커밋	19
풀 요청	248
개정	248
워크플로	19
설정	249
Git 설치	250
개인용 액세스 토큰 생성	250
소스 리포지토리 시작하기	250

블루프린트를 사용하여 프로젝트 생성	251
프로젝트의 리포지토리 보기	253
개발 환경 생성	255
풀 요청 생성	256
풀 요청 병합	258
배포된 코드 보기	259
리소스 정리	260
리포지토리에 소스 코드 저장	260
소스 리포지토리 생성	261
기존 Git 리포지토리를 소스 리포지토리로 복제	263
소스 리포지토리 연결	266
소스 리포지토리 보기	269
소스 리포지토리의 설정 편집	270
소스 리포지토리 복제	271
소스 리포지토리 삭제	273
브랜치를 사용하여 소스 코드 작업 구성	274
브랜치 생성	275
기본 브랜치 관리	276
브랜치 규칙을 사용하여 허용되는 작업 관리	277
브랜치에 대한 Git 명령	280
브랜치 및 세부 정보 보기	281
브랜치 삭제	282
소스 코드 파일 관리	283
파일 생성 또는 추가	283
파일 보기	285
파일 변경 내역 보기	286
파일 편집	287
파일 이름 바꾸기 또는 삭제	288
풀 요청을 사용하여 코드 검토	288
풀 요청 생성	290
풀 요청 보기	293
승인 규칙과 병합하기 위한 요구 사항 관리	295
풀 요청 검토	297
풀 요청 업데이트	300
풀 요청 병합	301
풀 요청 닫기	304

커밋을 사용하여 소스 코드의 변경 사항 이해	305
브랜치에 대한 커밋 보기	305
커밋 표시 방식 변경(CodeCatalyst 콘솔)	306
소스 리포지토리의 할당량	307
개발 환경으로 코드 작성 및 수정	311
개발 환경 생성	312
개발 환경에 지원되는 통합 개발 환경	313
CodeCatalyst에서 개발 환경 만들기	313
IDE에서 개발 환경 생성	316
개발 환경 중지	316
개발 환경 재개	318
개발 환경 편집	319
개발 환경 삭제	320
SSH를 사용하여 개발 환경에 연결	321
devfile 구성 및 사용	323
devfile 편집	324
CodeCatalyst에서 지원하는 Devfile 기능	325
개발 환경용 devfile의 예	326
복구 모드를 사용하여 리포지토리 devfile 문제 해결	327
범용 devfile 이미지 지정	327
Devfile 명령	332
Devfile 이벤트	333
Devfile 구성 요소	333
VPC 연결을 개발 환경에 연결	334
개발 환경 할당량	335
소프트웨어 패키지 게시 및 공유	336
패키지 개념	337
Packages	337
패키지 네임스페이스	337
패키지 버전	338
자산	338
패키지 리포지토리	338
업스트림 리포지토리	338
게이트웨이 리포지토리	339
패키지 리포지토리 구성 및 사용	339
패키지 리포지토리 생성	340

패키지 리포지토리에 연결	340
패키지 리포지토리 삭제	341
업스트림 리포지토리 구성 및 사용	341
업스트림 리포지토리 추가	342
업스트림 리포지토리의 검색 순서 편집	343
업스트림 리포지토리가 포함된 패키지 버전 요청	344
업스트림 리포지토리 제거	347
퍼블릭 외부 리포지토리에 연결	347
지원되는 외부 패키지 리포지토리 및 해당 리포지토리의 게이트웨이 리포지토리	348
패키지 게시 및 수정	349
패키지 게시	349
패키지 버전 세부 정보 보기	351
패키지 버전 삭제	351
패키지 버전의 상태 업데이트	352
패키지 원본 제어 편집	353
npm 사용	358
npm 구성 및 사용	358
npm 태그 처리	367
Maven 사용	368
Gradle Groovy 구성 및 사용	369
mvn 구성 및 사용	378
curl을 사용하여 패키지 게시	385
Maven 체크섬 및 스냅샷 사용	387
NuGet 사용	389
Visual Studio에서 CodeCatalyst 사용	389
nuget 또는 dotnet CLI 구성 및 사용	391
NuGet 패키지 이름, 버전 및 자산 이름 표준화	394
NuGet 호환성	395
Python 사용	396
pip 구성 및 Python 패키지 설치	396
Twine 구성 및 Python 패키지 게시	400
Python 패키지 이름 정규화	402
Python 호환성	402
패키지 할당량	402
워크플로를 사용하여 빌드, 테스트 및 배포	404
워크플로 정의 파일 정보	404

CodeCatalyst 콘솔의 시각적 및 YAML 편집기 사용	406
워크플로 검색	408
워크플로 실행 세부 정보 보기	408
다음 단계	409
워크플로 개념	409
워크플로	409
워크플로 정의 파일	410
작업	410
작업 그룹	410
아티팩트	410
컴퓨팅	410
환경	411
게이트	411
Reports	411
실행	411
소스	412
변수	412
워크플로 트리거	412
워크플로 시작하기	412
사전 조건	413
1단계: 워크플로 생성 및 구성	414
2단계: 커밋을 사용하여 워크플로 저장	415
3단계: 실행 결과 보기	416
(선택 사항) 4단계: 정리	416
워크플로로 빌드하기	416
애플리케이션을 빌드하려면 어떻게 해야 하나요?	417
빌드 작업의 이점	418
빌드 작업의 대안	418
빌드 작업 추가	418
빌드 작업 결과 보기	419
자습서: Amazon S3에 아티팩트 업로드	420
YAML - 빌드 및 테스트 작업	429
워크플로를 사용한 테스트	456
품질 보고서 유형	456
테스트 작업 추가	459
테스트 작업 결과 보기	461

실패한 테스트 건너뛰기	461
universal-test-runner와 통합	462
품질 보고서 구성	464
테스트 모범 사례	469
SARIF 속성	472
워크플로를 사용하여 배포	480
애플리케이션을 배포하려면 어떻게 해야 하나요?	480
배포 작업 목록	481
작업 배포의 이점	482
작업을 배포하는 대안	482
Amazon ECS에 배포	483
Amazon EKS에 배포	533
CloudFormation 스택 배포	579
AWS CDK 앱 배포	629
AWS CDK 앱 부트스트래핑	653
Amazon S3에 게시	668
AWS 계정 및 VPCs에 배포	682
앱 URL 표시	693
배포 대상 제거	697
커밋별 배포 상태 추적	698
배포 로그 보기	699
배포 정보 보기	700
워크플로 생성	702
워크플로 실행	705
수동으로 실행 시작	706
트리거를 사용하여 자동으로 실행 시작	706
수동 전용 트리거 구성	722
워크플로 실행 중지	723
워크플로 게이팅	724
워크플로 실행에 대한 승인 요구	727
실행의 대기열 동작 구성	739
실행 간 파일 캐싱	745
워크플로 실행 상태 및 세부 정보 보기	749
워크플로 작업 구성	753
작업 유형	754
작업 추가	757

작업 제거	759
사용자 지정 작업 개발	760
작업을 작업 그룹으로 그룹화	761
작업 순서 지정	763
작업 간 아티팩트 및 파일 공유	767
사용할 작업 버전 지정	780
사용 가능한 작업 버전 나열	782
작업의 소스 코드 보기	783
GitHub Actions와 통합	784
컴퓨팅 및 런타임 이미지 구성	820
컴퓨팅 유형	821
컴퓨팅 플릿	821
온디맨드 플릿 속성	822
프로비저닝된 플릿 속성	823
프로비저닝된 플릿 생성	825
프로비저닝된 플릿 편집	826
프로비저닝된 플릿 삭제	826
작업에 플릿 또는 컴퓨팅 할당	827
작업 간에 컴퓨팅 공유	828
런타임 환경 이미지 지정	835
워크플로에 소스 리포지토리 연결	843
워크플로 파일의 소스 리포지토리 지정	844
워크플로 작업의 소스 리포지토리 지정	845
소스 리포지토리 파일 참조	846
변수 - 'BranchName' 및 'CommitId'	847
워크플로에 패키지 리포지토리 연결	848
자습서: 패키지 리포지토리에서 가져오기	848
워크플로에서 CodeCatalyst 패키지 리포지토리 지정	864
워크플로 작업에서 권한 부여 토큰 사용	865
예시: 워크플로의 패키지 리포지토리	867
Lambda 함수 호출	869
이 작업을 사용해야 하는 경우	870
'AWS Lambda 호출' 작업에서 사용하는 런타임 이미지	870
예시: Lambda 함수를 간접적으로 호출합니다.	870
'AWS Lambda 호출' 작업 추가	872
변수 - 'AWS Lambda 호출'	874

YAML - 'AWS Lambda 호출' 작업	875
Amazon ECS 작업 정의 수정	888
이 작업을 사용해야 하는 경우	889
'Amazon ECS 작업 정의 렌더링' 작업 작동 방식	889
'Amazon ECS 작업 정의 렌더링' 작업에 사용되는 런타임 이미지	891
예시: Amazon ECS 작업 정의 수정	891
'Amazon ECS 작업 정의 렌더링' 작업 추가	894
업데이트된 작업 정의 파일 보기	896
변수 - 'Amazon ECS 작업 정의 렌더링'	897
YAML - 'Amazon ECS 작업 정의 렌더링'	897
워크플로에서 변수 사용	906
사용자 정의된 변수 사용	906
사전 정의된 변수 사용	918
보안 암호를 사용하여 데이터 마스킹	923
보안 암호 생성	923
보안 암호 편집	924
보안 암호 사용	925
보안 암호 삭제	926
워크플로의 상태 보기	927
워크플로 할당량	928
워크플로 실행 상태	930
워크플로 상태	930
워크플로 YAML 정의	931
워크플로 정의 파일의 예시	932
구문 지침 및 규칙	933
최상위 속성	935
문제 작업 추적 및 구성	946
문제 개념	947
활성 문제	947
보관된 문제	947
담당자	947
사용자 지정 필드	948
추정	948
문제	948
Label	948
우선순위	948

상태 및 상태 범주	948
작업	949
보기	949
문제 관련 작업 추적	949
문제 생성	949
문제 추정	953
문제 편집 및 공동 작업	954
문제 찾기 및 보기	962
문제 진행	965
문제 보관	967
문제 내보내기	968
백로그, 레이블 및 보드로 작업 구성	968
레이블로 작업 범주화	968
사용자 지정 필드를 사용하여 작업 구성	970
사용자 지정 상태로 작업 추적	970
문제에 대한 노력 추정 구성	972
다수의 담당자 활성화 또는 비활성화	973
문제 보기 생성	973
문제에 대한 할당량	974
CodeCatalyst에서 ID, 권한 및 액세스 구성	975
사용자 역할로 액세스 권한 부여	976
스페이스 및 프로젝트의 사용자 역할 이해	976
각 역할이 사용할 수 있는 권한 보기	978
사용자 역할 보기 및 변경	1007
개인 액세스 토큰을 사용하여 사용자 리포지토리 액세스 권한 부여	1008
PAT 생성	1009
PAT 보기	1011
PAT 삭제	1012
.....	1013
개인 연결 생성	1014
개인 연결 삭제	1015
다중 인증(MFA)으로 로그인하도록 AWS Builder ID 구성	1016
다중 인증에 사용할 디바이스를 등록하는 방법	1017
인증 애플리케이션	1019
MFA 디바이스 변경	1019
보안	1020

데이터 개인 정보 보호	1021
데이터 보호	1022
CodeCatalyst 및 ID 및 액세스 관리	1024
규정 준수 확인	1075
복원성	1075
인프라 보안	1075
구성 및 취약성 분석	1076
Amazon CodeCatalyst의 데이터 및 개인 정보 보호	1076
워크플로 작업에 대한 모범 사례	1076
CodeCatalyst 신뢰 모델 이해	1077
로그를 사용하여 이벤트 및 API 직접 호출 모니터링	1079
AWS CloudTrail 로그를 사용하여 AWS 계정의 API 직접 호출 모니터링	1082
이벤트 로그를 사용하여 로그인 이벤트에 액세스	1089
ID, 권한 및 액세스 할당량	1092
문제 해결	1093
가입 문제	1093
로그인 문제	1094
로그아웃 문제	1095
실패한 워크플로에서 역할이 존재하지 않는다는 오류가 발생했습니다.	1095
실패한 워크플로에 대한 역할 오류가 발생합니다.	1096
프로젝트 워크플로에서 IAM 역할을 업데이트해야 합니다.	1096
개인 연결을 생성한 후 GitHub 계정에 대한 검토 요청이 있습니다.	1096
지원 문의 양식은 어떻게 작성하나요?	1097
확장 프로그램이 있는 프로젝트에 기능 추가	1098
사용 가능한 타사 확장	1098
CodeCatalyst에 GitHub 리포지토리 통합	1099
CodeCatalyst에 Bitbucket 리포지토리 통합	1099
CodeCatalyst에 GitLab 리포지토리 통합	1100
CodeCatalyst에 Jira 문제 통합	1101
확장 프로그램 개념	1102
확장 프로그램	1102
CodeCatalyst 카탈로그	1102
연결 및 링크	1102
빠른 시작: 확장 프로그램 설치, 공급자 연결 및 리소스 연결	1102
1단계: CodeCatalyst 카탈로그에서 타사 확장 프로그램 설치	1105
2단계: 타사 공급자를 CodeCatalyst 스페이스에 연결	1106

3단계: 타사 리소스를 CodeCatalyst 프로젝트에 연결	1109
다음 단계	1112
스페이스에 확장 프로그램 설치	1113
스페이스에 확장 프로그램 제거	1114
GitHub 계정, Bitbucket 작업 영역, GitLab 사용자 및 Jira 사이트 연결	1115
GitHub 계정, Bitbucket 작업 영역, GitLab 사용자 및 Jira 사이트 연결 해제	1119
GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결	1120
연결된 타사 공급자의 리소스 연결	1123
프로젝트 생성 중에 타사 리포지토리 연결	1129
GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제	1129
CodeCatalyst에서 타사 리포지토리 보기 및 Jira 문제 검색	1131
CodeCatalyst에서 타사 리포지토리 보기	1131
CodeCatalyst에서 Jira 문제 검색	1132
타사 리포지토리 이벤트 후 워크플로 실행 자동 시작	1133
워크플로 실행을 시작하기 위한 트리거 추가	1133
타사 리포지토리 공급자를 통한 IP 액세스 제한	1134
타사 리포지토리 확장 프로그램에 사용되는 IP 주소	1135
워크플로 실패 시 타사 병합 차단	1135
Jira 문제를 풀 요청에 연결	1136
Jira 문제에서 CodeCatalyst 이벤트 보기	1137
코드, 문제, 프로젝트 및 사용자 검색	1139
검색 쿼리 구체화	1140
유형별 구체화	1140
필드별 구체화	1140
부울 연산자를 사용한 구체화	1141
프로젝트별 구체화	1141
검색 작업 시 고려 사항	1141
검색 가능한 필드 참조	1142
문제 해결	1148
일반적인 액세스 문제 해결	1148
암호를 잊어버렸습니다	1148
Amazon CodeCatalyst의 일부 또는 전부를 사용할 수 없습니다	1149
CodeCatalyst에서 프로젝트를 생성할 수 없습니다	1149
지원 문제 관련 문제 해결	1149
Amazon CodeCatalyst용 지원에 액세스할 때 오류가 발생합니다	1149

내 스페이스에 대한 기술 지원 사례를 생성할 수 없습니다	1150
지원 사례에 대한 내 계정이 CodeCatalyst의 내 스페이스에 더 이상 연결되지 않습니다	1150
Amazon CodeCatalyst용 지원에서 다른 AWS 서비스에 대한 지원 사례를 열 수 없습니다. .	1150
Amazon CodeCatalyst의 일부 또는 전부를 사용할 수 없습니다	1149
CodeCatalyst에서 프로젝트를 생성할 수 없습니다	1149
사용자 이름이 잘려서 새 사용자로 BID 스페이스에 액세스할 수 없거나 새 SSO 사용자로 추가 할 수 없습니다	1151
SSO 사용자를 새 사용자로 페더레이션 스페이스에 추가하면 중복된 사용자가 생성됩니다	1152
CodeCatalyst에서 피드백을 제출하고 싶습니다.	1153
소스 리포지토리 문제 해결	1153
내 스페이스의 스토리지가 최대 용량에 도달하여 경고 또는 오류가 표시됩니다	1153
Amazon CodeCatalyst 소스 리포지토리를 복제하거나 푸시하려고 할 때 오류가 발생합니 다.	1154
Amazon CodeCatalyst 소스 리포지토리를 커밋하거나 푸시하려고 할 때 오류가 발생합니 다.	1155
프로젝트를 위한 소스 리포지토리가 필요합니다.	1155
소스 리포지토리가 완전히 새 것이지만 커밋이 포함되어 있습니다.	1155
다른 브랜치를 기본 브랜치로 사용하겠습니다.	1156
풀 요청의 활동에 대한 이메일을 받고 있습니다.	1156
개인 액세스 토큰(PAT)을 잊어버렸습니다.	1156
풀 요청에는 예상 변경 사항이 표시되지 않습니다.	1156
풀 요청은 병합할 수 없음 상태를 표시합니다.	1157
프로젝트 및 블루프린트 문제 해결	1157
apache-maven-3.8.6에 대한 종속성이AWS Fargate 블루프린트에서 누락된 Java API	1157
Amazon CodeGuru에 대한 권한 오류로 최신 3계층 웹 애플리케이션 블루프린트 OnPullRequest 워크플로 실패	1159
여전히 문제가 해결되지 않았나요?	1161
개발 환경 문제 해결	1161
할당량 문제로 인해 내 개발 환경 생성에 실패했습니다.	1162
개발 환경에서 리포지토리의 특정 브랜치로 변경 사항을 푸시할 수 없습니다.	1162
내 개발 환경이 재개되지 않습니다.	1163
내 개발 환경 연결이 해제됩니다.	1163
VPC 연결 개발 환경이 실패합니다.	1163
프로젝트가 있는 디렉터리를 찾을 수 없습니다.	1163
SSH를 통해 개발 환경에 연결할 수 없습니다.	1164
로컬 SSH 구성이 누락되어 SSH를 통해 내 개발 환경에 연결할 수 없습니다.	1164

codecatalyst 프로파일에 대한 AWS Config에 문제가 있어 SSH를 통해 개발 환경에 연결할 수 없습니다.	1164
Single Sign-On 계정을 사용하여 CodeCatalyst에 로그인할 때는 개발 환경을 생성할 수 없습니다.	1164
IDE 문제 해결	1165
devfile 문제 해결	1166
워크플로 문제 해결	1168
'Workflow is inactive' 메시지를 해결하려면 어떻게 해야 하나요?	1169
'Workflow definition has <i>n</i> errors' 오류를 해결하려면 어떻게 해야 하나요?	1170
'Unable to locate credentials' 및 'ExpiredToken' 오류를 해결하려면 어떻게 해야 하나요? ...	1172
'Unable to connect to the server' 오류를 해결하려면 어떻게 해야 하나요?	1173
시각적 편집기에서 CodeDeploy 필드가 누락된 이유는 무엇인가요?	1174
IAM 기능 오류를 해결하려면 어떻게 해야 하나요?	1174
'npm install' 오류를 해결하려면 어떻게 해야 하나요?	1176
여러 워크플로의 이름이 같은 이유는 무엇인가요?	1179
워크플로 정의 파일을 다른 폴더에 저장할 수 있나요?	1180
워크플로에 작업을 순차적으로 추가하려면 어떻게 해야 하나요?	1180
워크플로가 성공적으로 검증되었지만 런타임에 실패하는 이유는 무엇인가요?	1180
자동 검색이 내 작업에 대한 보고서를 찾지 못함	1181
성공 기준을 구성한 후 내 작업이 자동 검색된 보고서에 대한 작업이 실패해요.	1181
자동 검색이 내가 원하지 않는 보고서를 생성해요.	1182
자동 검색이 단일 테스트 프레임워크에 대해 작은 보고서를 많이 생성함	1182
CI/CD에 나열된 워크플로가 소스 리포지토리의 워크플로와 일치하지 않음	1182
워크플로를 생성하거나 업데이트할 수 없음	1183
문제 해결	1184
내 문제에 대한 담당자를 선택할 수 없습니다.	1184
검색 문제 해결	1184
프로젝트에서 사용자를 찾을 수 없음	1184
프로젝트 또는 공간에서 원하는 것이 보이지 않음	1185
페이지를 탐색할 때 검색 결과 수가 계속 변경됨	1185
검색 쿼리가 완료되지 않음	1185
확장 프로그램 문제 해결	1185
연결된 타사 리포지토리의 변경 내용을 보거나 해당 변경 결과를 검색할 수 없습니다.	1186
연결된 계정 문제 해결	1186
내 AWS 계정 연결 요청에 잘못된 토큰 오류가 발생했습니다	1186

내 Amazon CodeCatalyst 프로젝트 워크플로가 구성된 계정, 환경 또는 IAM 역할에 대한 오류로 실패합니다	1187
프로젝트를 생성하려면 연결된 계정, 역할 및 환경이 필요해요	1188
AWS Management Console에서 Amazon CodeCatalyst 스페이스 페이지에 액세스할 수 없어요	1189
다른 계정을 청구 계정으로 사용하고 싶어요	1189
연결 이름 오류와 함께 프로젝트 워크플로가 실패합니다.	1189
AWS CLI 및 SDK 문제 해결	1190
명령줄 또는 터미널에서 aws codecatalyst라고 입력하면 잘못된 선택이라는 메시지가 표시됩니다.	1190
aws codecatalyst 명령을 실행할 때 자격 증명 오류가 발생합니다.	1190
CodeCatalyst 상태 보고서를 사용하여 현재 서비스 상태 이해	1191
CodeCatalyst 상태 보고서 개념	1191
인시던트	1191
상태 표시기	1192
영향을 받는 기능	1192
업데이트 시기	1192
Amazon CodeCatalyst의 지원	1193
Amazon CodeCatalyst의 지원에 대한 청구	1193
Amazon CodeCatalyst용 지원의 스페이스 설정	1195
AWS Management Console에서 CodeCatalyst에 대한 지원 액세스	1196
CodeCatalyst에서 CodeCatalyst 지원 사례 생성	1197
CodeCatalyst에서 지원 사례 해결	1200
CodeCatalyst에서 지원 사례 다시 열기	1201
할당량	1202
문서 이력	1204
AWS 용어집	1230

Amazon CodeCatalyst는 더 이상 신규 고객에게 공개되지 않습니다. 기존 고객은 정상적으로 서비스를 계속 이용할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 마이그레이션하는 방법](#) 단원을 참조하십시오.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.

Amazon CodeCatalyst란?

Amazon CodeCatalyst는 소프트웨어 개발 프로세스에 지속적인 통합 및 배포 관행을 채택하는 소프트웨어 개발 팀을 위한 통합 서비스입니다. CodeCatalyst는 필요한 도구를 모두 한 곳에 배치합니다. 지속적 통합/지속적 전송(CI/CD) 도구를 사용하여 작업을 계획하고, 코드를 공동 작업하고, 애플리케이션을 빌드, 테스트 및 배포할 수 있습니다. AWS 계정을 CodeCatalyst 스페이스에 연결하여 프로젝트와 AWS 리소스를 통합할 수도 있습니다. 애플리케이션 수명 주기의 모든 단계와 측면을 하나의 도구에서 관리함으로써 소프트웨어를 빠르고 자신 있게 제공할 수 있습니다.

CodeCatalyst에서 회사, 부서 또는 그룹을 대표하는 스페이스를 만든 다음 개발 팀과 태스크를 지원하는 데 필요한 리소스가 포함된 프로젝트를 만듭니다. CodeCatalyst 리소스는 스페이스 내에 사는 프로젝트 내에서 구조화됩니다. 팀이 빠르게 시작할 수 있도록 CodeCatalyst는 언어 또는 도구 기반 프로젝트 블루프린트를 제공합니다. 프로젝트 청사진에서 프로젝트를 생성할 때 프로젝트는 샘플 코드가 있는 소스 리포지토리, 빌드 스크립트, 배포 작업, 가상 서버 또는 서버리스 리소스 등과 같은 리소스와 함께 제공됩니다.

CodeCatalyst로 할 수 있는 작업은 무엇인가요?

사용자와 개발팀은 작업 계획부터 애플리케이션 배포에 이르기까지 소프트웨어 개발의 각 측면을 수행하는 데 CodeCatalyst를 사용할 수 있습니다. CodeCatalyst를 사용하여 다음을 수행할 수 있습니다.

- 코드 반복 및 공동 작업 - 소스 코드 리포지토리의 브랜치, 병합, 풀 요청 및 주석이 포함된 코드에 대해 팀과 공동 작업합니다. 리포지토리에 대한 연결을 복제하거나 설정하지 않고도 코드를 빠르게 처리할 수 있는 개발 환경을 생성합니다.
- 워크플로를 사용하여 애플리케이션 빌드, 테스트 및 배포 - 애플리케이션의 지속적 통합 및 전송을 처리하기 위한 빌드, 테스트 및 배포 작업을 사용하여 워크플로를 구성합니다. 워크플로를 수동으로 시작하거나 코드 푸시 또는 풀 요청 생성 또는 종료와 같은 이벤트에 따라 자동으로 시작하도록 구성할 수 있습니다.
- 문제 추적을 통해 팀의 작업 우선 순위 지정 - 문제를 사용하여 백로그를 생성하고 보드를 사용하여 진행 중인 작업의 상태를 모니터링합니다. 팀이 작업할 항목의 정상 백로그를 생성하고 유지하는 것은 소프트웨어 개발의 중요한 부분입니다.
- 모니터링 및 알림 설정 - 팀 활동 및 리소스 상태를 모니터링하고 중요한 변경 사항을 최신 상태로 유지하도록 알림을 구성합니다.

CodeCatalyst는 어떻게 시작할 수 있나요?

스페이스가 없거나 스페이스를 설정하고 관리하는 방법을 배우고 싶으시다면 [Amazon CodeCatalyst 관리자 안내서](#)를 시작하시는 것을 추천합니다.

프로젝트 또는 스페이스에서 처음 작업하는 경우 다음을 통해 시작하는 것이 좋습니다.

- 검토 [CodeCatalyst 개념](#)
- [스페이스 생성](#)
- [자습서: 최신 3계층 웹 애플리케이션 블루프린트로 프로젝트 만들기](#)의 단계에 따라 첫 번째 프로젝트 생성

CodeCatalyst에 대해 자세히 알아보기

이 사용 설명서와 다음 리소스에서 CodeCatalyst의 기능에 대해 자세히 알아볼 수 있습니다.

- [Amazon CodeCatalyst에 대한 AWS DevOps 블로그 문서](#)
- [Amazon CodeCatalyst API 참조 가이드](#)
- [Amazon CodeCatalyst 작업 개발 키트 개발자 안내서](#)
- [CodeCatalyst FAQ](#)
- [추천사](#)

CodeCatalyst에서 마이그레이션하는 방법

신중한 고려 후 2025년 11월 7일부터 Amazon CodeCatalyst에 대한 새 고객 액세스를 종료하기로 결정했습니다. 기존 Amazon CodeCatalyst 고객은 기존 스페이스에서 서비스를 계속 사용할 수 있지만 새 스페이스를 생성할 수는 없습니다. AWS는 Amazon CodeCatalyst의 보안 및 가용성에 계속 투자하지만 새로운 기능을 도입할 계획은 없습니다.

고객은 Amazon CodeCatalyst에서 다른 공급자로 데이터를 수동으로 마이그레이션할 수 있습니다. 이 문서에서는 CodeCatalyst 및 AWS Management 콘솔에서 데이터를 마이그레이션, 추출 또는 삭제하는 기본 방법을 설명합니다. CodeCatalyst 콘솔을 통해 다른 AWS 또는 3P 서비스에서 생성된 리소스 및 데이터는 요금 발생을 중지하기 위해 해당 서비스를 통해 삭제해야 합니다(해당하는 경우).

고객은 2025년 4월 17일 정식 출시가 발표된 [GitLab Duo with Amazon Q](#)로 마이그레이션하는 것을 고려할 수 있습니다. 이 새로운 오퍼링은 [GitLab](#)의 DevSecOps 플랫폼과 Amazon Q의 생성형 AI 기능을 결합한 통합 제품입니다. GitLab Duo with Amazon Q는 Amazon Q 에이전트 기능을 GitLab의 DevSecOps 플랫폼에 직접 내장하여 전체 소프트웨어 개발 수명 주기에서 복잡한 다단계 작업을 가속화합니다.

리포지토리 마이그레이션

CodeCatalyst 리포지토리를 GitLab 리포지토리로 마이그레이션

사전 조건 URL을 HTTPS Git 리포지토리 자격 증명과 함께 사용하여 [URL을 통해 리포지토리](#)에서 [소스 코드 가져오기](#)에 대한 GitLab 설명서의 지침을 따릅니다.

CodeCatalyst 리포지토리를 GitHub 리포지토리로 마이그레이션

사전 조건 URL을 HTTPS Git 리포지토리 자격 증명과 함께 사용하여 [소스 코드 가져오기](#)에 대한 GitHub 설명서의 지침을 따릅니다.

다른 리포지토리 공급자로 일반 마이그레이션

1. CodeCatalyst 리포지토리 복제

Git을 사용하여 Amazon CodeCatalyst 리포지토리를 로컬 시스템에 복제합니다. HTTPS를 사용하는 경우 다음 명령을 실행하여 이 작업을 수행할 수 있습니다.

```
git clone --mirror https://your-aws-repository-url your-aws-repository
```

`your-aws-repository-url`을 Amazon CodeCatalyst 리포지토리의 URL로 바꿉니다.

`your-aws-repository`를 이 리포지토리의 이름으로 바꿉니다.

예제:

```
git clone https://git-codecatalyst.us-east-2.amazonaws.com/v1/repos/MyDemoRepo my-demo-repo
```

2. 새 원격 리포지토리 포인터 설정

복제된 Amazon CodeCatalyst 리포지토리의 디렉터리로 이동합니다. 그런 다음 새 리포지토리 공급자의 리포지토리 URL을 원격으로 추가합니다.

```
git remote add <provider name> <provider-repository-url>
```

`<provider name>`을 원하는 공급자 이름으로 바꿉니다. (예: gitlab)

`<provider-repository-url>`을 새 리포지토리 공급자의 리포지토리 URL로 바꿉니다.

3. 로컬 리포지토리를 새 원격 리포지토리로 푸시

그러면 모든 브랜치와 태그가 새 리포지토리 공급자의 리포지토리로 푸시됩니다. 공급자 이름은 2 단계의 공급자 이름과 일치해야 합니다.

```
git push <provider name> --mirror
```

참고:

- 원격 리포지토리는 비어 있어야 합니다.
- 원격 리포지토리에는 공급자에 따라 강제 푸시를 허용하지 않는 보호된 브랜치가 있을 수 있습니다. 이 경우 새 리포지토리 공급자로 이동해 브랜치 보호를 비활성화하여 강제 푸시를 허용해야 합니다.

4. 마이그레이션 확인

푸시가 완료되면 모든 파일, 브랜치 및 태그가 새 리포지토리 공급자로 성공적으로 마이그레이션되었는지 확인합니다. 리포지토리를 온라인으로 검색하거나 다른 위치로 복제하고 로컬에서 확인하여 이 작업을 수행할 수 있습니다.

5. 원격 URL 업데이트(선택 사항)

마이그레이션된 리포지토리를 로컬에서 계속 사용하려면 Amazon CodeCatalyst 대신 새 공급자의 리포지토리를 가리키도록 원격 URL을 업데이트해야 할 수 있습니다. 다음 명령을 사용하여 확인할 수 있습니다.

```
git remote set-url origin <provider-repository-url>
```

<provider-repository-url>을 새 리포지토리 공급자의 리포지토리 URL로 바꿉니다.

CodeCatalyst에서 데이터 추출

아티팩트 다운로드

Amazon CodeCatalyst 워크플로 작업에서 생성된 아티팩트를 다운로드하여 검사할 수 있습니다. 다운로드할 수 있는 아티팩트에는 다음과 같이 두 가지 유형이 있습니다.

- 소스 아티팩트 - 실행이 시작될 때 존재했던 소스 리포지토리 콘텐츠의 스냅샷이 포함된 아티팩트입니다.
- 워크플로 아티팩트 - 워크플로 구성 파일의 Outputs 속성에 정의된 아티팩트입니다.

워크플로에서 아티팩트 출력 다운로드

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 워크플로 이름에서 실행을 선택합니다.
6. 실행 기록의 실행 ID 옆에서 실행을 선택합니다. Run-95a4d를 예로 들 수 있습니다.
7. 실행 이름에서 아티팩트를 선택합니다.
8. 아티팩트 옆에 있는 다운로드를 선택합니다. 아카이브 파일이 다운로드됩니다. 파일 이름은 7개의 무작위 문자로 구성됩니다.
9. 원하는 아카이브 추출 유틸리티를 사용하여 아카이브를 추출합니다.

문제 첨부 파일 다운로드

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 첨부 파일을 관리할 문제를 선택합니다. 문제 찾기에 대한 도움말은 [문제 찾기 및 보기](#)를 참조하세요.
3. 첨부 파일을 다운로드하려면 다운로드하려는 첨부 파일 옆의 줄임표 메뉴를 선택하고 다운로드를 선택합니다.

작업의 소스 코드 다운로드

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 코드를 보려는 작업을 찾습니다.
 - a. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
 - b. 워크플로의 이름을 선택하거나 새로 워크플로를 생성합니다. 워크플로 생성에 대한 자세한 내용은 [워크플로 생성](#)을 참조하세요.
 - c. 편집을 선택합니다.
 - d. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.
 - e. 드롭다운 목록에서 Amazon CodeCatalyst를 선택하여 CodeCatalyst, CodeCatalyst Labs 및 타사 작업을 확인합니다.
 - f. 작업을 검색하고 해당 이름을 선택합니다. +(더하기 부호)를 선택합니다.

작업에 대한 세부 정보가 표시됩니다.

4. 작업 세부 정보 대화 상자 하단에서 다운로드를 선택합니다.

작업의 소스 코드가 있는 Amazon S3 버킷을 보여주는 페이지가 나타납니다. 자세한 내용은 Amazon Simple Storage Service 사용 안내서의 [Amazon S3란 무엇인가요?](#)를 참조하세요.

CodeCatalyst에서 데이터 삭제

CodeCatalyst에서 데이터를 삭제하기 전에 여러분의 팀에 서비스 마이그레이션에 대해 알리고 리소스가 여전히 필요하지 않은지 확인합니다. 데이터와 리소스가 삭제되면 복구할 수 없습니다.

서비스 팀에 데이터를 대신 삭제해 주도록 요청

스페이스 관리자는 CodeCatalyst 콘솔의 지원 센터를 통해 당사에 연락하여 서비스 팀이 스페이스를 대신 삭제해 주도록 요청할 수 있습니다. 스페이스 삭제를 요청하려면 스페이스 관리자가 CodeCatalyst 콘솔에서 인증되어야 합니다. 요청을 제출하면 서비스 팀이 대신 조치를 취하기 전에 요청을 확인하기 위해 연락을 드립니다.

CodeCatalyst 스페이스 삭제

스페이스를 삭제하여 스페이스의 모든 리소스에 대한 액세스를 제거할 수 있습니다. 스페이스를 삭제하려면 스페이스 관리자 역할이 있어야 합니다.

참고:스페이스 삭제는 실행 취소할 수 없으며 스페이스가 삭제된 후에는 데이터를 검색할 수 없습니다.

스페이스를 삭제한 후에는 모든 스페이스 멤버가 스페이스 리소스에 액세스할 수 없게 됩니다. 스페이스 리소스에 대한 청구도 중지되며 타사 소스 리포지토리에서 프롬프트되는 모든 워크플로도 중지됩니다.

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

스페이스 삭제

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다.

Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

3. 설정을 선택한 삭제를 선택합니다.
4. **delete**를 입력하여 삭제를 확인합니다.
5. 삭제를 선택합니다.

Note

두 개 이상의 스페이스에 소속되어 있는 경우 스페이스 개요 페이지로 리디렉션됩니다. 하나의 스페이스에 소속되어 있는 경우 스페이스 생성 페이지로 리디렉션됩니다.

두 개 이상의 스페이스에 소속되어 있는 경우 스페이스 하나를 삭제하면 스페이스 개요 페이지로 리디렉션됩니다. 하나의 스페이스에 소속되어 있는 경우 스페이스 생성 페이지로 리디렉션됩니다.

CodeCatalyst 콘솔에서 다른 AWS 또는 타사 서비스에 리소스를 생성한 경우 해당 서비스로 개별적으로 이동하여 생성된 결제 계정의 리소스를 종료해야 합니다. 스페이스를 삭제하면 CodeCatalyst 데이터 및 리소스만 삭제됩니다.

프로젝트 삭제

프로젝트를 삭제하여 프로젝트의 리소스에 대한 모든 액세스를 제거할 수 있습니다. 프로젝트를 삭제하려면 스페이스 관리자 또는 프로젝트 관리자 역할이 있어야 합니다. 프로젝트를 삭제하면 프로젝트 멤버가 프로젝트 리소스에 액세스할 수 없게 되고 타사 소스 리포지토리에서 프롬프트하는 워크플로가 중지됩니다.

프로젝트 삭제

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 보려는 프로젝트가 있는 스페이스로 이동합니다. 프로젝트에서 프로젝트를 선택합니다.
3. 탐색 창에서 프로젝트 설정을 선택합니다.
4. 프로젝트 삭제를 선택합니다.
5. **delete**를 입력하여 삭제를 확인합니다.
6. 프로젝트 삭제를 선택합니다.

CodeCatalyst 프로젝트에서 다른 AWS 또는 타사 서비스에 리소스를 생성한 경우 해당 서비스로 개별적으로 이동하여 생성된 결제 계정의 리소스를 종료해야 합니다. 스페이스를 삭제하면 CodeCatalyst 데이터 및 리소스만 삭제됩니다.

소스 리포지토리 삭제

Amazon CodeCatalyst 프로젝트의 소스 리포지토리를 삭제할 수 있습니다. 소스 리포지토리를 삭제하면 리포지토리에 저장된 모든 프로젝트 정보도 삭제됩니다. 워크플로가 소스 리포지토리에 종속된 경우 리포지토리가 삭제된 후 해당 워크플로는 프로젝트 워크플로 목록에서 삭제됩니다. 소스 리포지토리를 참조하는 문제는 삭제되거나 변경되지 않지만 리포지토리가 삭제되면 문제에 추가된 소스 리포지토리 링크는 모두 실패합니다.

중요: 소스 리포지토리 삭제는 실행 취소할 수 없습니다. 소스 리포지토리를 삭제한 후에는 더 이상 복제하거나, 리포지토리에서 데이터를 가져오거나, 리포지토리로 데이터를 푸시할 수 없습니다. 소스 리

포지토리를 삭제한다고 해서 해당 리포지토리의 모든 로컬 사본(로컬 리포지토리)이 삭제되는 것은 아닙니다. 로컬 리포지토리를 삭제하려면 로컬 컴퓨터의 디렉터리 및 파일 관리 도구를 사용합니다.

참고: 연결된 리포지토리는 CodeCatalyst 콘솔에서 삭제할 수 없습니다. 연결된 리포지토리를 삭제하려면 리포지토리 목록에서 링크를 선택하여 해당 리포지토리를 호스팅하는 서비스에서 해당 리포지토리를 연 다음 삭제합니다. 자세한 내용은 연결된 리포지토리를 호스팅하는 서비스에 대한 설명서를 참조하세요.

프로젝트에서 연결된 리포지토리를 제거하려면 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#)를 참조하세요.

소스 리포지토리 삭제

1. 삭제하려는 소스 리포지토리가 포함된 프로젝트로 이동합니다.
2. 프로젝트의 요약 페이지에서 목록에서 원하는 리포지토리를 선택한 다음 리포지토리 보기를 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다.
3. 리포지토리의 홈 페이지에서 추가를 선택한 다음 설정 관리와 리포지토리 삭제를 차례로 선택합니다.
4. 브랜치, 풀 요청 및 관련 워크플로 정보를 검토하여 아직 사용 중이거나 완료되지 않은 작업이 있는 리포지토리를 삭제하지 않는지 확인합니다. 계속하려면 delete 입력한 다음 삭제를 선택합니다.

사용자 지정 블루프린트 삭제

Amazon CodeCatalyst 스페이스에서 블루프린트를 삭제하면 블루프린트 프로젝트 또는 블루프린트 버전의 리소스에 대한 모든 액세스 권한이 제거됩니다. 블루프린트를 삭제하면 프로젝트 멤버가 프로젝트 리소스에 액세스할 수 없게 되고 타사 소스 리포지토리에서 프롬프트하는 워크플로가 중지됩니다.

블루프린트를 삭제해도 블루프린트가 적용된 프로젝트에는 영향을 주지 않습니다. 블루프린트의 리소스는 프로젝트에서 제거되지 않습니다.

중요: 스페이스에서 게시된 사용자 지정 블루프린트 또는 사용자 지정 블루프린트의 카탈로그 버전을 삭제하려면 스페이스에 스페이스 관리자 또는 파워 유저 역할이 있는 계정으로 로그인해야 합니다.

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 사용자 지정 블루프린트를 삭제할 스페이스로 이동합니다.

3. 스페이스 대시보드에서 설정 탭을 선택한 다음 블루프린트를 선택합니다.
4. 설정 테이블에서 삭제할 사용자 지정 블루프린트의 라디오 버튼을 선택한 다음 블루프린트 삭제를 선택합니다.
5. `delete`를 입력하여 블루프린트 카탈로그 버전의 삭제를 확인합니다.
6. 삭제를 선택합니다.

문제 첨부 파일 삭제

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 첨부 파일을 관리할 문제를 선택합니다. 문제 찾기에 대한 도움말은 [문제 찾기 및 보기](#)를 참조하세요.
3. 첨부 파일을 제거하려면, 제거하려는 첨부 파일 옆의 줄임표 메뉴를 선택하고 삭제를 선택합니다.

Amazon CodeCatalyst를 통해 액세스하는 개발 환경에서 파일 삭제

개발 환경에서, 컴퓨터의 로컬 환경에서 또는 통합 개발 환경(IDE)에서 파일을 삭제할 수 있습니다. Amazon CodeCatalyst 콘솔에서는 파일을 삭제할 수 없습니다.

스페이스의 개발 환경 삭제

개발 환경 삭제 고려 사항에 대한 자세한 내용은 [개발 환경 삭제](#)를 참조하세요.

이 페이지를 보고 스페이스 수준에서 개발 환경을 관리하려면 스페이스 관리자 역할이 있어야 합니다. 둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

개발 환경을 삭제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다.
3. 설정을 선택한 다음 개발 환경을 선택합니다.
4. 관리하려는 개발 환경 옆의 선택기를 선택합니다. 삭제를 선택합니다.
5. **delete**를 입력하여 개발 환경 삭제를 확인합니다.
6. 삭제를 선택합니다.

계정 연결 삭제

이전에 스페이스에 추가한 계정 연결을 CodeCatalyst 콘솔에서 삭제할 수 있습니다. 계정 연결이 삭제된 후에는 해당 연결을 재연결할 수 없으며 새 연결을 설정해야 합니다.

스페이스에 대한 사용량이 프리 티어를 초과하지 않더라도 CodeCatalyst 스페이스에 대한 청구 계정을 지정해야 합니다. 지정된 청구 계정인 계정의 스페이스를 제거하려면 먼저 해당 스페이스에 다른 계정을 추가해야 합니다. 스페이스에 대한 결제 계정을 삭제하려는 경우 스페이스를 삭제해야 합니다. Amazon CodeCatalyst 관리자 안내서의 [Managing billing](#)을 참조하세요.

스페이스의 계정 연결을 관리하려면 스페이스 관리자 또는 파워 유저 역할이 있어야 합니다.

제거된 계정은 나중에 다시 추가할 수 있지만 계정과 스페이스 간에 새 연결을 생성해야 합니다. IAM 역할을 계정에 다시 연결해야 합니다.

계정 연결을 삭제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다. 설정을 선택한 다음 AWS 계정을 선택합니다.
3. Amazon CodeCatalyst 표시 이름에서 제거할 계정 연결 옆의 선택기를 선택합니다.
4. AWS 계정제거를 선택합니다. 필드에 이름을 입력하여 삭제를 확인한 다음 제거를 선택합니다.

성공 배너가 표시되고 계정 연결이 연결 목록에서 제거됩니다.

AWS 관리 콘솔의 CodeCatalyst 스페이스에서 계정 제거

의 CodeCatalyst 페이지를 사용하여 스페이스 AWS 에 추가된 계정을 제거할 수 있습니다. 이 절차에서는 관리 중인 특정 계정에 대한 관리 권한을 사용하여 AWS 관리 콘솔의 Amazon CodeCatalyst Spaces 페이지에 로그인하여 스페이스에서 AWS 계정을 제거합니다. CodeCatalyst 스페이스에 지정된 청구 계정인 계정을 제거하려면 먼저 새 청구 계정을 지정해야 합니다.

제거된 계정은 나중에 다시 추가할 수 있지만 계정과 스페이스 간에 새 연결을 생성해야 합니다. IAM 역할을 추가된 계정에 다시 연결해야 합니다.

스페이스에 대한 사용량이 프리 티어를 초과하지 않더라도 CodeCatalyst 스페이스에 대한 청구 계정을 지정해야 합니다. 지정된 청구 계정인 계정의 스페이스를 제거하려면 먼저 해당 스페이스에 다른 계정을 추가해야 합니다.

스페이스의 계정 연결을 관리하려면 스페이스 관리자 또는 파워 유저 역할이 있어야 합니다.

추가된 계정 제거

1. 에서 관리하려는 것과 동일한 계정으로 로그인했는지 AWS Management Console 확인합니다.
2. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
3. CodeCatalyst 스페이스로 이동합니다. 설정을 선택한 다음 결제를 선택합니다.
4. 페이지에서 결제 계정 정보를 보고 제거하려는 계정이 스페이스에 지정된 결제 계정이 아닌지 확인합니다.
5. 결제 관리를 AWS 선택합니다. AWS Management Console에서 Amazon CodeCatalyst 스페이스가 열립니다. 로그인하라는 메시지가 표시되면 AWS에 로그인한 다음 버튼을 다시 선택하여 페이지를 로드합니다.
6. Amazon CodeCatalyst 스페이스 페이지에서 제거하려는 계정이 있는 스페이스를 선택합니다. 해당 스페이스의 세부 정보 페이지가 표시됩니다.
7. 스페이스 제거를 선택합니다.
8. 이 계정에서 CodeCatalyst 스페이스 제거에 확인할 스페이스 이름을 입력합니다. 제거를 선택합니다.

보안 암호 삭제

다음 절차에 따라 보안 암호와 보안 암호 참조 식별자를 삭제합니다. 보안 암호를 삭제하기 전에 모든 워크플로 작업에서 해당 보안 암호의 참조 식별자를 제거하는 것이 좋습니다. 참조 식별자를 삭제하지 않고 보안 암호를 삭제하면 다음에 실행할 때 작업이 실패합니다.

워크플로에서 보안 암호의 참조 식별자 삭제

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
3. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
4. 편집을 선택합니다.
5. YAML을 선택합니다.
6. 워크플로에서 다음 문자열을 검색합니다.

```
`${Secrets.
```

이렇게 하면 모든 보안 암호의 모든 참조 식별자가 검색됩니다.

7. 선택한 보안 암호의 참조 식별자를 삭제하거나 일반 텍스트 값으로 바꿉니다.
8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

보안 암호를 삭제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 CI/CD를 선택한 후, 보안 암호를 선택합니다.
3. 보안 암호 목록에서 삭제할 보안 암호를 선택합니다.
4. 삭제를 선택합니다.
5. **delete**를 입력하여 삭제를 확인합니다.
6. 삭제를 선택합니다.

팀 삭제

더 이상 필요 없는 팀은 삭제할 수 있습니다. 팀을 삭제하면 스페이스의 모든 프로젝트 및 리소스에서 모든 팀원에 대해 관련 권한이 제거됩니다. 팀을 관리하려면 스페이스 관리자 역할이 있어야 합니다.

팀 삭제

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다. 설정을 선택한 팀을 선택합니다.
3. 작업에서 팀 삭제를 선택합니다. 이렇게 하면 전체 팀의 역할이 변경됩니다.
4. 삭제를 선택합니다.

프로비저닝된 플릿 삭제

다음 지침에 따라 프로비저닝된 플릿을 삭제합니다.

프로비저닝된 플릿을 삭제하기 전에 작업의 YAML 코드에서 Fleet 속성을 삭제하여 모든 작업에서 제거합니다. 프로비저닝된 플릿을 삭제한 후에도 계속 참조하는 작업은 다음 번에 작업이 실행될 때 실패합니다.

프로비저닝된 플릿을 삭제하려면

1. 탐색 창에서 CI/CD를 선택한 후, 컴퓨팅을 선택합니다.

2. 프로비저닝된 플릿 목록에서 삭제할 플릿을 선택합니다.
3. 삭제를 선택합니다.
4. **delete**를 입력하여 삭제를 확인합니다.
5. 삭제를 선택합니다.

패키지 리포지토리 삭제

CodeCatalyst에서 패키지 리포지토리를 삭제하려면 다음 단계를 수행합니다.

패키지 리포지토리를 삭제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 삭제하려는 패키지 리포지토리가 포함된 프로젝트로 이동합니다.
3. 탐색 창에서 패키지를 선택합니다.
4. 패키지 리포지토리 페이지에서 삭제할 리포지토리를 선택합니다.
5. 삭제를 선택합니다.
6. 패키지 리포지토리 삭제의 영향에 대해 제공된 정보를 검토합니다.
7. 입력 필드에 delete를 입력한 다음 삭제를 선택합니다.

스페이스가 삭제되면 Amazon CodeCatalyst에 저장된 다른 모든 리소스가 삭제됩니다. 여기에는 CodeCatalyst 콘솔을 통해 다른 AWS 또는 3P 서비스에서 생성된 리소스 및 데이터는 포함되지 않습니다. 요금이 발생하지 않도록 하려면 CodeCatalyst 콘솔 외부의 서비스에서 생성된 모든 리소스를 해당 서비스를 통해 삭제해야 합니다.

추가 질문이 있는 경우 aws-codecatalyst-service@amazon.com으로 문의하거나 Amazon CodeCatalyst 콘솔의 지원 센터를 통해 문의하세요.

CodeCatalyst 개념

Amazon CodeCatalyst에서 협업 및 애플리케이션 개발을 가속화하는 데 도움이 되는 주요 개념을 숙지합니다. 이러한 개념에는 소스 제어, 지속적 통합 및 지속적 전송(CI/CD), 자동 릴리스 프로세스 모델링 및 구성에 사용되는 용어가 포함됩니다.

추가 개념 정보는 다음 주제를 참조하세요.

- [프라이빗 리포지토리 개념](#)
- [워크플로 개념](#)

주제

- [AWS CodeCatalyst의 Builder ID 스페이스](#)
- [CodeCatalyst에서 ID 페더레이션을 지원하는 스페이스](#)
- [Projects](#)
- [블루프린트](#)
- [계정 연결](#)
- [VPC 연결](#)
- [AWS 빌더 ID](#)
- [CodeCatalyst의 사용자 프로필](#)
- [소스 리포지토리](#)
- [커밋](#)
- [개발 환경](#)
- [워크플로](#)
- [작업](#)
- [문제](#)
- [개인 액세스 토큰\(PAT\)](#)
- [개인 연결](#)
- [역할](#)

AWS CodeCatalyst의 Builder ID 스페이스

스페이스 관리자는 멤버 페이지에서 개별 초대 이메일을 보내 사용자를 CodeCatalyst에 초대합니다. CodeCatalyst에 초대되거나 가입한 사용자는 자체 AWS Builder ID를 생성합니다. 프로필은 AWS Builder ID로 관리되며 CodeCatalyst의 사용자 설정에 사용자 이름 및 프로필 정보로 표시됩니다.

CodeCatalyst에서 ID 페더레이션을 지원하는 스페이스

IAM Identity Center 인스턴스의 SSO 사용자 및 그룹에 추가되고 Identity Store에서 관리되는 사용자는 IAM Identity Center를 통해 사용자의 스페이스에 초대됩니다. 스페이스 관리자는 최신 업데이트를 위해 CodeCatalyst 멤버 페이지를 동기화합니다. 사용자는 회사 IAM Identity Center 인스턴스에 설정된 대로 SSO 로그인 포털을 사용하여 로그인합니다. ID 페더레이션을 지원하는 스페이스는 Identity Center 애플리케이션과 Identity Store ID에 대한 매핑을 통해 Identity Store 인스턴스에 연결됩니다.

Projects

프로젝트는 개발 팀과 작업을 지원하는 CodeCatalyst의 협업 스페이스입니다. 프로젝트를 만든 후에는 사용자와 리소스를 추가, 업데이트 또는 제거하고, 프로젝트 대시보드를 사용자 지정하고, 팀의 작업 진행 상황을 모니터링할 수 있습니다. 스페이스 내에 여러 프로젝트를 포함할 수 있습니다.

프로젝트에 대한 자세한 내용은 [CodeCatalyst에서 프로젝트 작업 구성](#) 섹션을 참조하세요.

블루프린트

블루프린트는 콘솔에서 CodeCatalyst 프로젝트를 생성하는 동시에 애플리케이션 지원 파일 및 종속성을 생성하고 확장하는 프로젝트 합성기입니다. CodeCatalyst의 블루프린트 선택에서 프로젝트 유형을 선택하고, README 파일을 보고, 생성할 프로젝트 리포지토리와 리소스를 미리 봅니다. 프로젝트는 블루프린트에 지정된 기본 구성에서 생성됩니다. 프로젝트 블루프린트로 주기적으로 합성하여 소프트웨어 종속성과 같은 프로젝트 파일을 업데이트하고 리소스를 재생성합니다. 프로젝트는 Projen이라는 도구를 사용하여 최신 프로젝트 업데이트를 동기화하고 지원 파일을 생성하여 프로젝트를 합성합니다. 이러한 파일에는 애플리케이션 유형 및 언어에 따라 package.json, Makefile, eslint 등이 포함될 수 있습니다. 프로젝트 블루프린트는 CDK 구문, CloudFormation 템플릿 및 AWS Serverless Application Model 템플릿과 같은 AWS 리소스를 지원하는 파일을 생성할 수 있습니다.

프로젝트 블루프린트에 대한 자세한 내용은 [CodeCatalyst 블루프린트를 사용하여 포괄적인 프로젝트 생성](#) 섹션을 참조하세요.

계정 연결

계정 연결은 CodeCatalyst 스페이스를 AWS 계정에 연결합니다. 계정 연결이 설정되면 스페이스에서 AWS 계정을 사용할 수 있습니다. 그런 다음 CodeCatalyst에 IAM 역할을 추가하여 리소스에 액세스할 수 있습니다 AWS 계정. CodeCatalyst 워크플로 작업에 이러한 역할을 사용할 수도 있습니다.

프로젝트 제한 계정 연결을 활성화하여 계정 연결에 액세스할 수 있는 프로젝트와 리소스를 제한할 수 있습니다. 스페이스의 지정된 프로젝트만 액세스할 수 AWS 계정 있는 프로젝트 제한 계정 연결이 연결됩니다. 이를 통해 스페이스의 팀은 프로젝트별 통합 AWS 리소스에 대한 사용을 AWS 계정 제한할 수 있습니다. 예를 들어 특정 프로젝트의 배포 워크플로 및 VPC 연결에 사용되는 계정은 프로젝트 제한 계정 연결에서만 사용할 수 있습니다. 자세한 내용은 [Configuring project-restricted account connections](#)를 참조하세요.

계정 연결에 대한 자세한 정보는 [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 섹션을 참조하세요.

VPC 연결

VPC 연결은 워크플로가 VPC에 액세스하는 데 필요한 모든 구성을 포함하는 CodeCatalyst 리소스입니다. 스페이스 관리자는 스페이스 멤버를 대신하여 Amazon CodeCatalyst 콘솔에 자체 VPC 연결을 추가할 수 있습니다. VPC 연결을 추가하면 스페이스 멤버가 워크플로 작업을 실행하고 네트워크 규칙을 준수하고 연결된 VPC의 리소스에 액세스할 수 있는 개발 환경을 생성할 수 있습니다.

VPC 연결에 대한 자세한 내용은 CodeCatalyst 관리자 안내서의 [Amazon Virtual Private Clouds 관련](#)을 참조하세요.

AWS 빌더 ID

AWS Builder ID는 CodeCatalyst 및 기타 참여 애플리케이션에 가입하고 로그인하는 데 사용할 수 있는 개인 자격 증명입니다. 와 동일하지 않습니다 AWS 계정. AWS Builder ID는 사용자 별칭 및 이메일 주소와 같은 메타데이터를 관리합니다. AWS Builder ID는 CodeCatalyst의 모든 스페이스에서 사용자를 지원하는 고유한 자격 증명입니다. AWS Builder ID 프로파일에 액세스하는 방법에 대한 자세한 내용은 섹션을 참조하세요 [프로파일 업데이트](#). AWS Builder ID에 대한 자세한 내용은 [AWS Builder ID](#)를 참조하세요 AWS 일반 참조.

가입에 대한 자세한 내용은 [CodeCatalyst 설정 및 로그인](#) 섹션을 참조하세요.

CodeCatalyst의 사용자 프로파일

CodeCatalyst의 모든 페이지에서 로그인 이니셜 아래의 드롭다운에서 프로파일 옵션을 선택하여 CodeCatalyst 사용자 프로파일에 액세스합니다. 프로파일 페이지에서 개인 액세스 토큰(PATs)을 생성할 수 있지만 AWS CLI를 사용하여 PATs만 보거나 삭제할 수 있습니다. 사용자 이름은 가입 시 선택한 별칭입니다. 사용자 이름은 변경할 수 없습니다. 다른 CodeCatalyst 사용자의 프로파일 페이지를 보려면 프로젝트의 멤버 탭으로 이동하여 적절한 사용자를 선택합니다.

CodeCatalyst 프로필을 확인한 다음 AWS Builder ID로 이동하도록 선택하여 AWS Builder ID에 액세스합니다. AWS Builder ID 프로필 페이지로 리디렉션됩니다. 프로필의 전체 이름, 이메일 주소 및 암호는 AWS Builder ID로 관리되며 AWS Builder ID 페이지를 사용하여 해당 정보를 편집할 수 있습니다. 가입할 때 이 정보를 입력했습니다. 로그인에 인증자 애플리케이션을 사용하도록 MFA를 설정할 준비가 되면 AWS Builder ID 페이지를 사용합니다. AWS Builder ID 프로파일 보기에 대한 자세한 내용은 [섹션을 참조하세요](#) [프로파일 업데이트](#).

가입에 대한 자세한 내용은 [CodeCatalyst 설정 및 로그인](#) 섹션을 참조하세요.

소스 리포지토리

소스 리포지토리는 프로젝트에 대한 코드 및 파일을 안전하게 저장합니다. 또한 파일의 버전 기록도 저장합니다. 기본적으로 소스 리포지토리는 CodeCatalyst 프로젝트의 다른 사용자와 공유됩니다. 프로젝트에 대한 소스 리포지토리가 두 개 이상 있을 수 있습니다. CodeCatalyst의 프로젝트에 대한 소스 리포지토리를 생성하거나, 설치된 확장 프로그램에서 해당 서비스가 지원되는 경우 다른 서비스에서 호스팅하는 기존 소스 리포지토리를 연결하도록 선택할 수 있습니다. 예를 들어 GitHub 리포지토리 확장 프로그램을 설치한 후 GitHub 리포지토리를 프로젝트에 연결할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 프로젝트의 리포지토리에 소스 코드 저장](#) 및 [빠른 시작: CodeCatalyst에서 확장 프로그램 설치, 공급자 연결 및 리소스 연결](#) 섹션을 참조하세요.

소스 리포지토리는 CI/CD 워크플로의 속성 및 작업을 정의하는 구성 파일과 같이 CodeCatalyst 프로젝트에 대한 구성 정보가 저장되는 위치이기도 합니다. 블루프린트를 사용하여 프로젝트를 생성하는 경우 프로젝트 구성 정보가 저장된 소스 리포지토리가 생성됩니다. 빈 프로젝트를 생성하는 경우 워크플로와 같은 구성 정보가 필요한 리소스를 생성하기 전에 소스 리포지토리를 생성해야 합니다.

소스 리포지토리 및 소스 제어 작업에 도움이 되는 자세한 개념은 [섹션을 참조하세요](#) [프라이빗 리포지토리 개념](#).

커밋

커밋은 파일 또는 파일 집합의 변경입니다. Amazon CodeCatalyst 콘솔에서 커밋은 변경 사항을 저장하고 소스 리포지토리로 푸시합니다. 커밋에는 변경한 사용자의 ID, 변경 시간 및 날짜, 커밋 제목, 변경에 포함된 모든 메시지를 포함한 변경에 대한 정보가 포함됩니다. 자세한 내용은 [Amazon CodeCatalyst에서 커밋을 사용한 소스 코드 변경 사항 이해](#) 섹션을 참조하세요.

CodeCatalyst의 소스 리포지토리 맥락에서 커밋은 리포지토리의 콘텐츠 변경 사항에 대한 스냅샷입니다. 사용자가 변경 사항을 커밋하고 푸시할 때마다 CodeCatalyst는 변경 사항을 커밋한 사람, 커밋 날짜 및 시간, 커밋의 일부로 수행된 변경 사항을 포함하는 정보를 저장합니다. 또한 커밋에 Git 태그를 추가하여 특정 커밋을 쉽게 식별할 수 있습니다.

커밋에 대한 자세한 내용은 [Amazon CodeCatalyst에서 커밋을 사용한 소스 코드 변경 사항 이해](#) 섹션을 참조하세요.

개발 환경

개발 환경은 프로젝트의 소스 리포지토리에 저장된 코드를 빠르게 작업하기 위해 CodeCatalyst에서 사용할 수 있는 클라우드 기반 개발 환경입니다. 개발 환경에 포함된 프로젝트 도구 및 애플리케이션 라이브러리는 프로젝트의 소스 리포지토리에 있는 devfile로 정의됩니다. 소스 리포지토리에 devfile이 없는 경우 기본 devfile이 자동으로 적용됩니다. 기본 devfile에는 가장 자주 사용되는 프로그래밍 언어 및 프레임워크를 위한 도구가 포함되어 있습니다. 기본적으로 개발 환경은 2코어 프로세서, 4GB RAM, 16GiB 영구 저장소를 갖도록 구성됩니다.

워크플로

워크플로는 지속적 통합 및 지속적 전송(CI/CD) 시스템의 일부로 코드를 빌드, 테스트 및 배포하는 방법을 설명하는 자동화된 절차입니다. 워크플로는 워크플로 실행 중에 수행할 일련의 단계 또는 작업을 정의합니다. 또한 워크플로는 워크플로를 시작하게 하는 이벤트 또는 트리거를 정의합니다. 워크플로를 설정하려면 CodeCatalyst 콘솔의 [시각적 또는 YAML 편집기](#)를 사용하여 워크플로 정의 파일을 생성합니다.

Tip

프로젝트에서 워크플로를 사용하는 방법을 간략하게 알아보려면 [블루프린트가 있는 프로젝트를 생성](#)합니다. 각 블루프린트는 검토, 실행 및 실험할 수 있는 기능 워크플로를 배포합니다.

워크플로에 대한 자세한 내용은 [워크플로를 사용하여 빌드, 테스트 및 배포](#) 섹션을 참조하세요.

작업

작업은 워크플로의 기본 구성 요소이며 워크플로 실행 중에 수행할 논리적 작업 단위 또는 태스크를 정의합니다. 일반적으로 워크플로에는 구성 방식에 따라 순차적으로 또는 병렬로 실행되는 여러 작업이 포함됩니다.

작업에 대한 자세한 내용은 [워크플로 작업 구성](#) 섹션을 참조하세요.

문제

문제는 프로젝트와 관련된 작업을 추적하는 레코드입니다. 기능, 작업, 버그 또는 프로젝트와 관련된 기타 작업에 대한 문제를 생성할 수 있습니다. 애자일 개발을 사용하는 경우 문제가 에픽 또는 사용자 스토리를 설명할 수도 있습니다.

이러한 문제에 대한 자세한 내용은 [CodeCatalyst의 문제 처리 추적 및 구성](#) 섹션을 참조하세요.

개인 액세스 토큰(PAT)

개인 액세스 토큰(PAT)은 암호와 유사합니다. CodeCatalyst의 모든 스페이스와 프로젝트에서 사용할 수 있도록 사용자 ID와 연결됩니다. PAT를 사용하여 통합 개발 환경(IDE) 및 Git 기반 소스 리포지토리가 포함된 CodeCatalyst 리소스에 액세스합니다. PAT는 CodeCatalyst에서 사용자를 나타내며 사용자 설정에서 이를 관리할 수 있습니다. 사용자는 둘 이상의 PAT를 가질 수 있습니다. 개인 액세스 토큰은 한 번만 표시됩니다. 가장 좋은 방법은 로컬 컴퓨터에 안전하게 저장하는 것입니다. 기본적으로 PAT는 1년 후에 만료됩니다.

PAT에 대한 자세한 내용은 [개인 액세스 토큰을 사용하여 사용자 리포지토리 액세스 권한 부여](#) 섹션을 참조하세요.

개인 연결

개인 연결은 CodeCatalyst ID와 GitHub와 같은 외부 소스 공급자 간의 권한 부여입니다. 개인 연결을 사용하여 CodeCatalyst 사용자가 타사 소스 리포지토리를 추가할 수 있도록 허용합니다. 예를 들어 GitHub 리포지토리를 CodeCatalyst 스페이스에 연결할 수 있습니다. 설치된 커넥터 애플리케이션은 계정 소유자가 지정한 리포지토리와 함께 사용할 수 있도록 GitHub 계정에 설치됩니다. GitHub와 같은 특정 공급자 유형의 모든 스페이스에 걸쳐 하나의 사용자 ID(CodeCatalyst 별칭)에 대해 하나의 개인 연결을 생성할 수 있습니다. 개인 연결은 AWS Builder ID 또는 SSO 사용자와 연결됩니다.

자세한 내용은 [개인 연결을 사용하여 GitHub 리소스에 액세스](#) 단원을 참조하십시오.

역할

역할은 프로젝트 또는 스페이스의 리소스에 대한 사용자의 액세스 권한과 사용자가 수행할 수 있는 작업을 정의합니다. 프로젝트에 사용자를 초대할 때 사용자의 역할을 선택합니다. CodeCatalyst에는 스페이스 수준 역할 및 프로젝트 수준 역할이 있습니다. 관리 역할이 올바른 수준인 사용자는 할당된 역할을 변경할 수 있습니다. 예를 들어 프로젝트의 프로젝트 관리자 역할을 가진 사용자는 해당 프로젝트를 완전히 제어할 수 있으며 해당 프로젝트에서 사용자의 역할을 변경할 수 있습니다. 사용 가능한 역할과 각 역할에 있는 권한에 대한 자세한 내용은 섹션을 참조하세요 [사용자 역할로 액세스 권한 부여](#).

역할에 관한 자세한 내용은 [사용자 역할로 액세스 권한 부여](#) 섹션을 참조하세요.

CodeCatalyst 설정 및 로그인

CodeCatalyst에서는 두 가지 유형의 스페이스를 설정할 수 있습니다. 하나는 AWS Builder ID 사용자를 지원하는 스페이스이며, 다른 하나는 IAM Identity Center에서 SSO 사용자 및 그룹을 관리하는 ID 페더레이션을 지원하는 스페이스입니다. AWS Builder ID 스페이스의 사용자는 자신의 AWS Builder ID로 CodeCatalyst에 로그인하고, ID 페더레이션을 위해 설정된 스페이스의 사용자는 해당 스페이스와 연결된 회사의 SSO 포털을 사용하여 CodeCatalyst에 로그인합니다.

Note

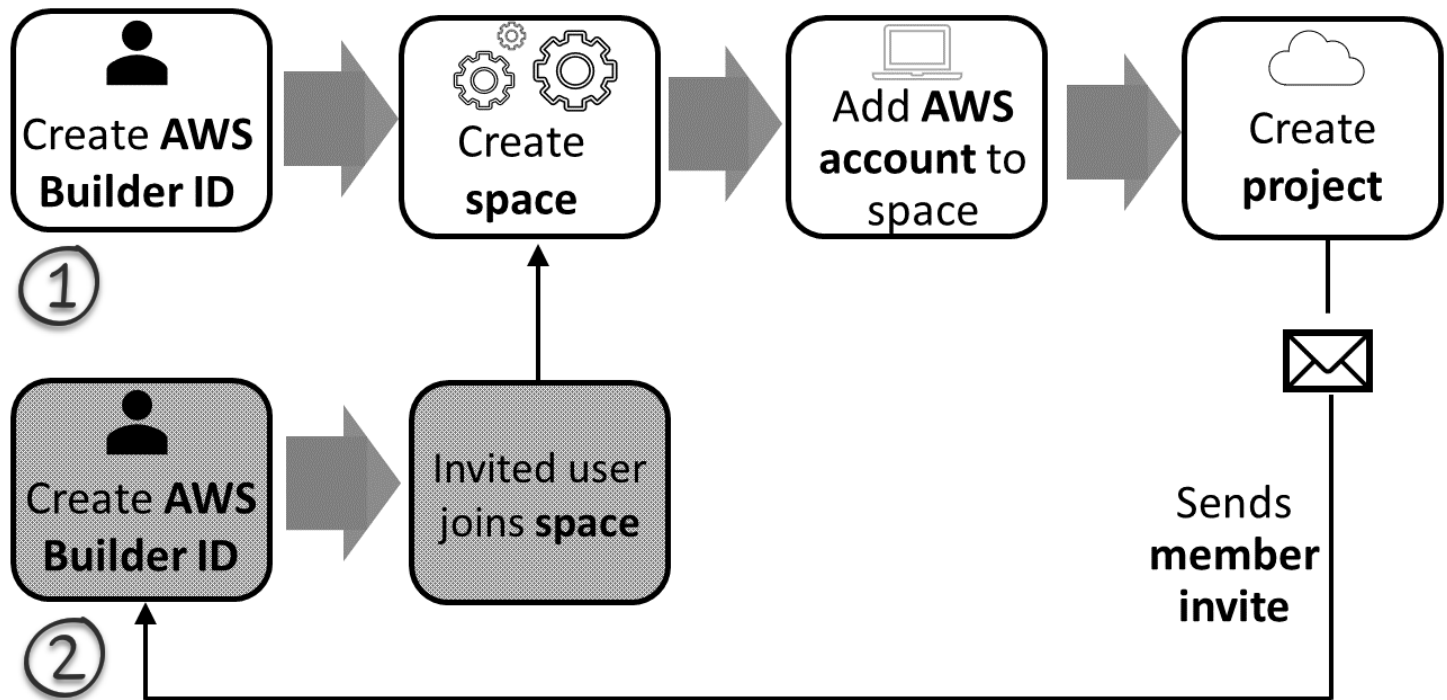
CodeCatalyst 사용자 이름의 최소 길이는 3자, 최대 길이는 100자입니다. 100자보다 긴 제공된 사용자 이름은 잘립니다. 이로 인해 사용자 이름이 다른 100자 길이의 사용자 이름과 중복되는 것처럼 보일 수 있습니다. 자세한 내용은 [사용자 이름이 잘려서 새 사용자로 BID 스페이스에 액세스할 수 없거나 새 SSO 사용자로 추가할 수 없습니다](#) 섹션을 참조하세요.

AWS Builder ID 스페이스를 설정하고 관리하는 단계는 이 안내서에 나와 있습니다. CodeCatalyst AWS Builder ID 스페이스를 사용하려면, CodeCatalyst에 로그인하는 데 사용하는 사용자 설정 및 AWS Builder ID를 사용하여 CodeCatalyst를 설정합니다.

ID 페더레이션을 지원하는 스페이스를 설정하고 관리하는 단계는 CodeCatalyst 관리자 안내서에 나와 있습니다. ID 페더레이션을 위해 설정된 스페이스로 작업하려면, Amazon CodeCatalyst 관리자 안내서의 [CodeCatalyst 스페이스 설정 및 관리](#)를 참조하세요.

이 섹션에서는 AWS Builder ID 스페이스를 사용하여 Amazon CodeCatalyst에서 작동하도록 설정하는 두 가지 일반적인 경로, 즉 스페이스와 프로젝트를 첫 번째 사용자로서 생성하는 경로와 기존 스페이스 또는 프로젝트에 대한 초대를 수락하는 경로를 제공합니다. 이러한 설정 워크플로에는 반드시 큰 차이가 나타날 수 밖에 없습니다. 다음 다이어그램은 다음과 같이 두 가입 프로세스를 보여줍니다.

1. 첫 번째 경우에는 회사, 팀 또는 그룹을 위한 스페이스를 생성 및 설정하고, 이러한 리소스에 다른 사람을 초대하기 전에 프로젝트를 생성합니다. 청구를 위해 AWS 계정을 제공해야 합니다. 이 경우 여전히 프리 티어를 기본값으로 설정할 수 있습니다.
2. 두 번째 경우 프로젝트 초대를 수락하여 CodeCatalyst에 가입하면 다른 사람이 이미 생성한 스페이스와 프로젝트가 있습니다. 하지만, 프로파일을 구성해 다른 사용자와 작업을 시작할 준비해야 합니다.



Tip

CodeCatalyst는 스페이스를 사용하여 프로젝트와 리소스를 그룹화합니다. CodeCatalyst에 처음 가입하면 프로젝트뿐만 아니라 스페이스를 생성하라는 메시지가 표시됩니다.

스페이스 및 프로젝트를 생성하기 위해 가입하든 초대 수락 과정으로 가입하든 상관 없이 CodeCatalyst에 로그인하는 데 사용할 AWS Builder ID를 생성합니다. AWS Builder ID를 생성하려면 AWS 애플리케이션에 로그인하는 데 사용하는 전체 이름, 암호 및 이메일 주소를 입력합니다. 이 시점 이후에는 이메일과 암호를 사용하여 CodeCatalyst에 로그인합니다. 이 AWS Builder ID를 사용하여 AWS Builder ID 자격 증명을 사용하는 다른 애플리케이션에 로그인할 수도 있습니다.

CodeCatalyst 및 AWS Builder ID에서 프로파일은 로그인 정보를 기반으로 생성됩니다. 프로파일에는 CodeCatalyst 프로젝트의 언어 및 알림 설정에 대한 CodeCatalyst 기본 설정이 포함되어 있습니다.

Tip

Amazon CodeCatalyst 프로파일에 가입하는 동안 문제가 발생하면 해당 페이지에 제공된 단계를 따르세요. 추가 도움이 필요한 경우 [가입 문제](#) 섹션을 참조하세요.

주제

- [새 스페이스 및 개발 역할 생성\(초대 없이 시작\)](#)
- [초대 수락 및 AWS Builder ID 생성](#)
- [AWS Builder ID로 로그인](#)
- [SSO로 로그인](#)
- [사용자의 모든 스페이스 및 프로젝트 보기](#)
- [CodeCatalyst 프로파일 보기 및 관리](#)
- [CodeCatalyst AWS CLI 에서를 사용하도록 설정](#)

새 스페이스 및 개발 역할 생성(초대 없이 시작)

기존 스페이스 또는 프로젝트에 초대하지 않고도 Amazon CodeCatalyst에 가입할 수 있습니다. 이렇게 하면 AWS Builder ID를 생성한 후 스페이스와 프로젝트가 생성됩니다. 스페이스 생성의 일환으로 청구를 위해 AWS 계정을 추가해야 합니다.

Tip

Amazon CodeCatalyst 프로파일에 가입하는 동안 문제가 발생하면 해당 페이지에 제공된 단계를 따르세요. 추가 도움이 필요한 경우 [가입 문제](#) 섹션을 참조하세요.

다음은 프로젝트 또는 스페이스에 대한 초대 없이 CodeCatalyst로 시작하는 사용자에게 가능한 흐름 중 하나입니다.

Mary Major는 CodeCatalyst에 관심이 있는 개발자로, CodeCatalyst를 사용해 보기로 결정합니다. 그는 CodeCatalyst 콘솔로 이동하여, 등록하고 AWS Builder ID를 생성하는 옵션을 선택합니다. Mary는 이메일 주소와 암호를 제공하여 AWS Builder ID를 생성합니다. 그는 AWS Builder ID를 사용하여 CodeCatalyst 및 기타 애플리케이션에 로그인할 수 있습니다. 별칭을 선택하라는 메시지가 표시되면 MaryMajor를 CodeCatalyst에 표시될 CodeCatalyst 사용자 이름으로 지정하고 다른 프로젝트 멤버가 @mention Mary에 사용할 수 있도록 지정합니다.

다음으로 Mary는 자동으로 스페이스를 생성하라는 지시를 받습니다. 이 흐름의 일환으로 Mary는 첫 번째 프로젝트 빌드 및 배포에서 샘플 코드를 볼 수 있도록 AWS 계정을 자신이 생성하는 스페이스와 연결하라는 요청을 받습니다. 그는 해당 정보를 추가하고 스페이스를 생성하여 새 스페이스의 프로젝트에 사용할 수 있는 미리 보기 개발 역할을 생성하는 옵션을 선택합니다. Mary는 프로젝트를 생성하기로 선택한 다음 프로젝트의 블루프린트 목록을 봅니다. 사용 가능한 블루프린트에 대한 정보를 검토

한 후, 첫 번째 프로젝트에 대한 최신 3계층 웹 애플리케이션 블루프린트를 사용해 보기로 결정합니다. Mary는 필수 필드를 작성하고 프로젝트를 생성합니다. 프로젝트가 준비되자마자 Mary는 최근 활동뿐만 아니라 프로젝트 코드에 대한 링크와 해당 코드를 자동으로 빌드하고 배포하는 워크플로가 포함된 프로젝트 요약 페이지로 이동합니다. 배포된 샘플 웹 애플리케이션을 보는 것을 포함하여 코드와 워크플로를 모두 살펴봅니다. Mary는 자신이 본 것을 마음에 들어 CodeCatalyst 탐색을 시작하기 위해 프로젝트에 동료를 초대하기로 결정합니다.

잠시 시간이 있을 때, Mary는 다중 인증(MFA)을 사용하여 CodeCatalyst에 로그인하도록 AWS Builder ID를 구성합니다. MFA를 구성하면 Mary는 CodeCatalyst 암호와 승인된 타사 인증 앱의 암호 또는 토큰을 조합하여 CodeCatalyst에 로그인할 수 있습니다.

새 스페이스 및 IAM 역할 생성

이 단계에 따라 Amazon CodeCatalyst 프로파일에 가입하고, 스페이스를 만들고, 스페이스에 대한 계정, 지원 역할, 개발자 역할을 추가합니다.

최종 절차에서는 개발자 역할을 생성하고 추가합니다. 개발자 역할은 CodeCatalyst 워크플로가 AWS 리소스에 액세스할 수 있도록 하는 AWS IAM 역할입니다. 개발자 역할은 AWS 서비스를 관리하는 데 사용되는 서비스 역할이며 로그인된 계정에 생성됩니다. 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 [IAM 역할](#)입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 역할에는 CodeCatalystWorkflowDevelopmentRole-*spaceName* 이름이 있습니다. 역할 및 역할 정책에 대한 자세한 내용은 [CodeCatalystWorkflowDevelopmentRole-*spaceName* 서비스 역할 이해](#) 섹션을 참조하세요.

Note

보안 모범 사례로, 스페이스의 AWS 리소스에 대한 액세스를 관리해야 하는 관리 사용자 및 개발자에게만 관리 액세스 권한을 할당합니다.

시작하기 전에 관리 권한이 있는 계정의 AWS 계정 ID를 제공할 준비가 되어 있어야 합니다. 12자리로 이루어진 AWS 계정 ID를 준비하세요. AWS 계정 ID를 찾는 방법에 대한 자세한 내용은 [AWS 계정 ID 및 별칭](#)을 참조하세요.

새 사용자 등록

1. CodeCatalyst 콘솔에서 시작하기 전에 AWS Management Console을 연 다음 스페이스를 생성하는 데 사용할 것과 동일한 AWS 계정으로 로그인했는지 확인합니다.

2. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
3. 시작 페이지에서 가입을 선택합니다. AWS Builder ID 만들기 페이지가 표시됩니다. AWS Builder ID는 로그인하기 위해 생성하는 ID입니다. AWS 계정과 동일하지 않습니다.
4. 이메일 주소에 CodeCatalyst에 연결하여 사용하려는 이메일 주소를 입력합니다. 그런 다음 다음을 선택합니다.
5. 이름에는 AWS Builder ID를 사용하는 애플리케이션에 표시할 이름과 성을 입력합니다. 공백을 사용할 수 있습니다. Mary Major 와 같은 AWS Builder ID 프로파일 이름입니다. 이 이름은 나중에 변경할 수 있습니다.

다음을 선택합니다. 이메일 확인 페이지가 표시됩니다.

6. 지정한 이메일로 확인 코드가 전송됩니다. 확인 코드에 이 코드를 입력한 다음, 확인을 선택합니다. 5분 후에도 코드를 받지 못하고 스팸 폴더나 정크 폴더에서 코드를 찾을 수 없다면, 코드 재전송을 선택하세요.
7. 코드를 확인한 후 암호 및 암호 확인의 요구 사항을 충족하는 암호를 입력합니다.

AWS 이용계약 및 AWS 서비스 약관에 동의함을 확인하는 확인란을 선택한 다음, AWS Builder ID 생성을 선택합니다.

8. CodeCatalyst 별칭 생성 페이지에서 CodeCatalyst에서 고유한 사용자 식별자에 사용할 별칭을 입력합니다. MaryMajor와 같이 공백이 없는 이름의 단축 버전을 선택합니다. 다른 CodeCatalyst 사용자는 댓글과 폴 요청에서 이 별칭을 사용하여 사용자에게 @mention을 보냅니다. CodeCatalyst 프로파일에는 AWS Builder ID의 전체 이름과 CodeCatalyst 별칭이 모두 포함됩니다. CodeCatalyst 별칭은 나중에 변경할 수 없습니다.

전체 이름과 별칭은 CodeCatalyst의 여러 영역에 표시됩니다. 예를 들어 프로파일 이름은 활동 피드에서 나열된 액티비티에 표시되지만, 프로젝트 멤버는 사용자의 별칭을 사용자를 @mention하는 데 사용합니다.

다음을 선택합니다. 페이지가 업데이트되어 CodeCatalyst 스페이스 생성 섹션이 표시됩니다.

9. 스페이스 이름 지정에서 스페이스 이름을 입력합니다. 이 이름은 나중에 변경할 수 없습니다.

Note

스페이스 이름은 CodeCatalyst 전체에서 고유해야 합니다. 삭제된 스페이스의 이름은 재 사용할 수 없습니다.

10. AWS 리전 드롭다운 메뉴에서 스페이스 및 프로젝트 데이터를 저장할 리전을 선택합니다. 이 이름은 나중에 변경할 수 없습니다.

11. 다음을 선택합니다. 페이지가 업데이트되어 AWS 계정을 추가하기 위한 페이지가 표시됩니다. 이 계정은 스페이스의 청구 계정으로 사용됩니다.
12. AWS 계정 ID의 경우, 스페이스에 연결하려는 계정의 12자리 ID를 입력합니다.

AWS 계정 확인 토큰에서 생성된 토큰 ID를 복사합니다. 토큰은 자동으로 복사되지만 AWS 연결 요청을 승인하는 동안에 토큰을 저장할 수 있습니다.

13. AWS Console로 이동하여 확인을 선택합니다.
14. AWS Management Console에서 Amazon CodeCatalyst 스페이스 확인 페이지가 열립니다. Amazon CodeCatalyst 스페이스 페이지입니다. 페이지에 액세스하려면 로그인해야 할 수 있습니다.

AWS Management Console에서 스페이스를 만들려는 동일한 AWS 리전을 선택해야 합니다.

페이지에 직접 액세스하려면 <https://console.aws.amazon.com/codecatalyst/home/>의 AWS Management Console에서 Amazon CodeCatalyst 스페이스에 로그인합니다.

AWS Management Console의 확인 토큰 필드는 CodeCatalyst에서 생성된 토큰으로 자동으로 채워집니다.

15. (선택 사항) 승인된 유료 티어에서 유료 티어 승인(표준, 엔터프라이즈)을 선택하여 청구 계정의 유료 티어를 활성화할 수 있습니다.

Note

이렇게 해도 청구 계층은 유료 티어로 업그레이드되지 않습니다. 그러나 이는 CodeCatalyst에서 언제든지 스페이스의 청구 계층을 변경할 수 있도록 AWS 계정을 구성합니다. 언제든지 유료 티어를 활성화할 수 있습니다. 이러한 변경이 없다면, 스페이스는 프리 티어만 사용할 수 있습니다.

16. 스페이스 확인을 선택합니다.

계정이 스페이스에 추가되었음을 보여주는 계정 확인 성공 메시지가 표시됩니다.

17. Amazon CodeCatalyst 스페이스 확인 페이지에 머무릅니다. 다음 링크를 선택합니다. 이 스페이스에 대한 IAM 역할을 추가하려면 스페이스 세부 정보를 봅니다.

AWS Management Console에서 CodeCatalyst 스페이스 세부 정보 페이지가 열립니다. Amazon CodeCatalyst 스페이스 페이지입니다. 페이지에 액세스하려면 로그인해야 할 수 있습니다.

18. CodeCatalyst 페이지로 돌아가 다음을 선택합니다.

19. 스페이스가 생성되는 동안 상태 메시지가 표시됩니다. 스페이스가 생성되면 CodeCatalyst에 다음과 같은 메시지가 표시됩니다. 스페이스가 준비되었습니다. 마지막 단계는 프로젝트 생성입니다. 다음 중 하나를 수행할 수 있습니다.

- 지금 건너뛰기를 선택합니다.
- 스페이스에 대한 첫 번째 프로젝트 생성을 선택합니다. 블루프린트를 사용하여 프로젝트를 생성하는 방법을 보여주는 자습서는 [자습서: 최신 3계층 웹 애플리케이션 블루프린트로 프로젝트 만들기](#) 섹션을 참조하세요.

 Note

권한 오류 또는 배너가 표시되면 페이지를 새로 고치고 페이지를 다시 확인해 봅니다.

CodeCatalyst CodeCatalystWorkflowDevelopmentRole-*spaceName*을 생성하고 추가하려면

1. CodeCatalyst 콘솔에서 시작하기 전에 AWS Management Console을 연 다음 스페이스에 대해 동일한 AWS 계정으로 로그인했는지 확인합니다.
2. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
3. CodeCatalyst 스페이스로 이동합니다. 설정을 선택한 다음 AWS 계정을 선택합니다.
4. 역할을 생성하려는 AWS 계정의 연결을 선택합니다. AWS 계정 세부 정보 페이지가 표시됩니다.
5. AWS Management Console에서 역할 관리를 선택합니다.

AWS Management Console에서 Amazon CodeCatalyst 스페이스에 IAM 역할 추가 페이지가 열립니다. Amazon CodeCatalyst 스페이스 페이지입니다. 페이지에 액세스하려면 로그인해야 할 수 있습니다.

6. IAM에서 CodeCatalyst 개발 관리자 역할 생성을 선택합니다. 이 옵션은 개발 역할에 대한 권한 정책 및 신뢰 정책을 포함하는 서비스 역할을 생성합니다. 역할에는 CodeCatalystWorkflowDevelopmentRole-*spaceName* 이름이 있습니다. 역할 및 역할 정책에 대한 자세한 내용은 [CodeCatalystWorkflowDevelopmentRole-*spaceName* 서비스 역할 이해](#) 섹션을 참조하세요.

Note

이 역할은 개발자 계정에서만 사용하는 것이 권장되며 AdministratorAccess AWS 관리형 정책을 사용하여 이 AWS 계정에서 새 정책 및 리소스를 생성할 수 있는 전체 액세스 권한을 부여합니다.

7. 개발 역할을 생성을 선택합니다.
8. 연결 페이지에 있는 CodeCatalyst에서 사용할 수 있는 IAM 역할에서 계정에 추가된 IAM 역할 목록의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 봅니다.
9. 스페이스로 돌아가려면 Amazon CodeCatalyst로 이동을 선택합니다.

CodeCatalyst AWSSRoleForCodeCatalystSupport을 생성하고 추가하려면

1. CodeCatalyst 콘솔에서 시작하기 전에 AWS Management Console을 연 다음 스페이스에 대해 동일한 AWS 계정으로 로그인했는지 확인합니다.
2. CodeCatalyst 스페이스로 이동합니다. 설정을 선택한 다음 AWS 계정을 선택합니다.
3. 역할을 생성하려는 AWS 계정의 연결을 선택합니다. AWS 계정 세부 정보 페이지가 표시됩니다.
4. AWS Management Console에서 역할 관리를 선택합니다.

AWS Management Console에서 Amazon CodeCatalyst 스페이스에 IAM 역할 추가 페이지가 열립니다. Amazon CodeCatalyst 스페이스 페이지입니다. 페이지에 액세스하려면 로그인해야 할 수 있습니다.

5. CodeCatalyst 스페이스 세부 정보에서 CodeCatalyst 지원 역할 추가를 선택합니다. 이 옵션은 미리 보기 개발 역할에 대한 권한 정책 및 신뢰 정책을 포함하는 서비스 역할을 생성합니다. 역할에는 고유한 식별자가 추가된 AWSSRoleForCodeCatalystSupport 이름이 있습니다. 역할 및 역할 정책에 대한 자세한 내용은 [AWSSRoleForCodeCatalystSupport 서비스 역할 이해](#) 섹션을 참조하세요.
6. CodeCatalyst 지원에 역할 추가 페이지에서 기본값을 선택한 상태로 두고 역할 생성을 선택합니다.
7. CodeCatalyst에서 사용할 수 있는 IAM 역할에서 계정에 추가된 IAM 역할 목록의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 봅니다.
8. 스페이스로 돌아가려면 Amazon CodeCatalyst로 이동을 선택합니다.

AWS Builder ID를 생성하고, 첫 번째 스페이스를 생성하고, 계정을 추가한 후에 프로젝트를 생성할 수 있습니다. 자세한 내용은 [프로젝트 생성](#) 섹션을 참조하세요. CodeCatalyst 처음 사용하는 경우 [자습서: 최신 3계층 웹 애플리케이션 블루프린트로 프로젝트 만들기](#) 섹션부터 시작하는 것이 좋습니다.

초대 수락 및 AWS Builder ID 생성

프로젝트 또는 스페이스에 대한 초대를 수락하는 과정에서 Amazon CodeCatalyst에 처음으로 가입하는 방법을 알아봅니다. 초대를 수락하는 과정에서 AWS Builder ID를 생성하라는 메시지가 표시됩니다. AWS Builder ID를 사용하여 CodeCatalyst의 리소스에 액세스합니다.

Tip

추가 도움이 필요한 경우 [가입 문제](#) 섹션을 참조하세요.

다음은 프로젝트 또는 스페이스에 대한 초대로 CodeCatalyst를 시작하는 사용자에게 가능한 흐름 중 하나입니다.

Saanvi Sarkar는 프로젝트 관리자로 CodeCatalyst 프로젝트에 초대를 받은 개발자입니다. Saanvi는 초대를 수락하면 CodeCatalyst에 대한 로그인 페이지가 열립니다. Saanvi는 가입하기로 선택하고 이메일 주소와 암호를 제공하여 AWS Builder ID를 생성합니다. Saanvi는 AWS Builder ID를 사용하여 CodeCatalyst 및 기타 애플리케이션에 로그인할 수 있습니다. 나중에 프로파일을 편집하여 로그인 이메일 주소 또는 암호를 변경할 수 있습니다. 별칭을 선택하라는 메시지가 표시되면 Saanvi는 CodeCatalyst에 표시될 CodeCatalyst 별칭으로 SaanviSarkar를 지정하고 다른 프로젝트 멤버가 @mention Saanvi에 사용할 수 있도록 지정합니다. Saanvi는 가입한 후 AWS Builder ID 자격 증명을 사용하는 다른 애플리케이션에도 로그인 자격 증명을 사용할 수 있습니다.

가입이 완료되면 Saanvi는 초대에 지정된 CodeCatalyst 프로젝트와 스페이스에 자동으로 참여합니다. 초대는 프로젝트 및 스페이스에서 역할에 대한 사전 결정된 권한도 제공합니다. 프로젝트 설정에서 Saanvi의 별칭은 할당된 프로젝트 역할과 함께 멤버 목록에 표시됩니다. CodeCatalyst에서 소스 리포지토리를 사용하기 위해 Saanvi는 개인 액세스 토큰(PAT)을 생성합니다. PAT는 인증 토큰이 필요한 소스 변경 또는 작업을 수행할 때 CodeCatalyst에서 인증을 위해 사용됩니다.

Saanvi가 프로젝트에서 작업하면 프로젝트의 작업 활동 로그에 별칭이 나열됩니다. Saanvi의 문제와 의견은 다른 프로젝트 멤버가 그녀를 답글로 @mention할 수 있는 별칭을 보여줍니다. 다른 프로젝트 멤버를 @mention하기 위해 Saanvi는 CodeCatalyst 프로파일에서 별칭을 조회합니다.

잠시 시간이 되면 Saanvi는 멀티 팩터 인증(MFA)을 사용하여 CodeCatalyst에 로그인하도록 AWS Builder ID를 구성합니다. MFA를 구성하면 Saanvi는 CodeCatalyst 암호와 승인된 타사 인증 앱의 암호 또는 토큰을 조합하여 CodeCatalyst에 로그인할 수 있습니다.

초대 수락 및 AWS Builder ID 생성

Amazon CodeCatalyst의 프로젝트 또는 스페이스에 초대를 받으면 `notify@codecatalyst.aws`에서 초대 수락을 요청하는 이메일을 받게 됩니다. AWS Builder ID가 이미 있고 CodeCatalyst에 로그인한 경우 초대 수락을 선택하면 브라우저 탭에서 프로젝트 또는 공간이 자동으로 열립니다. 콘솔에 로그인하지 않았지만 AWS Builder ID가 있는 경우 로그인 페이지로 이동합니다. 자세한 내용은 [AWS Builder ID로 로그인](#) 단원을 참조하십시오.

AWS Builder ID가 없는 경우 초대 수락을 선택하면 로그인 페이지로 이동합니다. 로그인 페이지에서 AWS Builder ID를 생성하는 옵션을 선택해야 합니다.

Important

초대를 수락하고 개발 환경에서 리포지토리를 열면 스크립트가 CodeCatalyst 자격 증명에 액세스하여 실행될 수 있습니다. 계속하기 전에 소스를 신뢰해야 합니다.

초대를 수락하고 AWS Builder ID를 생성하려면

1. 초대 이메일에서 초대 수락을 선택합니다.
2. 로그인 페이지에서 가입하지 않았나요?을 선택합니다. AWS Builder ID를 생성합니다.

Tip

AWS Builder ID는 로그인하기 위해 생성한 자격 증명입니다. AWS 계정과 동일하지 않습니다.

3. AWS Builder ID 생성 페이지의 이메일 주소에 AWS Builder ID에 사용할 이메일 주소를 입력합니다.

이름에 AWS Builder ID를 사용하는 애플리케이션에 표시할 이름과 성을 입력합니다. 공백을 사용할 수 있습니다. Mary Major와 같은 AWS Builder ID 프로파일 이름이 됩니다. 이 이름은 나중에 변경할 수 있습니다.

다음을 선택합니다.

지정한 이메일로 확인 코드가 전송됩니다. 확인 코드에 이 코드를 입력한 다음, 확인을 선택합니다. 5분 후에도 코드를 받지 못하고 스팸 폴더나 정크 폴더에서 코드를 찾을 수 없다면, 코드 재전송을 선택하세요.

4. 코드가 확인되면, 암호 및 암호 확인의 요구 사항을 충족하는 암호를 입력합니다.
5. AWS Builder ID 생성을 선택합니다.
6. 별칭 생성 페이지에서 CodeCatalyst에서 고유한 사용자 식별자에 사용할 별칭을 입력합니다. MaryMajor와 같이 공백이 없는 이름의 단축 버전을 선택합니다. 다른 CodeCatalyst 사용자는 댓글과 풀 요청에서 이 별칭을 사용하여 사용자에게 @mention을 보냅니다. CodeCatalyst 프로필에는 AWS Builder ID의 전체 이름과 CodeCatalyst 별칭이 모두 포함됩니다. CodeCatalyst 별칭은 변경할 수 없습니다.

전체 이름과 별칭은 CodeCatalyst의 여러 영역에 표시됩니다. 예를 들어 프로필 이름은 활동 피드에서 나열된 액티비티에 표시되지만, 프로젝트 멤버는 사용자의 별칭을 사용자를 @mention하는 데 사용합니다.

별칭 생성을 선택합니다. 초대받은 프로젝트 또는 스페이스로 이동합니다.

AWS Builder ID로 로그인

다음 단계에 따라 Amazon CodeCatalyst 프로필에 로그인합니다.

Note

다중 인증(MFA)용 디바이스를 아직 등록하지 않으셨습니까? 보안을 높이려면 Amazon CodeCatalyst에서 MFA를 구성하는 것이 좋습니다. 자세한 내용은 [다중 인증에 사용할 디바이스를 등록하는 방법](#) 섹션을 참조하세요.

AWS Builder ID로 로그인하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 이메일 주소를 입력합니다. 원하는 경우 향후 로그인을 위해 이메일 주소를 저장하려면 내 이메일 주소 저장을 선택합니다. 계속을 선택합니다.
3. 암호를 입력합니다. 로그인을 선택합니다. 암호가 기억나지 않는 경우 [암호를 잊어버렸습니다](#)의 단계를 따르세요.

신뢰할 수 있는 디바이스

로그인 페이지에서 신뢰할 수 있는 디바이스 옵션을 선택하면, Amazon CodeCatalyst는 향후 해당 디바이스를 통한 모든 로그인에 승인된 것으로 간주합니다. Amazon CodeCatalyst는 신뢰할 수 있는 디바이스를 사용하는 한 MFA 코드를 입력하는 옵션을 제공하지 않습니다. 브라우저에서 로그인하거나 디바이스에 알 수 없는 IP 주소가 발급된 경우 등 몇 가지 예외가 있습니다.

SSO로 로그인

SSO를 사용하여 Amazon CodeCatalyst 로그인하려면 다음 단계를 따르세요.

AWS Builder ID를 대신 사용하여 로그인하려면 [AWS Builder ID로 로그인](#) 섹션을 참조하세요.

SSO로 로그인하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 로그인 옵션 선택에서 AWS Single Sign-On(SSO) 사용을 선택합니다.
3. AWS Identity Center 애플리케이션 이름에 ID 페더레이션 관리자가 제공한 애플리케이션 이름을 입력합니다.
4. IAM Identity Center로 계속 진행을 선택합니다.

사용자의 모든 스페이스 및 프로젝트 보기

사용자 홈 페이지에서 스페이스 및 프로젝트 목록을 볼 수 있습니다. 사용자 홈 페이지에는 사용자가 속한 각 스페이스, 스페이스 관리자와 같은 해당 스페이스의 사용자 역할, 사용자가 멤버십을 보유한 각 스페이스의 프로젝트 목록이 표시됩니다.

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 브라우저에 다음 주소를 입력합니다. <https://codecatalyst.aws/home>

The screenshot displays the Amazon CodeCatalyst interface. At the top, there's a header with a search bar for spaces, a 'Manage AWS Builder ID' button, and a 'Create space' button. Below this, three space cards are shown:

- EnchantedForest** (Space administrator):
 - Projects (2):
 - WildWaves**: Pull requests, Workflows, Source repositories, Environments.
 - FracturedFairyTales**: Pull requests, Workflows, Source repositories, Environments.
- org** (Space administrator):
 - Projects (4):
 - migration**: Pull requests, Workflows, Source repositories, Environments.
 - test**: Pull requests, Workflows, Source repositories, Environments.
 - 12597**: Pull requests, Workflows, Source repositories, Environments.
- AnyCompany** (Space member):
 - Projects (1):
 - newproject**: Pull requests, Workflows, Source repositories, Environments.

- 열려 있는 스페이스 또는 프로젝트를 선택합니다. 예상했던 스페이스나 프로젝트가 보이지 않으면 다른 사용자로 로그인해야 할 수 있습니다.

CodeCatalyst 프로파일 보기 및 관리

Amazon CodeCatalyst에서 사용자 프로파일을 확인하여 이메일 주소 및 CodeCatalyst 별칭과 같은 정보를 얻을 수 있습니다. 프로파일과 AWS Builder ID를 업데이트할 수도 있습니다. 암호를 잊어버린 경우 암호 재설정을 요청하면 됩니다.

CodeCatalyst 프로파일 보기

가입 시 Amazon CodeCatalyst에 로그인하기 위한 자격 증명으로 사용되며 프로파일에서 관리되는 정보를 제공합니다. 여기에는 이름, 별명, CodeCatalyst에 로그인할 때 사용하는 이메일 주소가 포함됩니다.

Note

AWS Builder ID 별명은 CodeCatalyst 별칭이 아닙니다. 가입 시 CodeCatalyst 별칭을 선택했습니다.

CodeCatalyst 프로파일 보기

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 오른쪽 상단에서 첫 번째 이니셜이 있는 아이콘 옆의 화살표를 선택한 다음 내 설정을 선택합니다. CodeCatalyst 내 설정 페이지가 열립니다.
3. AWS Builder ID 이메일 주소 또는 암호를 업데이트하거나 MFA를 설정하려면 AWS Builder ID 관리 선택합니다. AWS Builder ID 페이지가 열립니다.

다른 사용자의 CodeCatalyst 프로파일 보기

다른 사용자의 CodeCatalyst 프로파일 보기

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 측면 탐색에서 프로젝트 설정을 선택합니다. 멤버 탭을 선택합니다. CodeCatalyst 프로젝트의 멤버 목록을 확인합니다.
3. 조회하려는 멤버 이름을 선택하거나 @멘션합니다. 내 설정 페이지에는 사용자의 별칭, 이메일 주소, 성명이 표시됩니다. 프로젝트 멤버를 @멘션할 때 CodeCatalyst 별칭을 사용합니다.

Note

사용자의 AWS Builder ID 닉네임은 CodeCatalyst 별칭이 아닙니다. 가입 시 CodeCatalyst 별칭을 선택했습니다.

프로젝트에서 다른 사용자의 프로파일을 보려면 목록에서 해당 사용자의 이름을 선택하세요.

프로파일 업데이트

CodeCatalyst에서 프로파일은 AWS Builder ID로 관리되는 개인 정보와 CodeCatalyst에서 관리되는 설정으로 구성됩니다.

- 프로파일의 전체 이름, 이메일 주소 및 암호는 AWS Builder ID에서 관리합니다. 가입할 때 이 정보를 입력했습니다. 애플리케이션 로그인에 인증 앱을 사용하도록 MFA를 설정하면 CodeCatalyst는 AWS Builder ID 페이지로 이동합니다.
- 개인 액세스 토큰(PAT), CodeCatalyst 알림 및 언어 기본 설정에 대한 CodeCatalyst 설정은 CodeCatalyst의 내 설정 페이지에서 관리합니다. 자세한 내용은 [개인 액세스 토큰을 사용하여 사용자 리포지토리 액세스 권한 부여](#) 섹션을 참조하세요.

Note

AWS Builder ID 전체 이름(CodeCatalyst 표시 이름)과 이름을 업데이트할 수 있습니다. 단, CodeCatalyst 별칭은 변경할 수 없습니다.

AWS Builder ID 또는 이메일 주소 업데이트

AWS Builder ID 또는 이메일 주소 업데이트

- <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
- 오른쪽 상단에서 첫 번째 이니셜이 있는 아이콘 옆의 화살표를 선택한 다음 내 설정을 선택합니다. CodeCatalyst 내 설정 페이지가 열립니다.
- 프로파일 페이지에서 AWS Builder ID 관리를 선택합니다. AWS Builder ID 페이지가 열립니다.
- 페이지 왼쪽에서 내 세부 정보를 선택합니다.
- 프로파일 정보에서 편집을 선택하여 이름 또는 별칭을 업데이트합니다. 별명을 지정하지 않은 경우 별명 필드에 전체 이름의 이름이 반영됩니다. CodeCatalyst 별칭이 아닙니다.

Note

그러면 AWS Builder ID 전체 이름과 이름이 업데이트됩니다. 이렇게 해도 CodeCatalyst 별칭은 업데이트되지 않습니다.

연락처 정보에서 편집을 선택하여 이메일 주소를 업데이트합니다.

Note

이렇게 하면 CodeCatalyst 에 로그인하는 데 사용할 이메일 주소가 업데이트됩니다.

AWS Builder ID와 연결된 CodeCatalyst 암호 변경

다음 지침에 따라 AWS Builder ID와 연결된 Amazon CodeCatalyst 암호를 변경합니다.

Note

SSO를 사용하여 CodeCatalyst에 로그인하는 경우 암호 변경에 대해 관리자에게 문의하세요.

CodeCatalyst 암호를 변경하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 오른쪽 상단에서 첫 번째 이니셜이 있는 아이콘 옆의 화살표를 선택한 다음 사용자 프로파일을 선택합니다. CodeCatalyst 내 설정 페이지가 열립니다.
3. 프로파일 페이지에서 AWS Builder ID 관리를 선택합니다. AWS Builder ID 페이지가 열립니다.
4. 페이지 왼쪽에서 보안을 선택합니다.
5. 암호 변경을 선택하고 지침을 따릅니다.

CodeCatalyst AWS CLI 에서를 사용하도록 설정

Amazon CodeCatalyst 콘솔에서는 대부분의 일상 작업을 수행합니다. 그러나 CodeCatalyst에서 개발 환경, 개인 액세스 토큰 또는 이벤트 로그를 사용할 AWS CLI 때를 설정하고 구성할 수 있습니다. CodeCatalyst AWS CLI 와 함께 사용하려면 먼저 설치하고 프로필을 구성해야 합니다.

CodeCatalyst AWS CLI 용를 설정하려면

1. AWS CLI의 최신 버전을 설치합니다. 버전이 이미 AWS CLI 설치되어 있는 경우 버전이 최신 버전이고 CodeCatalyst에 대한 명령이 포함되어 있는지 확인하고 필요한 경우 업데이트합니다. CodeCatalyst 명령이 포함된 버전이 설치되어 있는지 확인하려면 명령 프롬프트를 열고 다음 명령을 실행합니다.

```
aws codecatalyst help
```

CodeCatalyst 명령 목록이 표시되면 CodeCatalyst를 지원하는 버전이 설치되어 있는 것입니다. 명령이 인식되지 않으면 버전을 AWS CLI 최신 버전으로 업데이트합니다. 자세한 내용은 AWS Command Line Interface 사용 설명서의 [의 최신 버전 설치 또는 업데이트를 AWS CLI](#) 참조하세요.

2. 프로파일이 없거나 CodeCatalyst 전용으로 이름이 지정된 프로파일을 사용하려는 경우, `aws configure` 명령을 실행하여 프로파일을 생성합니다. 특별히 CodeCatalyst 와 함께 사용하기 위해 명명된 프로파일을 생성하는 것을 권장하지만, 기본 프로파일도 사용할 수 있습니다. 자세한 정보는 [구성 기본 사항](#)을 참조하세요.
3. 다음과 같이 프로파일의 config 파일을 편집하여 CodeCatalyst에 연결하기 위한 섹션을 추가합니다. config 파일은 `~/.aws/config`(Linux 또는 macOS) 또는 `C:\Users\USERNAME\.aws\config`(Windows)에 저장됩니다.

```
[profile codecatalyst]
region = us-west-2
sso_session = codecatalyst

[sso-session codecatalyst]
sso_region = us-east-1
sso_start_url = https://view.awsapps.com/start
sso_registration_scopes = codecatalyst:read_write
```

4. 파일을 저장합니다.
5. CodeCatalyst 명령을 실행해보기 전에 새 터미널 또는 명령 프롬프트를 열고 다음 명령을 실행하여 `aws codecatalyst` 명령을 실행할 자격 증명을 요청하고 검색합니다. 필요한 경우 `codecatalyst`를 프로파일 이름으로 바꿉니다.

```
aws sso login --profile codecatalyst
```

`codecatalyst` 명령의 예를 보려면 다음 주제를 참조하세요.

- [개인 액세스 토큰을 사용하여 사용자 리포지토리 액세스 권한 부여](#)
- [이벤트 로깅을 사용하여 로깅된 이벤트에 액세스](#)

시작하기 자습서

Amazon CodeCatalyst는 프로젝트를 시작하는 데 도움이 되는 다양한 템플릿을 제공합니다. 빈 프로젝트로 시작하고 여기에 리소스를 추가하도록 선택할 수도 있습니다. 이 자습서의 단계에 따라 CodeCatalyst에서 작업할 수 있는 몇 가지 방법을 알아봅니다.

CodeCatalyst 처음 사용하는 경우 [자습서: 최신 3계층 웹 애플리케이션 블루프린트로 프로젝트 만들기](#) 섹션부터 시작하는 것이 좋습니다.

Note

이러한 자습서를 따르려면 먼저 설정을 완료해야 합니다. 자세한 내용은 [CodeCatalyst 설정 및 로그인](#) 섹션을 참조하세요.

주제

- [자습서: 최신 3계층 웹 애플리케이션 블루프린트로 프로젝트 만들기](#)
- [자습서: 빈 프로젝트로 시작하고 수동으로 리소스 추가](#)
- [자습서: CodeCatalyst 생성형 AI 기능을 사용하여 개발 작업 속도 향상](#)

CodeCatalyst의 특정 기능 영역에 초점을 맞춘 추가 자습서는 다음을 참조하세요.

- [스택 알림 시작하기](#)
- [CodeCatalyst 소스 리포지토리 및 단일 페이지 애플리케이션 블루프린트 시작하기](#)
- [워크플로 시작하기](#)
- [사용자 지정 블루프린트 시작하기](#)
- [Amazon CodeCatalyst 작업 개발자 안내서 시작하기](#)

심층 자습서는 다음을 참조하세요.

- [자습서: Amazon S3에 아티팩트 업로드](#)
- [자습서: 서버리스 애플리케이션 배포](#)
- [자습서: Amazon ECS에 애플리케이션 배포](#)
- [자습서: Amazon EKS에 애플리케이션 배포](#)

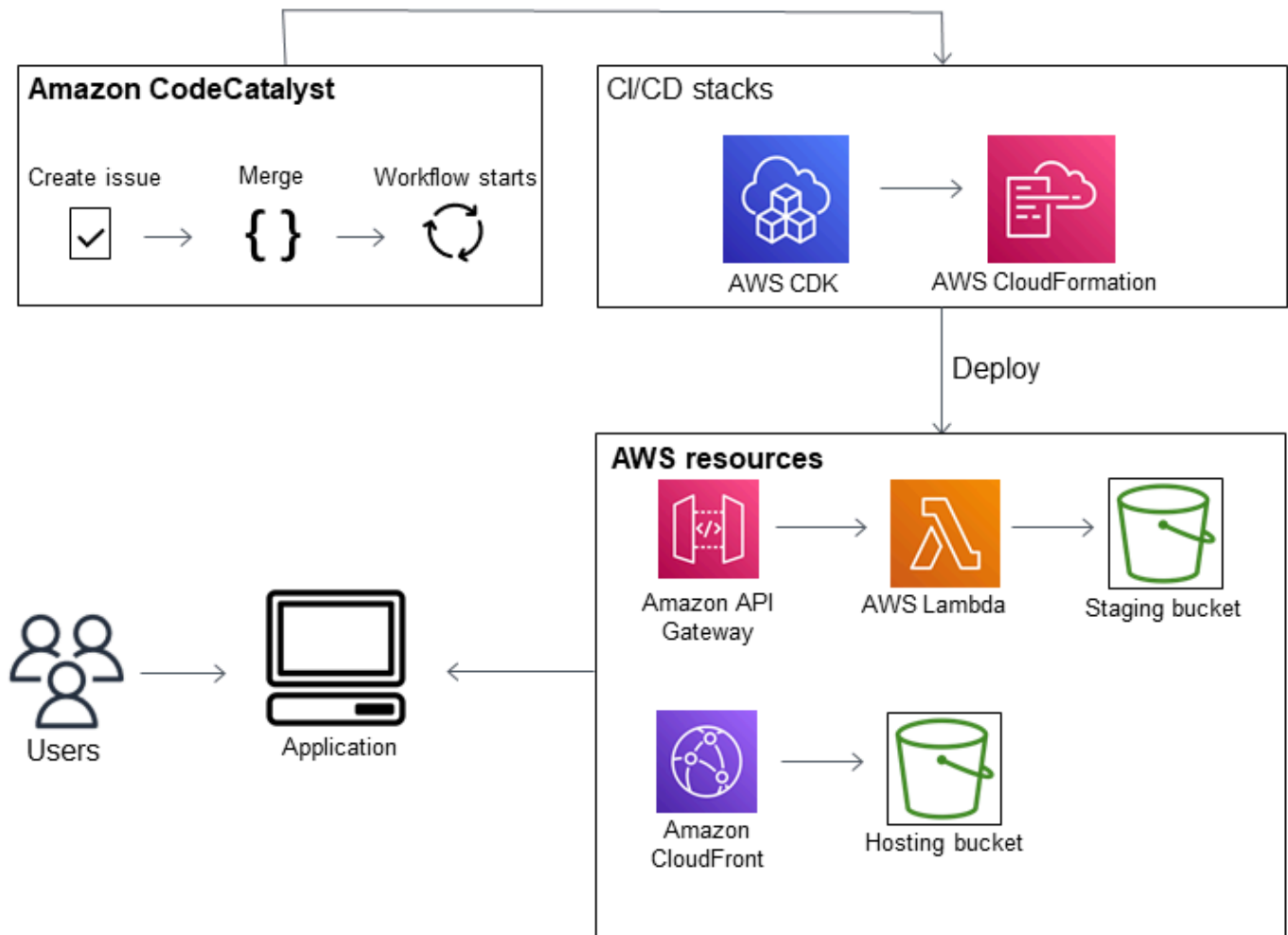
- [자습서: GitHub 작업을 사용한 린트 코드](#)
- [자습서: React 애플리케이션 생성 및 업데이트](#)

자습서: 최신 3계층 웹 애플리케이션 블루프린트로 프로젝트 만들기

블루프린트로 프로젝트를 생성하여 소프트웨어 개발을 더 빠르게 시작할 수 있습니다. 블루프린트로 생성된 프로젝트에는 코드를 관리하는 소스 리포지토리와 애플리케이션을 빌드하고 배포하는 워크플로를 포함하여 필요한 리소스가 포함됩니다. 이 자습서에서는 최신 3계층 웹 애플리케이션 블루프린트를 사용하여 Amazon CodeCatalyst에서 프로젝트를 생성하는 방법을 안내합니다. 또한 자습서에는 배포된 샘플 보기, 작업할 다른 사용자 초대, 풀 요청이 병합될 때 연결된 AWS 계정의 리소스에 자동으로 빌드되고 배포되는 풀 요청으로 코드 변경도 포함됩니다. CodeCatalyst가 보고서, 활동 피드 및 기타 도구를 사용하여 프로젝트를 생성하는 경우 블루프린트는 프로젝트와 연결된 AWS 계정에 AWS 리소스를 생성합니다. 블루프린트 파일을 사용하면 샘플 최신 애플리케이션을 빌드 및 테스트하고 AWS 클라우드의 인프라에 배포할 수 있습니다.

다음 그림은 CodeCatalyst의 도구를 사용하여 변경 사항을 추적, 병합 및 자동으로 빌드하는 문제를 생성한 다음 CodeCatalyst 프로젝트에서 AWS CDK 및 CloudFormation을 허용하고 인프라를 프로비저닝하는 워크플로를 시작하는 방법을 보여줍니다.

작업은 연결된 AWS 계정에서 리소스를 생성하고 API Gateway 엔드포인트를 사용하여 애플리케이션을 서버리스 AWS Lambda 함수에 배포합니다. 이 AWS Cloud Development Kit (AWS CDK) 작업은 하나 이상의 AWS CDK 스택을 CloudFormation 템플릿으로 변환하고 스택을 AWS 계정에 배포합니다. 스택의 리소스에는 동적 웹 콘텐츠를 배포하기 위한 Amazon CloudFront 리소스, 애플리케이션 데이터에 대한 Amazon DynamoDB 인스턴스, 배포된 애플리케이션을 지원하는 역할 및 정책이 포함됩니다.



최신 3계층 웹 애플리케이션 블루프린트로 프로젝트를 생성하면 다음 리소스로 프로젝트가 생성됩니다.

CodeCatalyst 프로젝트에서:

- 샘플 코드 및 워크플로 YAML이 있는 [소스 리포지토리](#)
- 기본 브랜치를 변경할 때마다 샘플 코드를 빌드하고 배포하는 [워크플로](#)
- 작업을 계획하고 추적하는 데 사용할 수 있는 문제 보드 및 백로그
- 샘플 코드에 자동 보고서가 포함된 테스트 보고서 제품군

연결된 AWS 계정에서:

- 애플리케이션에 필요한 리소스를 생성하는 세 개의 AWS CloudFormation 스택.

이 자습서의 일부로 AWS 및 CodeCatalyst에서 생성할 리소스에 대한 자세한 내용은 [레퍼런스](#) 섹션을 참조하세요.

Note

프로젝트에 포함된 리소스와 샘플은 선택한 블루프린트에 따라 달라집니다. Amazon CodeCatalyst는 정의된 언어 또는 프레임워크와 관련된 리소스를 정의하는 여러 프로젝트 블루프린트를 제공합니다. 블루프린트에 대한 자세한 내용은 [CodeCatalyst 블루프린트를 사용하여 포괄적인 프로젝트 생성](#) 섹션을 참조하세요.

주제

- [사전 조건](#)
- [1단계: 최신 3계층 웹 애플리케이션 프로젝트 생성](#)
- [2단계: 프로젝트에 초대하기](#)
- [3단계: 협업 및 작업 추적을 위한 문제 생성](#)
- [4단계: 소스 리포지토리 보기](#)
- [5단계: 테스트 브랜치를 사용하여 개발 환경 생성 및 빠른 코드 변경](#)
- [6단계: 최신 애플리케이션을 빌드하는 워크플로 보기](#)
- [7단계: 다른 사람에게 변경 사항을 검토하도록 요청](#)
- [8단계: 문제 종료](#)
- [리소스 정리](#)
- [레퍼런스](#)

사전 조건

이 자습서에서 최신 애플리케이션 프로젝트를 생성하려면 다음과 같이 [CodeCatalyst 설정 및 로그인](#)의 작업을 완료해야 합니다.

- CodeCatalyst에 로그인하기 위한 AWS Builder ID가 있어야 합니다.
- 스페이스에 속해 해당 스페이스에서 스페이스 관리자 또는 파워 유저 역할을 할당받습니다. 자세한 내용은 [스페이스 생성](#), [사용자에게 스페이스 권한 부여](#), [스페이스 관리자 역할](#) 섹션을 참조하세요.
- AWS 계정이 사용자 스페이스와 연결되어 있고 가입 중에 생성한 IAM 역할이 있어야 합니다. 예를 들어, 가입 중에 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 정책이라는

역할 정책을 사용하여 서비스 역할을 생성하도록 선택할 수 있습니다. 역할에는 고유한 식별자가 추가된 `CodeCatalystWorkflowDevelopmentRole-spaceName` 이름이 있습니다. 역할 및 역할 정책에 대한 자세한 내용은 [CodeCatalystWorkflowDevelopmentRole-*spaceName* 서비스 역할 이해](#) 섹션을 참조하세요. 역할을 생성하는 단계는 [계정 및 스페이스의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 생성](#) 섹션을 참조하세요.

1단계: 최신 3계층 웹 애플리케이션 프로젝트 생성

생성한 후에는 코드를 개발 및 테스트하고, 개발 작업을 조정하고, 프로젝트 지표를 볼 수 있습니다. 프로젝트에는 개발 도구 및 리소스도 포함되어 있습니다.

이 자습서에서는 최신 3계층 웹 애플리케이션 블루프린트를 사용하여 대화형 애플리케이션을 생성합니다. 프로젝트의 일부로 자동으로 생성되고 실행되는 워크플로는 애플리케이션을 빌드하고 배포합니다. 워크플로는 모든 역할 및 계정 정보가 스페이스에 구성된 후에만 성공적으로 실행됩니다. 워크플로가 성공적으로 실행되면 엔드포인트 URL을 방문하여 애플리케이션을 볼 수 있습니다.

블루프린트를 사용하여 프로젝트를 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 프로젝트를 생성하려는 스페이스로 이동합니다.
3. 프로젝트 생성을 선택합니다.
4. 블루프린트로 시작을 선택합니다.
5. 검색 창에 **modern**를 입력합니다.
6. 최신 3계층 웹 애플리케이션 블루프린트를 선택한 다음 다음을 선택합니다.
7. 프로젝트 이름에 프로젝트의 이름을 입력합니다. 예:

MyExampleProject.

Note

이름은 스페이스 내에서 고유해야 합니다.

8. 계정에서 가입 중에 추가한 AWS 계정을 선택합니다. 블루프린트는 이 계정에 리소스를 설치합니다.
9. 배포 역할에서 등록 중에 추가한 역할을 선택합니다. 예를 들어, `CodeCatalystWorkflowDevelopmentRole-spaceName`를 선택합니다.

나열된 역할이 없는 경우 하나를 추가합니다. 역할을 추가하려면 IAM 역할을 추가를 선택하고 AWS 계정에 역할을 추가합니다. 자세한 내용은 [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 섹션을 참조하세요.

10. 컴퓨팅 플랫폼에서 Lambda를 선택합니다.
11. 프론트엔드 호스팅 옵션에서 Amplify Hosting을 선택합니다. AWS Amplify에 대한 자세한 내용은 AWS Amplify 사용 설명서의 [AWS Amplify 호스팅이란 무엇입니까?](#)를 참조하세요.
12. 배포 리전에 블루프린트가 Mysfits 애플리케이션 및 지원 리소스를 배포하려는 AWS 리전의 리전 코드를 입력합니다. 리전 코드 목록은 AWS 일반 참조의 [리전 엔드포인트](#)를 참조하세요.
13. 애플리케이션 이름에서 기본값인 `mysfitsstring`을 그대로 둡니다.
14. (선택 사항) 프로젝트 미리 보기 생성에서 코드 보기를 선택하여 블루프린트가 설치할 소스 파일을 미리 봅니다. 워크플로 보기를 선택하여 블루프린트가 설치할 CI/CD 워크플로 정의 파일을 미리 봅니다. 미리 보기는 선택 사항에 따라 동적으로 업데이트됩니다.
15. 프로젝트 생성을 선택합니다.

프로젝트 워크플로는 프로젝트를 생성하는 즉시 시작됩니다. 코드 구축 및 배포를 완료하는 데 약간의 시간이 걸릴 것입니다. 그 동안 다른 사람을 프로젝트에 초대하세요.

2단계: 프로젝트에 초대하기

이제 프로젝트를 설정했으므로 다른 사람을 초대하여 함께 작업하도록 하세요.

프로젝트에 누군가를 초대하려면

1. 사용자를 초대하려는 프로젝트로 이동합니다.
2. 탐색 창에서 프로젝트 설정을 선택합니다.
3. 멤버 탭에서 초대를 선택합니다.
4. 프로젝트의 사용자로 초대하려는 사람의 이메일 주소를 입력합니다. 여러 이메일 주소를 스페이스 또는 쉼표로 구분하여 입력할 수 있습니다. 프로젝트의 멤버가 아닌 스페이스의 멤버 중에서 선택할 수도 있습니다.
5. 사용자의 역할을 선택합니다.

사용자 추가를 완료했으면 초대를 선택합니다.

3단계: 협업 및 작업 추적을 위한 문제 생성

CodeCatalyst는 프로젝트와 관련된 기능, 작업, 버그 및 기타 작업과 문제를 추적하는 데 도움이 됩니다. 문제를 생성하여 필요한 작업과 아이디어를 추적할 수 있습니다. 기본적으로 문제를 생성하면 백로그에 추가됩니다. 진행 중인 작업을 추적하는 보드로 문제를 이동할 수 있습니다. 특정 프로젝트 멤버에게 문제를 할당할 수도 있습니다.

프로젝트에 대한 문제를 생성하려면

1. 탐색 창에서 문제를 선택합니다.
2. 문제 생성을 선택합니다.
3. 문제 제목에 문제의 이름을 입력합니다. 필요에 따라 문제에 대한 설명을 제공합니다. 이 예에서는 **make a change in the src/mysfit_data.json file.**을 사용합니다.
4. 우선 순위, 견적, 상태 및 레이블을 선택합니다. 담당자에서 +나 추가를 선택하여 문제를 자신에게 할당합니다.
5. 문제 생성을 선택합니다. 이제 보드에 문제가 표시됩니다. 카드를 선택하여 문제를 진행 중 열로 이동합니다.

자세한 내용은 [CodeCatalyst의 문제 처리 추적 및 구성](#) 섹션을 참조하세요.

4단계: 소스 리포지토리 보기

블루프린트는 애플리케이션 또는 서비스를 정의하고 지원하는 파일이 포함된 소스 리포지토리를 설치합니다. 소스 리포지토리의 몇 가지 주목할 만한 디렉터리와 파일은 다음과 같습니다.

- .cloud9 디렉터리 - AWS Cloud9 개발 환경에 대한 지원 파일이 포함되어 있습니다.
- .codecatalyst 디렉터리 - 블루프린트에 포함된 각 YAML 워크플로에 대한 워크플로 정의 파일을 포함합니다.
- .idea 디렉터리 - JetBrains 개발 환경에 대한 지원 파일이 포함되어 있습니다.
- .vscode 디렉터리 - Visual Studio Code 개발 환경에 대한 지원 파일이 포함되어 있습니다.
- cdkStacks 디렉터리 - AWS 클라우드에서 인프라를 정의하는 AWS CDK 스택 파일이 포함되어 있습니다.
- src 디렉터리 - 애플리케이션 소스 코드를 포함합니다.
- tests 디렉터리 - 애플리케이션을 빌드하고 테스트할 때 실행되는 자동 CI/CD 워크플로의 일부로 실행되는 통합 및 단위 테스트용 파일이 포함되어 있습니다.

- web 디렉터리 - 프론트엔드 소스 코드를 포함합니다. 다른 파일에는 프로젝트에 대한 중요한 메타데이터가 포함된 `package.json` 파일, 웹 사이트 `index.html` 페이지, Lint 코드용 `.eslintrc.cjs` 파일, 루트 파일 및 컴파일러 옵션을 지정하기 위한 `tsconfig.json` 파일과 같은 프로젝트 파일이 포함됩니다.
- Dockerfile 파일 - 애플리케이션의 컨테이너를 설명합니다.
- README.md 파일 - 프로젝트 구성 정보를 포함합니다.

프로젝트의 소스 리포지토리로 이동하려면

1. 프로젝트로 이동하여 다음 중 하나를 수행합니다.
 - 프로젝트의 요약 페이지에서 목록에서 원하는 리포지토리를 선택한 다음 리포지토리 보기를 선택합니다.
 - 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다. 소스 리포지토리의 경우 목록에서 리포지토리의 이름을 선택합니다. 필터 표시줄에 리포지토리 이름의 일부를 입력하여 리포지토리 목록을 필터링할 수 있습니다.
2. 리포지토리의 홈 페이지에서 리포지토리의 내용과 풀 요청 수 및 워크플로와 같은 관련 리소스에 대한 정보를 확인합니다. 기본적으로 기본 브랜치의 내용이 표시됩니다. 드롭다운 목록에서 다른 브랜치를 선택하여 보기를 변경할 수 있습니다.

5단계: 테스트 브랜치를 사용하여 개발 환경 생성 및 빠른 코드 변경

개발 환경을 생성하여 소스 리포지토리의 코드를 빠르게 처리할 수 있습니다. 이 자습서에서는 다음을 가정합니다.

- AWS Cloud9 개발 환경을 생성합니다.
- 개발 환경을 생성할 때 기본 브랜치의 새 브랜치에서 작업할 옵션을 선택합니다.
- 이 새 브랜치의 이름을 `test`로 합니다.

이후 단계에서는 개발 환경을 사용하여 코드를 변경하고 풀 요청을 생성합니다.

새 브랜치로 개발 환경을 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 개발 환경을 생성하려는 프로젝트로 이동합니다.

3. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다. 또는 탐색 창에서 코드를 선택하고 소스 리포지토리를 선택한 다음 개발 환경을 생성할 리포지토리를 선택합니다.
4. 리포지토리 홈 페이지에서 개발 환경 생성을 선택합니다.
5. 드롭다운 메뉴에서 지원되는 IDE를 선택합니다. 자세한 정보는 [개발 환경에 지원되는 통합 개발 환경을 참조하세요](#).
6. 복제할 리포지토리를 선택하고, 새 브랜치에서 작업을 선택하고, 브랜치 이름 필드에 브랜치 이름을 입력하고, 다음에서 브랜치 생성 드롭다운 메뉴에서 새 브랜치를 만들 브랜치를 선택합니다.
7. 원하는 경우 개발 환경의 별칭을 추가할 수 있습니다.
8. 선택적으로 개발 환경 구성 편집 버튼을 선택하여 개발 환경의 컴퓨팅, 스토리지 또는 제한 시간 구성을 편집합니다.
9. 생성(Create)을 선택합니다. 개발 환경이 생성되는 동안 개발 환경 상태 열에 시작 중이 표시되고, 개발 환경이 생성되면 상태 열에 실행 중이 표시됩니다. 선택한 IDE의 개발 환경과 함께 새 탭이 열립니다. 코드를 편집하고 변경 사항을 커밋하고 푸시할 수 있습니다.

이 섹션에서는 풀 요청이 병합될 때 연결된 AWS 계정의 리소스에 자동으로 빌드되고 배포되는 풀 요청으로 코드를 변경하여 CodeCatalyst에서 생성된 샘플 애플리케이션을 사용합니다.

src/mysfit_data.json 파일을 변경하려면

1. 프로젝트 개발 환경으로 이동합니다. AWS Cloud9에서 사이드 탐색 메뉴를 확장하여 파일을 찾습니다. mysfits, src를 확장하고 src/mysfit_data.json를 엽니다.
2. 파일에서 "Age": 필드 값을 6에서 12로 변경합니다. 라인은 다음과 같을 것입니다.

```
{
  "Age": 12,
  "Description": "Twilight's personality sparkles like the night sky and is looking for a forever home with a Greek hero or God. While on the smaller side at 14 hands, he is quite adept at accepting riders and can fly to 15,000 feet. Twilight needs a large area to run around in and will need to be registered with the FAA if you plan to fly him above 500 feet. His favorite activities include playing with chimeras, going on epic adventures into battle, and playing with a large inflatable ball around the paddock. If you bring him home, he'll quickly become your favorite little Pegasus.",
  "GoodEvil": "Good",
  "LawChaos": "Lawful",
  "Name": "Twilight Glitter",
  "ProfileImageUri": "https://www.mythicalmysfits.com/images/pegasus_hover.png",
```

```
"Species": "Pegasus",  
  "ThumbImageUri": "https://www.mythicalmysfits.com/images/pegasus_thumb.png"  
},
```

3. 파일을 저장합니다.
4. **cd /projects/mysfits** 명령을 사용하여 mysfits 리포지토리로 변경합니다.
5. git add, git commit, git push 명령을 사용하여 변경 사항을 추가, 커밋 및 푸시합니다.

```
git add .  
git commit -m "make an example change"  
git push
```

6단계: 최신 애플리케이션을 빌드하는 워크플로 보기

최신 애플리케이션 프로젝트를 생성한 후 CodeCatalyst는 워크플로를 포함하여 사용자를 대신하여 여러 리소스를 생성합니다. 워크플로는 코드를 빌드, 테스트 및 배포하는 방법을 설명하는 .yaml 파일에 정의된 자동화된 절차입니다.

이 자습서에서 CodeCatalyst는 워크플로를 생성하고 프로젝트를 생성할 때 자동으로 워크플로를 시작했습니다. (프로젝트를 생성한 시간에 따라 워크플로가 계속 실행 중일 수 있습니다.) 다음 절차에 따라 워크플로의 진행 상황을 확인하고, 생성된 로그 및 테스트 보고서를 검토한 다음, 마지막으로 배포된 애플리케이션의 URL로 이동합니다.

워크플로 진행 상황을 확인하려면

1. CodeCatalyst 콘솔의 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.

워크플로 목록이 나타납니다. 다음은 프로젝트를 생성할 때 CodeCatalyst 블루프린트가 생성하고 시작한 워크플로입니다.

2. 워크플로 목록을 관찰합니다. 네 가지가 표시됩니다.
 - 상단의 두 워크플로는 [5단계: 테스트 브랜치를 사용하여 개발 환경 생성 및 빠른 코드 변경](#)에서 이전에 생성한 test 브랜치에 해당합니다. 이러한 워크플로는 main 브랜치의 워크플로를 복제합니다. ApplicationDeploymentPipeline은 main 브랜치와 함께 사용하도록 구성되어 있으므로 활성화되지 않습니다. 풀 요청이 이루어지지 않아 OnPullRequest 워크플로가 실행되지 않았습 니다.

- 하단의 두 워크플로는 이전에 블루프린트를 실행할 때 생성된 main 브랜치에 해당합니다. ApplicationDeploymentPipeline 워크플로가 활성 상태이며 진행 중(또는 완료됨)으로 실행됩니다.

Note

Build@cdk_bootstrap 또는 DeployBackend 오류로 인해 ApplicationDeploymentPipeline 실행이 실패하는 경우, 이전에 최신 3계층 웹 애플리케이션을 실행했고 기존 리소스가 현재 블루프린트와 충돌하지 않기 때문일 수 있습니다. 이러한 이전 리소스를 삭제한 다음 워크플로를 다시 실행해야 합니다. 자세한 내용은 [리소스 정리](#) 섹션을 참조하세요.

3. 하단에서 main 브랜치와 연결된 ApplicationDeploymentPipeline 워크플로를 선택합니다. 이 워크플로는 main 브랜치의 소스 코드를 사용하여 실행되었습니다.

워크플로 다이어그램이 나타납니다. 다이어그램에는 태스크 또는 작업을 각각 나타내는 여러 블록이 나와 있습니다. 대부분의 작업은 세로로 정렬되며 맨 위의 작업은 아래 작업보다 먼저 실행됩니다. 나란히 배열된 작업은 병렬로 실행됩니다. 함께 그룹화된 작업은 모두 성공적으로 실행되어야 아래의 작업을 시작할 수 있습니다.

기본 블록은 다음과 같습니다.

- WorkflowSource – 이 블록은 소스 리포지토리를 나타냅니다. 여기에는 무엇보다도 소스 리포지토리 이름(mysfits)과 워크플로 실행을 자동으로 시작한 커밋이 표시됩니다. CodeCatalyst는 프로젝트를 생성할 때 이 커밋을 생성했습니다.
 - 빌드 - 이 블록은 다음 작업을 시작하려면 두 작업을 모두 성공적으로 완료해야 하는 두 작업의 그룹화를 나타냅니다.
 - DeployBackend – 이 블록은 애플리케이션의 백엔드 구성 요소를 AWS 클라우드에 배포하는 작업을 나타냅니다.
 - 테스트 - 이 블록은 다음 작업을 시작하기 위해 모두 성공적으로 완료해야 하는 두 가지 테스트 작업의 그룹화를 나타냅니다.
 - DeployFrontend – 이 블록은 애플리케이션의 프론트엔드 구성 요소를 AWS 클라우드에 배포하는 작업을 나타냅니다.
4. 정의 탭(상단 근처)을 선택합니다. [워크플로 정의 파일](#)이 오른쪽에 나타납니다. 파일에는 다음과 같은 주목할 만한 섹션이 있습니다.

- 상단의 Triggers 섹션. 이 섹션에서는 코드를 소스 리포지토리의 main 브랜치로 푸시할 때마다 워크플로를 시작해야 함을 나타냅니다. 다른 브랜치(예: test)로 푸시해도 이 워크플로가 시작되지 않습니다. 워크플로는 main 브랜치의 파일을 사용하여 실행됩니다.
 - Actions 섹션, Triggers 아래. 이 섹션에서는 워크플로 다이어그램에 표시되는 작업을 정의합니다.
5. 최신 상태 탭(상단 근처)을 선택하고 워크플로 다이어그램에서 작업을 선택합니다.
 6. 오른쪽에서 구성 탭을 선택하여 최근 실행 중에 작업에 사용되는 구성 설정을 확인합니다. 각 구성 설정에는 워크플로 정의 파일에 일치하는 속성이 있습니다.
 7. 콘솔을 열어 두고 다음 절차로 이동합니다.

빌드 로그 및 테스트 보고서를 검토하려면

1. 최신 상태 탭을 선택합니다.
2. 워크플로 다이어그램에서 DeployFrontend 작업을 선택합니다.
3. 작업을 마칠 때까지 기다리세요. '진행 중' 아이콘
)
 이 '성공' 아이콘
)
 으로 변경될 때까지 기다립니다.
4. build_backend 작업을 선택합니다.
5. 로그 탭을 선택하고 몇 개의 섹션을 확장하여 이러한 단계에 대한 로그 메시지를 확인합니다. 백엔드 설정과 관련된 메시지를 볼 수 있습니다.
6. 보고서 탭을 선택한 다음 backend-coverage.xml 보고서를 선택합니다. CodeCatalyst는 연결된 보고서를 표시합니다. 이 보고서는 실행된 코드 적용 범위 테스트를 보여주며, 80%와 같이 테스트에서 성공적으로 검증한 코드 줄의 비율을 나타냅니다.

테스트 보고서에 대한 자세한 내용은 [워크플로를 사용한 테스트](#)의 내용을 참조하세요.

Tip

탐색 창에서 보고서를 선택하여 테스트 보고서를 볼 수도 있습니다.

7. CodeCatalyst 콘솔을 열어 두고 다음 절차로 이동합니다.

최신 애플리케이션이 성공적으로 배포되었는지 확인하려면

1. ApplicationDeploymentPipeline 워크플로로 돌아가서 최신 실행의 Run-**string** 연결을 선택합니다.
2. 워크플로 다이어그램에서 DeployFrontend 작업을 찾아 앱 보기 연결을 선택합니다. Mysfit 웹 사이트가 나타납니다.

 Note

DeployFrontend 작업 내에 앱 보기 연결이 보이지 않는 경우 실행 ID 연결을 선택했는지 확인하세요.

3. Twilight Glitter라는 pegasus Mysfit을 검색합니다. 연령 값을 기록해 둡니다. 6입니다. 코드를 변경하여 기간을 업데이트합니다.

7단계: 다른 사람에게 변경 사항을 검토하도록 요청

이제 test 브랜치에 변경 사항이 있으므로 풀 요청을 생성하여 다른 사람에게 검토하도록 요청할 수 있습니다. 다음 단계를 수행하여 test 브랜치의 변경 사항을 main 브랜치로 병합하기 위한 풀 요청을 생성합니다.

풀 요청을 생성하려면

1. 프로젝트로 이동합니다.
2. 다음 중 하나를 수행합니다.
 - 탐색 창에서 코드를 선택하고 풀 요청을 선택한 다음 풀 요청 생성을 선택합니다.
 - 리포지토리 홈 페이지에서 추가를 선택한 다음 풀 요청 생성을 선택합니다.
 - 프로젝트 페이지에서 풀 요청 생성을 선택합니다.
3. 소스 리포지토리에서 지정된 소스 리포지토리가 커밋된 코드가 포함된 리포지토리인지 확인합니다. 이 옵션은 리포지토리의 메인 페이지에서 풀 요청을 생성하지 않은 경우에만 나타납니다.
4. 대상 브랜치에서 코드를 검토한 후 병합할 브랜치를 선택합니다.
5. 소스 브랜치에서 커밋된 코드가 포함된 브랜치를 선택합니다.
6. 풀 요청 제목에 다른 사용자가 검토해야 할 사항과 이유를 이해하는 데 도움이 되는 제목을 입력합니다.
7. (선택 사항) 풀 요청 설명에서 문제에 대한 연결 또는 변경 사항에 대한 설명과 같은 정보를 제공합니다.

i Tip

CodeCatalyst가 풀 요청에 포함된 변경 사항에 대한 설명을 자동으로 생성하도록 설명 쓰기를 선택할 수 있습니다. 자동으로 생성된 설명은 풀 요청에 추가한 후에 변경할 수 있습니다.

이 기능을 사용하려면 스페이스에 생성형 AI 기능을 활성화해야 하며 연결된 리포지토리의 풀 요청에 사용할 수 없습니다. 자세한 내용은 [생성형 AI 기능 관리](#)를 참조하세요.

8. (선택 사항) 문제에서 문제 연결을 선택한 다음 목록에서 문제를 선택하거나 해당 ID를 입력합니다. 문제를 연결 해제하려면 연결 해제 아이콘을 선택합니다.
9. (선택 사항) 필수 검토자에서 필수 검토자 추가를 선택합니다. 프로젝트 멤버 목록에서 추가할 멤버를 선택합니다. 필수 검토자는 풀 요청을 대상 브랜치에 병합하기 전에 변경 사항을 승인해야 합니다.

i Note

검토자를 추가할 때 필수 검토자이면서 동시에 선택적 검토자로 추가할 수 없습니다. 자신을 검토자로 추가할 수 없습니다.

10. (선택 사항) 선택적 검토자에서 선택적 검토자 추가를 선택합니다. 프로젝트 멤버 목록에서 추가할 멤버를 선택합니다. 선택적 검토자는 풀 요청을 대상 브랜치에 병합하기 전에 변경 사항을 요구 사항으로 승인할 필요가 없습니다.
11. 브랜치 간의 차이점을 검토합니다. 풀 요청에 표시되는 차이점은 소스 브랜치의 수정본과 풀 요청이 생성된 시점의 대상 브랜치의 헤드 커밋인 병합 기반 간의 변경 사항입니다. 변경 사항이 표시되지 않으면 브랜치가 동일하거나 소스와 대상 모두에 대해 동일한 브랜치를 선택했을 수 있습니다.
12. 풀 요청에 검토하려는 코드와 변경 사항이 포함되어 있다고 판단되면 생성을 선택합니다.

i Note

풀 요청을 생성한 후 설명을 추가할 수 있습니다. 풀 요청 또는 파일의 개별 줄 및 풀 요청 전반에 설명을 추가할 수 있습니다. @ 기호와 파일 이름을 사용하여 파일과 같은 리소스에 대한 연결을 추가할 수 있습니다.

풀 요청을 생성하면 OnPullRequest 워크플로가 test 브랜치의 소스 파일을 사용하기 시작합니다. 검토자가 코드 변경을 승인하는 동안 워크플로를 선택하고 테스트 출력을 확인하여 결과를 관찰할 수 있습니다.

변경 사항을 검토한 후 코드를 병합할 수 있습니다. 코드를 기본 브랜치에 병합하면 변경 사항을 빌드하고 배포하는 워크플로가 자동으로 시작됩니다.

CodeCatalyst 콘솔에서 풀 요청을 병합하려면

1. 최신 애플리케이션 프로젝트로 이동합니다.
2. 프로젝트 페이지의 풀 요청 열기에서 병합할 풀 요청을 선택합니다. 풀 요청이 표시되지 않으면 모두 보기를 선택한 다음 목록에서 선택합니다. 병합을 선택합니다.
3. 풀 요청에서 사용 가능한 병합 전략 중에서 선택합니다. 선택적으로 풀 요청을 병합한 후 소스 브랜치를 삭제하는 옵션을 선택하거나 선택 취소한 다음 병합을 선택합니다.

Note

병합 버튼이 활성화 상태가 아니거나 병합할 수 없음 레이블이 표시되면 하나 이상의 필수 검토자가 아직 풀 요청을 승인하지 않았거나 CodeCatalyst 콘솔에서 풀 요청을 병합할 수 없는 것입니다. 풀 요청을 승인하지 않은 검토자는 풀 요청 세부 정보 영역의 개요에 시계 아이콘으로 표시됩니다. 필요한 모든 검토자가 풀 요청을 승인했지만 병합 버튼이 여전히 활성화되어 있지 않은 경우 병합 충돌이 발생할 수 있습니다. CodeCatalyst 콘솔에서 대상 브랜치의 병합 충돌을 해결한 다음 풀 요청을 병합하거나 충돌을 해결하고 로컬에서 병합한 다음 병합이 포함된 커밋을 CodeCatalyst에 푸시할 수 있습니다. 자세한 내용은 [풀 요청 병합\(Git\)](#) 섹션 및 Git 설명서를 참조하세요.

test 브랜치의 변경 사항을 브랜치로 병합하면 **main** 변경 사항이 변경 사항을 빌드하고 배포하는 ApplicationDeploymentPipeline 워크플로를 자동으로 시작합니다.

ApplicationDeploymentPipeline 워크플로를 통해 병합된 커밋 실행을 보려면

1. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
2. 워크플로의 ApplicationDeploymentPipeline에서 최근 실행을 확장합니다. 병합 커밋으로 시작된 워크플로 실행을 볼 수 있습니다. 선택적으로 실행 진행 상황을 보려면 선택합니다.
3. 실행이 완료되면 이전에 방문한 URL을 다시 로드합니다. 페가수스를 보고 연령이 변경되었는지 확인합니다.



8단계: 문제 종료

문제가 해결되면 CodeCatalyst 콘솔에서 종료할 수 있습니다.

프로젝트의 문제를 종료하려면

1. 프로젝트로 이동합니다.
2. 탐색 창에서 문제를 선택합니다.
3. 문제를 완료 열로 끌어다 놓습니다.

자세한 내용은 [CodeCatalyst의 문제 처리 추적 및 구성](#) 섹션을 참조하세요.

리소스 정리

CodeCatalyst 및 AWS에서 정리하여 환경에서 이 자습서의 트레이스를 제거합니다.

이 자습서에 사용한 프로젝트를 계속 사용하도록 선택하거나 프로젝트 및 관련 리소스를 삭제할 수 있습니다.

Note

이 프로젝트를 삭제하면 모든 멤버에 대한 프로젝트의 모든 리포지토리, 문제 및 아티팩트가 삭제됩니다.

프로젝트를 삭제하려면 다음과 같이 하세요.

1. 프로젝트로 이동한 다음 프로젝트 설정을 선택합니다.
2. 일반 탭을 선택합니다.
3. 프로젝트 이름에서 프로젝트 삭제를 선택합니다.

CloudFormation 및 Amazon S3에서 리소스를 삭제하려면

1. CodeCatalyst 스페이스에 추가한 것과 동일한 계정으로 AWS Management Console에 로그인합니다.
2. CloudFormation 서비스로 이동합니다.
3. mysfits**string** 스택을 삭제합니다.
4. development-mysfits**string** 스택을 삭제합니다.
5. CDKToolkit 스택을 선택합니다(삭제하지 않음). 리소스 탭을 선택합니다. StagingBucket 연결을 선택하고 Amazon S3에서 버킷 및 버킷 콘텐츠를 삭제합니다.

Note

이 버킷을 수동으로 삭제하지 않으면 최신 3계층 웹 애플리케이션 블루프린트를 다시 실행할 때 오류가 발생할 수 있습니다.

6. (선택 사항) CDKToolkit 스택을 삭제합니다.

레퍼런스

최신 3계층 웹 애플리케이션 블루프린트는 리소스를 CodeCatalyst 스페이스 및 AWS 클라우드의 AWS 계정에 배포합니다. 이러한 리소스는 다음과 같습니다.

- CodeCatalyst 스페이스에서:
 - 다음 리소스를 포함하는 CodeCatalyst 프로젝트:

- [소스 리포지토리](#) - 이 리포지토리에는 'Mysfits' 웹 애플리케이션의 샘플 코드가 포함되어 있습니다.
- [워크플로](#) - 이 워크플로는 기본 브랜치를 변경할 때마다 Mysfits 애플리케이션 코드를 빌드하고 배포합니다.
- [문제 보드](#) 및 백로그 - 이 보드 및 백로그를 사용하여 작업을 계획하고 추적할 수 있습니다.
- [테스트 보고서 제품군](#) - 이 제품군에는 샘플 코드에 포함된 자동 보고서가 포함됩니다.
- 연결된 AWS 계정에서:
 - CDKToolkit 스택 - 이 스택은 다음 리소스를 배포합니다.
 - Amazon S3 스테이징 버킷, 버킷 정책 및 버킷을 암호화하는 데 사용되는 AWS KMS 키입니다.
 - 배포 작업에 대한 IAM 배포 역할입니다.
 - 스택의 리소스를 지원하는 AWS IAM 역할 및 정책입니다.

 Note

CDKToolkit는 각 배포에 대해 분해되거나 다시 생성되지 않습니다. AWS CDK 지원을 위해 각 계정에서 시작되는 스택입니다.

- development-mysfits**string** BackEnd 스택 - 이 스택은 다음과 같은 백엔드 리소스를 배포합니다.
 - Amazon API Gateway 엔드포인트입니다.
 - 스택의 리소스를 지원하는 AWS IAM 역할 및 정책입니다.
 - AWS Lambda 함수와 계층은 최신 애플리케이션을 위한 서버리스 컴퓨팅 플랫폼을 제공합니다.
 - 버킷 배포 및 Lambda 함수에 대한 IAM 정책 및 역할입니다.
- mysfits**string** 스택 - 이 스택은 AWS Amplify 프론트엔드 애플리케이션을 배포합니다.

다음 사항도 참조하세요.

이 자습서의 일부로 리소스가 생성되는 AWS 서비스에 대한 자세한 내용은 다음을 참조하세요.

- Amazon S3 - 업계 최고의 확장성, 데이터 고가용성, 보안 및 성능을 제공하는 객체 스토리지 서비스에 프론트엔드 자산을 저장하는 서비스입니다. 자세한 내용은 [Amazon S3 사용 설명서](#)를 참조하세요.

- Amazon API Gateway - 규모에 관계없이 REST, HTTP 및 WebSocket API를 생성, 게시, 유지 관리, 모니터링 및 보호하기 위한 서비스입니다. 자세한 내용은 [API Gateway 개발자 가이드](#)를 참조하세요.
- Amplify - 프런트엔드 애플리케이션을 호스팅하기 위한 서비스입니다. 자세한 내용은 [AWS Amplify 호스팅 사용 설명서](#)를 참조하세요.
- AWS Cloud Development Kit (AWS CDK) - 코드에서 클라우드 인프라를 정의하고 AWS CloudFormation을 통해 프로비저닝하기 위한 프레임워크입니다. AWS CDK에는 AWS CDK 앱 및 스택과 상호 작용하기 위한 명령줄 도구인 AWS CDK Toolkit이 포함되어 있습니다. 자세한 정보는 [AWS Cloud Development Kit \(AWS CDK\) 개발자 안내서](#)를 참조하세요.
- Amazon DynamoDB - 데이터 저장을 위한 완전 관리형 NoSQL 데이터베이스 서비스입니다. 자세한 내용은 [Amazon DynamoDB 개발자 안내서](#)를 참조하세요.
- AWS Lambda - 서버를 프로비저닝하거나 관리하지 않고 고가용성 컴퓨팅 인프라에서 코드를 간접 호출하는 서비스입니다. 자세한 정보는 [AWS Lambda 개발자 안내서](#)를 참조하세요.
- AWS IAM - AWS 및 리소스에 대한 액세스를 안전하게 제어하기 위한 서비스입니다. 자세한 내용은 [IAM 사용 설명서](#)를 참조하세요.

자습서: 빈 프로젝트로 시작하고 수동으로 리소스 추가

프로젝트를 생성할 때 빈 프로젝트 블루프린트를 선택하여 사전 정의된 리소스 없이 빈 프로젝트를 생성할 수 있습니다. 빈 프로젝트를 생성한 후 프로젝트 요구 사항에 따라 리소스를 생성하고 추가할 수 있습니다. 블루프린트 없이 생성된 프로젝트는 생성 시 비어 있으므로 이 옵션을 시작하려면 CodeCatalyst 리소스를 생성하고 구성하는 방법에 대한 더 많은 지식이 필요합니다.

주제

- [사전 조건](#)
- [빈 프로젝트 생성](#)
- [소스 리포지토리 생성](#)
- [코드 변경을 빌드, 테스트 및 배포하기 위한 워크플로 생성](#)
- [프로젝트에 초대하기](#)
- [작업을 공동 작업하고 추적하기 위한 문제 생성](#)

사전 조건

빈 프로젝트를 생성하려면 스페이스 관리자 또는 파워 유저 역할이 할당되어 있어야 합니다. CodeCatalyst 처음 로그인하는 경우 [CodeCatalyst 설정 및 로그인](#) 섹션을 참조하세요.

빈 프로젝트 생성

프로젝트 생성은 함께 작업할 수 있는 첫 번째 단계입니다. 소스 리포지토리 및 워크플로와 같은 자체 리소스를 생성하려면 빈 프로젝트로 시작할 수 있습니다.

빈 프로젝트를 만들려면

1. 프로젝트를 생성하려는 스페이스로 이동합니다.
2. 스페이스 대시보드에서 프로젝트 생성을 선택합니다.
3. 처음부터 시작을 선택합니다.
4. 프로젝트에 이름 부여에서 프로젝트에 할당할 이름을 입력합니다. 이름은 스페이스 내에서 고유해야 합니다.
5. 프로젝트 생성을 선택합니다.

이제 프로젝트가 비어 있으므로 다음 단계는 소스 리포지토리를 생성하는 것입니다.

소스 리포지토리 생성

프로젝트 코드를 저장하고 공동 작업할 소스 리포지토리를 생성합니다. 프로젝트 멤버는 이 리포지토리를 로컬 컴퓨터에 복제하여 코드 작업을 수행할 수 있습니다. 또는 지원되는 서비스에서 호스팅되는 리포지토리를 연결하도록 선택할 수 있지만 이 자습서에서는 다루지 않습니다. 자세한 내용은 [소스 리포지토리 연결](#) 섹션을 참조하세요.

소스 리포지토리를 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트로 이동합니다.
3. 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
4. 리포지토리 추가를 선택하고 리포지토리 생성을 선택합니다.
5. 리포지토리 이름에 리포지토리 이름을 제공합니다. 이 안내서에서는 *codecatalyst-source-repository*를 사용하지만 다른 이름을 선택할 수 있습니다. 리포지토리 이름은 프로젝트에서

고유해야 합니다. 리포지토리 이름 요구 사항에 대한 자세한 내용은 [CodeCatalyst의 소스 리포지토리 할당량](#) 섹션을 참조하세요.

6. (선택 사항) 설명에 리포지토리에 대한 설명을 추가하여 프로젝트의 다른 사용자가 리포지토리의 용도를 이해하는 데 도움이 되도록 합니다.
7. 리포지토리 생성(기본)을 선택합니다. 이 옵션은 기본 브랜치와 README.md 파일을 포함하는 리포지토리를 생성합니다. 빈 리포지토리보다 달리 이 리포지토리가 생성되는 즉시 사용할 수 있습니다.
8. 기본 브랜치에서 다른 이름을 선택할 이유가 없는 한 이름을 main으로 둡니다. 이 가이드의 예시는 모두 기본 브랜치의 이름으로 main을 사용합니다.
9. (선택 사항) 푸시하려는 코드 유형에 맞는 .gitignore 파일을 추가합니다.
10. 생성(Create)을 선택합니다.

Note

CodeCatalyst는 README.md 파일을 만들 때 리포지토리에 파일을 추가합니다. 또한 CodeCatalyst는 main이라는 기본 브랜치에 리포지토리에 대한 초기 커밋을 생성합니다. README.md 파일을 편집하거나 삭제할 수 있지만 기본 브랜치를 삭제할 수는 없습니다.

개발 환경을 생성하여 리포지토리에 코드를 빠르게 추가할 수 있습니다. 이 자습서에서는 AWS Cloud9을 사용하여 개발 환경을 생성하고 개발 환경을 생성할 때 메인 브랜치에서 브랜치를 생성하는 옵션을 선택하는 것이 좋습니다. 이 브랜치에 **test** 이름을 사용하지만 원하는 경우 다른 브랜치 이름을 입력할 수 있습니다.

새 브랜치로 개발 환경을 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 개발 환경을 생성하려는 프로젝트로 이동합니다.
3. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다. 또는 탐색 창에서 코드를 선택하고 소스 리포지토리를 선택한 다음 개발 환경을 생성할 리포지토리를 선택합니다.
4. 리포지토리 홈 페이지에서 개발 환경 생성을 선택합니다.
5. 드롭다운 메뉴에서 지원되는 IDE를 선택합니다. 자세한 정보는 [개발 환경에 지원되는 통합 개발 환경](#)을 참조하세요.
6. 복제할 리포지토리를 선택하고, 새 브랜치에서 작업을 선택하고, 브랜치 이름 필드에 브랜치 이름을 입력하고, 다음에서 브랜치 생성 드롭다운 메뉴에서 새 브랜치를 만들 브랜치를 선택합니다.

- 원하는 경우 개발 환경의 별칭을 추가할 수 있습니다.
- 선택적으로 개발 환경 구성 편집 버튼을 선택하여 개발 환경의 컴퓨팅, 스토리지 또는 제한 시간 구성을 편집합니다.
- 생성(Create)을 선택합니다. 개발 환경이 생성되는 동안 개발 환경 상태 옆에 시작 중이 표시되고, 개발 환경이 생성되면 상태 옆에 실행 중이 표시됩니다. 선택한 IDE의 개발 환경과 함께 새 탭이 열립니다. 코드를 편집하고 변경 사항을 커밋하고 푸시할 수 있습니다.

코드 변경을 빌드, 테스트 및 배포하기 위한 워크플로 생성

CodeCatalyst에서는 워크플로에서 애플리케이션 또는 서비스의 구축, 테스트 및 배포를 구성합니다. 워크플로는 작업으로 구성되며 코드 푸시 또는 풀 요청 열기 또는 업데이트와 같이 지정된 소스 리포지토리 이벤트가 발생한 후 자동으로 실행되도록 구성할 수 있습니다. 워크플로에 대한 자세한 내용은 [워크플로를 사용하여 빌드, 테스트 및 배포](#) 섹션을 참조하세요.

[워크플로 시작하기](#)의 지침에 따라 첫 번째 워크플로를 생성합니다.

프로젝트에 초대하기

이제 사용자 지정 프로젝트를 설정했으므로 다른 사용자가 함께 작업하도록 초대합니다.

프로젝트에 누군가를 초대하려면

- 사용자를 초대하려는 프로젝트로 이동합니다.
- 탐색 창에서 프로젝트 설정을 선택합니다.
- 멤버 탭에서 초대를 선택합니다.
- 프로젝트의 사용자로 초대하려는 사람의 이메일 주소를 입력합니다. 여러 이메일 주소를 스페이스 또는 쉼표로 구분하여 입력할 수 있습니다. 프로젝트의 멤버가 아닌 스페이스의 멤버 중에서 선택할 수도 있습니다.
- 사용자의 역할을 선택합니다.

사용자 추가를 완료했으면 초대를 선택합니다.

작업을 공동 작업하고 추적하기 위한 문제 생성

CodeCatalyst는 프로젝트와 관련된 기능, 작업, 버그 및 기타 작업과 문제를 추적하는 데 도움이 됩니다. 문제를 생성하여 필요한 작업과 아이디어를 추적할 수 있습니다. 기본적으로 문제를 생성하면 백로

그에 추가됩니다. 진행 중인 작업을 추적하는 보드로 문제를 이동할 수 있습니다. 특정 프로젝트 멤버에게 문제를 할당할 수도 있습니다.

프로젝트에 대한 문제를 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.

문제를 생성하려는 프로젝트를 탐색하고 있는지 확인합니다. 모든 프로젝트를 보려면 탐색 창에서 Amazon CodeCatalyst를 선택하고 필요한 경우 모든 프로젝트 보기를 선택합니다. 문제를 생성하거나 처리할 프로젝트를 선택합니다.

2. 탐색 창에서 트랙을 선택한 후, 백로그를 선택합니다.
3. 문제 생성을 선택합니다.
4. 문제 제목에 문제의 이름을 입력합니다. 필요에 따라 문제에 대한 설명을 제공합니다. 원하는 경우 문제의 상태, 우선 순위 및 추정치를 선택합니다. 프로젝트 멤버 목록에서 프로젝트 멤버에게 문제를 할당할 수도 있습니다.

 Tip

Amazon Q가 문제를 해결하도록 Amazon Q에 문제를 할당할 수 있습니다. 시도가 성공하면 풀 요청이 생성되고 코드를 검토하고 테스트할 수 있도록 문제의 상태가 검토 중으로 변경됩니다. 자세한 내용은 [자습서: CodeCatalyst 생성형 AI 기능을 사용하여 개발 작업 속도 향상](#) 섹션을 참조하세요.

이 기능을 사용하려면 스페이스에 생성형 AI 기능을 활성화해야 합니다. 자세한 내용은 [생성형 AI 기능 관리](#)를 참조하세요.

5. 저장을 선택합니다.

문제를 생성한 후 프로젝트 멤버에게 할당하고, 예측하고, Kanban 보드에서 추적할 수 있습니다. 자세한 내용은 [CodeCatalyst의 문제 처리 추적 및 구성](#) 섹션을 참조하세요.

자습서: CodeCatalyst 생성형 AI 기능을 사용하여 개발 작업 속도 향상

생성형 AI 기능이 활성화된 스페이스에 Amazon CodeCatalyst에 프로젝트와 소스 리포지토리가 있는 경우 이러한 기능을 사용하여 소프트웨어 개발 속도를 높일 수 있습니다. 개발자는 이를 달성하기 위해 시간보다 더 많은 작업을 수행해야 하는 경우가 많습니다. 그들은 종종 이러한 변경 사항에 대한 검

토를 위한 풀 요청을 생성할 때 팀 동료에게 코드 변경 사항을 설명하는 데 시간을 할애하지 않으며, 다른 사용자가 자체 설명으로 변경 사항을 찾을 것으로 기대합니다. 풀 요청 생성자 및 검토자도 풀 요청에 대한 모든 설명을 찾고 읽을 시간이 없습니다. 특히 풀 요청에 여러 개정이 있는 경우 더욱 그렇습니다. CodeCatalyst는 소프트웨어 개발용 Amazon Q Developer Agent와 통합되어 팀원이 작업을 더 빠르게 수행하고 작업의 가장 중요한 부분에 집중해야 하는 시간을 늘릴 수 있는 생성형 AI 기능을 제공합니다.

Amazon Q Developer는 AWS 애플리케이션을 이해, 구축, 확장 및 운영하는 데 도움이 되는 생성형 AI 기반 대화형 어시스턴트입니다. 빌드 속도를 높이기 위해 Amazon Q AWS를 지원하는 모델을 고품질 AWS 콘텐츠로 보강하여 더 완전하고 실행 가능하며 참조된 답변을 생성합니다. 자세한 내용은 Amazon Q Developer 사용 설명서의 [Amazon Q Developer란 무엇인가요?](#)를 참조하세요.

Note

Amazon Bedrock 기반: [자동화된 침해 탐지](#) AWS 를 구현합니다. 소프트웨어 개발용 Amazon Q Developer Agent의 설명 쓰기, 콘텐츠 요약 생성, 권장 작업, Amazon Q를 사용하여 프로젝트에 기능을 생성하거나 추가, Amazon Q에 문제 할당 기능은 Amazon Bedrock에 구축되므로, 사용자는 Amazon Bedrock에서 구현된 제어 기능을 최대한 활용하여 안전, 보안 및 인공지능(AI)을 책임 있게 사용할 수 있습니다.

이 자습서에서는 CodeCatalyst의 생성형 AI 기능을 사용하여 블루프린트가 있는 프로젝트를 생성하고 기존 프로젝트에 블루프린트를 추가하는 방법을 알아봅니다. 또한 풀 요청을 생성할 때 브랜치 간의 변경 사항을 요약하고 풀 요청에 남아 있는 설명을 요약하는 방법을 알아봅니다. 또한 코드 변경 또는 개선에 대한 아이디어와 관련된 문제를 생성하고 Amazon Q에 할당하는 방법도 알아봅니다. Amazon Q에 할당된 문제 작업의 일환으로 Amazon Q가 작업을 제안하도록 허용하는 방법과 문제 작업의 일부로 생성된 모든 작업을 할당하고 작업하는 방법을 알아봅니다.

주제

- [사전 조건](#)
- [Amazon Q를 사용하여 프로젝트를 생성하거나 기능을 추가할 때 블루프린트 선택](#)
- [풀 요청을 생성할 때 브랜치 간의 코드 변경 사항 요약 생성](#)
- [풀 요청의 코드 변경에 대해 남긴 주석 요약 생성](#)
- [문제를 생성하고 Amazon Q에 할당](#)
- [문제를 생성하고 Amazon Q에서 해당 이슈에 대한 작업 추천받기](#)
- [리소스 정리](#)

사전 조건

이 자습서에서 CodeCatalyst 기능을 사용하려면 먼저 다음 리소스를 완료하고 액세스할 수 있어야 합니다.

- CodeCatalyst에 로그인하기 위한 AWS Builder ID 또는 Single Sign-On(SSO) 자격 증명이 있습니다.
- 생성형 AI 기능이 활성화된 스페이스에 있습니다. 자세한 내용은 [생성형 AI 기능 관리](#)를 참조하세요.
- 해당 스페이스의 프로젝트에서 기고자 또는 프로젝트 관리자 역할이 있습니다.
- 생성형 AI로 프로젝트를 생성하지 않는 한 기존 프로젝트에는 해당 프로젝트에 대해 구성된 소스 리포지토리가 하나 이상 있습니다. 연결된 리포지토리는 지원되지 않습니다.
- 생성형 AI에서 생성된 초기 솔루션을 갖도록 문제를 할당할 때는 Jira 소프트웨어 확장 프로그램을 사용하여 프로젝트를 구성할 수 없습니다. 이 기능에는 확장 프로그램이 지원되지 않습니다.

자세한 내용은 [스페이스 생성](#), [CodeCatalyst의 문제 처리 추적 및 구성](#), [CodeCatalyst에서 확장 프로그램이 있는 프로젝트에 기능 추가 및 사용자 역할로 액세스 권한 부여](#) 부분을 참조하세요.

이 자습서는 Python을 사용한 최신 3계층 웹 애플리케이션 블루프린트를 사용하여 생성된 프로젝트를 기반으로 합니다. 다른 블루프린트로 생성된 프로젝트를 사용하는 경우에도 단계를 따를 수 있지만 샘플 코드 및 언어와 같은 일부 세부 사항은 달라집니다.

Amazon Q를 사용하여 프로젝트를 생성하거나 기능을 추가할 때 블루프린트 선택

프로젝트 개발자는 새 프로젝트를 생성하거나 기존 프로젝트에 구성 요소를 추가할 때 생성형 AI 어시스턴트인 Amazon Q와 협업할 수 있습니다. 채팅과 유사한 인터페이스에서 Amazon Q와 상호 작용하여 프로젝트의 요구 사항을 Amazon Q에 제공할 수 있습니다. 요구 사항에 따라 Amazon Q는 블루프린트를 제안하고 충족할 수 없는 요구 사항을 간략하게 설명합니다. 스페이스에 사용자 지정 블루프린트가 있는 경우 Amazon Q는 이러한 블루프린트를 학습하고 권장 사항에도 포함합니다. 그런 다음 만족하면 Amazon Q의 제안을 진행할 수 있으며, 요구 사항에 맞는 코드가 포함된 소스 리포지토리와 같은 필요한 리소스를 생성합니다. 또한 Amazon Q는 블루프린트로 만족할 수 없는 요구 사항에 대한 문제를 생성합니다. 사용 가능한 CodeCatalyst 블루프린트에 대한 자세한 내용은 [CodeCatalyst 블루프린트를 사용하여 포괄적인 프로젝트 생성](#) 섹션을 참조하세요. Amazon Q를 블루프린트와 함께 사용하는 방법에 대한 자세한 내용은 [Amazon Q를 사용하여 프로젝트를 생성하거나 블루프린트를 사용하여 기능을 추가할 때의 모범 사례](#) 섹션을 참조하세요.

Amazon Q로 프로젝트를 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 블루프린트를 생성할 스페이스로 이동합니다.
3. 스페이스 대시보드에서 Amazon Q로 생성을 선택합니다.
4. Amazon Q 프롬프트 텍스트 입력 필드에서 빌드하려는 프로젝트에 대한 간략한 설명을 작성하여 지침을 제공합니다. 예시: "I want to create a project in Python that has a presentation layer responsible for how the data is presented, an application layer that contains the core logic and functionality of the application, and a data layer that manages the storage and retrieval of the data."

(선택 사항) 연습 예시에서 블루프린트를 선택하여 미리 작성된 프롬프트를 사용할 수 있습니다. 예를 들어 React 앱을 선택하면 다음 프롬프트가 제공됩니다. "I want to create a project in Python that has a presentation layer responsible for how the data is presented, an application layer that contains the core logic and functionality of the application, and a data layer that manages the storage and retrieval of the data. I also want to add authentication and authorization mechanisms for security and allowable actions."

5. 전송을 선택하여 Amazon Q에 지침을 제출합니다. 생성형 AI 어시스턴트는 블루프린트에서 충족할 수 없는 요구 사항을 제안하고 간략하게 설명합니다. 예를 들어 Amazon Q는 사용자의 기준에 따라 다음을 제안할 수 있습니다.

I recommend using the Modern three-tier web application blueprint based on your requirements. Blueprints are dynamic and can always be updated and edited later.

Modern three-tier web application

By Amazon Web Services

This blueprint creates a Mythical Mysfits 3-tier web application with a modular presentation, application, and data layers.

The application leverages containers, infrastructure as code (IaC), continuous integration and continuous delivery (CI/CD), and serverless code functions.

Version: 0.1.163

[View details](#)

The following requirements could not be met so I will create issues for you.

- Add authentication and authorization mechanisms for security and allowable actions.

- (선택 사항) 제안된 블루프린트의 세부 정보를 보려면 세부 정보 보기를 선택합니다.
- 다음 중 하나를 수행하세요.
 - 제안에 만족하는 경우 예, 이 블루프린트를 사용합니다를 선택합니다.
 - 프롬프트를 수정하려면 프롬프트 편집을 선택합니다.
 - 프롬프트를 완전히 지우려면 다시 시작을 선택합니다.
- 다음 중 하나를 수행하세요.
 - 제안된 블루프린트를 구성하려면 구성을 선택합니다. 나중에 블루프린트를 구성할 수도 있습니다.
 - 현재 블루프린트 구성을 수정하지 않으려면 건너뛰기를 선택합니다.
- 블루프린트를 구성하기로 선택한 경우 프로젝트 리소스를 수정한 후 계속을 선택합니다.
- 메시지가 표시되면 프로젝트에 할당할 이름과 관련 리소스 이름을 입력합니다. 이름은 스페이스 내에서 고유해야 합니다.
- 프로젝트 생성을 선택하여 블루프린트로 프로젝트를 생성합니다. Amazon Q는 블루프린트를 사용하여 리소스를 생성합니다. 예를 들어 단일 페이지 애플리케이션 블루프린트로 프로젝트를 생성하면 CI/CD에 대한 관련 코드 및 워크플로에 대한 소스 리포지토리가 생성됩니다.
- (선택 사항) 기본적으로 Amazon Q는 블루프린트에 부합하지 않는 요구 사항에 대한 문제도 생성합니다. 문제를 생성하지 않으려는 항목을 선택할 수 있습니다. Amazon Q에서 문제를 생성하도록 선택한 후 Amazon Q에도 문제를 할당할 수 있습니다. 지정된 소스 리포지토리의 컨텍스트에서 문제를 분석하여 관련 소스 파일 및 코드의 요약を提供합니다. 자세한 내용은 [문제 찾기 및 보기](#), [문제를 생성하고 Amazon Q에 할당](#), [Amazon Q에 할당된 문제를 생성하고 작업할 때의 모범 사례](#) 섹션을 참조하세요.

Amazon Q를 사용하여 프로젝트를 생성한 후 마찬가지로 Amazon Q를 사용하여 새 구성 요소를 추가할 수 있습니다. Amazon Q는 요구 사항에 따라 CodeCatalyst 블루프린트를 제안합니다.

Amazon Q에 블루프린트를 추가하려면

- <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
- CodeCatalyst 콘솔에서 블루프린트를 추가하려는 프로젝트로 이동합니다.

3. Amazon Q로 추가를 선택합니다.
4. Amazon Q 프롬프트 텍스트 입력 필드에서 빌드하려는 프로젝트에 대한 간략한 설명을 작성하여 지침을 제공합니다. 예시: “I want to create a project in Python that has a presentation layer responsible for how the data is presented, an application layer that contains the core logic and functionality of the application, and a data layer that manages the storage and retrieval of the data.”

(선택 사항) 연습 예시에서 블루프린트를 선택하여 미리 작성된 프롬프트를 사용할 수 있습니다. 예를 들어 React 앱을 선택하면 다음 프롬프트가 제공됩니다. “I want to create a project in Python that has a presentation layer responsible for how the data is presented, an application layer that contains the core logic and functionality of the application, and a data layer that manages the storage and retrieval of the data. I also want to add authentication and authorization mechanisms for security and allowable actions.”

5. 전송을 선택하여 Amazon Q에 지침을 제출합니다. 생성형 AI 어시스턴트는 블루프린트에서 충족할 수 없는 요구 사항을 제안하고 간략하게 설명합니다. 예를 들어 Amazon Q는 사용자의 기준에 따라 다음을 제안할 수 있습니다.

I recommend using the Single-page application blueprint based on your requirements. Blueprints are dynamic and can always be updated and edited later.

Single-page application

By Amazon Web Services

This blueprint creates a SPA (single-page application) using React, Vue, or Angular frameworks and deploys to AWS Amplify Hosting.

Version: 0.2.15

[View details](#)

The following requirements could not be met so I will create issues for you.

- The application should have reusable UI components
- The application should support for client-side routing
- The application may require server-side rendering for improved performance and SEO

6. (선택 사항) 제안된 블루프린트의 세부 정보를 보려면 세부 정보 보기를 선택합니다.
7. 다음 중 하나를 수행하세요.

- a. 제안에 만족하는 경우 예, 이 블루프린트를 사용합니다를 선택합니다.
 - b. 프롬프트를 수정하려면 프롬프트 편집을 선택합니다.
 - c. 프롬프트를 완전히 지우려면 다시 시작을 선택합니다.
8. 다음 중 하나를 수행하세요.
- a. 제안된 블루프린트를 구성하려면 구성을 선택합니다. 나중에 블루프린트를 구성할 수도 있습니다.
 - b. 현재 블루프린트 구성을 수정하지 않으려면 건너뛰기를 선택합니다.
9. 블루프린트를 구성하기로 선택한 경우 프로젝트 리소스를 수정한 후 계속을 선택합니다.
10. 프로젝트에 추가를 선택하여 블루프린트로 프로젝트에 리소스를 추가합니다. Amazon Q는 블루프린트를 사용하여 리소스를 생성합니다. 예를 들어 단일 페이지 애플리케이션 블루프린트를 사용하여 프로젝트에 추가하면 CI/CD에 대한 관련 코드 및 워크플로에 대한 소스 리포지토리가 생성됩니다.
11. (선택 사항) 기본적으로 Amazon Q는 블루프린트에 부합하지 않는 요구 사항에 대한 문제도 생성합니다. 문제를 생성하지 않으려는 항목을 선택할 수 있습니다. Amazon Q에서 문제를 생성하도록 선택한 후 Amazon Q에도 문제를 할당할 수 있습니다. 지정된 소스 리포지토리의 컨텍스트에서 문제를 분석하여 관련 소스 파일 및 코드의 요약を提供합니다. 자세한 내용은 [문제를 생성하고 Amazon Q에 할당](#) 및 [Amazon Q에 할당된 문제를 생성하고 작업할 때의 모범 사례](#) 섹션을 참조하세요.

풀 요청을 생성할 때 브랜치 간의 코드 변경 사항 요약 생성

풀 요청은 나와 프로젝트 멤버가 검토하고, 주석을 추가하며, 한 브랜치에서 다른 브랜치로 코드를 변경하기 위한 기본적인 방법입니다. 풀 요청을 사용하여 중요하지 않은 변경 내용이나 수정 사항, 중요 기능 추가 또는 릴리스된 소프트웨어의 새 버전에 대한 코드 변경 내용을 공동으로 검토할 수 있습니다. 풀 요청 설명의 일부로 코드 변경 사항과 변경의 의도를 요약하면 코드를 검토하는 다른 사용자에게 도움이 되며 시간 경과에 따른 코드 변경 사항 내역을 이해하는 데도 도움이 됩니다. 하지만 개발자는 검토자가 검토 중인 내용이나 코드 변경의 의도를 이해할 수 있도록 변경 사항을 충분히 자세히 설명하기보다는 코드의 자체 설명에 의존하거나 모호한 세부 정보를 제공하는 경우가 많습니다.

풀 요청을 생성할 때 나를 위해 설명 작성 기능을 사용하여 Amazon Q가 풀 요청에 포함된 변경 사항에 대한 설명을 생성하도록 할 수 있습니다. 이 옵션을 선택하면 Amazon Q는 코드 변경 사항이 포함된 소스 브랜치와 이러한 변경 사항을 병합할 대상 브랜치 간의 차이를 분석합니다. 그런 다음 이러한 변경 사항에 대한 요약과 이러한 변경의 의도 및 효과에 대한 최상의 해석을 생성합니다.

Note

이 기능은 Git 하위 모듈에서 작동하지 않습니다. 풀 요청의 일부인 Git 하위 모듈의 변경 사항은 요약되지 않습니다.
연결된 리포지토리의 풀 요청에는 이 기능을 사용할 수 없습니다.

생성한 풀 요청으로 이 기능을 시도할 수 있지만 이 자습서에서는 Python 기반 최신 3계층 웹 애플리케이션 블루프린트에서 생성된 프로젝트에 포함된 코드를 몇 가지 간단하게 변경하여 테스트합니다.

Tip

다른 블루프린트 또는 자체 코드로 생성된 프로젝트를 사용하는 경우에도 이 자습서를 따를 수 있지만 이 자습서의 예시는 프로젝트의 코드와 일치하지 않습니다. 아래 제안된 예 대신 브랜치에서 프로젝트 코드를 간단히 변경한 뒤 다음 단계에 표시된 대로 풀 요청을 생성하여 기능을 테스트합니다.

먼저 소스 리포지토리에 브랜치를 생성합니다. 그런 다음 콘솔의 텍스트 편집기를 사용하여 해당 브랜치의 파일을 빠르게 코드 변경합니다. 그런 다음 풀 요청을 생성하고 설명 쓰기 기능을 사용하여 변경 내용을 요약합니다.

브랜치를 생성하려면(콘솔)

1. CodeCatalyst 콘솔에서 소스 리포지토리가 있는 프로젝트로 이동합니다.
2. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
3. 브랜치를 생성하려는 리포지토리의 이름을 선택합니다.
4. 리포지토리의 개요 페이지에서 추가를 선택한 다음 브랜치 생성을 선택합니다.
5. 브랜치의 이름을 입력합니다.
6. 브랜치를 생성할 브랜치를 선택한 다음 생성을 선택합니다.

브랜치가 있으면 간단한 변경으로 해당 브랜치의 파일을 편집합니다. 이 예시에서는 `test_endpoint.py` 파일을 편집하여 테스트 재시도 횟수를 3에서 5로 변경합니다.

i Tip

개발 환경을 생성하거나 사용하여 이 코드를 변경할 수도 있습니다. 자세한 내용은 [개발 환경 생성](#) 섹션을 참조하세요.

콘솔에서 **test_endpoint.py** 파일을 찾으려면

1. **mysfits** 소스 리포지토리의 개요 페이지에서 브랜치 드롭다운을 선택하고 이전 절차에서 생성한 브랜치를 선택합니다.
2. 파일에서 편집하려는 파일로 이동합니다. 예를 들어 `test_endpoint.py` 파일을 편집하려면 테스트를 확장하고 통합을 확장한 다음 `test_endpoint.py`를 선택합니다.
3. 편집을 선택합니다.
4. 7번 줄에서 모든 테스트를 다시 시도할 횟수를 변경합니다. 변경 전:

```
def test_list_all(retry=3):
```

변경 후:

```
def test_list_all(retry=5):
```

5. 커밋을 선택하고 변경 사항을 브랜치에 커밋합니다.

이제 변경 사항이 있는 브랜치가 있으므로 풀 요청을 생성할 수 있습니다.

변경 사항 요약을 사용하여 풀 요청 생성

1. 리포지토리의 개요 페이지에서 추가를 선택한 다음 풀 요청 생성을 선택합니다.
2. 대상 브랜치에서 코드를 검토한 후 병합할 브랜치를 선택합니다.

i Tip

이 기능을 가장 간단하게 시연하려면 이전 절차에서 브랜치를 생성한 브랜치를 선택합니다. 예를 들어 리포지토리의 기본 브랜치에서 브랜치를 생성한 경우 풀 요청의 대상 브랜치로 해당 브랜치를 선택합니다.

3. 소스 브랜치에서 `test_endpoint.py` 파일에 방금 커밋한 변경 사항이 포함된 브랜치를 선택합니다.
4. 풀 요청 제목에 다른 사용자가 검토해야 할 사항과 이유를 이해하는 데 도움이 되는 제목을 입력합니다.
5. 풀 요청 설명에서 설명 쓰기를 선택하여 Amazon Q가 풀 요청에 포함된 변경 사항에 대한 설명을 생성하도록 합니다.
6. 변경 사항 요약이 나타납니다. 제안된 텍스트를 검토한 다음 수락을 선택하고 설명에 추가합니다.
7. 선택적으로 코드에 대한 변경 사항을 더 잘 반영하도록 요약을 수정합니다. 검토자를 추가하거나 이 풀 요청에 문제를 연결하도록 선택할 수도 있습니다. 원하는 추가 변경을 완료했으면 생성을 선택합니다.

풀 요청의 코드 변경에 대해 남긴 주석 요약 생성

사용자가 풀 요청을 검토할 때 해당 풀 요청의 변경 사항에 대해 여러 개의 주석을 남기는 경우가 많습니다. 많은 검토자의 의견이 많으면 피드백에서 일반적인 주제를 선택하거나 모든 개정에서 모든 의견을 검토했는지 확인하는 것이 어려울 수 있습니다. 설명 요약 생성 기능을 사용하여 Amazon Q가 풀 요청의 코드 변경에 남아 있는 모든 설명을 분석하고 해당 설명의 요약을 생성하도록 할 수 있습니다.

Note

주석 요약은 일시적입니다. 풀 요청을 새로 고치면 요약이 사라집니다. 콘텐츠 요약에는 전체 풀 요청에 대한 주석이 포함되지 않으며 풀 요청 개정의 코드 차이에 대한 주석만 남았습니다. 이 기능은 Git 하위 모듈의 코드 변경에 대한 주석이 남아 있는 경우 작동하지 않습니다. 연결된 리포지토리의 풀 요청에는 이 기능을 사용할 수 없습니다.

풀 요청에서 주석 요약을 생성하려면

1. 이전 절차에서 생성한 풀 요청으로 이동합니다.

Tip

원하는 경우 프로젝트에서 열려 있는 풀 요청을 사용할 수 있습니다. 탐색 모음에서 코드를 선택하고 풀 요청을 선택한 다음 열려 있는 풀 요청을 선택합니다.

2. 풀 요청에 아직 주석이 없는 경우 변경 사항의 풀 요청에 몇 가지 주석을 추가합니다.

3. 개요에서 의견 요약 생성을 선택합니다. 완료되면 주식 요약 섹션이 확장됩니다.
4. 풀 요청 개정의 코드 변경에 대해 남아 있는 의견 요약을 검토하고 풀 요청의 의견과 비교합니다.

문제를 생성하고 Amazon Q에 할당

개발 팀은 작업을 추적하고 관리하기 위해 문제를 발생시키지만, 누가 문제를 처리해야 하는지 명확하지 않거나, 문제에 코드 기반에서 특정 부분을 조사해야 하거나, 다른 긴급 작업에 먼저 참석해야 하기 때문에 문제가 지속되는 경우가 있습니다. CodeCatalyst에는 소프트웨어 개발용 Amazon Q Developer Agent와의 통합이 포함됩니다. 제목과 설명을 기반으로 문제를 분석할 수 있는 Amazon Q라는 생성형 AI 어시스턴트에 문제를 할당할 수 있습니다. Amazon Q에 문제를 할당하면 평가할 초안 솔루션을 생성하려고 시도합니다. 이렇게 하면 사용자와 팀이 주의가 필요한 문제에 집중하고 작업을 최적화하는 데 도움이 되며, Amazon Q는 즉시 해결할 리소스가 없는 문제에 대한 솔루션을 제공합니다.

Tip

Amazon Q는 간단하고 단순한 문제를 다룰 때 최상의 성능을 발휘합니다. 최상의 결과를 얻으려면 일반적인 언어를 사용하여 수행하려는 작업을 명확하게 설명하세요.

Amazon Q에 문제를 할당하면 CodeCatalyst는 Amazon Q가 문제를 어떻게 처리할지 확인할 때까지 해당 문제를 차단됨으로 표시합니다. 계속하려면 세 가지 질문에 답해야 합니다.

- 필요한 모든 단계를 확인할지 또는 피드백 없이 진행할지 여부. 각 단계를 확인하기로 선택하는 경우, Amazon Q에 생성하는 접근 방식에 대한 피드백을 회신하여 필요한 경우 해당 접근 방식을 반복할 수 있습니다. Amazon Q는 또한 이 옵션을 선택하면 사용자가 생성한 풀 요청에 대해 남기는 피드백을 검토할 수 있습니다. 각 단계를 확인하지 않기로 선택하면 Amazon Q가 작업을 더 빨리 완료할 수 있지만 문제 또는 해당 단계에서 생성한 풀 요청에서 제공한 피드백은 검토되지 않습니다.
- 작업의 일부로 워크플로 파일을 업데이트하도록 허용할지 여부입니다. 프로젝트에 풀 요청 이벤트에 대한 실행을 시작하도록 구성된 워크플로가 있을 수 있습니다. 그렇다면 Amazon Q가 생성했으며 워크플로 YAML 생성 또는 업데이트를 포함하고 있는 풀 요청은 해당 풀 요청에 포함된 워크플로의 실행을 시작할 수 있습니다. 프로젝트에서 생성한 풀 요청을 검토하고 승인하기 전에 이러한 워크플로를 자동으로 실행하는 워크플로가 없다는 사실이 확인되지 않았다면 Amazon Q가 워크플로 파일에서 작업하도록 허용하지 않는 것이 가장 좋습니다.
- 문제의 작업을 Amazon Q 자체를 포함하여 사용자에게 개별적으로 할당할 수 있는 더 작은 증분으로 세분화하는 작업을 생성하도록 허용할지 여부입니다. Amazon Q가 작업을 제안하고 생성하도록

허용하면 여러 사람이 문제의 개별 부분을 처리할 수 있도록 하여 복잡한 문제에 대한 개발을 가속화하는 데 도움이 될 수 있습니다. 또한 각 작업을 완료하는 데 필요한 작업이 속한 문제보다 더 간단하기 때문에 작업 전체를 이해하는 복잡성을 줄이는 데 도움이 될 수 있습니다.

- 작업할 소스 리포지토리입니다. 프로젝트에 여러 소스 리포지토리가 있더라도 Amazon Q는 하나의 소스 리포지토리에서만 코드 작업을 수행할 수 있습니다. 연결된 리포지토리는 지원되지 않습니다.

선택을 하고 확인한 후 Amazon Q는 지정된 리포지토리의 코드뿐만 아니라 문제 제목 및 설명을 기반으로 요청이 무엇인지 확인하려고 시도하는 동안 문제를 진행 중 상태로 이동합니다. 고정된 주석이 생성되어 작업 상태에 대한 업데이트를 제공합니다. 데이터를 검토한 후 Amazon Q는 솔루션에 대한 잠재적 접근 방식을 공식화합니다. Amazon Q는 고정된 주석을 업데이트하고 모든 단계에서 문제에 대한 진행 상황에 주석을 달아 작업을 기록합니다. 고정된 주석 및 답글과 달리 작업에 대한 엄격한 시간적 기록은 유지되지 않습니다. 대신 고정 주석의 최상위 수준에 작업에 대한 가장 적절한 정보를 추가합니다. 리포지토리에 이미 있는 코드에 대한 접근 방식 및 분석을 기반으로 코드를 생성하려고 시도합니다. 잠재적 솔루션을 성공적으로 생성하면 브랜치를 생성하고 해당 브랜치에 코드를 커밋합니다. 그런 다음 해당 브랜치를 기본 브랜치와 병합하는 풀 요청을 생성합니다. Amazon Q가 작업을 완료하면 문제를 검토 중으로 이동하므로 사용자와 팀이 평가할 수 있는 코드가 있는지 알 수 있습니다.

Note

이 기능은 미국 서부(오리건) 리전의 문제를 통해서만 사용할 수 있습니다. Jira 소프트웨어 확장 프로그램과 함께 Jira를 사용하도록 프로젝트를 구성한 경우에는 사용할 수 없습니다. 또한 보드의 레이아웃을 사용자 지정한 경우 상태가 변경되지 않을 수 있습니다. 최상의 결과를 얻으려면 표준 보드 레이아웃이 있는 프로젝트에서만 이 기능을 사용합니다.

이 기능은 Git 하위 모듈에서 작동하지 않습니다. 리포지토리에 포함된 Git 하위 모듈은 변경할 수 없습니다.

Amazon Q에 문제를 할당한 후에는 문제의 제목 또는 설명을 변경하거나 다른 사람에게 할당할 수 없습니다. 문제에서 Amazon Q를 할당 취소하면 현재 단계가 완료된 다음 작업이 중지됩니다. 작업을 재개하거나 할당 해제된 후에는 문제에 재할당할 수 없습니다.

사용자가 작업 생성을 허용하기로 선택한 경우 Amazon Q에 할당된 경우 검토 중 열로 문제를 자동으로 이동할 수 있습니다. 그러나 검토 중 문제에는 진행 중 상태와 같이 다른 상태의 태스크가 여전히 있을 수 있습니다.

자습서의 이 부분에서는 최신 3계층 웹 애플리케이션 블루프린트로 생성된 프로젝트에 포함된 코드에 대한 잠재적 기능을 기반으로 세 가지 문제를 생성합니다. 하나는 `a`를 추가하여 새 `mysfit` 객체를 생성하고, 다른 하나는 정렬 기능을 추가하고, 다른 하나는 `test`라는 브랜치를 포함하도록 워크플로를 업데이트하는 것입니다.

Note

코드가 다른 프로젝트에서 작업하는 경우 해당 코드 기반과 관련된 제목 및 설명 문제를 생성합니다.

문제를 생성하고 평가할 솔루션을 생성하려면

1. 탐색 창에서 문제를 선택하고 보드 보기에 있는지 확인합니다.
2. 문제 생성을 선택합니다.
3. 문제에 일반적인 언어로 하려는 작업을 설명하는 제목을 지정합니다. 예를 들어 이 문제에는 **Create another mysfit named Quokkapus** 제목을 입력합니다. 설명에 다음 세부 정보를 제공합니다.

Expand the table of mysfits to 13, and give the new mysfit the following characteristics:

Name: Quokkapus

Species: Quokka-Octopus hybrid

Good/Evil: Good

Lawful/Chaotic: Chaotic

Age: 216

Description: Australia is full of amazing marsupials, but there's nothing there quite like the Quokkapus.

She's always got a friendly smile on her face, especially when she's using her eight limbs to wrap you up

in a great big hug. She exists on a diet of code bugs and caffeine. If you've got some gnarly code that needs a

assistance, adopt Quokkapus and put her to work - she'll love it! Just make sure you leave enough room for

her to grow, and keep that coffee coming.

4. (선택 사항) mysfit을 위한 축소판 그림 및 프로필 사진으로 사용할 이미지를 문제에 연결합니다. 이렇게 하면 설명을 업데이트하여 사용하려는 이미지와 그 이유에 대한 세부 정보를 포함합니다. 예를 들어 설명에 'mysfit을 사용하려면 웹 사이트에 이미지 파일을 배포해야 합니다. 이 문제에 연결된 이미지를 작업의 일부로 소스 리포지토리에 추가하고 웹 사이트에 이미지를 배포합니다.'

Note

연결된 이미지는 이 자습서의 상호 작용 중에 웹 사이트에 배포되거나 배포되지 않을 수 있습니다. 웹 사이트에 직접 이미지를 추가한 다음 Amazon Q에 대한 설명을 남겨 코드를 업데이트하여 풀 요청을 생성한 후 사용하려는 이미지를 가리키도록 할 수 있습니다.

설명을 검토하고 다음 단계로 진행하기 전에 필요할 수 있는 모든 세부 정보가 포함되어 있는지 확인합니다.

5. 담당자에서 Amazon Q에 할당을 선택합니다.
6. 소스 리포지토리의 경우 소스 코드가 포함된 리포지토리의 이름을 선택합니다.
7. 필요한 경우 각 단계 후 중지하고 작업 검토를 대기하도록 Amazon Q에 요구 선택기를 활성 상태로 합니다.

Note

각 단계 후 Amazon Q 중지 옵션을 선택하면 문제 또는 생성된 작업에 대해 의견을 제시할 수 있으며, 의견을 기반으로 Amazon Q가 접근 방식을 최대 3회 변경할 수 있습니다. 작업을 검토할 수 있도록 각 단계 후 Amazon Q 중지를 설정하지 않도록 옵션을 선택하면 Amazon Q가 피드백을 기다리지 않으므로 작업이 더 빨리 진행될 수 있지만 의견을 남기면 Amazon Q가 취하는 방향에 영향을 미칠 수 없습니다. 또한 Amazon Q는 해당 옵션을 선택한 경우 풀 요청에 남아 있는 주석에 응답하지 않습니다.

8. Amazon Q가 워크플로 파일을 수정하도록 허용 선택기를 비활성 상태로 둡니다.
9. Amazon Q가 작업을 제안하도록 허용을 활성 상태로 설정합니다.
10. 문제 생성을 선택합니다. 보기가 문제 보드로 변경됩니다.
11. 문제 생성을 선택하여 이번에는 제목이 **Change the get_all_mysfits() API to return mysfits sorted by the Age attribute**인 다른 문제를 생성합니다. 이 문제를 Amazon Q에 할당하고 문제를 생성합니다.
12. 문제 생성을 선택하여 이번에는 제목이 **Update the OnPullRequest workflow to include a branch named test in its triggers**인 다른 문제를 생성합니다. 선택적으로 설명의 워크플로에 연결합니다. 이 문제를 Amazon Q에 할당하되 이번에는 Amazon Q가 워크플로 파일을 수정하도록 허용 선택기가 활성 상태로 설정되어 있는지 확인합니다. 문제를 생성하여 문제 보드로 돌아갑니다.

i Tip

워크폴로 파일을 포함한 파일은 at 기호(@)를 입력하고 파일 이름을 입력하여 검색할 수 있습니다.

문제를 생성하고 할당하면 문제가 진행 중으로 이동합니다. Amazon Q는 고정된 주석에 문제 내에서 진행 상황을 추적하는 주석을 추가합니다. 솔루션에 대한 접근 방식을 정의할 수 있는 경우 코드 기반에 대한 분석이 포함된 백그라운드 섹션과 솔루션 생성에 대해 제안된 접근 방식을 자세히 설명하는 접근 섹션으로 문제의 설명을 업데이트합니다. Amazon Q가 문제에 설명된 문제에 대한 솔루션을 성공적으로 실행하는 경우 해당 브랜치에 제안된 솔루션을 구현하는 브랜치 및 코드 변경 사항이 생성됩니다. 제안된 코드에 Amazon Q가 알고 있는 오픈 소스 코드와 유사한 점이 포함된 경우, 해당 코드에 대한 연결이 포함된 파일을 제공하여 검토할 수 있습니다. 코드가 준비되면 제안된 코드 변경 사항을 검토하고, 해당 풀 요청에 대한 연결을 문제에 추가하고, 문제를 검토 중으로 이동할 수 있도록 풀 요청을 생성합니다.

A Important

풀 요청의 코드 변경 사항을 병합하기 전에 항상 검토해야 합니다. 다른 코드 변경과 마찬가지로 Amazon Q에서 변경한 코드를 병합하면 병합된 코드가 제대로 검토되지 않고 병합될 때 오류가 발생하는 경우 코드 기반 및 인프라 코드에 부정적인 영향을 미칠 수 있습니다.

Amazon Q의 변경 사항이 포함된 문제 및 연결된 풀 요청을 검토하려면

1. 문제에서 진행 중인 Amazon Q에 할당된 문제를 선택합니다. 의견을 검토하여 Amazon Q의 진행 상황을 모니터링합니다. 의견이 있는 경우 배경을 검토하고 문제에 대한 설명에 있는 레코드에 접근합니다. Amazon Q가 작업을 제안하도록 허용하기로 선택한 경우 제안된 작업을 검토하고 필요한 조치를 취합니다. 예를 들어 Amazon Q에서 태스크를 제안하고 순서를 변경하거나 특정 사용자에게 태스크를 할당하려는 경우 태스크 변경, 추가 또는 재정렬을 선택하고 필요한 업데이트를 수행합니다. 문제 보기를 완료했으면 X를 선택하여 문제 창을 닫습니다.

i Tip

태스크의 진행 상황을 보려면 문제의 태스크 목록에서 태스크를 선택합니다. 작업은 보드에 별도의 항목으로 표시되지 않으며 문제를 통해서만 액세스할 수 있습니다. 작업이 Amazon Q에 할당된 경우 작업을 열어 수행하려는 작업을 승인해야 합니다. 연결된 풀 요

청이 문제에 연결로 표시되지 않고 태스크에만 표시되므로 연결된 풀 요청을 보려면 태스크를 열어야 합니다. 작업에서 문제로 돌아가려면 문제에 대한 연결을 선택합니다.

- 이제 검토 중인 Amazon Q에 할당된 문제를 선택합니다. 배경을 검토하고 문제에 대한 설명에서 기록으로 접근합니다. 설명을 검토하여 수행한 작업을 이해합니다. 진행 상황, 취해야 할 조치, 의견을 포함하여 이 문제와 관련된 작업을 위해 생성된 모든 작업을 검토합니다. 풀 요청에서 오픈 레이블 옆의 풀 요청에 대한 연결을 선택하여 코드를 검토합니다.

Tip

태스크에 대해 생성된 풀 요청은 태스크 보기에 연결된 풀 요청으로만 표시됩니다. 문제에 대한 연결된 풀 요청으로 표시되지 않습니다.

- 풀 요청에서 코드 변경 사항을 검토합니다. 자세한 내용은 [풀 요청 검토](#) 섹션을 참조하세요. Amazon Q가 제안된 코드를 변경하도록 하려면 풀 요청에 대한 설명을 남겨 둡니다. 최상의 결과를 얻으려면 Amazon Q에 대한 의견을 남길 때 구체적으로 설명하세요.

예를 들어, **Create another mysfit named Quokkapus**에 대해 생성된 풀 요청을 검토할 때 설명에 오타가 있음을 확인할 수 있습니다. "'needsa'라는 오타를 'needs'와 'a' 사이에 공백을 추가하여 수정하도록 설명을 변경하라"고 Amazon Q에 대한 주석을 남길 수 있습니다. 또는 Amazon Q에 설명을 업데이트하고 통합하도록 수정된 전체 설명을 제공하도록 지시하는 주석을 남길 수 있습니다.

새 mysfit에 대한 이미지를 웹 사이트에 업로드한 경우 Amazon Q에 대한 주석을 남겨 새 mysfit에 사용할 이미지 및 축소판에 대한 포인터로 mysfit을 업데이트할 수 있습니다.

Note

Amazon Q는 개별 의견에 응답하지 않습니다. Amazon Q는 문제를 생성할 때 승인을 위해 모든 단계 후에 중지하는 기본 옵션을 선택한 경우에만 풀 요청에 주석에 남아 있는 피드백을 통합합니다.

- (선택 사항) 사용자와 다른 프로젝트 사용자가 코드 변경에 필요한 모든 설명을 남긴 후 리비전 생성을 선택하여 Amazon Q가 요청한 변경 사항을 설명에 통합하는 풀 요청의 개정을 생성하도록 합니다. 기비전 생성 진행 상황은 변경 사항이 아닌 개요에서 Amazon Q에 보고됩니다. 리비전 생성 시 Amazon Q의 최신 업데이트를 보려면 브라우저를 새로 고쳐야 합니다.

Note

문제를 생성한 사용자만 풀 요청의 리비전을 생성할 수 있습니다. 풀 요청의 리비전은 하나만 요청할 수 있습니다. 리비전 생성을 선택하기 전에 의견과 관련된 모든 문제를 해결하고 의견의 내용에 만족해야 합니다.

- 이 예시 프로젝트의 각 풀 요청에 대해 워크플로가 실행됩니다. 풀 요청을 병합하기 전에 워크플로가 성공적으로 실행되었는지 확인합니다. 코드를 병합하기 전에 코드를 테스트할 추가 워크플로 및 환경을 생성하도록 선택할 수도 있습니다. 자세한 내용은 [워크플로 시작하기](#) 섹션을 참조하세요.
- 풀 요청의 최신 개정에 만족하면 병합을 선택합니다.

문제를 생성하고 Amazon Q에서 해당 이슈에 대한 작업 추천받기

문제에는 복잡하거나 긴 양의 작업이 포함될 수 있습니다. CodeCatalyst에는 소프트웨어 개발용 Amazon Q Developer Agent와의 통합이 포함됩니다. Amazon Q에 제목과 설명을 기준으로 문제를 분석하도록 요청하고 작업을 논리적으로 구분하여 별도의 태스크로 분할하는 것이 좋습니다. 그러면 권장 작업 목록을 생성하려고 시도합니다. 그러면 이 목록을 검토, 수정하고 생성할지 여부를 선택할 수 있습니다. 이렇게 하면 사용자와 팀이 작업의 개별 부분을 더 빠르게 달성할 수 있는 관리 가능한 방식으로 사용자에게 할당하는 데 도움이 될 수 있습니다.

문제에 권장되는 작업 목록을 생성하고 검토하려면

- 탐색 창에서 문제를 선택하고 보드 보기에 있는지 확인합니다.
- 문제 생성을 선택합니다.
- 문제에 일반적인 언어로 하려는 작업을 설명하는 제목을 지정합니다. 예를 들어 이 문제에는 **Change the `get_all_mysfits()` API to return mysfits sorted by the Good/Evil attribute** 제목을 입력합니다. 설명에 다음 세부 정보를 제공합니다.

```
Update the API to allow sorting of mysfits by whether they are Good, Neutral, or Evil. Add a button on the website that allows users to quickly choose this sort and to exclude alignments that they don't want to see.
```

- 설명을 검토하고 다음 단계로 진행하기 전에 필요할 수 있는 모든 세부 정보가 포함되어 있는지 확인합니다.
- 담당자에서 문제를 자신에게 할당하도록 선택합니다.

6. 문제 생성을 선택합니다. 보기가 문제 보드로 변경됩니다.
7. 방금 생성한 문제를 선택하여 엽니다. 작업 추천을 선택합니다.
8. 리포지토리의 경우 소스 코드가 포함된 리포지토리의 이름을 선택합니다. 작업 추천 시작을 선택합니다.

대화 상자가 닫히고 Amazon Q가 문제의 복잡성 분석을 시작합니다. 문제가 복잡한 경우 작업을 별도의 순차적 작업으로 세분화하는 것이 좋습니다. 목록이 준비되면 추천 작업 보기를 선택합니다. 작업을 추가하고 추천 작업을 수정하고 작업 순서를 변경할 수 있습니다. 추천 사항에 동의하는 경우 작업 생성을 선택하면 작업이 생성됩니다. 그런 다음 작업을 수행할 사용자에게 할당하거나 Amazon Q 자체에도 할당할 수 있습니다.

리소스 정리

이 자습서를 완료한 후에는 다음 작업을 수행하여 이 자습서 중에 생성한 리소스 중 더 이상 필요하지 않은 리소스를 정리하는 것이 좋습니다.

- 더 이상 작업하지 않는 모든 문제에서 Amazon Q 할당을 취소합니다. Amazon Q가 문제에 대한 작업을 완료했거나 솔루션을 찾을 수 없는 경우 생성형 AI 기능의 최대 할당량에 도달하지 않도록 Amazon Q를 할당 취소해야 합니다. 자세한 내용은 [생성형 AI 기능 관리](#) 및 [요금](#)을 참조하세요.
- 작업이 완료된 모든 문제를 완료로 이동합니다.
- 프로젝트가 더 이상 필요하지 않은 경우 프로젝트를 삭제합니다.

CodeCatalyst에서 스페이스로 리소스 구성

사용자, 회사, 부서 또는 그룹을 나타내는 스페이스를 생성하고 개발 팀이 프로젝트를 관리할 수 있는 스페이스를 제공합니다. Amazon CodeCatalyst에서 만든 프로젝트, 멤버 및 관련 클라우드 리소스를 추가하려면 스페이스를 만들어야 합니다.

Note

스페이스 이름은 CodeCatalyst 전체에서 고유해야 합니다. 삭제된 스페이스의 이름은 재사용할 수 없습니다.

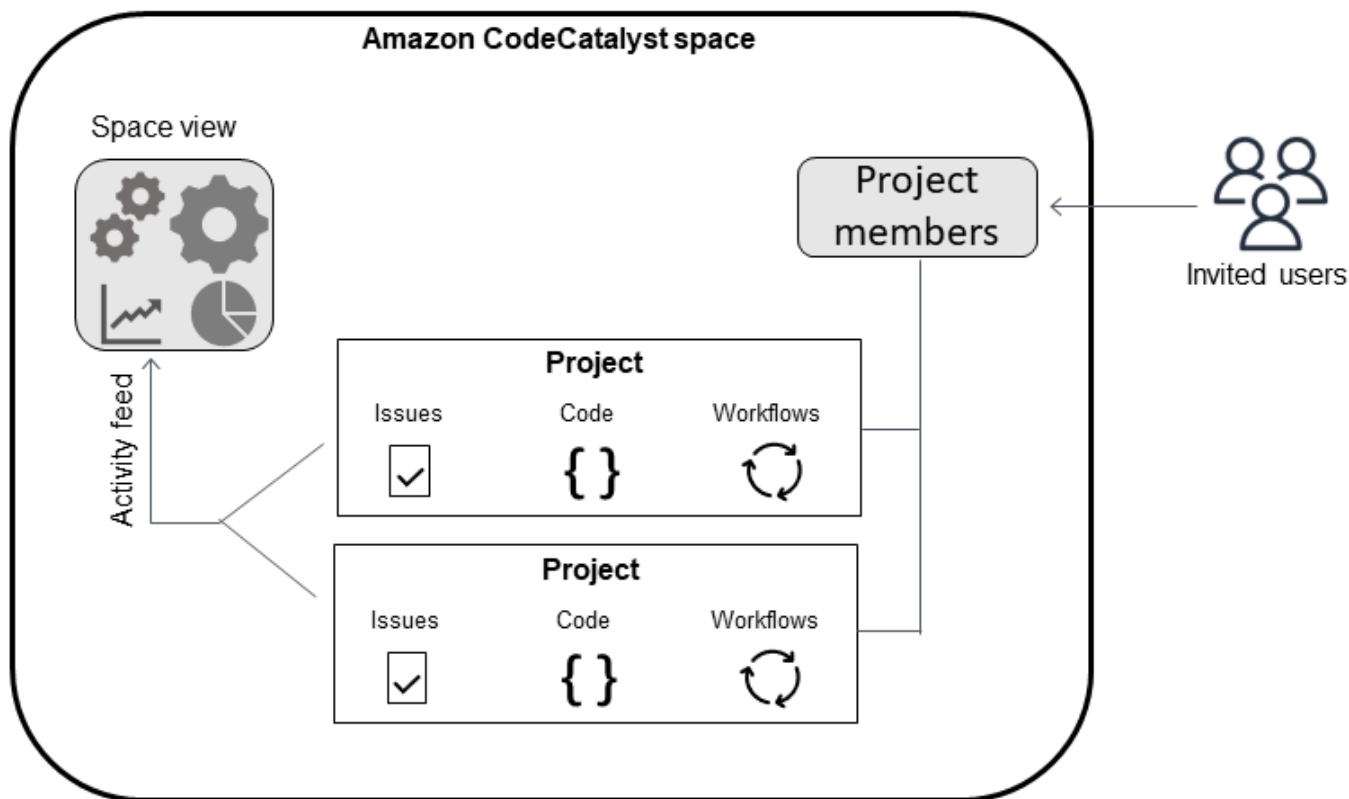
스페이스를 만들면 CodeCatalyst는 스페이스 관리자 역할을 자동으로 할당합니다. 스페이스의 다른 사용자에게 이 역할을 추가할 수 있습니다.

스페이스 관리자 역할을 사용하면 다음과 같이 스페이스를 관리할 수 있습니다

- 스페이스에 다른 스페이스 관리자 추가
- 멤버 역할 및 권한 변경
- 스페이스 편집 또는 삭제
- 프로젝트 생성 및 프로젝트에 멤버 초대
- 스페이스의 모든 프로젝트 목록 보기
- 스페이스의 모든 프로젝트에 대한 활동 피드 보기

스페이스를 만들면 스페이스 관리자 역할과 스페이스 만들기의 일부로 만든 프로젝트의 프로젝트 관리자 역할이라는 두 가지 역할이 스페이스에 자동으로 추가됩니다. 프로젝트에 대한 초대를 수락하면 추가 사용자가 자동으로 스페이스에 멤버로 추가됩니다. 스페이스의 이 멤버십은 해당 스페이스에 대한 권한을 부여하지 않습니다. 사용자가 스페이스에서 수행할 수 있는 작업은 사용자가 특정 프로젝트에서 수행하는 역할에 따라 결정됩니다.

역할에 관한 자세한 내용은 [사용자 역할로 액세스 권한 부여](#) 섹션을 참조하세요.



다음은 추가된 계정에 대한 추가 고려 사항입니다.

- AWS 계정 CodeCatalyst 스페이스에 추가되는 해당 스페이스의 모든 프로젝트에서 사용할 수 있습니다.
- 각 환경은 여러을 지원할 수 있지만 작업에서 환경당 하나의 계정만 사용할 AWS 계정수 있습니다.
- 청구는 스페이스 수준에서 구성됩니다. 청구를 위해 여러 계정을 구성할 수 있지만 CodeCatalyst 스페이스에서는 하나만 활성화할 수 있습니다. 는 CodeCatalyst에서 둘 이상의 스페이스에 대한 결제 계정으로 사용할 AWS 계정 수 있습니다. CodeCatalyst 스페이스의 결제 계정으로 AWS 계정 지정된 에는 스페이스의 다른 계정 연결과 다른 할당량이 있습니다. 자세한 내용은 [CodeCatalyst 할당량 단원](#)을 참조하십시오.
- 연결을 생성한 후 워크플로가 CodeCatalyst 환경에서 해당 AWS IAM 역할에 액세스해야 하는 경우 연결에 IAM 역할을 추가해야 합니다. 환경 사용 방식에 대한 자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 섹션을 참조하세요.

주제

- [스페이스 생성](#)

- [스페이스 편집](#)
- [스페이스 삭제](#)
- [스페이스 내 사용자 및 리소스에 대한 활동 모니터링](#)
- [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#)
- [연결된 계정에 대한 IAM 역할 구성](#)
- [사용자에게 스페이스 권한 부여](#)
- [팀을 사용하여 스페이스 액세스 허용](#)
- [시스템 리소스에 대한 스페이스 액세스 허용](#)
- [스페이스의 개발 환경 관리](#)
- [스페이스 할당량](#)

스페이스 생성

AWS Builder ID로 Amazon CodeCatalyst에 처음 가입할 때는 스페이스를 생성해야 합니다. 자세한 내용은 [CodeCatalyst 설정 및 로그인](#) 단원을 참조하십시오. 비즈니스 요구 사항에 맞게 추가 스페이스를 생성하도록 선택할 수 있습니다.

Note

스페이스 이름은 CodeCatalyst 전체에서 고유해야 합니다. 삭제된 스페이스의 이름은 재사용할 수 없습니다.

이 사용 설명서의 정보는 CodeCatalyst에서 AWS Builder ID 사용자를 지원하는 스페이스에서 사용자를 생성하기 위해 제공됩니다. ID 페더레이션을 지원하는 스페이스를 설정하고 관리하는 단계는 CodeCatalyst 관리자 안내서에 나와 있습니다. ID 페더레이션을 위해 설정된 스페이스로 작업하려면, Amazon CodeCatalyst 관리자 안내서의 [CodeCatalyst 스페이스 설정 및 관리](#)를 참조하세요.

AWS Builder ID 사용자를 지원하는 추가 스페이스를 생성하려면 스페이스 관리자 역할이 할당되어야 합니다.

Note

추가 스페이스를 생성할 때 프로젝트를 생성하라는 메시지가 표시되지 않습니다. 스페이스에서 프로젝트를 만드는 방법을 알아보려면 [프로젝트 생성](#) 섹션을 참조하세요.

다른 인스턴스 생성

1. 에서 CodeCatalyst 스페이스와 연결하려는 AWS 계정 것과 동일한 로 로그인했는지 AWS Management Console 확인합니다.
2. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
3. 스페이스로 이동합니다.

Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

4. 스페이스 생성을 선택합니다.
5. 스페이스 생성 페이지의 스페이스 이름 에 스페이스의 이름을 입력합니다. 이 이름은 나중에 변경할 수 없습니다.

Note

스페이스 이름은 CodeCatalyst 전체에서 고유해야 합니다. 삭제된 스페이스의 이름은 재 사용할 수 없습니다.

6. AWS 리전에서 스페이스 및 프로젝트 데이터를 저장할 리전을 선택합니다. 이 이름은 나중에 변경할 수 없습니다.
7. AWS 계정 ID의 경우, 스페이스에 연결하려는 계정의 12자리 ID를 입력합니다.

AWS 계정 확인 토큰에서 생성된 토큰 ID를 복사합니다. 토큰은 자동으로 복사되지만 AWS 연결 요청을 승인하는 동안 토큰을 저장할 수 있습니다.

8. 확인을 AWS 선택합니다.
9. AWS Management Console에서 Amazon CodeCatalyst 스페이스 확인 페이지가 열립니다. Amazon CodeCatalyst 스페이스 페이지입니다. 페이지에 액세스하려면 로그인해야 할 수 있습니다.

에서 AWS Management Console 스페이스를 생성하려는 AWS 리전 동일한를 선택해야 합니다.

페이지에 직접 액세스하려면 <https://console.aws.amazon.com/codecatalyst/home/>에서 Amazon CodeCatalyst Spaces AWS Management Console 에 로그인합니다.

검증 토큰은 검증 토큰에 자동으로 입력됩니다. 성공 배너는 토큰이 유효한 토큰이라는 메시지를 보여줍니다.

10. 스페이스 확인을 선택합니다.

계정이 스페이스에 추가되었음을 보여주는 계정 확인 성공 메시지가 표시됩니다.

11. Amazon CodeCatalyst 스페이스 확인 페이지에 머무릅니다. 다음 링크를 선택합니다. 이 스페이스에 대한 IAM 역할을 추가하려면 스페이스 세부 정보를 봅니다.

AWS Management Console에서 CodeCatalyst 스페이스 세부 정보 페이지가 열립니다. Amazon CodeCatalyst 스페이스 페이지입니다. 페이지에 액세스하려면 로그인해야 할 수 있습니다.

12. CodeCatalyst에 사용 가능한 IAM 역할에서 IAM 역할 추가를 선택합니다.

CodeCatalyst에 사용 가능한 IAM 역할 추가 페이지가 표시됩니다.

13. IAM에서 CodeCatalyst 개발 관리자 역할 생성을 선택합니다. 이 옵션은 개발 역할에 대한 권한 정책 및 신뢰 정책을 포함하는 서비스 역할을 생성합니다.

개발자 역할은 CodeCatalyst 워크플로가 Amazon S3, Lambda, 등의 AWS 리소스에 액세스할 수 있도록 하는 AWS IAM 역할입니다 CloudFormation. 역할에는 고유한 식별자가 추가된 CodeCatalystWorkflowDevelopmentRole-*spaceName* 이름이 있습니다. 역할 및 역할 정책에 대한 자세한 내용은 [CodeCatalystWorkflowDevelopmentRole-*spaceName* 서비스 역할 이해](#) 섹션을 참조하세요.

14. 개발 역할 생성을 선택합니다.

15. 연결 페이지에 있는 CodeCatalyst에 사용 가능한 IAM 역할에서 계정에 추가된 IAM 역할 목록의 개발자 역할을 봅니다.

16. Amazon CodeCatalyst로 이동을 선택합니다.

17. CodeCatalyst의 생성 페이지에서 스페이스 생성을 선택합니다.

스페이스 편집

스페이스의 설명을 변경하여 사용자가 스페이스의 용도를 더 잘 이해할 수 있도록 도울 수 있습니다.

스페이스 세부 정보를 편집하려면 스페이스 관리자 역할이 있어야 합니다.

이 사용 설명서의 정보는 CodeCatalyst에서 AWS Builder ID 사용자를 지원하는 스페이스에서 사용자를 편집하기 위해 제공됩니다. ID 페더레이션을 지원하는 스페이스를 설정하고 관리하는 단계에 대해

자세히 알아보려면 Amazon CodeCatalyst 관리자 안내서의 [CodeCatalyst 스페이스 설정 및 관리](#)를 참조하세요.

스페이스 설명 편집

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다.

Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

3. 스페이스 설정 탭에서 편집을 선택합니다. 원하는 대로 스페이스 설명을 변경한 후 저장을 선택합니다.

스페이스 삭제

스페이스를 삭제하여 스페이스의 모든 리소스에 대한 액세스를 제거할 수 있습니다. 스페이스를 삭제하려면 스페이스 관리자 역할이 있어야 합니다.

Note

스페이스 삭제는 실행 취소할 수 없습니다.

스페이스를 삭제한 후에는 모든 스페이스 멤버가 스페이스 리소스에 액세스할 수 없게 됩니다. 스페이스 리소스에 대한 청구도 중지되며 타사 소스 리포지토리에서 프롬프트되는 모든 워크플로도 중지됩니다.

Note

스페이스 이름은 CodeCatalyst 전체에서 고유해야 합니다. 삭제된 스페이스의 이름은 재사용할 수 없습니다.

이 사용 설명서의 정보는 CodeCatalyst에서 AWS Builder ID 사용자를 지원하는 스페이스에서 사용자를 삭제하기 위해 제공됩니다. ID 페더레이션을 지원하는 스페이스를 설정하고 관리하는 단계에 대해

자세히 알아보려면 Amazon CodeCatalyst 관리자 안내서의 [CodeCatalyst 스페이스 설정 및 관리](#)를 참조하세요.

스페이스 삭제

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다.

Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

3. 설정을 선택한 삭제를 선택합니다.
4. **delete**를 입력하여 삭제를 확인합니다.
5. 삭제를 선택합니다.

Note

두 개 이상의 스페이스에 소속되어 있는 경우 스페이스 개요 페이지로 리디렉션됩니다. 하나의 스페이스에 소속되어 있는 경우 스페이스 생성 페이지로 리디렉션됩니다.

스페이스 내 사용자 및 리소스에 대한 활동 모니터링

최근에 만든 프로젝트와 상태 업데이트를 확인하려면 CodeCatalyst 콘솔에서 스페이스 리소스에 대한 업데이트가 표시된 활동 피드를 볼 수 있습니다.

활동 피드에서 실패한 워크플로 실행 및 생성된 프로젝트와 같은 지표를 볼 수 있습니다.

스페이스 내 활동 보기

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다.

i Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

3. 활동을 선택합니다.
4. 활동에서 정보를 봅니다.
5. 활동별로 필터링하려면 오른쪽 상단의 선택기를 선택합니다.
6. 스페이스의 모든 활동을 보려면 모든 활동 유형을 선택합니다.

연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정

Amazon CodeCatalyst 스페이스 AWS 계정 에서의 리소스를 사용할 수 있습니다. 이렇게 하려면 CodeCatalyst에서 AWS 계정 와 스페이스 간에 연결을 설정해야 합니다. 이와 같은 연결을 생성하면 CodeCatalyst 스페이스 내의 프로젝트 및 워크플로가 AWS 계정의 리소스와 상호 작용할 수 있습니다. CodeCatalyst 스페이스 AWS 계정 와 함께 사용할 각에 대해 하나의 연결을 생성해야 합니다.

연결을 생성한 후 AWS IAM 역할을 연결하도록 선택할 수 있습니다.

주제

- [스페이스 AWS 계정 에 추가](#)
- [계정 연결에 IAM 역할 추가](#)
- [배포 환경에 계정 연결 및 IAM 역할 추가](#)
- [계정 연결 보기](#)
- [계정 연결 삭제\(CodeCatalyst\)](#)
- [스페이스에 대한 청구 계정 구성](#)

스페이스에 계정을 AWS 계정 추가하여 승인된를 사용하도록 CodeCatalyst를 설정할 수 있습니다. CodeCatalyst 스페이스 AWS 계정 에를 추가하면 프로젝트 워크플로에 AWS 계정 리소스 및 결제 구성에 대한 액세스 권한을 부여할 수 있습니다.

를 추가하면 CodeCatalyst가이 계정을 사용하도록 승인하는 연결이 AWS 계정 생성됩니다. 추가된 AWS 계정 를 사용하여 다음을 수행할 수 있습니다.

- CodeCatalyst 스페이스에 대한 청구를 설정합니다. Amazon CodeCatalyst 관리자 안내서의 [Managing billing](#)을 참조하세요. CodeCatalyst 스페이스의 청구 계정으로 지정된 AWS 계정에는 스페이스의 다른 계정 연결과 다른 할당량이 있습니다. 자세한 내용은 [CodeCatalyst 할당량](#) 단원을 참조하십시오.
- CodeCatalyst가 AWS 리소스에 액세스하고 계정 AWS 서비스 의에 배포하기 위해 IAM 역할을 수입하도록 허용합니다. [연결된 계정에 대한 IAM 역할 구성](#)(를) 참조하세요.

계정 연결은 AWS 계정으로 인증을 완료하여 생성됩니다. 연결이 생성된 후 IAM 역할을 추가하여 워크플로 및 프로젝트가 사용할 연결을 추가로 구성합니다.

CodeCatalyst의 AWS Management Console 페이지에서 AWS 계정 및 스페이스의 관리자로 계정 연결을 구성하는 단계는 CodeCatalyst 관리자 안내서의 [연결된 계정 관리](#)를 참조하세요. 특정 프로젝트에 대한 제한을 위해 계정 연결을 구성할 수 있습니다. 워크플로 또는 VPC 연결은 프로젝트에 액세스할 수 있는 AWS 계정 있는에만 연결할 수 있습니다. 자세한 내용은 [Configuring project-restricted account connections](#)를 참조하세요.

스페이스 AWS 계정 에 추가

CodeCatalyst 콘솔과를 사용하여 스페이스 AWS Management Console 를에 연결합니다 AWS 계정.

CodeCatalyst의 스페이스에 AWS 계정 를 추가하기 전에 다음 사전 조건을 완료합니다.

- 를 AWS 계정 생성하고 연결하려는 계정에서 IAM 역할을 생성할 수 있는 권한을 얻습니다.
- 역할에 대한 권한이 있는 IAM 정책을 포함하여 계정 연결과 연결할 IAM 역할 또는 역할을 생성합니다.
- 연결을 생성하려는 CodeCatalyst 스페이스에서 스페이스 관리자 역할을 획득합니다.

주제

- [1단계: 연결 요청 생성](#)
- [2단계: 계정 연결 요청 수락](#)
- [3단계: 승인된 연결 검토](#)
- [4단계: 연결에 IAM 역할 추가](#)
- [다음 단계: 계정 연결을 위한 추가 IAM 역할 생성](#)

1단계: 연결 요청 생성

CodeCatalyst 콘솔에서 연결 요청을 생성하면 권한 부여를 완료하는 데 사용할 수 있는 연결 토큰이 생성됩니다.

연결을 생성하려는 CodeCatalyst 스페이스에 스페이스 관리자 또는 파워 유저 역할이 있어야 합니다. 추가하려는 AWS 계정 에 대한 관리 권한도 있어야 합니다.

연결을 생성하는 방법

1. 에서 연결을 생성하려는 것과 동일한 계정으로 로그인했는지 AWS Management Console 확인합니다.
2. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
3. CodeCatalyst 스페이스로 이동합니다. 설정을 선택한 다음 AWS 계정을 선택합니다.
4. 추가를 AWS 계정 선택합니다.
5. Amazon CodeCatalyst AWS 계정 와 연결 페이지의 AWS 계정 ID에 스페이스에 연결할 계정의 12자리 ID를 입력합니다. AWS 계정 ID 찾기에 대한 자세한 내용은 [AWS 계정 ID 및 해당 별칭을 참조하세요](#).
6. Amazon CodeCatalyst 표시 이름에 계정의 참조 이름을 입력합니다.
7. (선택 사항) 연결 설명에 계정 및 역할 또는 역할이 적용될 프로젝트를 선택하는 데 도움이 되는 계정에 대한 설명을 입력합니다.
8. AWS 계정 연결을 선택합니다.
9. 페이지가 성공 배너가 표시되는 AWS 계정 세부 정보 페이지로 돌아갑니다.

2단계: 계정 연결 요청 수락

CodeCatalyst 콘솔에서 연결하기 위한 요청을 제출한 후 AWS 관리자와 AWS 계정 협력하여 제공된 연결 토큰과 함께 제출하여 연결 요청을 수락합니다.

계정에 대한 관리자 권한이 있고 연결을 생성하는 것과 동일한 AWS Management Console 로 AWS 계정 에 로그인했는지 확인합니다.

연결 요청을 승인하려면(콘솔)

1. 에서 연결을 생성하려는 것과 동일한 계정으로 로그인했는지 AWS Management Console 확인합니다.
2. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.

3. CodeCatalyst 스페이스로 이동합니다. 설정을 선택한 다음 AWS 계정을 선택합니다.
4. AWS 계정 세부 정보 페이지에서 AWS Management Console에서 설정 완료를 선택합니다.
5. AWS Management Console에서 Amazon CodeCatalyst 스페이스 확인 페이지가 열립니다. Amazon CodeCatalyst 스페이스 페이지입니다. 페이지에 액세스하려면 로그인해야 할 수 있습니다.

페이지에 직접 액세스하려면 <https://console.aws.amazon.com/codecatalyst/home/>에서 Amazon CodeCatalyst Spaces AWS Management Console 에 로그인합니다.

검증 토큰은 검증 토큰에 자동으로 입력됩니다. 성공 메시지는 토큰이 유효한 토큰이라는 메시지를 보여줍니다.

6. (선택 사항) 승인된 유료 티어에서 유료 티어 승인(표준, 엔터프라이즈)을 선택하여 청구 계정의 유료 티어를 활성화할 수 있습니다.

 Note

이렇게 해도 청구 계층은 유료 티어로 업그레이드되지 않습니다. 그러나 CodeCatalyst에서 언제든지 스페이스의 결제 계층을 변경할 수 AWS 계정 있도록 구성합니다. 언제든지 유료 티어를 활성화할 수 있습니다. 이러한 변경이 없다면, 스페이스는 프리 티어만 사용할 수 있습니다.

7. 스페이스 확인을 선택합니다.

계정이 스페이스에 추가되었음을 보여주는 계정 확인 성공 메시지가 표시됩니다.

3단계: 승인된 연결 검토

연결이 승인되면 콘솔에서 연결에 추가한 IAM 역할과 함께 연결을 볼 수 있습니다.

승인된 연결을 검토하려면

1. CodeCatalyst 스페이스로 이동합니다. 설정을 선택한 다음 AWS 계정을 선택합니다.
2. 계정 연결은 생성된 날짜와 함께 나열됩니다.
3. 계정 표시 이름을 선택합니다. AWS 계정 세부 정보 페이지가 표시됩니다.

4단계: 연결에 IAM 역할 추가

CodeCatalyst 배포 작업에 대해 구성된 IAM 역할을 사용하는 경우 배포 환경에 역할을 추가합니다. 자세한 내용은 [계정 연결에 IAM 역할 추가](#) 섹션을 참조하세요.

다음 단계: 계정 연결을 위한 추가 IAM 역할 생성

연결을 생성한 후 추가 IAM 역할을 생성하여 추가할 수 있습니다. 추가하는 IAM 역할은 워크플로에 따라 다릅니다. 예를 들어 CodeCatalyst 빌드 작업에는 CodeCatalyst 빌드 역할이 필요합니다.

계정을 연결하려면 생성한 역할의 Amazon 리소스 이름(ARN)이 필요합니다. 여기에 설명된 대로 역할 또는 역할에 대한 ARN을 복사합니다. IAM 역할에서 ARN으로 작업하는 방법에 대한 추가 정보는 [Amazon 리소스 이름\(ARN\)](#)을 참조하세요.

IAM 역할 ARN에 액세스하려면

1. IAM 콘솔(<https://console.aws.amazon.com/iam/>)을 엽니다.
2. 탐색 창에서 역할을 선택합니다.
3. 검색 상자에 추가하려는 역할의 이름을 입력합니다.
4. 목록에서 역할을 선택합니다.

역할의 요약 페이지가 나타납니다.

5. 상단에서 역할 ARN 값을 복사합니다.

계정 연결에 IAM 역할 추가

계정 연결 생성의 일부로는 CodeCatalyst 스페이스의 프로젝트에 사용할 IAM 역할 또는 역할을 추가하는 것이 포함됩니다.

Note

계정 연결과 함께 IAM 역할을 사용하려면 CodeCatalyst 서비스 위탁자를 사용하도록 신뢰 정책이 업데이트되었는지 확인합니다.

계정 연결에 IAM 역할 추가(콘솔)

1. 에서 관리하려는 것과 동일한 계정으로 로그인했는지 AWS Management Console 확인합니다.

2. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
3. CodeCatalyst 스페이스로 이동합니다. 설정을 선택한 다음 AWS 계정을 선택합니다.
4. 계정 연결의 Amazon CodeCatalyst 표시 이름을 선택한 다음 역할 관리를 AWS Management Console 선택합니다.

Amazon CodeCatalyst 스페이스에 IAM 역할 추가 페이지가 표시됩니다.

5. 다음 중 하나를 수행하세요.
 - 개발자 역할에 대한 권한 정책 및 신뢰 정책이 포함된 서비스 역할을 생성하려면 IAM에서 CodeCatalyst 개발 관리자 역할 생성을 선택합니다. 역할에는 고유한 식별자가 추가된 CodeCatalystWorkflowDevelopmentRole-*spaceName* 이름이 있습니다. 역할 및 역할 정책에 대한 자세한 내용은 [CodeCatalystWorkflowDevelopmentRole-*spaceName* 서비스 역할 이해](#) 섹션을 참조하세요.

개발 역할 생성을 선택합니다.

- 이미 IAM에서 생성한 역할을 추가하려면 기존 IAM 역할 추가를 선택합니다. 기존 IAM 역할 선택의 드롭다운 목록에서 역할을 선택합니다.

[Add role]을 선택합니다.

페이지가 AWS Management Console에서 열립니다. 페이지에 액세스하려면 로그인해야 할 수 있습니다.

6. Amazon CodeCatalyst 스페이스 페이지 탐색 창에서 스페이스를 선택합니다.

페이지에 직접 액세스하려면 <https://console.aws.amazon.com/codecatalyst/home/>에서 Amazon CodeCatalyst Spaces AWS Management Console 에 로그인합니다.

7. CodeCatalyst 스페이스에 추가된 계정을 선택합니다. 연결 페이지가 표시됩니다.
8. 연결 페이지의 CodeCatalyst에서 사용할 수 있는 IAM 역할에서 계정에 추가된 IAM 역할 목록을 확인합니다. IAM 역할을 CodeCatalyst에 연결을 선택합니다.
9. IAM 역할 연결 팝업의 역할 ARN에 CodeCatalyst 스페이스와 연결할 IAM 역할의 Amazon 리소스 이름(ARN)을 입력합니다.

목적에서 계정 연결에서 역할을 사용할 방법을 설명하는 역할 목적을 선택합니다. 워크플로에서 작업을 실행하는 데 사용하는 역할에 RUNNER를 지정합니다. 다른 서비스에 액세스하는 데 사용하는 역할에 SERVICE를 지정합니다.

목적은 둘 이상 지정할 수 있습니다.

Note

역할 ARN의 목적을 선택해야 합니다.

10. IAM 역할 연결을 선택합니다. 추가 IAM 역할에 대해 이 단계를 반복합니다.

배포 환경에 계정 연결 및 IAM 역할 추가

Amazon ECS 또는 배포 AWS 용 리소스와 같은 AWS Lambda 리소스에 액세스하려면 CodeCatalyst 빌드 및 배포 작업에 해당 리소스에 액세스할 수 있는 권한이 있는 IAM 역할이 필요합니다. 스페이스 관리자 또는 파워 유저 역할을 사용하면 CodeCatalyst 계정을 리소스 AWS 계정 가 생성되는에 연결할 수 있습니다. 그런 다음 계정 연결에 IAM 역할을 추가합니다. 배포 작업의 경우 CodeCatalyst 환경에 IAM 역할을 추가해야 합니다.

프로젝트의 배포 환경에 사용할 IAM 역할을 추가해야 합니다. 계정 연결에 역할을 추가해도 역할 및 프로젝트 배포 환경에 대한 연결이 추가되지 않습니다. 배포 환경에 계정 연결 및 IAM 역할을 추가하려면 계정 연결 및 역할이 [4단계: 연결에 IAM 역할 추가](#)에 설명된 대로 생성되었는지 확인합니다.

그런 다음 CodeCatalyst 콘솔의 환경 페이지를 사용하여 프로젝트의 배포 환경에 계정 연결 및 IAM 역할을 추가합니다.

Note

IAM 역할이 필요한 CodeCatalyst 작업에 사용되는 경우에만 환경에 IAM 역할을 추가합니다. 빌드 작업을 포함하여 IAM 역할이 필요한 모든 워크플로 작업은 CodeCatalyst 환경을 사용해야 합니다.

배포 환경에 계정 연결 및 IAM 역할을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 계정 연결 및 IAM 역할을 추가하려는 배포 환경이 있는 프로젝트로 이동합니다.
3. CI/CD를 확장한 다음 환경을 선택합니다.
4. 환경을 선택하면 추가 탭이 표시됩니다.
5. AWS 계정 연결 탭을 선택합니다. 연결 이름 아래에 환경에 추가된 계정이 있는 경우 나열됩니다.
6. AWS 계정연결을 선택합니다. AWS 계정 를 <environment_name>에 연결 페이지가 표시됩니다.

7. 연결에서 추가하려는 IAM 역할과 계정 연결의 이름을 선택합니다. 연결을 선택합니다.

계정 연결 보기

연결 목록을 보고 각 연결에 대한 세부 정보를 볼 수 있습니다.

스페이스의 연결을 관리하려면 스페이스 관리자 또는 파워 유저 역할이 있어야 합니다.

CodeCatalyst 스페이스의 모든 연결을 보려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 보려는 계정 연결이 있는 스페이스로 이동합니다.
3. AWS 계정 탭을 선택합니다.
4. AWS 계정에서 각 연결의 계정 ID 및 상태를 포함하여 스페이스의 계정 연결 목록을 확인합니다.

계정 연결 세부 정보를 보려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다. 설정을 선택한 다음 AWS 계정을 선택합니다.
3. Amazon CodeCatalyst 표시 이름에서 연결 이름을 선택합니다. 세부 정보 페이지에서 다른 세부 정보와 함께 연결과 연결된 IAM 역할 목록을 봅니다.

계정 연결 삭제(CodeCatalyst)

계정 연결이 더 이상 필요하지 않으면 삭제할 수 있습니다. 이 절차에서는 CodeCatalyst를 사용하여 이전에 스페이스에 추가한 계정 연결을 삭제합니다. 이렇게 하면 계정이 스페이스의 청구 계정이 아닌 경우 스페이스의 계정 연결이 삭제됩니다.

Important

계정 연결이 삭제된 후에는 다시 연결할 수 없습니다. 새 계정 연결을 생성한 다음 필요에 따라 IAM 역할과 환경을 연결하거나 청구를 설정해야 합니다.

스페이스에 대한 사용량이 프리 티어를 초과하지 않더라도 CodeCatalyst 스페이스에 대한 청구 계정을 지정해야 합니다. 지정된 청구 계정인 계정의 스페이스를 제거하려면 먼저 해당 스페이스에 다른 계정을 추가해야 합니다. Amazon CodeCatalyst 관리자 안내서의 [Managing billing](#)을 참조하세요.

Important

이 단계를 사용하여 계정을 제거할 수 있지만, 이는 권장되지 않습니다. CodeCatalyst의 워크플로를 지원하도록 계정을 설정할 수도 있습니다.

스페이스의 계정 연결을 관리하려면 스페이스 관리자 또는 파워 유저 역할이 있어야 합니다.

제거된 계정은 나중에 다시 추가할 수 있지만 계정과 스페이스 간에 새 연결을 생성해야 합니다. IAM 역할을 추가된 계정에 다시 연결해야 합니다.

계정 연결을 삭제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다. 설정을 선택한 다음 AWS 계정을 선택합니다.
3. Amazon CodeCatalyst 표시 이름에서 제거할 계정 연결 옆의 선택기를 선택합니다.
4. AWS 계정제거를 선택합니다. 필드에 이름을 입력하여 삭제를 확인한 다음 제거를 선택합니다.

성공 배너가 표시되고 계정 연결이 연결 목록에서 제거됩니다.

스페이스에 대한 청구 계정 구성

스페이스에 대한 사용량이 프리 티어를 초과하지 않더라도 CodeCatalyst 스페이스에 대한 청구 계정을 지정해야 합니다.

청구 계정을 구성하려면 CodeCatalyst 관리자 안내서의 [청구](#)를 참조하세요. CodeCatalyst 스페이스의 청구 계정으로 지정된 AWS 계정에는 스페이스의 다른 계정 연결과 다른 할당량이 있습니다. 자세한 내용은 [CodeCatalyst 할당량](#) 섹션을 참조하세요.

CodeCatalyst 스페이스에 지정된 청구 계정인 계정을 제거하려면 먼저 새 청구 계정을 지정해야 합니다.

연결된 계정에 대한 IAM 역할 구성

CodeCatalyst에 추가하려는 계정에 대해 AWS Identity and Access Management (IAM)에서 역할을 생성합니다. 청구 계정을 추가하는 경우 역할을 생성할 필요가 없습니다.

에는 스페이스에 추가할에 대한 역할을 생성할 수 AWS 계정 있는 권한이 AWS 계정있어야 합니다. IAM 참조 및 예시 정책을 포함한 IAM 역할 및 정책에 대한 자세한 내용은 [Amazon CodeCatalyst의 ID 및 액세스 관리](#) 섹션을 참조하세요. CodeCatalyst에 사용되는 신뢰 정책 및 서비스 위탁자에 대한 자세한 내용은 [CodeCatalyst 신뢰 모델 이해](#) 섹션을 참조하세요.

CodeCatalyst에서 스페이스 관리자 역할로 로그인하여 계정에 계정(및 해당하는 경우 역할)을 추가하는 단계를 완료해야 합니다.

다음 방법 중 하나를 사용하여 계정 연결에 역할을 추가할 수 있습니다.

- CodeCatalystWorkflowDevelopmentRole-**spaceName** 역할에 대한 권한 정책 및 신뢰 정책이 포함된 서비스 역할을 생성하려면 [CodeCatalystWorkflowDevelopmentRole-**spaceName** 역할](#) 섹션을 참조하세요.
- 역할을 생성하고 블루프린트에서 프로젝트를 생성하기 위한 정책을 추가하는 예시는 [IAM 역할 생성 및 CodeCatalyst 신뢰 정책 사용](#) 섹션을 참조하세요.
- IAM 역할을 생성할 때 사용할 샘플 역할 정책 목록은 [IAM 역할을 사용하여 프로젝트 AWS 리소스에 대한 액세스 권한 부여](#) 섹션을 참조하세요.
- 워크플로 작업에 대한 역할을 생성하는 자세한 단계는 다음과 같이 해당 작업에 대한 워크플로 자습서를 참조하세요.
 - [자습서: Amazon S3에 아티팩트 업로드](#)
 - [자습서: 서버리스 애플리케이션 배포](#)
 - [자습서: Amazon ECS에 애플리케이션 배포](#)
 - [자습서: GitHub 작업을 사용한 린트 코드](#)

주제

- [CodeCatalystWorkflowDevelopmentRole-spaceName 역할](#)
- [AWSRoleForCodeCatalystSupport 역할](#)
- [IAM 역할 생성 및 CodeCatalyst 신뢰 정책 사용](#)

CodeCatalystWorkflowDevelopmentRole-**spaceName** 역할

개발자 역할을 IAM에서 원클릭 역할로 생성합니다. 계정을 추가하려는 스페이스에서 스페이스 관리자 또는 파워 유저 역할이 있어야 합니다. 추가 AWS 계정 하려는에 대한 관리 권한도 있어야 합니다.

아래 절차를 시작하기 전에 CodeCatalyst AWS Management Console 스페이스에 추가하려는 것과 동일한 계정으로 로그인해야 합니다. 그렇지 않으면 콘솔이 알 수 없는 계정 오류를 반환합니다.

CodeCatalyst CodeCatalystWorkflowDevelopmentRole-*spaceName*을 생성하고 추가하려면

1. CodeCatalyst 콘솔에서 시작하기 전에 열고 스페이스 AWS 계정 에 대해 동일한 로 로그인했는지 AWS Management Console 확인합니다.
2. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
3. CodeCatalyst 스페이스로 이동합니다. 설정을 선택한 다음 AWS 계정을 선택합니다.
4. 역할을 생성하려는 AWS 계정 의 링크를 선택합니다. AWS 계정 세부 정보 페이지가 표시됩니다.
5. 역할 관리를 AWS Management Console 선택합니다.

AWS Management Console에서 Amazon CodeCatalyst 스페이스에 IAM 역할 추가 페이지가 열립니다. Amazon CodeCatalyst 스페이스 페이지입니다. 페이지에 액세스하려면 로그인해야 할 수 있습니다.

6. IAM에서 CodeCatalyst 개발 관리자 역할 생성을 선택합니다. 이 옵션은 개발 역할에 대한 권한 정책 및 신뢰 정책을 포함하는 서비스 역할을 생성합니다. 역할에는 CodeCatalystWorkflowDevelopmentRole-*spaceName* 이름이 있습니다. 역할 및 역할 정책에 대한 자세한 내용은 [CodeCatalystWorkflowDevelopmentRole-*spaceName* 서비스 역할 이해](#) 섹션을 참조하세요.

Note

이 역할은 개발자 계정에서만 사용하는 것이 권장되며 AdministratorAccess AWS 관리형 정책을 사용하여 여기에 새 정책 및 리소스를 생성할 수 있는 전체 액세스 권한을 부여합니다 AWS 계정.

7. 개발 역할 생성을 선택합니다.
8. 연결 페이지에 있는 CodeCatalyst에서 사용할 수 있는 IAM 역할에서 계정에 추가된 IAM 역할 목록의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 봅니다.
9. 스페이스로 돌아가려면 Amazon CodeCatalyst로 이동을 선택합니다.

AWSRoleForCodeCatalystSupport 역할

IAM에서 원클릭 역할로 지원 역할을 생성합니다. 계정을 추가하려는 스페이스에서 스페이스 관리자 또는 파워 유저 역할이 있어야 합니다. 추가 AWS 계정 하려는에 대한 관리 권한도 있어야 합니다.

아래 절차를 시작하기 전에 CodeCatalyst AWS Management Console 스페이스에 추가하려는 것과 동일한 계정으로 로그인해야 합니다. 그렇지 않으면 콘솔이 알 수 없는 계정 오류를 반환합니다.

CodeCatalyst AWSSRoleForCodeCatalystSupport을 생성하고 추가하려면

1. CodeCatalyst 콘솔에서 시작하기 전예를 열고 스페이스 AWS 계정 에 대해 동일한 로 로그인했는지 AWS Management Console 확인합니다.
2. CodeCatalyst 스페이스로 이동합니다. 설정을 선택한 다음 AWS 계정을 선택합니다.
3. 역할을 생성하려는 AWS 계정 의 링크를 선택합니다. AWS 계정 세부 정보 페이지가 표시됩니다.
4. 역할 관리를 AWS Management Console 선택합니다.

AWS Management Console에서 Amazon CodeCatalyst 스페이스에 IAM 역할 추가 페이지가 열립니다. Amazon CodeCatalyst 스페이스 페이지입니다. 페이지에 액세스하려면 로그인해야 할 수 있습니다.

5. CodeCatalyst 스페이스 세부 정보에서 CodeCatalyst 지원 역할 추가를 선택합니다. 이 옵션은 미리 보기 개발 역할에 대한 권한 정책 및 신뢰 정책을 포함하는 서비스 역할을 생성합니다. 역할에는 고유한 식별자가 추가된 AWSSRoleForCodeCatalystSupport 이름이 있습니다. 역할 및 역할 정책에 대한 자세한 내용은 [AWSSRoleForCodeCatalystSupport 서비스 역할 이해](#) 섹션을 참조하세요.
6. CodeCatalyst 지원에 역할 추가 페이지에서 기본값을 선택한 상태로 두고 역할 생성을 선택합니다.
7. CodeCatalyst에서 사용할 수 있는 IAM 역할에서 계정에 추가된 IAM 역할 목록의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 봅니다.
8. 스페이스로 돌아가려면 Amazon CodeCatalyst로 이동을 선택합니다.

IAM 역할 생성 및 CodeCatalyst 신뢰 정책 사용

CodeCatalyst에서 AWS 계정 연결과 함께 사용할 IAM 역할은 여기에 제공된 신뢰 정책을 사용하도록 구성해야 합니다. 다음 단계를 사용하여 IAM 역할을 생성하고 CodeCatalyst의 블루프린트에서 프로젝트를 생성할 수 있는 정책을 연결합니다.

또는 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할에 대한 권한 정책 및 신뢰 정책이 포함된 서비스 역할을 생성할 수 있습니다. 자세한 내용은 [계정 연결에 IAM 역할 추가](#) 단원을 참조하십시오.

1. 에 로그인 AWS Management Console 하고 <https://console.aws.amazon.com/iam/> IAM 콘솔을 엽니다.
2. 역할을 선택한 다음 역할 생성을 선택합니다.
3. 사용자 지정 신뢰 정책을 선택합니다.

4. 사용자 지정 신뢰 정책 양식에서 다음 신뢰 정책을 붙여넣습니다.

```
"Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "codecatalyst-runner.amazonaws.com",
          "codecatalyst.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:codecatalyst:::space/spaceId/project/*"
        }
      }
    }
  ]
```

5. 다음을 선택합니다.
6. 권한 추가에서 IAM에서 이미 생성한 사용자 지정 정책을 검색하고 선택합니다.
7. 다음을 선택합니다.
8. 역할 이름에 역할 이름을 입력합니다(예시: codecatalyst-project-role).
9. 역할 생성을 선택합니다.
10. Amazon 리소스 이름(ARN) 역할을 복사합니다. 계정 연결 또는 환경에 역할을 추가할 때 이 정보를 제공해야 합니다.

사용자에게 스페이스 권한 부여

스페이스에 참가하는 사용자의 멤버를 확인, 추가, 제거 또는 역할 변경을 통해 스페이스의 멤버를 관리할 수 있습니다.

이 가이드의 정보는 AWS Builder ID 사용자를 지원하는 CodeCatalyst의 스페이스에서 사용자를 초대하고 관리하기 위해 제공됩니다. ID 페더레이션을 지원하는 스페이스를 설정하고 관리하는 단계에 대

해 자세히 알아보려면 Amazon CodeCatalyst 관리자 안내서의 [CodeCatalyst 스페이스 설정 및 관리](#)를 참조하세요.

스페이스에서 멤버 보기

사용자의 표시 이름, 별칭 및 해당 스페이스에 대한 역할에 대한 정보를 포함하여 사용자의 스페이스를 볼 수 있습니다. 스페이스의 멤버에게는 세 가지 역할이 있습니다.

- 스페이스 관리자 - 이 역할에는 프로젝트 생성을 포함하여 CodeCatalyst 모든 권한이 있습니다. 스페이스의 모든 프로젝트에 액세스하는 등 스페이스의 모든 측면을 관리해야 하는 사용자에게만 이 역할을 할당합니다.

사용자를 먼저 제거하지 않으면 나중에 이 역할을 변경할 수 없습니다. 자세한 내용은 [스페이스 관리자 역할](#) 섹션을 참조하세요.

- 파워 유저 - 이 역할은 Amazon CodeCatalyst 스페이스에서 두 번째로 강력한 역할이지만 스페이스의 프로젝트에 액세스할 수는 없습니다. 스페이스 내에 프로젝트를 생성하고 스페이스의 사용자와 리소스를 관리하는 데 도움이 필요한 사용자를 위해 설계되었습니다. 자세한 내용은 [파워 유저 역할](#) 섹션을 참조하세요.
- 제한된 액세스 - 이 역할은 기본적으로 스페이스의 프로젝트에 대한 초대를 수락하여 스페이스에 참여하는 사용자에게 할당됩니다. 프로젝트 멤버는 프로젝트에서 역할이 할당됩니다. 프로젝트 멤버 관리에 대한 자세한 내용은 [사용자에게 프로젝트 권한 부여](#) 섹션을 참조하세요.

스페이스 관리자 테이블에는 스페이스 관리자 역할을 지닌 사용자가 표시됩니다. 이러한 사용자는 스페이스의 모든 프로젝트에 자동으로(암시적으로) 할당되고 프로젝트에서 역할이 없기 때문에 스페이스 멤버에 표시되지 않습니다.

스페이스 멤버 테이블에는 스페이스 관리자 역할은 없지만 프로젝트에서 역할이 있는 스페이스의 모든 멤버가 표시됩니다.

사용자가 CodeCatalyst에서 스페이스 관리자 역할을 가지고 있는지 여부에 따라 다음과 같이 표시됩니다.

- 나중에 프로젝트 초대 및 역할을 수락하는 스페이스 관리자 역할을 가진 사용자는 스페이스 멤버 테이블의 공백 또는 프로젝트 아래의 프로젝트 멤버 테이블에 표시되지 않습니다. 두 위치의 스페이스 관리자 테이블에 계속 표시됩니다. 각 프로젝트에서 스페이스 관리자 역할을 가진 모든 사용자는 해당 프로젝트의 프로젝트 스페이스 관리자 테이블에 표시됩니다.
- 프로젝트 역할로 참여하라는 프로젝트 초대를 수락하는 사용자는 제한된 액세스 역할이 있는 스페이스에 추가됩니다. 나중에 사용자의 역할이 스페이스 관리자 역할로 변경되는 경우에도 스페이스

멤버 테이블에서 스페이스 관리자 테이블로 이동합니다. 프로젝트에서 사용자는 프로젝트 멤버 테이블에서 스페이스 관리자 테이블로 이동합니다.

스페이스의 사용자 및 역할 보기

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다.

Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

3. 설정을 선택한 다음 멤버를 선택합니다.

스페이스의 멤버인 사용자는 스페이스 멤버 테이블에 표시됩니다.

Tip

스페이스 관리자 역할이 있는 경우 직접 초대된 프로젝트를 볼 수 있습니다. 프로젝트의 프로젝트 설정으로 이동한 다음, 내 프로젝트를 선택합니다.

상태 열에 유효한 값은 다음과 같습니다.

- 초대됨 - CodeCatalyst가 초대를 보냈지만 사용자가 아직 수락하거나 거부하지 않았습니다.
- 멤버 - 사용자가 초대를 수락했습니다.

사용자를 직접 스페이스에 초대

CodeCatalyst 스페이스에 사용자를 직접 초대할 수 있습니다. 해당 사용자에게 스페이스 관리자 또는 파워 유저 역할을 할당하여 스페이스 관리를 도와주도록 초대하려는 경우에 유용합니다. 이러한 역할 중 하나를 다른 사용자에게 할당하면 해당 사용자를 프로젝트에 초대하지 않고도 더 많은 사람에게 스페이스 관리 책임을 분산할 수 있습니다.

Note

멤버를 초대하려면 스페이스 관리자 또는 파워 유저 역할이 있어야 합니다.

스페이스 관리자 테이블에는 스페이스 관리자 역할을 지닌 사용자가 표시됩니다. 이러한 사용자는 스페이스의 모든 프로젝트에 자동으로(암시적으로) 할당되고 프로젝트에서 역할이 없기 때문에 스페이스 멤버 테이블에 표시되지 않습니다.

프로젝트 초대를 수락한 멤버는 기본적으로 스페이스에 추가됩니다. 프로젝트 멤버 테이블에는 프로젝트에서 역할을 가진 모든 스페이스 멤버가 표시됩니다.

초대를 수락하고 처음 로그인하는 방법에 대한 자세한 내용은 [CodeCatalyst 설정 및 로그인](#) 섹션을 참조하세요.

스페이스에 사용자 초대

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다.
3. 설정을 선택한 다음 멤버를 선택합니다.
4. 초대를 선택합니다.
5. 스페이스에 초대하려는 사람의 이메일을 입력합니다. 역할에서 해당 사용자에게 할당할 역할을 선택합니다.
6. 초대를 선택합니다.

스페이스 초대 취소

최근에 보낸 스페이스 참여 초대를 취소하고 싶지만 아직 수락되지 않은 경우 취소할 수 있습니다.

스페이스 초대를 관리하려면 스페이스 관리자 또는 파워 유저 역할이 있어야 합니다.

스페이스 멤버 초대 취소

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다.

i Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

3. 설정을 선택한 다음 멤버를 선택합니다.
4. 멤버의 상태가 초대됨인지 확인합니다.

i Note

아직 수락되지 않은 초대만 취소할 수 있습니다.

5. 초대된 멤버가 있는 행 옆의 옵션을 선택한 다음 초대 취소를 선택합니다.
6. 확인 창이 표시됩니다. 초대 취소를 선택하여 확인합니다.

스페이스 멤버의 역할 변경

스페이스 멤버에게 할당된 역할을 변경할 수 있습니다. 스페이스에서 사용자의 역할을 변경하려면 스페이스 관리자 역할이 있어야 합니다.

스페이스 관리자 테이블에는 스페이스 관리자 역할을 지닌 사용자가 표시됩니다. 이러한 사용자는 스페이스의 모든 프로젝트에 자동으로(암시적으로) 할당되기 때문에 스페이스 멤버 테이블에 표시되지 않습니다.

스페이스에서 사용자 역할 변경

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다.

i Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

3. 설정을 선택한 다음 멤버를 선택합니다.
4. 스페이스 멤버 테이블에서 역할을 변경할 사용자를 선택합니다. 역할 변경을 선택합니다.

스페이스 멤버 제거

스페이스 멤버가 스페이스 리소스에 액세스할 필요가 없는 경우 해당 멤버를 제거할 수 있습니다. 스페이스에서 멤버를 제거하려면 스페이스 관리자 역할이 있어야 합니다.

스페이스 관리자 테이블에는 스페이스 관리자 역할을 지닌 사용자가 표시됩니다. 이러한 사용자는 스페이스의 모든 프로젝트에 자동으로(암시적으로) 할당되고 프로젝트에서 역할이 없기 때문에 스페이스 멤버 테이블에 표시되지 않습니다. 이 테이블에서는 공간의 멤버만 직접 제거할 수 있습니다.

프로젝트 멤버 테이블에서 사용자 제거

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다.

Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

3. 설정을 선택한 다음 멤버를 선택합니다.
4. 프로젝트 멤버 테이블에서 사용자를 선택합니다. 제거를 선택합니다.

Note

스페이스에서 멤버를 제거하면 해당 프로젝트의 리소스와 관련된 권한과 함께 스페이스의 모든 프로젝트에서 해당 사용자가 제거됩니다.

스페이스 관리자 역할이 있는 사용자의 역할 제거 또는 변경

스페이스 관리자 역할이 있는 사용자의 역할을 제거하거나 변경할 수 있습니다.

스페이스 관리자 역할이 있는 사용자를 스페이스에서 제거하려면 스페이스 관리자 역할이 있어야 합니다. 스페이스 관리자 역할이 있는 사용자의 역할을 변경하면 기본적으로 해당 사용자가 스페이스 관리자 테이블에서 제거됩니다. 해당 사용자가 스페이스의 어떤 프로젝트에서도 프로젝트 역할이 없는 경우, 사용자로부터 스페이스 관리자 역할을 제거하면 스페이스에서 해당 사용자가 제거됩니다.

Note

스페이스 관리자 역할이 있는 사용자는 자신을 제거할 수 없습니다. 스페이스 관리자 역할이 있는 다른 사용자에게 문의하세요.

스페이스 멤버 테이블에서 스페이스 관리자 역할이 있는 사용자 제거

Note

프로젝트에 명시적으로 추가되지 않은 사용자의 경우 프로젝트 역할(프로젝트 관리자 또는 기여자)이 없습니다. 스페이스 관리자 역할이 사용자의 유일한 역할인 경우 해당 사용자는 스페이스에서 완전히 제거됩니다.

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스 관리자 역할이 있는 사용자의 역할을 제거하거나 변경하려는 스페이스로 이동합니다.
3. 설정을 선택한 다음 멤버를 선택합니다.
4. 멤버 목록의 초대 상태를 보고 스페이스에 대한 무단 보류 중인 초대가 목록에 없는지 확인합니다 (초대됨 상태).

Important

스페이스 관리자 역할이 있는 사용자를 제거하기 전에 보류 중인 초대가 시작되지 않았는지 확인해야 합니다.

5. 멤버 탭을 선택합니다. 스페이스 관리자 테이블에서 사용자를 선택한 다음 제거를 선택합니다.

멤버 제거 대화 상자에서 다음 중 하나를 수행합니다.

- 사용자의 스페이스 관리자 역할만 제거하려면 옵션을 선택합니다. 제거를 선택합니다.

Important

사용자에게 다른 역할이 할당되어 있지 않은 경우 스페이스 관리자에서 역할을 변경하면 해당 스페이스에서 사용자가 제거됩니다.

- 스페이스 관리자 역할이 있는 사용자를 스페이스 및 모든 프로젝트에서 제거하는 옵션을 선택합니다. 제거를 선택합니다.
6. 멤버 탭을 새로 고칩니다. 사용자는 프로젝트 역할을 통해 멤버십이 있는 모든 프로젝트의 프로젝트 멤버 목록에 자동으로 추가됩니다. 스페이스 관리자 역할이 사용자의 유일한 역할인 경우 해당 사용자는 스페이스에서 완전히 제거됩니다.

팀을 사용하여 스페이스 액세스 허용

스페이스를 생성한 후 팀을 추가할 수 있습니다. 팀을 사용하면 CodeCatalyst에서 권한을 공유하고 프로젝트, 문제 추적, 역할 및 리소스를 관리할 수 있도록 사용자를 그룹화할 수 있습니다.

팀을 관리하려면 스페이스 관리자 역할이 있어야 합니다.

또한 팀은 CodeCatalyst의 프로젝트/스페이스 수준에서 관리됩니다. 스페이스/프로젝트의 팀에 대한 자세한 내용은 [팀을 사용하여 스페이스 액세스 허용](#) 섹션을 참조하세요.

주제

- [팀 생성](#)
- [팀 보기](#)
- [팀에 스페이스 역할 부여](#)
- [스페이스 수준에서 팀에 프로젝트 역할 부여](#)
- [팀에 직접 사용자 추가](#)
- [팀에서 사용자 직접 제거](#)
- [팀에 SSO 그룹 추가](#)
- [팀 삭제](#)

팀 생성

팀은 한 스페이스에 파워 유저와 같은 역할 권한을 가질 수 있습니다. 또한 팀은 프로젝트 관리자와 같은 프로젝트 권한을 프로젝트에서 가질 수 있습니다. 팀은 각 프로젝트에 대한 다양한 역할이 있는 많은 프로젝트와 연결할 수 있습니다. 팀원이 AWS Builder ID 스페이스의 개별 사용자이거나 자격 증명 페더레이션을 지원하는 스페이스의 SSO 그룹인 팀을 관리할 수 있습니다.

스페이스 및 프로젝트 사용자의 멤버 페이지에서 사용자는 여러 역할을 가질 수 있습니다. 여러 역할을 지닌 사용자에게는 역할이 여러 개일 때 보이는 표시가 나타나며, 가장 많은 권한을 가진 역할이 먼저 표시됩니다.

Note

스페이스가 ID 페더레이션을 지원하는 경우, 이미 SSO 사용자 또는 SSO 그룹이 IAM Identity Center에 설정되어 있어야 합니다.

팀원을 관리하는 방법은 사용자를 추가하고 제거하는 방법에 따라 달라집니다. 팀원을 관리하기 위한 두 가지 옵션이 있습니다.

- 사용자 직접 추가 - 사용자를 개별적으로 추가하거나 제거합니다. 예를 들어, IAM Identity Center에 이미 설정된 AWS Builder ID 사용자 또는 SSO 사용자를 선택하여 팀에 사용자를 추가합니다. AWS Builder ID 사용자 또는 SSO 사용자를 직접 추가하여 팀원을 관리하도록 선택하면 SSO 그룹을 사용하는 옵션을 더 이상 사용할 수 없습니다.
- SSO 그룹 사용 - IAM Identity Center에 이미 설정된 SSO 그룹을 통해 팀원을 관리합니다. SSO 그룹을 사용하여 팀원을 관리하도록 선택하면, 사용자를 직접 추가하는 옵션을 더 이상 사용할 수 없습니다.

팀을 관리하려면 스페이스 관리자 역할이 있어야 합니다.

팀을 생성하는 방법

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다. 설정을 선택한 팀을 선택합니다.
3. 풀 생성을 선택합니다.
4. 팀 이름에 팀을 설명하는 이름을 입력합니다.

Note

팀 이름은 스페이스 내에서 고유해야 합니다.

(선택 사항) 팀 설명에 팀에 대한 설명을 입력합니다.

5. 스페이스 역할에서 CodeCatalyst에서 사용할 수 있는 스페이스 역할 목록에서 팀에 할당할 역할을 선택합니다. 역할은 팀의 모든 멤버가 상속합니다.
 - 스페이스 관리자 - 자세한 내용은 [스페이스 관리자 역할](#) 섹션을 참조하세요.
 - 제한된 액세스 - 자세한 내용은 [제한된 액세스 역할](#) 섹션을 참조하세요.

- 파워 유저 - 자세한 내용은 [파워 유저 역할](#) 섹션을 참조하세요.
6. 팀 멤버십에서 다음 중 하나를 선택하여 팀에 멤버를 추가하는 방법을 선택합니다.
- 사용자를 개별적으로 관리하려면 멤버 직접 추가를 선택합니다. 여기에는 스페이스에 AWS Builder ID 사용자를 추가하거나 자격 증명 페더레이션을 지원하는 스페이스에 SSO 사용자를 추가하는 것이 포함됩니다.
 - SSO 그룹 사용을 선택하여 IAM Identity Center에서 이미 설정한 SSO 그룹을 선택합니다.

SSO 그룹에서 추가하려는 그룹 옆의 상자를 선택합니다. 최대 5개의 SSO 그룹을 추가할 수 있습니다.

Note

이 이름은 나중에 변경할 수 없습니다. AWS Builder ID 사용자 또는 SSO 사용자를 직접 추가하여 팀원을 관리하도록 선택하면, SSO 그룹을 사용하는 옵션을 더 이상 사용할 수 없습니다. SSO 그룹을 사용하여 팀원을 관리하도록 선택하면, 사용자를 직접 추가하는 옵션을 더 이상 사용할 수 없습니다.

7. 생성(Create)을 선택합니다.

Note

SSO 그룹을 사용하도록 선택하면, 팀을 생성할 때 SSO 그룹의 사용자를 불러올 수 없습니다. 사용자는 CodeCatalyst에 로그인해야 그룹이 목록에 표시됩니다.

팀 보기

CodeCatalyst에서 팀의 프로젝트와 역할을 볼 수 있습니다. 멤버 페이지에서 프로젝트 역할과 사용자 목록을 볼 수 있습니다. SSO 그룹 유형 팀의 경우, 팀과 연결된 SSO 그룹 목록도 볼 수 있습니다.

팀을 보려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다. 설정을 선택한 팀을 선택합니다.
3. 스페이스 역할 에서 이 스페이스에 대해 팀에 할당된 역할을 봅니다.

4. 프로젝트 역할 탭에서 팀이 멤버로 추가된 스페이스의 각 CodeCatalyst 프로젝트에 대해 팀에 할당된 프로젝트 및 프로젝트 역할을 확인합니다(AWS 빌더 ID 스페이스만 해당).
5. 멤버 탭에서 팀에 할당된 멤버 목록을 확인합니다.
6. SSO 그룹 탭에서 팀에 할당된 SSO 그룹 목록을 봅니다(ID 페더레이션만 지원하는 스페이스의 경우).

팀에 스페이스 역할 부여

팀은 CodeCatalyst의 프로젝트에 대한 팀 액세스 권한을 부여하고 관리할 수 있도록 사용자를 그룹화하는 방법입니다. 예를 들어, 팀을 사용하여 팀에 사용자를 위한 스페이스 관리 기능을 제공하여 사용자의 역할과 권한을 빠르게 관리할 수 있습니다.

팀은 한 스페이스에 파워 유저와 같은 역할 권한을 가질 수 있습니다. 팀의 스페이스 역할은 변경할 수 있지만, 팀의 모든 멤버가 해당 권한을 상속한다는 점에 유의해야 합니다.

팀을 관리하려면 스페이스 관리자 역할이 있어야 합니다.

팀의 스페이스 역할 변경

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다. 설정을 선택한 팀을 선택합니다.
3. 작업에서 스페이스 역할 변경 선택합니다. 스페이스 역할을 다음 중 하나로 변경할 수 있습니다. 이렇게 하면 팀의 모든 멤버의 역할이 변경됩니다.
 - 스페이스 관리자 - 자세한 내용은 [스페이스 관리자 역할](#) 섹션을 참조하세요.
 - 제한된 액세스 - 자세한 내용은 [제한된 액세스 역할](#) 섹션을 참조하세요.
 - 파워 유저 - 자세한 내용은 [파워 유저 역할](#) 섹션을 참조하세요.
4. 저장을 선택합니다.

스페이스 수준에서 팀에 프로젝트 역할 부여

CodeCatalyst의 팀은 프로젝트에서 프로젝트 관리자와 같은 역할 권한을 팀원이 가질 수 있다는 점에서 사용자와 유사합니다. 역할 변경이 팀에 적용되며 팀의 모든 멤버가 해당 권한을 상속합니다. 팀에 자동으로 부여될 각 프로젝트에 대해 하나의 역할을 선택할 수 있습니다.

팀을 관리하려면 스페이스 관리자 역할이 있어야 합니다.

프로젝트 역할을 추가하거나 변경하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다. 설정을 선택한 팀을 선택합니다.
3. 프로젝트 역할 탭을 선택합니다.
4. 역할을 변경하려면 이 목록의 프로젝트 옆에 있는 선택기를 선택한 다음 역할 변경을 선택합니다. 역할을 추가하려면 프로젝트 역할 추가를 선택합니다. 프로젝트에서 추가할 프로젝트를 선택하고 역할에서 역할을 선택합니다. 사용 가능한 프로젝트 역할 중 하나를 선택합니다.
 - 프로젝트 관리자 - 자세한 내용은 [프로젝트 관리자 역할](#) 섹션을 참조하세요.
 - 기고자 - 자세한 내용은 [기고자 역할](#) 섹션을 참조하세요.
 - 검토자 - 자세한 내용은 [검토자 역할](#) 섹션을 참조하세요.
 - 읽기 전용 - 자세한 내용은 [읽기 전용 역할](#) 섹션을 참조하세요.
5. 저장을 선택합니다.

프로젝트 역할을 제거하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다. 설정을 선택한 팀을 선택합니다.
3. 프로젝트 역할 탭을 선택합니다.
4. 제거할 역할을 선택합니다.

Important

팀에서 역할을 제거하면 팀의 모든 사용자가 지닌 관련 권한이 제거됩니다.

5. 저장을 선택합니다.

팀에 직접 사용자 추가

팀에 팀원을 추가할 수 있습니다. 사용자를 추가하면 새 사용자는 팀의 모든 기존 역할에서 권한을 상속합니다.

스페이스가 AWS Builder ID 사용자 지원을 위해 설정되었든 자격 증명 페더레이션을 위해 설정되었든 상관없이 사용자를 직접 추가하도록 스페이스를 설정할 수 있습니다.

Note

SSO 그룹을 사용하여 팀원을 관리하도록 스페이스를 설정하면 사용자 직접 추가를 사용하는 옵션을 사용할 수 없습니다. SSO 그룹을 사용하려면 [팀에 SSO 그룹 추가](#) 섹션을 참조하세요.

팀을 관리하려면 스페이스 관리자 역할이 있어야 합니다.

사용자를 직접 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다. 설정을 선택한 팀을 선택합니다.
3. 멤버 탭을 선택합니다.
4. 멤버 추가를 선택하세요.

Note

팀에 추가되는 사용자는 이미 스페이스의 멤버여야 합니다. 스페이스의 멤버가 아닌 팀원은 추가하거나 초대할 수 없습니다.

5. 드롭다운 필드에서 사용자를 선택한 다음 저장을 선택합니다. IAM Identity Center에 이미 설정된 AWS Builder ID 사용자 또는 SSO 사용자를 선택합니다.

팀에서 사용자 직접 제거

팀에서 팀원을 제거할 수 있습니다. 모든 권한은 더 이상 사용자에게 상속되지 않습니다. 나중에 제거한 사용자를 팀에 다시 추가할 수 있습니다.

Note

팀원을 제거하면 스페이스의 모든 프로젝트 및 리소스에서 사용자에게 대해 관련 권한이 제거됩니다.

팀을 관리하려면 스페이스 관리자 역할이 있어야 합니다.

팀원을 제거하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다. 설정을 선택한 팀을 선택합니다.
3. 멤버 탭을 선택합니다.
4. 제거하려는 사용자 옆의 선택기를 선택한 다음 제거 선택합니다.
5. 텍스트 입력 필드에 remove를 입력하고 제거를 선택합니다.

팀에 SSO 그룹 추가

스페이스가 IAM Identity Center에서 관리되는 SSO 사용자 및 그룹이 있는 스페이스로 구성된 경우, 해당 스페이스에 별도의 팀으로 조인할 SSO 그룹을 추가할 수 있습니다.

Note

AWS Builder ID 사용자 또는 SSO 사용자를 직접 추가하여 팀원을 관리하도록 선택하면 SSO 그룹을 사용하는 옵션을 사용할 수 없습니다. 사용자를 직접 추가하려면 [팀에 직접 사용자 추가](#) 섹션을 참조하세요.

팀을 관리하려면 스페이스 관리자 역할이 있어야 합니다.

SSO 그룹을 팀으로 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스 페이지에서 팀을 선택합니다. SSO 그룹 탭을 선택합니다.
3. 추가할 SSO 그룹을 선택합니다. 최대 5개의 SSO 그룹을 추가할 수 있습니다.

팀 삭제

더 이상 필요 없는 팀은 삭제할 수 있습니다.

Note

팀을 삭제하면 스페이스의 모든 프로젝트 및 리소스에서 모든 팀원에 대해 관련 권한이 제거됩니다.

팀을 관리하려면 스페이스 관리자 역할이 있어야 합니다.

팀 삭제

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다. 설정을 선택한 팀을 선택합니다.
3. 작업에서 팀 삭제를 선택합니다. 이렇게 하면 전체 팀의 역할이 변경됩니다.
4. 삭제를 선택합니다.

시스템 리소스에 대한 스페이스 액세스 허용

시스템 리소스는 CodeCatalyst의 프로젝트 또는 스페이스에 대한 권한이 부여된 CodeCatalyst의 특정 리소스입니다.

Note

시스템 리소스라는 용어는 Amazon EC2 인스턴스와 같은 클라우드 인프라를 가리키는 것이 아니라 스페이스 또는 프로젝트에 대한 권한이 있는 블루프린트 또는 워크플로 리소스를 의미합니다.

시스템 리소스는 SSO를 통해 CodeCatalyst에 액세스할 때 승인된 리소스의 ID를 나타냅니다. 시스템 리소스는 블루프린트 및 워크플로와 같은 스페이스의 리소스에 권한을 부여하는 데 사용됩니다. 스페이스의 시스템 리소스는 확인할 수 있으며 스페이스의 시스템 리소스를 활성화하거나 비활성화하도록 선택할 수 있습니다. 예를 들어, 액세스를 관리하기 위해 시스템 리소스를 비활성화했다가 나중에 다시 활성화할 수도 있습니다.

이러한 작업은 시스템 리소스를 취소하거나 비활성화해야 하는 경우 시스템 리소스에 사용할 수 있습니다. 예를 들어 자격 증명에 손상되었을 수 있다고 의심되는 경우 시스템 리소스를 비활성화할 수 있습니다. 일반적으로 이러한 작업은 사용할 필요가 없습니다.

이 페이지를 보고 스페이스 수준에서 시스템 리소스를 관리하려면 스페이스 관리자 역할이 있어야 합니다.

또한 시스템 리소스는 CodeCatalyst의 프로젝트 수준에서 관리됩니다. 프로젝트의 팀에 대한 자세한 설명은 [시스템 리소스에 대한 스페이스 액세스 허용](#) 섹션을 참조하세요.

주제

- [시스템 리소스에 대한 스페이스 액세스 보기](#)
- [시스템 리소스에 대한 스페이스 액세스 비활성화](#)
- [시스템 리소스에 대한 스페이스 액세스 활성화](#)

시스템 리소스에 대한 스페이스 액세스 보기

스페이스에서 사용 중인 시스템 리소스 목록을 볼 수 있습니다.

시스템 리소스를 관리하려면 스페이스 관리자 역할이 있어야 합니다.

시스템 리소스를 보려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동한 다음 설정을 선택합니다. 시스템 리소스를 선택합니다.
3. 드롭다운에서 워크플로 작업을 선택하여 워크플로에 대한 시스템 리소스만 봅니다. 블루프린트에 대한 시스템 리소스만 보려면 블루프린트를 선택합니다.

필터 필드를 사용하여 이름을 필터링할 수도 있습니다.

시스템 리소스에 대한 스페이스 액세스 비활성화

스페이스에서 사용 중인 시스템 리소스를 비활성화하도록 선택할 수 있습니다.

Important

시스템 리소스를 비활성화하면 스페이스의 모든 관련 블루프린트 또는 워크플로에 대한 모든 권한이 제거됩니다.

시스템 리소스를 관리하려면 스페이스 관리자 역할이 있어야 합니다.

시스템 리소스를 비활성화하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동한 다음 설정을 선택합니다. 시스템 리소스를 선택합니다.
3. 다음 중 하나를 선택합니다.

⚠ Important

시스템 리소스를 비활성화하면 스페이스의 모든 관련 블루프린트 또는 워크플로에 대한 모든 권한이 제거됩니다.

- 개별적으로 비활성화하려면, 비활성화하려는 하나 이상의 시스템 리소스 옆에 있는 선택기를 선택합니다. 비활성화를 선택한 다음 이 리소스를 선택합니다.
- 모든 리소스를 비활성화하려면 비활성화를 선택한 다음 모든 리소스를 선택합니다.
- 모든 워크플로 작업을 비활성화하려면 비활성화를 선택한 다음 모든 워크플로 작업을 선택합니다.
- 모든 블루프린트를 비활성화하려면 비활성화를 선택한 다음 모든 블루프린트를 선택합니다.

시스템 리소스에 대한 스페이스 액세스 활성화

스페이스에서 사용 중이고 비활성화된 시스템 리소스를 활성화하도록 선택할 수 있습니다.

시스템 리소스를 관리하려면 스페이스 관리자 역할이 있어야 합니다.

시스템 리소스를 활성화하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동한 다음 설정을 선택합니다. 시스템 리소스를 선택합니다.
3. 다음 중 하나를 선택합니다.
 - 개별적으로 활성화하려면 활성화하려는 하나 이상의 시스템 리소스 옆에 있는 선택기를 선택합니다. 활성화를 선택한 다음 이 리소스를 선택합니다.
 - 모든 리소스를 활성화하려면 활성화를 선택한 다음 모든 리소스를 선택합니다.
 - 모든 워크플로 작업을 활성화하려면 활성화를 선택한 다음 모든 워크플로 작업을 선택합니다.
 - 모든 블루프린트를 활성화하려면 활성화를 선택한 다음 모든 블루프린트를 선택합니다.

스페이스의 개발 환경 관리

모든 개발 환경은 스페이스 내에서 프로젝트의 일부로 생성됩니다. 스페이스 멤버는 프로젝트 내에서 소스 리포지토리 수준에서 자신만의 개발 환경을 만들 수 있습니다. 그러면 스페이스 관리자는

Amazon CodeCatalyst 콘솔을 사용하여 스페이스 멤버를 대신하여 개발 환경을 보고, 편집하고, 삭제하고, 중지할 수 있습니다. 간단히 말해, 스페이스 관리자는 스페이스 수준에서 개발 환경을 유지 관리합니다.

개발 환경 관리에 대한 고려 사항

- 설정에서 개발 환경 페이지를 보고 스페이스 수준에서 개발 환경을 관리하려면 스페이스 관리자 역할이 있어야 합니다.
- 스페이스 멤버는 자신의 CodeCatalyst 계정을 통해 프로젝트에서 생성한 개발 환경을 관리합니다. 스페이스 관리자로서 개발 환경을 관리할 때는 스페이스 멤버를 대신하여 이러한 리소스를 유지 관리하게 됩니다.
- 개발 환경은 기본적으로 특정 컴퓨팅 및 스토리지 구성으로 설정됩니다. 구성 업그레이드에 대한 청구 및 요금에 대한 자세한 내용은 [Amazon CodeCatalyst 가격 페이지](#)를 참조하세요.

Important

Active Directory가 ID 제공업체로 사용되는 스페이스의 사용자는 개발 환경을 사용할 수 없습니다. 자세한 내용은 [Single Sign-On 계정을 사용하여 CodeCatalyst에 로그인할 때는 개발 환경을 생성할 수 없습니다](#) 섹션을 참조하세요.

인스턴스 실행 중지, 기본 컴퓨팅 구성, 컴퓨팅 업그레이드, 비용 발생, 제한 시간 구성 등 개발 환경에 대한 기타 고려 사항은 [CodeCatalyst에서 개발 환경으로 코드 작성 및 수정](#) 섹션을 참조하세요.

주제

- [스페이스의 개발 환경 보기](#)
- [스페이스의 개발 환경 편집](#)
- [스페이스의 개발 환경 중지](#)
- [스페이스의 개발 환경 삭제](#)

스페이스의 개발 환경 보기

스페이스에 있는 모든 개발 환경의 유형, 상태 및 세부 정보를 볼 수 있습니다. 개발 환경 생성 및 실행에 대한 자세한 내용은 [개발 환경 생성](#) 섹션을 참조하세요.

이 페이지를 보고 스페이스 수준에서 개발 환경을 관리하려면 스페이스 관리자 역할이 있어야 합니다.

스페이스의 개발 환경 보기

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다.

Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

3. 설정을 선택한 다음 개발 환경을 선택합니다.

이 페이지에는 스페이스의 모든 개발 환경이 나열됩니다. 리소스 이름, 해당되는 경우 리소스 별칭, IDE 유형, 기본값 또는 구성된 컴퓨팅 및 스토리지, 각 개발 환경에 대해 구성된 제한 시간을 볼 수 있습니다.

스페이스의 개발 환경 편집

유휴 개발 환경의 실행을 중지하기 위해 구성된 제한 시간(있는 경우)과 같은 개발 환경의 구성을 편집할 수 있습니다. 개발 환경 편집에 대한 자세한 내용은 [개발 환경 편집](#) 섹션을 참조하세요.

이 페이지를 보고 스페이스 수준에서 개발 환경을 관리하려면 스페이스 관리자 역할이 있어야 합니다.

스페이스의 개발 환경 편집

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다.

Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

3. 설정을 선택한 다음 개발 환경을 선택합니다.
4. 관리하려는 개발 환경 옆의 선택기를 선택합니다. 편집을 선택합니다.
5. 개발 환경의 컴퓨팅 또는 비활성 시간 제한을 원하는 대로 변경합니다.
6. 저장을 선택합니다.

스페이스의 개발 환경 중지

개발 환경이 제한 시간을 갖도록 구성된 경우 실행 중인 개발 환경이 유휴 상태가 되기 전에 중지할 수 있습니다. 그렇지 않으면 제한 시간이 경과한 개발 환경이 이미 중지됩니다. 개발 환경 중지 에 대한 자세한 내용은 [개발 환경 중지](#) 섹션을 참조하세요.

이 페이지를 보고 스페이스 수준에서 개발 환경을 관리하려면 스페이스 관리자 역할이 있어야 합니다.

스페이스에서 개발 환경 중지

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다.

Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

3. 설정을 선택한 다음 개발 환경을 선택합니다.
4. 관리하려는 개발 환경 옆의 선택기를 선택합니다. 중지를 선택합니다.

스페이스의 개발 환경 삭제

더 이상 필요하지 않거나 더 이상 소유자가 없는 개발 환경을 삭제할 수 있습니다. 개발 환경 삭제 고려 사항에 대한 자세한 내용은 [개발 환경 삭제](#) 섹션을 참조하세요.

이 페이지를 보고 스페이스 수준에서 개발 환경을 관리하려면 스페이스 관리자 역할이 있어야 합니다.

스페이스의 개발 환경 삭제

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다.

Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

3. 설정을 선택한 다음 개발 환경을 선택합니다.

4. 관리하려는 개발 환경 옆의 선택기를 선택합니다. 삭제를 선택합니다. 확인하려면 delete를 입력하여 확인한 후, 삭제를 선택합니다.

스페이스 할당량

다음 테이블에서는 Amazon CodeCatalyst의 공간 할당량 및 제한에 대해 설명합니다. Amazon CodeCatalyst의 할당량에 대한 자세한 내용은 [CodeCatalyst 할당량](#) 섹션을 참조하세요.

스페이스의 최대 Slack 채널 수	500
이메일 주소에 보낼 수 있는 최대 초대 수	25
사용자에게 보낼 수 있는 최대 초대 수	500
당 사용자당 최대 활성 공간 수 AWS 리전	5
리전당 한 명의 사용자에게 허용된 월간 최대 스페이스 생성 수	5
팀의 최대 SSO 그룹 수	5
스페이스의 최대 팀 수	100
팀의 최대 사용자 수	1000
스페이스 설명	스페이스 설명은 선택 사항입니다. 지정할 경우, 설명의 길이는 0~200자여야 합니다. 문자, 숫자, 공백, 마침표, 밑줄, 쉼표, 대시 및 다음과 같은 특수 문자의 조합을 포함할 수 있습니다. ? & \$ % + = / \ ; : \n \t \r
스페이스 이름	스페이스 이름은 CodeCatalyst 전체에서 고유해야 합니다. 삭제된 스페이스의 이름은 재사용할 수 없습니다. 스페이스 이름은 3~63자 이내로 작성해야 합니다. 또한 영숫자 문자로 시작해야 합니다. 스페이스 이름은 문자, 숫자, 마침표, 대시 및 밑줄의

조합을 포함할 수 있습니다. 이름에는 다음 문자를 포함할 수 없습니다.

! ? @ # \$ % ^ & * () + = { } []
| / \ > < ~ ` ' " ; :

CodeCatalyst에서 프로젝트 작업 구성

Amazon CodeCatalyst의 프로젝트를 사용하여 개발 팀이 공유된 지속적 통합/지속적 전달(CI/CD) 워크플로 및 리포지토리를 사용하여 개발 작업을 수행할 수 있는 협업 공간을 구축합니다. 프로젝트를 생성한 후 리소스를 추가, 업데이트 또는 제거할 수 있습니다. 팀의 작업 진행 상황을 모니터링할 수도 있습니다. 스페이스 내에 여러 프로젝트를 포함할 수 있습니다.

CodeCatalyst의 스페이스는 프로젝트로 구성됩니다. 스페이스 내의 모든 프로젝트를 볼 수 있지만 멤버인 프로젝트만 사용할 수 있습니다. 프로젝트를 생성하면 프로젝트의 기본 역할이 생성되어 프로젝트에 초대하는 사용자에게 할당합니다.

- 기고자 역할과 같이 프로젝트 역할이 있는 프로젝트에 할당된 모든 사용자는 소스 리포지토리와 같은 프로젝트 리소스에 액세스할 수 있습니다.
- 스페이스 관리자 또는 프로젝트 관리자 역할이 있는 사람은 누구나 프로젝트 참여 초대장을 보낼 수 있습니다.
- 프로젝트 관리자 역할을 가진 사용자는 공유 리소스에서 활동, 상태 및 기타 설정을 추적할 수 있습니다.
- 제한된 액세스 역할을 지닌 사용자는 CI/CD 워크플로의 일부로서 기능, 코드 수정 및 테스트에 대한 프로젝트 할당을 관리할 수 있습니다.

워크플로는 CI/CD 파이프라인으로서 애플리케이션을 빌드, 테스트, 릴리스 또는 업데이트하는 데 사용됩니다. 소스 아티팩트를 전송하고 작업하는 작업을 추가하여 워크플로를 조합할 수 있습니다. 작업을 실행하면, 프로젝트 클라우드 리소스는 워크플로 작업에 온디맨드 컴퓨팅 기능을 제공하는 데 사용됩니다. 설정하려는 활동 및 출력에 따라 더 많은 CI/CD 워크플로를 구성할 수 있습니다. 예를 들어, 빌드 및 테스트 작업에 대해서만 워크플로를 생성할 수 있습니다. 여기서 버그를 수정하는 동안 배포 없이 테스트 결과를 보고 워크플로를 완료할 수 있습니다. 그런 다음 다른 워크플로를 생성하여 애플리케이션을 빌드하고 스테이징 환경에 배포할 수 있습니다.

프로젝트를 생성할 때 블루프린트를 사용하여 샘플 코드를 포함하거나 리소스를 생성하는 프로젝트를 생성하거나 빈 프로젝트로 시작할 수 있습니다. 블루프린트를 사용하여 프로젝트를 생성하는 경우, 선택한 블루프린트에 따라 프로젝트에 추가되는 리소스와 CodeCatalyst가 생성하거나 구성하는 도구가 결정되어 프로젝트 리소스를 추적하고 사용할 수 있습니다. 프로젝트를 생성한 후 수동으로 리소스를 추가 또는 제거할 수 있습니다.

각 프로젝트는 프로젝트가 생성되거나 리소스가 수정되는 경우와 같이 사용자별 이벤트 목록으로 프로젝트 활동을 추적합니다. 프로젝트 활동은 스페이스 수준에서 모니터링 및 집계됩니다. 활동 데이터 작업에 대한 자세한 내용은 [한 스페이스에서 모든 프로젝트 보기](#) 섹션을 참조하세요.

프로젝트에서 AWS 리소스를 사용하는 경우 CodeCatalyst 계정을 AWS 프로젝트의 리소스를 통합할 관리 권한이 있는 계정에 연결할 수 있습니다.

소스 리포지토리, 문제 및 기타 리소스를 생성한 후 프로젝트에 추가할 수 있습니다. 프로젝트를 생성하려면 스페이스 관리자 역할이 있어야 합니다.

프로젝트 생성

CodeCatalyst 프로젝트를 사용하면 공유된 지속적 통합/지속적 전송(CI/CD) 워크플로 및 리포지토리를 사용하여 개발 작업을 수행하고, 리소스를 관리하며, 문제를 추적하고, 사용자를 추가할 수 있습니다.

프로젝트를 생성하기 전에 스페이스 관리자 또는 파워 유저 역할이 있어야 합니다.

주제

- [Amazon CodeCatalyst에서 빈 프로젝트 생성](#)
- [연결된 타사 리포지토리를 사용하여 프로젝트 생성](#)
- [블루프린트를 사용하여 프로젝트 생성](#)
- [Amazon Q를 사용하여 프로젝트를 생성하거나 블루프린트를 사용하여 기능을 추가할 때의 모범 사례](#)
- [프로젝트에서 블루프린트를 사용하는 모범 사례](#)
- [생성된 프로젝트에 리소스 및 작업 추가](#)

Amazon CodeCatalyst에서 빈 프로젝트 생성

리소스 없이 빈 프로젝트를 생성하고 나중에 원하는 리소스를 수동으로 추가할 수 있습니다.

프로젝트를 생성하기 전에 스페이스 관리자 또는 파워 유저 역할이 있어야 합니다.

빈 프로젝트를 만들려면

1. 프로젝트를 생성하려는 스페이스로 이동합니다.

2. 스페이스 대시보드에서 프로젝트 생성을 선택합니다.
3. 처음부터 시작을 선택합니다.
4. 프로젝트에 이름 부여에서 프로젝트에 할당할 이름을 입력합니다. 이름은 스페이스 내에서 고유해야 합니다.
5. 프로젝트 생성을 선택합니다.

연결된 타사 리포지토리를 사용하여 프로젝트 생성

프로젝트의 소스 코드를 선호하는 타사 공급자에 보관하면서도, 블루프린트, 수명 주기 관리, 워크플로 등과 같은 모든 CodeCatalyst 기능을 계속 사용할 수 있습니다. 이를 위해, GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리에 연결되는 새 CodeCatalyst 프로젝트를 생성할 수 있습니다. 그런 다음 CodeCatalyst 프로젝트에서 연결된 소스 리포지토리를 사용할 수 있습니다.

CodeCatalyst 프로젝트를 생성하기 전에 스페이스 관리자 또는 파워 유저 역할이 있어야 합니다. 자세한 내용은 [스페이스 생성](#) 및 [사용자를 직접 스페이스에 초대](#) 섹션을 참조하세요.

CodeCatalyst에서 GitHub 계정의 소스 리포지토리에 연결하는 프로젝트를 생성하려면 다음 세 가지 작업을 완료해야 합니다.

1. GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 리포지토리 확장 프로그램을 설치합니다. 외부 사이트에서 CodeCatalyst에 연결하고 리포지토리에 대한 액세스 권한을 제공하도록 프롬프트가 표시되며, 이는 다음 단계의 일부로 수행됩니다.

Important

GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 리포지토리 확장 프로그램을 CodeCatalyst 스페이스에 설치하려면 해당 스페이스에 스페이스 관리자 역할이 있는 계정으로 로그인해야 합니다.

2. GitHub 계정 또는 Bitbucket 작업 영역 또는 GitLab 사용자를 CodeCatalyst에 연결합니다.

Important

GitHub 계정, Bitbucket 작업 영역, GitLab 사용자를 CodeCatalyst 스페이스에 연결하려면, 타사 소스의 관리자면서 동시에 CodeCatalyst 스페이스 관리자여야 합니다.

 Important

리포지토리 확장 프로그램을 설치한 후 CodeCatalyst에 연결하는 모든 리포지토리의 코드가 CodeCatalyst에 인덱싱 및 저장됩니다. 이렇게 하면 CodeCatalyst에서 코드를 검색할 수 있습니다. CodeCatalyst에서 연결된 리포지토리를 사용할 때 코드에 대한 데이터 보호를 더 잘 이해하려면 Amazon CodeCatalyst 사용 설명서의 [데이터 보호](#)를 참조하세요.

3. GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리에 연결된 CodeCatalyst 프로젝트를 생성합니다.

 Important

GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 기고자로 연결할 수 있지만 타사 리포지토리를 스페이스 관리자 또는 프로젝트 관리자로서만 연결 해제할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#) 섹션을 참조하세요.

 Important

CodeCatalyst는 연결된 리포지토리에 대한 기본 브랜치의 변경 사항 감지를 지원하지 않습니다. 연결된 리포지토리의 기본 브랜치를 변경하려면 먼저 CodeCatalyst에서 연결을 해제하고 기본 브랜치를 변경한 다음 다시 연결해야 합니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결](#) 섹션을 참조하세요.

리포지토리를 연결하기 전에 항상 최신 버전의 확장 프로그램을 사용하는 것이 좋습니다.

 Note

- GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리는 한 스페이스에서 하나의 CodeCatalyst 프로젝트에만 연결할 수 있습니다.
- 비어있거나 보관된 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 CodeCatalyst 프로젝트와 함께 사용할 수 없습니다.

- CodeCatalyst 프로젝트의 리포지토리와 이름이 동일한 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 연결할 수 없습니다.
- GitHub 리포지토리 확장 프로그램은 GitHub Enterprise Server 리포지토리와 호환되지 않습니다.
- Bitbucket 리포지토리 확장 프로그램은 Bitbucket Data Center 리포지토리와 호환되지 않습니다.
- GitLab 리포지토리 확장 프로그램은 GitLab 자체 관리형 프로젝트 리포지토리와 호환되지 않습니다.
- 연결된 리포지토리에서는 설명 쓰기 또는 설명 요약 기능을 사용할 수 없습니다. 이러한 기능은 CodeCatalyst의 풀 요청에서만 사용할 수 있습니다.

자세한 내용은 [CodeCatalyst에서 확장 프로그램이 있는 프로젝트에 기능 추가](#) 단원을 참조하십시오.

타사 확장 프로그램을 설치하려면

1. 프로젝트를 생성하려는 스페이스로 이동합니다.
2. 스페이스 대시보드에서 프로젝트 생성을 선택합니다.
3. 자체 코드 가져오기를 선택합니다.
4. 기존 리포지토리 연결에서 사용하려는 타사 리포지토리 공급자에 따라 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 리포지토리를 선택합니다. 이전에 연결하지 않은 경우 GitHub 계정, Bitbucket 작업 영역 또는 GitLab 계정을 연결하라는 메시지가 표시됩니다. 선택한 타사 확장 프로그램이 아직 설치되지 않은 경우 설치 프롬프트가 표시됩니다.
5. 프롬프트가 표시되면 설치를 선택합니다. 확장 프로그램에 필요한 권한을 검토하고 계속하려면 설치를 다시 선택합니다.

타사 확장 프로그램을 설치한 후 다음 단계는 GitHub 계정, Bitbucket 작업 영역 또는 GitLab 사용자를 CodeCatalyst 스페이스에 연결하는 것입니다.

GitHub 계정, Bitbucket 작업 영역 또는 GitLab 사용자를 CodeCatalyst에 연결하려면

구성하도록 선택한 타사 확장 프로그램에 따라 다음 중 하나를 수행합니다.

- GitHub 리포지토리: GitHub 계정에 연결합니다.
 1. GitHub 계정 연결을 선택하여 GitHub의 외부 사이트로 이동합니다.

2. GitHub 자격 증명으로 GitHub 계정에 로그인한 다음 Amazon CodeCatalyst를 설치할 계정을 선택합니다.

 Tip

이전에 GitHub 계정을 스페이스에 연결한 경우 재인증하라는 메시지가 표시되지 않습니다. 대신 둘 이상의 GitHub 스페이스에 멤버 또는 공동 작업자인 경우 확장 프로그램을 설치할 위치를 묻는 대화 상자가 표시되고, GitHub 스페이스에만 속한 경우 Amazon CodeCatalyst 애플리케이션의 구성 페이지가 표시됩니다. 허용하려는 리포지토리 액세스에 대한 애플리케이션을 구성한 다음 저장을 선택합니다. 저장 버튼이 활성화되지 않은 경우 구성을 변경한 다음 다시 시도하세요.

3. CodeCatalyst가 현재 및 미래 리포지토리에 모두 액세스하도록 허용할지 여부를 선택하거나 CodeCatalyst에서 사용할 특정 GitHub 리포지토리를 선택합니다. 기본 옵션은 CodeCatalyst에서 액세스할 향후 리포지토리를 포함하여 GitHub 계정에 모든 GitHub 리포지토리를 포함하는 것입니다.
4. CodeCatalyst에 부여된 권한을 검토한 다음 설치를 선택합니다.

GitHub 계정을 CodeCatalyst에 연결하면 GitHub 리포지토리 확장 프로그램 세부 정보 페이지로 이동합니다. 여기서 연결된 GitHub 계정과 연결된 GitHub 리포지토리를 보고 관리할 수 있습니다.

- Bitbucket 리포지토리: Bitbucket 작업 영역에 연결합니다.

1. Bitbucket 작업 영역 연결을 선택하여 Bitbucket의 외부 사이트로 이동합니다.
2. Bitbucket 자격 증명을 사용하여 Bitbucket 작업 영역에 로그인하고 CodeCatalyst에 부여된 권한을 검토합니다.
3. 작업 영역에 대한 권한 부여 드롭다운 메뉴에서 CodeCatalyst 액세스를 제공할 Bitbucket 작업 영역을 선택한 다음 액세스 권한 부여를 선택합니다.

 Tip

이전에 Bitbucket 작업 영역을 스페이스에 연결한 경우 재인증하라는 메시지가 표시되지 않습니다. 대신 하나 이상의 Bitbucket 작업 영역에서 멤버 또는 공동 작업자인 경우 확장 프로그램을 설치할 위치를 묻는 대화 상자가 표시되고, 하나의 Bitbucket 작업 영역에만 속한 경우 Amazon CodeCatalyst 애플리케이션의 구성 페이지가 표시됩니다. 작업 영역 액세스를 허용하려는 애플리케이션을 선택하고, 액세스 허용을 선택합니다. 액세스 허용 버튼이 활성화되지 않은 경우 구성을 변경한 다음 다시 시도하세요.

Bitbucket 워크스페이스를 CodeCatalyst에 연결하면 연결된 Bitbucket 워크스페이스와 연결된 Bitbucket 리포지토리를 보고 관리할 수 있는 Bitbucket 리포지토리 확장 프로그램 세부 정보 페이지로 이동합니다.

- GitLab 리포지토리: GitLab 사용자에게 연결합니다.
 1. GitLab 사용자 연결을 선택하여 GitLab 외부 사이트로 이동합니다.
 2. GitLab 자격 증명을 사용하여 GitLab 사용자에게 로그인하고 CodeCatalyst에 부여된 권한을 검토합니다.

 Tip

이전에 GitLab 사용자를 스페이스에 연결한 경우 재인증하라는 메시지가 표시되지 않습니다. 대신 CodeCatalyst 콘솔로 다시 이동합니다.

3. GitLab용 AWS 커넥터 권한 부여를 선택합니다.

GitLab 사용자를 CodeCatalyst에 연결하면 GitLab 리포지토리 확장 프로그램 세부 정보 페이지로 이동하며, 여기서 연결된 GitLab 사용자 및 연결된 GitLab 프로젝트 리포지토리를 보고 관리할 수 있습니다.

타사 소스를 CodeCatalyst에 연결한 후 타사 리포지토리를 CodeCatalyst 프로젝트에 연결할 수 있습니다.

프로젝트를 생성하려면

1. 프로젝트 생성 페이지에서 연결한 GitHub 계정을 선택합니다.
2. 연결한 타사 리포지토리 공급자에 따라 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 리포지토리의 리포지토리 드롭다운 메뉴를 선택하여 타사 리포지토리를 확인한 다음 프로젝트에 연결할 리포지토리를 선택합니다.
3. 프로젝트 이름 지정 텍스트 입력 필드에 프로젝트에 할당할 이름을 입력합니다. 이름은 스페이스 내에서 고유해야 합니다.
4. 프로젝트 생성을 선택합니다.

GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 리포지토리 확장 프로그램을 설치하고, 리소스 공급자를 연결하고, 타사 리포지토리를 CodeCatalyst 프로젝트에 연결한 후 CodeCatalyst 워크플

로 및 개발 환경에서 사용할 수 있습니다. 블루프린트에서 생성된 코드를 사용하여 연결된 GitHub 계정, Bitbucket 작업 영역 또는 GitLab 사용자에서 타사 리포지토리를 생성할 수도 있습니다. Amazon Q Developer, 블루프린트 등과 함께 연결된 리포지토리를 사용할 수도 있습니다. 자세한 내용은 [타사 리포지토리 이벤트 후 워크플로 실행 자동 시작](#) 및 [개발 환경 생성](#) 섹션을 참조하세요.

블루프린트를 사용하여 프로젝트 생성

프로젝트 블루프린트를 사용하여 모든 프로젝트 리소스와 샘플 코드를 프로비저닝할 수 있습니다. 블루프린트에 대한 자세한 내용은 [CodeCatalyst 블루프린트를 사용하여 포괄적인 프로젝트 생성](#) 섹션을 참조하세요.

블루프린트를 사용하여 프로젝트를 생성하려면

1. CodeCatalyst 콘솔에서 프로젝트를 생성하려는 스페이스로 이동합니다.
2. 스페이스 대시보드에서 프로젝트 생성을 선택합니다.
3. 블루프린트로 시작을 선택합니다.

Tip

Amazon Q가 블루프린트를 제안하도록 프로젝트 요구 사항을 Amazon Q에 제공하여 블루프린트를 추가하도록 선택할 수 있습니다. 자세한 내용은 [Amazon Q를 사용하여 프로젝트를 생성하거나 기능을 추가할 때 블루프린트 선택](#) 및 [Amazon Q를 사용하여 프로젝트를 생성하거나 블루프린트를 사용하여 기능을 추가할 때의 모범 사례](#) 섹션을 참조하세요. 이 기능은 미국 서부(오리건) 리전에서만 사용할 수 있습니다. 이 기능을 사용하려면 스페이스에 생성형 AI 기능을 활성화해야 합니다. 자세한 내용은 [생성형 AI 기능 관리](#)를 참조하세요.

4. CodeCatalyst 블루프린트 또는 스페이스 블루프린트 탭에서 블루프린트를 선택한 다음 다음을 선택합니다.
5. 프로젝트 이름 지정에 프로젝트에 할당할 이름과 관련 리소스 이름을 입력합니다. 이름은 스페이스 내에서 고유해야 합니다.
6. (선택 사항) 기본적으로 블루프린트에서 생성된 소스 코드는 CodeCatalyst 리포지토리에 저장됩니다. 또는 블루프린트의 소스 코드를 타사 리포지토리에 저장하도록 선택할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 확장 프로그램이 있는 프로젝트에 기능 추가](#) 섹션을 참조하세요.

⚠ Important

CodeCatalyst는 연결된 리포지토리에 대한 기본 브랜치의 변경 사항 감지를 지원하지 않습니다. 연결된 리포지토리의 기본 브랜치를 변경하려면 먼저 CodeCatalyst에서 연결을 해제하고 기본 브랜치를 변경한 다음 다시 연결해야 합니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결](#) 섹션을 참조하세요.

리포지토리를 연결하기 전에 항상 최신 버전의 확장 프로그램을 사용하는 것이 좋습니다.

사용하려는 타사 리포지토리 공급자에 따라 다음 중 하나를 수행합니다.

- GitHub 리포지토리: GitHub 계정을 연결합니다.

고급 드롭다운 메뉴를 선택하고 GitHub를 리포지토리 공급자로 선택한 다음 블루프린트에서 생성된 소스 코드를 저장할 GitHub 계정을 선택합니다.

i Note

GitHub 계정을 연결하는 경우 CodeCatalyst ID와 GitHub ID 간에 ID 매핑을 설정하려면 개인 연결을 생성해야 합니다. 자세한 내용은 [개인 연결](#) 및 [개인 연결을 사용하여 GitHub 리소스에 액세스](#) 섹션을 참조하세요.

- Bitbucket 리포지토리: Bitbucket 작업 영역을 연결합니다.

고급 드롭다운 메뉴를 선택하고 Bitbucket을 리포지토리 공급자로 선택한 다음, 블루프린트에서 생성된 소스 코드를 저장할 Bitbucket 작업 영역을 선택합니다.

- GitLab 리포지토리: GitLab 사용자를 연결합니다.

고급 드롭다운 메뉴를 선택하고 GitLab을 리포지토리 공급자로 선택한 다음 블루프린트에서 생성된 소스 코드를 저장할 GitLab 사용자를 선택합니다.

7. 프로젝트 리소스에서 블루프린트 파라미터를 구성합니다. 블루프린트에 따라 소스 리포지토리의 이름을 지정할 수 있는 옵션이 있을 수 있습니다.
8. (선택 사항) 구성된 프로젝트 파라미터에 따라 업데이트가 포함된 정의 파일을 보려면 프로젝트 미리 보기 생성에서 코드 보기 또는 워크플로 보기를 선택합니다.

9. (선택 사항) 블루프린트의 아키텍처 개요, 필수 연결 및 권한, 블루프린트가 생성하는 리소스 유형 등 블루프린트에 대한 특정 세부 정보를 보려면 블루프린트 카드에서 세부 정보 보기를 선택합니다.
10. 프로젝트 생성을 선택합니다.

Amazon Q를 사용하여 프로젝트를 생성하거나 블루프린트를 사용하여 기능을 추가할 때의 모범 사례

프로젝트를 생성하거나 기존 프로젝트에 새 구성 요소를 추가하려는 경우, 어떤 블루프린트를 사용할지 또는 기능을 어떻게 통합할지 잘 모를 수 있습니다. CodeCatalyst에는 프로젝트 요구 사항을 분석하고 요구 사항에 가장 적합한 블루프린트를 제안할 수 있는 Amazon Q라는 생성형 AI 어시스턴트와의 통합 기능이 포함됩니다.

Amazon Q를 사용하여 요구 사항에 따라 구성 요소를 생성하는 블루프린트로 프로젝트를 만들거나 Amazon Q를 사용하여 기존 프로젝트에 블루프린트를 추가할 수 있습니다. 예를 들어 프로젝트에 웹 애플리케이션 또는 최신 애플리케이션의 리소스를 추가하려면, 요구 사항을 지정할 시 권장 블루프린트와 함께 리소스가 추가됩니다. 나머지 구성 요소에 대한 문제를 생성할 수 있습니다.

또한 Amazon Q는 제안된 블루프린트로 해결할 수 없는 요구 사항에 대한 문제를 생성합니다. 또한 이러한 문제를 Amazon Q에 할당할 수 있습니다. Amazon Q에 문제를 할당하면 평가하기 위해 솔루션 초안을 생성하려고 시도합니다. 이렇게 하면 사용자와 팀이 주의가 필요한 문제에 집중하고 작업을 최적화하는 데 도움이 되며, Amazon Q는 즉시 해결할 리소스가 없는 문제에 대한 솔루션을 제공합니다.

Note

Amazon Bedrock:implements 자동 침해 탐지로 구동됩니다. AWS <https://docs.aws.amazon.com/bedrock/latest/userguide/abuse-detection.html> 소프트웨어 개발용 Amazon Q Developer Agent의 설명 쓰기, 콘텐츠 요약 생성, 권장 작업, Amazon Q를 사용하여 프로젝트에 기능을 생성하거나 추가, Amazon Q에 문제 할당 기능은 Amazon Bedrock에 구축되므로, 사용자는 Amazon Bedrock에서 구현된 제어 기능을 최대한 활용하여 안전, 보안 및 인공 지능(AI)을 책임 있게 사용할 수 있습니다.

다음은 Amazon Q를 사용하여 프로젝트를 생성하고 블루프린트를 추가하는 데 도움이 되는 몇 가지 모범 사례입니다.

⚠ Important

생성형 AI 기능은 미국 서부(오리건) 리전에서만 사용할 수 있습니다.

- Amazon Q 에서 제공하는 기본 프롬프트를 사용합니다. Amazon Q는 제공된 프롬프트에서 블루프린트를 선택하는 것이 가장 좋습니다.
- Amazon Q에서 제안한 구성 옵션을 사용하여 블루프린트를 미리 봅니다. 블루프린트를 선택하여 블루프린트에서 생성할 샘플 코드와 리소스를 미리 봅니다.
- Amazon Q에 대해 활성화된 스페이스를 사용합니다. Amazon Q를 사용하여 프로젝트를 생성하거나 Amazon Q를 사용하여 블루프린트가 있는 프로젝트에 기능을 추가하려면, 생성형 AI 기능에 대해 활성화된 스페이스를 사용합니다. 자세한 내용은 [스페이스에 대한 생성형 AI 기능 활성화 또는 비활성화](#)를 참조하세요.
- Amazon Q에서 권장하는 블루프린트에 대한 자세한 내용을 확인하세요. 특정 권장 블루프린트로 생성된 프로젝트 리소스, 샘플 코드 및 구성 요소의 종류에 대해 자세히 찾아볼 수 있습니다. CodeCatalyst에서 사용 가능한 블루프린트에 대한 자세한 내용은 [CodeCatalyst 블루프린트를 사용하여 포괄적인 프로젝트 생성](#) 섹션을 참조하세요.
- Amazon Q가 문제를 처리하도록 허용합니다. Amazon Q가 문제를 생성하고, 해당 문제를 할당하고, 추적하도록 허용합니다. 자세한 내용은 [자습서: CodeCatalyst 생성형 AI 기능을 사용하여 개발 작업 속도 향상](#) 단원을 참조하십시오.
- 더 이상 작업하지 않는 문제에서 Amazon Q 할당을 취소합니다. 예시를 완료한 후, 더 이상 작업하지 않는 문제에서 Amazon Q 할당을 취소합니다. Amazon Q가 문제에 대한 작업을 완료했거나 솔루션을 찾을 수 없는 경우 생성형 AI 기능의 최대 할당량에 도달하지 않도록 Amazon Q를 할당 취소해야 합니다. 자세한 내용은 [생성형 AI 기능 관리](#) 및 [요금](#)을 참조하세요.
- Amazon Q의 사용량을 봅니다. 사용자 수준에서 생성형 AI 기능의 사용량을 볼 수 있습니다. 내 설정으로 이동하여 생성형 AI 할당량을 관리하고 Builder ID 또는 AWS Single Sign-On(SSO) ID별로 사용량을 확인합니다. 자세한 내용은 [스페이스에서 생성형 AI 기능 사용량 보기](#)를 참조하세요.

⚠ Important

CodeCatalyst의 생성형 AI 기능에는 할당량이 적용됩니다. 자세한 내용은 [Amazon Q Developer 요금](#), [스페이스에 대한 생성형 AI 기능 활성화 또는 비활성화](#) 및 [청구](#)를 참조하세요.

프로젝트에서 블루프린트를 사용하는 모범 사례

다음은 블루프린트로 프로젝트를 생성하거나 블루프린트를 추가하는 데 도움이 되는 몇 가지 모범 사례입니다.

- CodeCatalyst에서 제공하는 블루프린트를 사용하여 프로젝트를 생성하거나 에 추가합니다. 블루프린트를 사용하여 개발자를 위한 소스 코드 및 리소스가 포함된 전체 프로젝트를 생성할 수 있습니다. 예를 들어 웹 애플리케이션 블루프린트는 애플리케이션 및 인프라 리소스를 생성하고 웹 애플리케이션을 배포합니다. 블루프린트로 프로젝트를 생성하거나 기존 프로젝트에 사용자 지정 블루프린트를 추가할 수 있습니다. 자세한 내용은 [블루프린트를 사용하여 프로젝트 생성](#) 단원을 참조하십시오. CodeCatalyst의 블루프린트를 보고 블루프린트에서 생성할 샘플 코드와 리소스를 미리 봅니다.
- 조직에서 설계한 사용자 지정 블루프린트를 사용합니다. 사용자 지정 블루프린트를 사용하여 스페이스에 전체 프로젝트를 생성할 수 있습니다. 조직에서 설계한 사용자 지정 블루프린트는 표준화 및 모범 사례를 제공할 수 있으며, 이를 통해 새 프로젝트를 설정할 때 드는 노력을 줄일 수도 있습니다. 사용자 지정 블루프린트 작성자는 스페이스 전체에서 블루프린트를 사용하는 프로젝트에 대한 세부 정보를 볼 수 있습니다. 수명 주기 관리를 사용하면 모든 프로젝트의 소프트웨어 개발 수명 주기를 중앙에서 관리할 수 있으며, 블루프린트 사용자는 수명 주기 관리를 사용하여 업데이트된 옵션 또는 블루프린트 버전에서 코드베이스를 재생성할 수 있습니다. 자세한 내용은 [블루프린트 작성자로서 수명 주기 관리 작업](#) 단원을 참조하십시오.
- 프로젝트 계정에 개발자 역할 또는 적절한 IAM 역할을 추가합니다. 프로젝트 생성 단계 중 또는 완료 후, 스페이스에 연결된 AWS 계정에서 IAM 역할을 선택하거나 생성하여 블루프린트 권한을 구성할 수 있습니다.

생성된 프로젝트에 리소스 및 작업 추가

프로젝트가 준비되면 리소스와 작업을 추가할 수 있습니다.

- 프로젝트로 생성된 CI/CD 워크플로에 대한 자세한 내용은 [워크플로 시작하기](#) 섹션을 참조하세요.
- Amazon S3 버킷에 빌드 아티팩트를 배포하는 새 프로젝트의 작업과 유사한 빌드 작업을 수행하려면 [워크플로로 빌드하기](#) 및 [자습서: Amazon S3에 아티팩트 업로드](#) 섹션을 참조하세요.
- 빈 프로젝트로 시작하여 CloudFormation 스택 배포를 통해 유사한 서버리스 애플리케이션을 배포하는 작업은 단원을 참조하십시오 [자습서: 서버리스 애플리케이션 배포](#).
- 문제 계획 보드를 추가하려면 [CodeCatalyst의 문제 처리 추적 및 구성](#) 섹션을 참조하세요.
- 프로젝트 개요, 프로젝트 상태, 최근 팀 활동 및 할당된 작업을 보려면 [프로젝트 목록 가져오기](#) 섹션을 참조하세요.

- 소스 코드를 보거나 풀 요청을 생성하려면 [CodeCatalyst의 소스 리포지토리로 코드 저장 및 협업](#) 섹션을 참조하세요.
- 워크플로 실행 성공 또는 실패에 대한 상태 알림을 보내는 알림을 설정하려면 [CodeCatalyst에서 Slack 및 이메일 알림 전송](#) 섹션을 참조하세요.
- 프로젝트에 멤버를 초대하려면 [사용자에게 프로젝트 권한 부여](#) 섹션을 참조하세요.
- 개발 환경을 설정하려면 [CodeCatalyst에서 개발 환경으로 코드 작성 및 수정](#) 섹션을 참조하세요.

프로젝트 목록 가져오기

CodeCatalyst 스페이스에서 프로젝트 권한이 있는 각 프로젝트의 세부 정보를 볼 수 있습니다.

프로젝트를 보려면 프로젝트의 멤버가거나 해당 공간에 대한 스페이스 관리자 역할이 있어야 합니다.

프로젝트를 아직 생성하지 않은 경우 [프로젝트 생성](#) 섹션을 참조하세요. 프로젝트를 생성하려면 스페이스 관리자 역할이 있어야 합니다.

- 프로젝트 개요에서 프로젝트 멤버, 소스 리포지토리, 워크플로 실행, 오픈 풀 요청, 프로젝트 개발 환경 및 문제를 볼 수 있습니다.
- 프로젝트 설정에서 프로젝트 세부 정보를 보고 관리하고, 프로젝트를 삭제하고, 프로젝트에 새 멤버를 초대하고, 프로젝트 멤버를 관리하고, 알림을 구성할 수 있습니다.

프로젝트 작업 및 개발 환경 보기

사용자에게 할당되거나 사용자가 생성한 아직 완료되지 않은 문제 및 풀 요청, 프로젝트의 관련 개발 환경과 같은 프로젝트 작업의 요약은 보려면 콘솔을 사용합니다.

프로젝트를 보려면 프로젝트의 멤버가거나 해당 공간에 대한 스페이스 관리자 역할이 있어야 합니다.

소스 리포지토리, 워크플로 실행, 문제, 풀 요청, 개발 환경 및 문제를 보려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 보려는 프로젝트가 있는 스페이스로 이동합니다. 프로젝트에서 프로젝트를 선택합니다.
3. 탐색 창에서 개요를 선택합니다.
4. 사용자에게 할당되고 사용자가 생성한 프로젝트 작업을 봅니다.
 - 멤버 + 전체 보기 목록을 통해 프로젝트 멤버 목록을 볼 수 있습니다.

- 리포지토리 카드를 보고 프로젝트와 연결된 소스 리포지토리를 확인합니다.
- 워크플로 실행 카드를 보고 프로젝트와 연결된 워크플로를 확인합니다.
- 풀 요청 열기 카드를 보고 코드 리포지토리 상태 요약과 더불어, 사용자에게 할당되고 사용자가 생성한 풀 요청도 확인합니다.
- 내 개발 환경 카드를 보고 프로젝트와 관련된 개발 환경의 요약을 확인합니다.
- 문제 카드를 보고 생성한 작업 또는 할당된 작업의 요약을 봅니다.

한 스페이스에서 모든 프로젝트 보기

스페이스의 프로젝트 목록에서 권한이 있는 모든 프로젝트를 볼 수 있습니다.

사용자에게 할당되거나 사용자가 생성한 아직 완료되지 않은 문제 및 풀 요청, 프로젝트의 관련 개발 환경과 같은 프로젝트 작업의 요약을 보려면 콘솔을 사용합니다.

프로젝트를 보려면 프로젝트의 멤버가거나 해당 공간에 대한 스페이스 관리자 역할이 있어야 합니다.

소스 리포지토리, 워크플로 실행, 문제, 풀 요청, 개발 환경 및 문제를 보려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 보려는 프로젝트가 있는 스페이스로 이동합니다. 프로젝트에서 프로젝트를 선택합니다.
3. 탐색 창에서 프로젝트 설정을 선택합니다.
4. 프로젝트 이름, 경로, 프로젝트 ID 및 설명을 봅니다.

프로젝트 설정 보기

프로젝트 설정 에서 프로젝트 멤버, 소스 리포지토리, 워크플로 실행, 오픈 풀 요청, 프로젝트 개발 환경 및 문제를 볼 수 있습니다.

사용자에게 할당되거나 사용자가 생성한 아직 완료되지 않은 문제 및 풀 요청, 프로젝트의 관련 개발 환경과 같은 프로젝트 작업의 요약을 보려면 콘솔을 사용합니다.

소스 리포지토리, 워크플로 실행, 문제, 풀 요청, 개발 환경 및 문제를 보려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 보려는 프로젝트가 있는 스페이스로 이동합니다. 프로젝트에서 프로젝트를 선택합니다.
3. 탐색 창에서 프로젝트 설정을 선택합니다.

4. 프로젝트 이름, 경로, 프로젝트 ID 및 설명을 봅니다.

CodeCatalyst에서 다른 프로젝트로 변경

다른 프로젝트로 변경하려면 콘솔을 사용하여 액세스할 수 있는 프로젝트 목록에서 선택합니다.

다른 프로젝트로 변경하려면

1. CodeCatalyst 콘솔에서 상단의 프로젝트 선택기를 선택합니다.
2. 드롭다운을 확장하고 탐색할 프로젝트를 선택합니다.

프로젝트 삭제

프로젝트를 삭제하여 프로젝트의 리소스에 대한 모든 액세스를 제거할 수 있습니다. 프로젝트를 삭제하려면 스페이스 관리자 또는 프로젝트 관리자 역할이 있어야 합니다. 프로젝트를 삭제하면 프로젝트 멤버가 프로젝트 리소스에 액세스할 수 없게 되고 타사 소스 리포지토리에서 프롬프트하는 워크플로가 중지됩니다.

프로젝트를 삭제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 보려는 프로젝트가 있는 스페이스로 이동합니다. 프로젝트에서 프로젝트를 선택합니다.
3. 탐색 창에서 프로젝트 설정을 선택합니다.
4. 프로젝트 삭제를 선택합니다.
5. **delete**를 입력하여 삭제를 확인합니다.
6. 프로젝트 삭제를 선택합니다.

사용자에게 프로젝트 권한 부여

Amazon CodeCatalyst 콘솔을 사용하여 프로젝트의 멤버를 관리할 수 있습니다. 사용자를 추가 또는 제거하고, 현재 멤버의 역할을 관리하고, 프로젝트 참여할 수 있는 초대장을 보내고, 아직 수락되지 않은 초대장을 취소할 수 있습니다.

스페이스 및 프로젝트 사용자의 멤버 페이지에서 사용자는 여러 역할을 가질 수 있습니다. 여러 역할을 지닌 사용자에게는 역할이 여러 개일 때 보이는 표시가 나타나며, 가장 많은 권한을 가진 역할이 먼저 표시됩니다.

멤버 및 해당 프로젝트 역할 목록 가져오기

프로젝트에 사용자를 추가할 때, 다음과 같이 프로젝트 권한을 부여하는 역할을 할당합니다.

- 프로젝트 관리자 역할에게는 프로젝트의 모든 권한이 있습니다. 프로젝트 설정 편집, 프로젝트 권한 관리, 프로젝트 삭제 등 프로젝트의 모든 측면을 관리해야 하는 사용자에게만 이 역할을 할당합니다. 자세한 내용은 [프로젝트 관리자 역할](#) 단원을 참조하십시오.
- 기고자 역할에게는 프로젝트에서 작업하는 데 필요한 권한이 있습니다. 프로젝트의 코드, 워크플로, 문제 및 작업을 수행해야 하는 사용자에게 이 역할을 할당합니다. 자세한 내용은 [기고자 역할](#) 단원을 참조하십시오.
- 검토자 역할에게는 검토 권한이 있습니다. 자세한 내용은 [검토자 역할](#)을 참조하세요.
- 읽기 전용 역할에게는 읽기 권한이 있습니다. 자세한 내용은 [읽기 전용 역할](#)을 참조하세요.

스페이스 관리자 역할을 가진 사용자는 이미 스페이스의 모든 프로젝트에 대한 암시적 액세스 권한이 있으므로 프로젝트에 초대할 필요가 없습니다.

사용자를 (스페이스 관리자 역할을 할당하지 않고) 프로젝트에 초대하면, 사용자는 프로젝트의 프로젝트 멤버 표와 스페이스의 프로젝트 멤버 표에 표시됩니다.

스페이스에서 사용자 및 역할을 보려면

- <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
- 보려는 프로젝트가 있는 스페이스로 이동합니다. 프로젝트에서 프로젝트를 선택합니다.
- 탐색 창에서 프로젝트 설정을 선택합니다.
- 멤버 탭을 선택합니다.

프로젝트 멤버 테이블에는 프로젝트에서 역할을 가진 모든 멤버가 표시됩니다.

Tip

스페이스 관리자 역할이 있는 경우 직접 초대된 프로젝트를 볼 수 있습니다. 프로젝트의 프로젝트 설정으로 이동한 다음, 내 프로젝트를 선택합니다.

스페이스 관리자 테이블에는 스페이스 관리자 역할을 지닌 사용자가 표시됩니다. 이러한 사용자는 스페이스의 모든 프로젝트에 자동으로(암시적으로) 할당되며 프로젝트에 역할이 없습니다.

상태 열에 유효한 값은 다음과 같습니다.

- 초대됨 - CodeCatalyst가 초대를 보냈지만 사용자가 아직 수락하거나 거부하지 않았습니다.
- 멤버 - 사용자가 초대를 수락했습니다.

주제

- [프로젝트에 사용자 초대](#)
- [초대 취소](#)
- [프로젝트에서 사용자 제거](#)
- [프로젝트 초대 수락 또는 거부](#)

프로젝트에 사용자 초대

콘솔을 사용하여 사용자를 프로젝트에 초대할 수 있습니다. 스페이스 외부에서 스페이스 멤버로 초대하거나 이름을 추가할 수 있습니다.

사용자를 프로젝트에 초대하려면 프로젝트 관리자 또는 스페이스 관리자 역할로 로그인해야 합니다.

스페이스 관리자 역할을 가진 사용자는 이미 스페이스의 모든 프로젝트에 대한 암시적 액세스 권한이 있으므로 프로젝트에 초대할 필요가 없습니다.

사용자를 (스페이스 관리자 역할을 할당하지 않고) 프로젝트에 초대하면, 사용자는 프로젝트의 프로젝트 멤버 표와 스페이스의 프로젝트 멤버 표에 표시됩니다.

프로젝트 설정 탭에서 프로젝트에 멤버를 초대하려면

1. 프로젝트로 이동합니다.

Tip

상단 탐색 모음에서 보려는 프로젝트를 선택할 수 있습니다.

2. 탐색 창에서 프로젝트 설정을 선택합니다.
3. 멤버 탭을 선택합니다.
4. 프로젝트 멤버에서 새 멤버 초대를 선택합니다.
5. 새 멤버의 이메일 주소를 입력하고 이 멤버의 역할을 선택한 다음 초대를 선택합니다. 역할에 관한 자세한 내용은 [사용자 역할로 액세스 권한 부여](#) 섹션을 참조하세요.

프로젝트 개요 페이지에서 프로젝트에 멤버를 초대하려면

1. 프로젝트로 이동합니다.

 Tip

상단 탐색 모음에서 보려는 프로젝트를 선택할 수 있습니다.

2. 멤버 + 버튼을 선택합니다.
3. 새 멤버의 이메일 주소를 입력하고 이 멤버의 역할을 선택한 다음 초대를 선택합니다. 역할에 관한 자세한 내용은 [사용자 역할로 액세스 권한 부여](#) 섹션을 참조하세요.

초대 취소

최근에 초대를 보낸 경우, 초대가 아직 수락되지 않은 한 취소할 수 있습니다.

프로젝트 초대를 관리하려면 프로젝트 관리자 또는 스페이스 관리자 역할이 있어야 합니다.

프로젝트 멤버 초대를 취소하려면

1. 취소하려는 초대장을 보낸 프로젝트로 이동합니다.
2. 탐색 창에서 프로젝트 설정을 선택합니다.
3. 멤버 탭을 보고 멤버의 상태가 초대됨인지 확인합니다.

 Note

아직 수락되지 않은 초대만 취소할 수 있습니다.

4. 초대된 멤버가 있는 행 옆의 옵션을 선택한 다음 초대 취소를 선택합니다.
5. 확인 창이 표시됩니다. 초대 취소를 선택하여 확인합니다.

프로젝트에서 사용자 제거

콘솔을 사용하여 프로젝트에서 사용자를 제거할 수 있습니다.

프로젝트에서 사용자를 제거하려면 프로젝트 관리자 또는 스페이스 관리자 역할로 로그인해야 합니다.

Note

한 명의 사용자를 한 스페이스 내의 모든 프로젝트에서 제거하면 해당 스페이스에서 해당 사용자가 자동으로 제거됩니다.

사용자를 프로젝트에서 제거하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 보려는 프로젝트가 있는 스페이스로 이동합니다. 프로젝트에서 프로젝트를 선택합니다.
3. 탐색 창에서 프로젝트 설정을 선택합니다.
4. 멤버 탭을 선택합니다.
5. 제거하려는 프로파일 옆의 선택기를 선택한 다음 제거 선택합니다.
6. 해당 사용자를 제거할지 확인한 다음, 제거를 선택합니다.

프로젝트 초대 수락 또는 거부

Amazon CodeCatalyst 프로젝트에 참여하도록 원하는 이메일 초대장을 받을 수 있습니다. 여러분은 초대장을 수락하거나 거부할 수 있습니다.

초대를 수락하거나 거부하려면

1. 초대 이메일을 엽니다.
2. 이메일에서 프로젝트 링크를 선택합니다.
3. 수락 또는 거부를 선택합니다.

거부를 선택하면 초대를 거부했음을 알리는 이메일이 프로젝트 관리 계정으로 전송됩니다.

팀을 통해 프로젝트 액세스 허용

프로젝트를 생성한 후 팀을 추가할 수 있습니다. 팀을 통해 CodeCatalyst에서 프로젝트 및 스페이스 멤버로서 권한을 공유하고 프로젝트, 문제 추적, 역할 및 리소스를 관리할 수 있도록 사용자를 그룹화할 수 있습니다.

프로젝트의 팀을 관리하려면 프로젝트 관리자 역할이 있어야 합니다.

또한 팀은 CodeCatalyst의 스페이스 수준에서 관리됩니다. 스페이스의 팀에 대한 자세한 내용은 [팀을 사용하여 스페이스 액세스 허용](#) 섹션을 참조하세요.

주제

- [프로젝트에 팀 추가](#)
- [팀에 프로젝트 역할 부여](#)
- [팀의 프로젝트 역할 제거](#)

프로젝트에 팀 추가

팀원이 프로젝트의 리소스에 액세스할 수 있는 팀을 관리할 수 있습니다.

스페이스 및 프로젝트 사용자의 멤버 페이지에서 사용자는 여러 역할을 가질 수 있습니다. 여러 역할을 지닌 사용자에게는 역할이 여러 개일 때 보이는 표시가 나타나며, 가장 많은 권한을 가진 역할이 먼저 표시됩니다.

팀을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트로 이동합니다. 프로젝트 설정을 선택하고 팀을 선택합니다.
3. 팀 추가를 선택합니다.
4. 팀의 사용 가능한 팀 목록에서 팀을 선택합니다.
5. 프로젝트 역할의 CodeCatalyst 내 사용할 수 있는 프로젝트 역할 목록에서 역할을 선택합니다.
 - 프로젝트 관리자 - 자세한 내용은 [프로젝트 관리자 역할](#) 섹션을 참조하세요.
 - 기고자 - 자세한 내용은 [기고자 역할](#) 섹션을 참조하세요.
 - 검토자 - 자세한 내용은 [검토자 역할](#) 섹션을 참조하세요.
 - 읽기 전용 - 자세한 내용은 [읽기 전용 역할](#) 섹션을 참조하세요.
6. 팀 추가를 선택합니다.

팀에 프로젝트 역할 부여

팀은 한 스페이스에 파워 유저와 같은 역할 권한을 가질 수 있습니다. 팀의 스페이스 역할은 변경할 수 있지만, 팀의 모든 멤버가 해당 권한을 상속한다는 점에 유의해야 합니다.

프로젝트 역할을 추가하거나 변경하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다. 프로젝트 설정을 선택하고 팀을 선택합니다.
3. 역할을 변경하려면 이 목록의 팀 옆에 있는 선택기를 선택한 다음 역할 변경을 선택합니다. 역할을 추가하려면 프로젝트 역할 추가를 선택합니다. 프로젝트에서 추가할 프로젝트를 선택하고 역할에서 역할을 선택합니다. 사용 가능한 프로젝트 역할 중 하나를 선택합니다.
 - 프로젝트 관리자 - 자세한 내용은 [프로젝트 관리자 역할](#) 섹션을 참조하세요.
 - 기고자 - 자세한 내용은 [기고자 역할](#) 섹션을 참조하세요.
 - 검토자 - 자세한 내용은 [검토자 역할](#) 섹션을 참조하세요.
 - 읽기 전용 - 자세한 내용은 [읽기 전용 역할](#) 섹션을 참조하세요.
4. 저장을 선택합니다.

팀의 프로젝트 역할 제거

CodeCatalyst에서 팀의 프로젝트 역할을 볼 수 있습니다. 팀의 멤버도 확인이 가능합니다. 또한 팀의 프로젝트 역할을 제거할 수도 있습니다.

프로젝트 역할을 제거하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다. 프로젝트 설정을 선택하고 팀을 선택합니다.
3. 프로젝트 역할 탭을 선택합니다.
4. 제거할 역할을 선택합니다.



Important

팀에서 역할을 제거하면 팀의 모든 사용자가 지닌 관련 권한이 제거됩니다.

5. 저장을 선택합니다.

시스템 리소스에 대한 프로젝트 액세스 허용

시스템 리소스는 CodeCatalyst의 프로젝트 또는 스페이스에 대한 권한이 부여된 CodeCatalyst의 특정 리소스입니다.

Note

시스템 리소스라는 용어는 EC2 인스턴스와 같은 클라우드 인프라를 가리키는 것이 아니라 스페이스 또는 프로젝트에 대한 권한이 있는 블루프린트 또는 워크플로 리소스를 의미합니다.

프로젝트에서 시스템 리소스를 사용하는 작업의 예로는 블루프린트 리소스가 사용자를 대신하여 프로젝트에 액세스할 수 있도록 하는 것이 포함됩니다.

시스템 리소스는 SSO를 통해 CodeCatalyst에 액세스할 때 승인된 리소스의 ID를 나타냅니다. 시스템 리소스는 블루프린트 및 워크플로와 같은 프로젝트의 리소스에 권한을 부여하는 데 사용됩니다. 프로젝트의 시스템 리소스는 확인할 수 있으며 프로젝트의 시스템 리소스를 활성화하거나 비활성화하도록 선택할 수 있습니다. 예를 들어, 액세스를 관리하기 위해 시스템 리소스를 비활성화했다가 나중에 다시 활성화할 수도 있습니다.

이러한 작업은 시스템 리소스를 취소하거나 비활성화해야 하는 경우 시스템 리소스에 사용할 수 있습니다. 예를 들어 자격 증명이 손상되었을 수 있다고 의심되는 경우 시스템 리소스를 비활성화할 수 있습니다. 일반적으로 이러한 작업은 사용할 필요가 없습니다.

이 페이지를 보고 프로젝트 수준에서 시스템 리소스를 관리하려면 스페이스 관리자 역할 또는 프로젝트 관리자 역할이 있어야 합니다.

또한 시스템 리소스는 CodeCatalyst의 스페이스 수준에서 관리됩니다. 스페이스/프로젝트의 팀에 대한 자세한 내용은 [시스템 리소스에 대한 스페이스 액세스 허용](#) 섹션을 참조하세요.

주제

- [시스템 리소스에 대한 프로젝트 액세스 보기](#)
- [시스템 리소스에 대한 프로젝트 액세스 비활성화](#)
- [시스템 리소스에 대한 프로젝트 액세스 활성화](#)

시스템 리소스에 대한 프로젝트 액세스 보기

프로젝트에서 사용 중인 시스템 리소스 목록을 볼 수 있습니다.

스페이스 관리자 역할 또는 프로젝트 관리자 역할이 있어야 합니다.

시스템 리소스를 보려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.

2. 프로젝트로 이동한 다음 프로젝트 설정을 선택합니다. 시스템 리소스를 선택합니다.
3. 드롭다운에서 워크플로 작업을 선택하여 워크플로에 대한 시스템 리소스만 봅니다. 블루프린트에 대한 시스템 리소스만 보려면 블루프린트를 선택합니다.

필터 필드를 사용하여 이름을 필터링할 수도 있습니다.

시스템 리소스에 대한 프로젝트 액세스 비활성화

프로젝트에서 사용 중인 시스템 리소스를 비활성화하도록 선택할 수 있습니다.

Important

시스템 리소스를 비활성화하면 스페이스의 모든 관련 블루프린트 또는 워크플로에 대한 모든 권한이 제거됩니다.

스페이스 관리자 역할 또는 프로젝트 관리자 역할이 있어야 합니다.

시스템 리소스를 비활성화하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트로 이동한 다음 프로젝트 설정을 선택합니다. 시스템 리소스를 선택합니다.
3. 다음 중 하나를 선택합니다.

Important

시스템 리소스를 비활성화하면 스페이스의 모든 관련 블루프린트 또는 워크플로에 대한 모든 권한이 제거됩니다.

- 개별적으로 비활성화하려면, 비활성화하려는 하나 이상의 시스템 리소스 옆에 있는 선택기를 선택합니다. 비활성화를 선택한 다음 이 리소스를 선택합니다.
- 모든 리소스를 비활성화하려면 비활성화를 선택한 다음 모든 리소스를 선택합니다.
- 모든 워크플로 작업을 비활성화하려면 비활성화를 선택한 다음 모든 워크플로 작업을 선택합니다.
- 모든 블루프린트를 비활성화하려면 비활성화를 선택한 다음 모든 블루프린트를 선택합니다.

시스템 리소스에 대한 프로젝트 액세스 활성화

프로젝트에서 사용 중이고 비활성화된 시스템 리소스를 활성화하도록 선택할 수 있습니다.

스페이스 관리자 역할 또는 프로젝트 관리자 역할이 있어야 합니다.

시스템 리소스를 활성화하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트로 이동한 다음 프로젝트 설정을 선택합니다. 시스템 리소스를 선택합니다.
3. 다음 중 하나를 선택합니다.
 - 개별적으로 활성화하려면 활성화하려는 하나 이상의 시스템 리소스 옆에 있는 선택기를 선택합니다. 활성화를 선택한 다음 이 리소스를 선택합니다.
 - 모든 리소스를 활성화하려면 활성화를 선택한 다음 모든 리소스를 선택합니다.
 - 모든 워크플로 작업을 활성화하려면 활성화를 선택한 다음 모든 워크플로 작업을 선택합니다.
 - 모든 블루프린트를 활성화하려면 활성화를 선택한 다음 모든 블루프린트를 선택합니다.

프로젝트 할당량

다음 표에서는 Amazon CodeCatalyst의 프로젝트에 대한 할당량 및 제한을 설명합니다. Amazon CodeCatalyst의 할당량에 대한 자세한 내용은 [CodeCatalyst 할당량](#) 섹션을 참조하세요.

스페이스당 최대 프로젝트 수	100
사용자가 속할 수 있는 최대 프로젝트 수	1,000
프로젝트에 속할 수 있는 최대 멤버 수	10,000
프로젝트 이름	프로젝트 이름은 스페이스에서 고유해야 합니다. 이름은 3~63자 이내로 작성해야 합니다. 이름은 대/소문자를 구분합니다. 프로젝트 이름은 영숫자 문자로 시작해야 합니다. 유효한 문자: a~z, A~Z, 0~9, 공백, .(마침표), _(밑줄), -(하이픈)

프로젝트 이름에 포함할 수 없는 문자: ! ? @ # \$ % ^ & * () + = { } [] | \ / > < ~ ` ' " ; :

프로젝트 설명

프로젝트 설명은 최대 200자까지 입력이 가능합니다. 유효한 문자: a~z, A~Z, 0~9, 공백, .(마침표), _(밑줄), -(하이픈) 프로젝트 설명은 선택 사항입니다.

CodeCatalyst에서 알림 전송

CodeCatalyst에서 프로젝트 및 리소스를 모니터링하도록 알림을 설정할 수 있습니다. 사용자는 자신이 멤버인 모든 프로젝트에서 이메일을 수신하려는 프로젝트 이벤트를 선택할 수 있습니다. CodeCatalyst 스페이스와 Slack 작업 영역 간의 액세스를 구성한 다음 해당 Slack 작업 영역의 하나 이상의 채널로 전송할 프로젝트에 대한 알림을 구성하여, Slack과 같은 팀 메시징 애플리케이션의 전체 팀에 전송되는 알림을 구성하도록 선택할 수도 있습니다. CodeCatalyst 스페이스와 Slack 작업 영역 간에 액세스를 구성한 후에는 연결된 Slack 작업 영역 및 채널의 CodeCatalyst 이벤트에 대해 직접 알림을 받을 수 있도록 프로젝트 멤버에게 자체 Slack 멤버 ID를 추가할 수도 있습니다.

Note

Slack으로 전송할 수 있는 프로젝트 이벤트 세트는 사용자가 이메일로 알림을 받기로 선택할 수 있는 이벤트 세트와 다릅니다.

주제

- [알림은 어떻게 작동합니까?](#)
- [스택 알림 시작하기](#)
- [CodeCatalyst에서 Slack 및 이메일 알림 전송](#)

알림은 어떻게 작동합니까?

Slack과 같은 팀 메시징 애플리케이션에 알림을 제공하도록 프로젝트를 설정할 수 있습니다.

알림에는 어떤 권한이 필요한가요?

모든 프로젝트 멤버는 CodeCatalyst의 채널에 대한 알림 설정을 구성, 확인, 업데이트 또는 삭제할 수 있습니다. 그러나 스페이스 관리자 역할을 가진 사용자만 Slack 작업 영역을 추가하거나 삭제할 수 있습니다. 모든 사용자는 CodeCatalyst에서 자신이 속한 프로젝트에 대해 이메일을 수신하려는 프로젝트 이벤트를 구성할 수 있습니다.

알림을 구성할 수 있는 CodeCatalyst 이벤트는 무엇인가요?

워크폴로 이벤트에 대한 알림을 하나 이상의 Slack 채널에 전달하도록 CodeCatalyst를 구성할 수 있습니다. CodeCatalyst 프로젝트와 Slack 간에 알림이 구성되면, 프로젝트 사용자는 자신의 Slack 멤버 ID를 추가하여 CodeCatalyst 이벤트에 대해 Slack 채널에서 다이렉트 메시지를 수신하도록 선택할 수 있습니다. Slack 멤버 ID를 추가하는 사용자는 프로젝트에 구성된 Slack 채널의 ID로 직접 언급을 받게 되므로 관심 있는 이벤트에 대한 인식을 높이는 데 도움이 됩니다.

이메일을 수신하려는 이벤트를 선택할 수도 있습니다. 이러한 이메일은 AWS Builder ID에 대해 구성된 이메일 주소로 전송됩니다.

알림은 어떻게 표시되나요?

하나 이상의 Slack 채널에 알림을 전달하도록 CodeCatalyst를 구성할 수 있습니다. CodeCatalyst에 Slack 작업 영역에 액세스할 수 있는 권한을 부여해야 합니다. 권한이 부여되면 CodeCatalyst는 사용자가 구성한 Slack 채널에 알림을 전송할 수 있습니다. 프로젝트 멤버가 Slack 멤버 ID를 추가하기로 선택하면, 해당 프로젝트에 대해 구성된 Slack 채널에서 CodeCatalyst 이벤트에 대한 언급을 받을 수 있습니다.

알림을 어떻게 설정할 수 있나요?

이메일 알림은 CodeCatalyst의 일부로 구성됩니다. 프로젝트 사용자는 내 설정 페이지에서 이메일을 수신할 이벤트를 선택할 수 있습니다.

프로젝트 리소스에 대한 Slack 알림을 설정하려면 다음과 같은 고급 작업을 완료해야 합니다.

알림을 설정하려면(고급 작업)

1. CodeCatalyst에서 CodeCatalyst와 Slack과 같은 메시징 클라이언트 간의 연결을 설정합니다. Slack 작업 영역이 연결되면 스페이스의 모든 프로젝트에서 사용할 수 있습니다.

Note

스페이스 관리자 역할을 가진 사용자만 Slack 작업 영역을 추가하거나 삭제할 수 있습니다.

2. CodeCatalyst의 프로젝트에 팀이 알림을 수신할 채널을 추가합니다.
3. CodeCatalyst에서는 워크플로 실행 실패와 같은 다양한 이벤트에 대한 알림을 켜고 전송하려는 채널을 지정합니다.

자세한 단계는 [스택 알림 시작하기](#) 섹션을 참조하세요.

CodeCatalyst 스페이스와 Slack 간에 알림이 구성되면, 사용자는 자체 Slack 멤버 ID를 추가하여 프로젝트에 구성된 Slack 채널에서 CodeCatalyst 이벤트에 대한 다이렉트 메시지를 수신하도록 선택할 수 있습니다.

스택 알림 시작하기

프로젝트를 생성한 후 팀이 프로젝트 리소스를 모니터링하는 데 도움이 되는 Slack 알림을 설정할 수 있습니다.

이 단계에서는 CodeCatalyst에서 처음으로 Slack 알림을 설정하는 방법을 안내합니다. 알림을 이미 구성한 경우 [CodeCatalyst에서 Slack 및 이메일 알림 전송](#) 섹션을 참조하세요.

Note

알림 채널로 전송할 수 있는 프로젝트 이벤트 세트는 사용자가 이메일로 알림을 받도록 선택할 수 있는 이벤트 세트와 다릅니다. 자세한 내용은 [CodeCatalyst에서 Slack 및 이메일 알림 전송](#) 섹션을 참조하세요.

주제

- [사전 조건](#)
- [1단계: CodeCatalyst를 Slack 작업 영역에 연결](#)
- [2단계: CodeCatalyst에 Slack 채널 추가](#)
- [3단계: CodeCatalyst에서 Slack으로 알림 테스트](#)
- [4단계: 다음 단계](#)

사전 조건

시작하려면 다음이 필요합니다.

- CodeCatalyst 스페이스. CodeCatalyst 스페이스를 생성하고 처음 로그인하는 방법에 대한 자세한 내용은 [CodeCatalyst 설정 및 로그인](#) 섹션을 참조하세요.
- CodeCatalyst 프로젝트. 자세한 내용은 [프로젝트 생성](#) 섹션을 참조하세요.
- 프로젝트 관리자 또는 스페이스 관리자 역할이 있는 CodeCatalyst 계정. 자세한 내용은 [사용자 역할로 액세스 권한 부여](#) 섹션을 참조하세요.
- CodeCatalyst에서 액세스할 수 있는 Slack 계정 및 Slack 작업 영역.
- CodeCatalyst가 알림을 보낼 Slack 채널. 이 채널은 퍼블릭 또는 프라이빗일 수 있습니다.

1단계: CodeCatalyst를 Slack 작업 영역에 연결

스페이스 관리자 역할을 가진 사용자만 Slack 작업 영역을 추가하거나 삭제할 수 있습니다. Slack 작업 영역을 추가하거나 삭제하면 스페이스의 모든 프로젝트에 영향을 미칩니다. CodeCatalyst와 Slack 간의 연결을 설정하기 위해 CodeCatalyst는 Slack 작업 영역에서 보안 OAuth 인증 핸드셰이크를 수행합니다.

다음 지침을 사용하여 CodeCatalyst를 Slack 작업 영역에 연결합니다.

Note

이는 각 Slack 작업 영역에 대해 한 번만 수행하면 됩니다. 그런 다음 Slack 채널을 통해 알림을 설정할 수 있습니다.

CodeCatalyst를 Slack 작업 영역에 연결하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트로 이동합니다.
3. 탐색 창에서 프로젝트 설정을 선택합니다.
4. 알림 탭을 선택합니다.
5. 알림 구성을 선택합니다.
6. Slack 작업 영역에 연결을 선택합니다.

7. 대화 상자 내용을 읽고 Slack 작업 영역에 연결을 선택합니다.
8. 채팅 애플리케이션의 Amazon Q Developer 메시지에서
 - a. 오른쪽 상단에서 채널이 포함된 Slack 작업 영역을 선택합니다.
 - b. 허용을 선택합니다.

CodeCatalyst 콘솔로 돌아갑니다.

9. 계속해서 [2단계: CodeCatalyst에 Slack 채널 추가](#)로 이동하세요.

2단계: CodeCatalyst에 Slack 채널 추가

CodeCatalyst에 채널을 추가하려면 Slack 채널 ID가 필요합니다.

Slack 채널 ID를 가져오려면

1. Slack에 로그인합니다. 자세한 내용은 [Slack에 로그인](#)을 참조하세요.
2. 알림을 보낼 채널이 포함된 Slack 작업 영역으로 이동합니다. 자세한 내용은 [Slack 작업 영역 간 전환](#) 또는 [추가 Slack 작업 영역에 로그인](#)을 참조하세요.
3. 탐색 창에서 알림을 보낼 채널의 컨텍스트(마우스 오른쪽 버튼 클릭) 메뉴를 열고 채널 세부 정보 열기를 선택합니다.

채널 ID는 대화 상자 하단에 표시됩니다.

4. 채널 ID 값을 복사합니다. 다음 단계에서 이 정보를 사용할 것입니다.

방금 복사한 채널 ID를 사용하여 이제 Slack 채널을 CodeCatalyst에 연결할 수 있습니다.

CodeCatalyst에 Slack 채널을 추가하려면

1. 시작하기 전에 Slack 채널이 프라이빗인 경우 다음과 같이 채팅 애플리케이션에서 채널에 Amazon Q Developer를 추가합니다.
 - a. Slack 채널의 메시지 상자에서 **@aws**를 입력하고 대화 상자에서 aws 앱을 선택합니다.
 - b. 입력을 누릅니다.

채팅 애플리케이션에서 Amazon Q Developer가 프라이빗 채널에 없음을 나타내는 Slackbot 메시지가 나타납니다.

- c. 채팅 애플리케이션에서 Amazon Q Developer를 채널에 초대하려면 초대를 선택합니다.

2. CodeCatalyst 콘솔에서 다음을 선택합니다.
3. 채널 ID에서 이전에 얻은 Slack 채널 ID를 붙여넣습니다.
4. 채널 이름에 이름을 입력합니다. Slack 채널 이름을 사용하는 것이 좋습니다.
5. 다음을 선택합니다.
6. 이벤트 알림 선택에서 알림을 받을 이벤트 유형을 선택합니다.
7. 마침을 클릭합니다.

3단계: CodeCatalyst에서 Slack으로 알림 테스트

워크플로 상태에 대한 알림을 보내도록 프로젝트를 구성한 후 Slack에서 알림을 볼 수 있습니다.

Slack에서 알림을 보려면

1. CodeCatalyst 프로젝트에서 워크플로 실행을 완료하고 실행이 완료되면 상태 알림을 받으려면 [워크플로를 수동으로 시작](#)합니다.
2. Slack에서 알림을 위해 설정한 채널을 확인합니다. 알림에는 각 워크플로 실행의 최신 상태와 실패 또는 성공 여부가 표시됩니다.

4단계: 다음 단계

CodeCatalyst 스페이스에 대해 Slack 작업 영역이 구성되면 기존 CodeCatalyst 프로젝트를 Slack 채널에 추가하고 생성한 후 새 프로젝트에 추가할 수 있습니다. 또한 프로젝트 사용자에게 Slack 멤버 ID에 대한 개인 Slack 알림을 구성하고 이메일을 수신할 이벤트를 구성할 수 있음을 알릴 수 있습니다. 자세한 내용은 [CodeCatalyst에서 Slack 및 이메일 알림 전송](#) 섹션을 참조하세요.

CodeCatalyst에서 Slack 및 이메일 알림 전송

프로젝트에서 발생하는 이벤트에 대한 알림을 보내도록 CodeCatalyst를 구성할 수 있습니다.

CodeCatalyst는 Slack 채널과 같은 메시징 클라이언트에 알림을 보낼 수 있습니다. CodeCatalyst가 Slack 채널로 메시지를 보내면 전체 팀이 워크플로 실패와 같은 중요한 이벤트를 인식하는 데 도움이 됩니다. 선택적으로 CodeCatalyst가 @mention을 보내는 Slack 메시지에 포함시켜 해당 다이렉트 메시지(DM)를 수신하도록 선택할 수 있습니다.

CodeCatalyst는 이메일로 직접 알림을 보낼 수도 있습니다. 멤버인 프로젝트의 이벤트에 대한 이메일 알림이 전송됩니다. 이러한 이메일은 AWS Builder ID에 대해 구성된 이메일 주소로 전송됩니다.

Note

Slack 채널로 전송할 수 있는 이벤트는 이메일로 전송할 수 있는 이벤트와 다를 수 있습니다.

주제

- [이메일 알림 구성](#)
- [Slack 채널로 알림 전송](#)
- [Slack 다이렉트 메시지 구성](#)
- [알림 채널의 알림 편집](#)
- [채널 제거](#)

이메일 알림 구성

멤버인 프로젝트의 이벤트에 대한 이메일 알림을 보내도록 선택할 수 있습니다. 이러한 이메일은 AWS Builder ID에 대해 구성된 이메일 주소로 전송됩니다. 기본적으로 이메일을 보낼 수 있는 모든 프로젝트 이벤트에 대한 이메일을 받게 됩니다.

프로젝트 이벤트에 대한 이메일 알림을 구성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 상단 메뉴 바에서 프로파일 배지를 선택한 다음 내 설정을 선택합니다. CodeCatalyst 내 설정 페이지가 열립니다.

Tip

프로젝트 또는 스페이스의 멤버 페이지로 이동하고 멤버 목록에서 이름을 선택하여 자신의 사용자 프로파일을 찾을 수도 있습니다.

3. 이메일 알림에서 이메일 알림을 구성하려는 목록에서 프로젝트를 찾아 편집을 선택합니다.
4. 이메일을 수신할 이벤트를 선택한 다음 저장을 선택합니다.

Slack 채널로 알림 전송

프로젝트 이벤트에 대한 알림을 팀의 Slack 채널로 보내도록 CodeCatalyst를 구성할 수 있습니다. 이렇게 하면 전체 팀이 워크플로 실행이 실패할 때와 같은 중요한 이벤트를 인식할 수 있습니다.

Note

프로젝트의 모든 멤버는 해당 프로젝트의 채널로 전송되는 알림을 관리할 수 있습니다. 그러나 스페이스 관리자 역할을 가진 사용자만 Slack 작업 영역을 추가하거나 삭제할 수 있습니다.

다음 지침에 따라 알림을 전송할 Slack 채널을 추가합니다.

알림을 전송하기 위해 Slack 채널을 추가하려면

1. 첫 번째 Slack 채널을 추가하는 경우 대신 [스택 알림 시작하기](#) 섹션을 참조하세요.

첫 번째 채널을 설정한 후 이 절차로 돌아와 추가 채널을 설정합니다.

2. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
3. 프로젝트로 이동합니다.
4. 탐색 창에서 프로젝트 설정을 선택합니다.
5. 알림 탭을 선택합니다.
6. Add channel(채널 추가)을 선택합니다.
7. 작업 영역 선택을 선택한 다음 알림을 보낼 채널이 포함된 Slack 작업 영역을 선택합니다.

Slack 작업 영역이 목록에 없는 경우 [스택 알림 시작하기](#)의 지침에 따라 추가할 수 있습니다.
8. 채널 ID를 입력하기 전에, 추가하려는 Slack 채널이 비공개 채널인 경우 다음 단계를 완료합니다.
 - a. Slack 채널의 메시지 상자에 팝업에서 **@aws**를 입력하고 aws 앱을 선택합니다.
 - b. Enter를 누릅니다.

채팅 애플리케이션에서 Amazon Q Developer가 프라이빗 채널에 없음을 나타내는 Slackbot 메시지가 나타납니다.

- c. 채팅 애플리케이션에서 Amazon Q Developer를 채널에 초대하려면 초대를 선택합니다.
9. CodeCatalyst 채널 ID 필드에 Slack 채널 ID를 입력합니다. ID를 찾으려면 Slack으로 이동하여 탐색 창에서 채널을 마우스 오른쪽 버튼으로 클릭하고 채널 세부 정보 열기를 선택합니다.

채널 ID는 대화 상자 하단에 표시됩니다.

10. 채널 이름에 이름을 입력합니다. Slack 채널 이름을 사용하는 것이 좋습니다.
11. 이벤트 알림 선택에서 알림을 받을 이벤트 유형을 선택합니다.
12. 추가를 선택합니다.

Slack 다이렉트 메시지 구성

CodeCatalyst 프로젝트가 [Slack 채널에 알림을 보내도록](#) 구성된 경우 이러한 알림은 다이렉트 메시지 (DM)로 전송할 수 있습니다. DM으로 알림을 직접 보내면 역할이 있는 프로젝트에서 발생하는 이벤트에 대한 인식을 높이는 데 도움이 될 수 있습니다. DM을 활성화하려면 CodeCatalyst에 Slack 멤버 ID를 추가해야 합니다.

Slack 다이렉트 메시지를 구성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 상단 메뉴 바에서 프로파일 배지를 선택한 다음 내 설정을 선택합니다. CodeCatalyst 내 설정 페이지가 열립니다.

 Tip

프로젝트 또는 스페이스의 멤버 페이지로 이동하고 멤버 목록에서 이름을 선택하여 자신의 사용자 프로파일을 찾을 수도 있습니다.

3. 개인 Slack 알림에서 Slack ID 연결을 선택한 다음 Slack 작업 영역에 연결을 선택합니다. 별도의 창이 열립니다.

 Tip

스페이스 관리자 역할이 있는 사용자가 CodeCatalyst 스페이스에 Slack 작업 영역을 추가하지 않는 한, 이 옵션은 구성할 수 없습니다. 자세한 내용은 [스택 알림 시작하기](#) 및 [Slack 채널로 알림 전송\(을\)](#)를 참조하세요.

4. 권한 요청 창에서 작업 영역의 이름이 CodeCatalyst 스페이스에 대해 구성된 Slack 작업 영역과 일치하는지 확인합니다. 채팅 애플리케이션에서 Amazon Q Developer가 워크스페이스에 액세스할 수 있도록 허용하려면 허용을 선택합니다. 창이 닫히고 Slack 작업 영역에 연결 상태가 연결됨으로 표시됩니다.

 Tip

연결 상태가 변경되지 않는 경우 Slack 작업 영역을 연결하는 동안 오류가 발생했는지 확인합니다. 오류를 보려면 위로 스크롤해야 할 수 있습니다.

5. 개인 Slack 알림 수신을 중지하려면 연결된 Slack 작업 영역을 선택한 다음 Slack ID 연결 해제를 선택합니다.

알림 채널의 알림 편집

알림이 전송되는 채널을 변경하고 특정 알림을 모두 끌 수 있습니다.

알림을 편집하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트로 이동합니다.
3. 탐색 창에서 프로젝트 설정을 선택합니다.
4. 알림 탭을 선택합니다.
5. 알림 편집을 선택합니다.
6. 다음 중 하나를 수행합니다.
 - 특정 채널에 알림을 보내려면 드롭다운 목록에서 채널을 선택합니다.
 - 전역적으로 알림을 끄려면 알림 옆의 토글을 선택합니다.
 - 특정 채널로 알림 전송을 중지하려면 채널에서 X를 선택합니다.
7. 저장을 선택합니다.

채널 제거

Amazon CodeCatalyst에서 Slack 채널을 제거할 수 있습니다. Slack 채널을 제거하면 선택한 CodeCatalyst 프로젝트에 대한 알림이 더 이상 채널로 전송되지 않습니다.

채널을 제거하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트로 이동합니다. 탐색 창에서 프로젝트 설정을 선택합니다.
3. 프로젝트 설정 페이지에서 알림 탭을 선택합니다.
4. 제거할 채널 옆의 표시기를 선택한 다음 채널 제거를 선택합니다. 확인 창에서 확인을 선택합니다.

블루프린트를 사용하여 CodeCatalyst 프로젝트 설정

블루프린트는 CodeCatalyst 프로젝트의 아키텍처 구성 요소를 나타내는 임의 코드 생성기입니다. 구성 요소는 단일 파일의 워크플로에서 샘플 코드가 포함된 전체 프로젝트까지 모든 것으로 구성될 수 있습니다. 블루프린트는 임의의 옵션 세트를 가져와 이를 사용하여 프로젝트로 전달되는 임의의 출력 코드 세트를 생성합니다. 블루프린트가 최신 모범 사례 또는 새 옵션으로 업데이트되면 해당 블루프린트가 포함된 프로젝트에서 코드베이스의 관련 부분을 재생성할 수 있습니다.

Amazon CodeCatalyst 블루프린트를 사용하여 소스 리포지토리, 샘플 소스 코드, CI/CD 워크플로, 빌드 및 테스트 보고서, 통합 문제 추적 도구를 사용하여 전체 프로젝트를 생성할 수 있습니다. CodeCatalyst 블루프린트는 구성 파라미터 세트를 기반으로 리소스와 소스 코드를 생성합니다. CodeCatalyst 관리형 블루프린트를 사용할 때 선택한 블루프린트에 따라 프로젝트에 추가되는 리소스와 CodeCatalyst가 생성하거나 구성하는 도구가 결정되므로 프로젝트 리소스를 추적하고 사용할 수 있습니다. 블루프린트 사용자는 블루프린트를 사용하여 프로젝트를 생성하거나 기존 CodeCatalyst 프로젝트에 추가할 수 있습니다. 프로젝트에 여러 블루프린트를 추가할 수 있으며 각각을 독립 구성 요소로 적용할 수 있습니다. 예를 들어 웹 애플리케이션 블루프린트로 생성된 프로젝트를 만든 다음 나중에 보안 블루프린트를 추가할 수 있습니다. 블루프린트 중 하나가 업데이트되면 수명 주기 관리를 통해 프로젝트에 변경 사항 또는 수정 사항을 통합할 수 있습니다. 자세한 내용은 [CodeCatalyst 블루프린트를 사용하여 포괄적인 프로젝트 생성](#) 및 [블루프린트 사용자로서 수명 주기 관리 작업](#) 섹션을 참조하세요.

블루프린트 작성자는 CodeCatalyst 스페이스 멤버가 프로젝트 리소스를 사용할 수 있도록 사용자 지정 블루프린트를 생성하고 게시할 수도 있습니다. 사용자 지정 블루프린트는 스페이스 프로젝트에 대한 지정된 요구 사항을 충족하도록 개발할 수 있습니다. 스페이스의 블루프린트 카탈로그에 사용자 지정 블루프린트를 추가한 후 블루프린트를 관리하고 계속 업데이트하여 스페이스의 프로젝트가 최신 모범 사례를 유지하도록 할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 사용자 지정 블루프린트를 사용하여 프로젝트 표준화](#) 섹션을 참조하세요. 블루프린트 SDK 및 샘플 블루프린트를 보려면 [오픈 소스 GitHub 리포지토리](#)를 참조하세요.

이미 표준화 및 모범 사례가 마련되어 있을 수 있습니다. 사용자 지정 블루프린트를 처음부터 생성하고 개발하는 대신 소스 코드가 있는 기존 소스 리포지토리를 사용자 지정 블루프린트로 변환하도록 선택할 수 있습니다. 자세한 내용은 [소스 리포지토리를 사용자 지정 블루프린트로 변환](#) 섹션을 참조하세요.

주제

- [블루프린트를 사용하여 프로젝트 생성](#)
- [프로젝트에 블루프린트를 추가하여 리소스 통합](#)
- [업데이트 중지를 위해 프로젝트에서 블루프린트 연결 해제](#)

- [프로젝트의 블루프린트 버전 변경](#)
- [프로젝트의 블루프린트에 대한 설명 편집](#)
- [블루프린트 사용자로서 수명 주기 관리 작업](#)
- [CodeCatalyst 블루프린트를 사용하여 포괄적인 프로젝트 생성](#)
- [CodeCatalyst에서 사용자 지정 블루프린트를 사용하여 프로젝트 표준화](#)
- [CodeCatalyst의 블루프린트 할당량](#)

블루프린트를 사용하여 프로젝트 생성

Amazon CodeCatalyst 블루프린트 카탈로그 또는 사용자 지정 블루프린트가 있는 팀의 스페이스 카탈로그에서 블루프린트를 사용하여 프로젝트를 빠르게 생성할 수 있습니다. 블루프린트에 따라 프로젝트는 특정 리소스로 생성됩니다. 새 프로젝트를 생성하거나 기존 프로젝트에 구성 요소를 추가할 때 생성형 AI 어시스턴트인 Amazon Q와 협업할 수도 있습니다. 채팅과 유사한 인터페이스에서 프로젝트와 상호 작용하여 프로젝트의 요구 사항을 Amazon Q에 제공할 수 있습니다. 요구 사항에 따라 Amazon Q는 블루프린트를 제안하고 충족할 수 없는 요구 사항을 간략하게 설명합니다. 그런 다음 만족하면 Amazon Q의 제안을 진행할 수 있으며, 요구 사항에 맞는 코드가 포함된 소스 리포지토리와 같은 필요한 리소스를 생성합니다. 자세한 내용은 [블루프린트를 사용하여 프로젝트 생성](#), [Amazon Q를 사용하여 프로젝트를 생성하거나 블루프린트를 사용하여 기능을 추가할 때의 모범 사례](#), [CodeCatalyst 블루프린트를 사용하여 포괄적인 프로젝트 생성](#) 섹션을 참조하세요.

프로젝트를 생성한 후 CodeCatalyst 카탈로그 또는 사용자 지정 블루프린트를 사용하여 스페이스의 카탈로그에서 CodeCatalyst 프로젝트에 블루프린트를 추가할 수 있습니다. 블루프린트는 아키텍처 구성 요소를 나타내므로 프로젝트에 여러 블루프린트를 함께 사용하여 팀의 모범 사례를 통합할 수 있습니다. 이렇게 하면 진화하는 구성 요소에 대한 최신 변경 사항을 프로젝트가 최신 상태로 유지되도록 할 수 있습니다. 프로젝트의 블루프린트 작업에 대한 자세한 내용은 섹션을 참조하세요. [블루프린트 사용자로서 수명 주기 관리 작업](#).

프로젝트에 블루프린트를 추가하여 리소스 통합

프로젝트에 여러 블루프린트를 추가하여 기능적 구성 요소, 리소스 및 거버넌스를 통합할 수 있습니다. 프로젝트는 별도의 블루프린트에서 독립적으로 관리되는 다양한 요소를 지원할 수 있습니다. 프로젝트에 블루프린트를 추가하면 리소스를 수동으로 생성하고 소프트웨어 구성 요소를 사용할 필요가 줄어듭니다. 요구 사항이 진화함에 따라 프로젝트도 최신 상태를 유지할 수 있습니다. 프로젝트에 블루프린트를 추가하는 방법에 대한 자세한 내용은 [블루프린트 사용자로서 수명 주기 관리 작업](#) 섹션을 참조하세요.

블루프린트의 세부 정보를 구성하는 동안 블루프린트의 소스 코드를 선호하는 타사 리포지토리에 저장하도록 선택할 수도 있습니다. 이 리포지토리에서는 블루프린트를 관리하고 수명 주기 관리 기능을 활용하여 프로젝트를 최신 상태로 유지할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 확장 프로그램이 있는 프로젝트에 기능 추가](#) 및 [블루프린트 사용자로서 수명 주기 관리 작업](#) 섹션을 참조하세요.

⚠ Important

CodeCatalyst 프로젝트에 블루프린트를 추가하려면 스페이스 관리자, 고급 사용자 또는 프로젝트 관리자 역할이 있는 계정으로 스페이스에 로그인해야 합니다.

💡 Tip

프로젝트에 블루프린트를 추가한 후 이메일 및 Slack 알림을 구성하여 블루프린트의 최신 변경 사항에 대한 업데이트를 제공할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 알림 전송](#) 섹션을 참조하세요.

프로젝트에 블루프린트를 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 스페이스로 이동한 다음 블루프린트를 추가할 프로젝트를 선택합니다.
3. 탐색 창에서 블루프린트를 선택한 다음 블루프린트 사용을 선택합니다.

💡 Tip

Amazon Q가 블루프린트를 제안하도록 프로젝트 요구 사항을 Amazon Q에 제공하여 블루프린트를 추가하도록 선택할 수 있습니다. 자세한 내용은 [Amazon Q를 사용하여 프로젝트를 생성하거나 기능을 추가할 때 블루프린트 선택](#) 및 [Amazon Q를 사용하여 프로젝트를 생성하거나 블루프린트를 사용하여 기능을 추가할 때의 모범 사례](#) 섹션을 참조하세요. 이 기능은 미국 서부(오리건) 리전에서만 사용할 수 있습니다.

이 기능을 사용하려면 스페이스에 생성형 AI 기능을 활성화해야 합니다. 자세한 내용은 [생성형 AI 기능 관리](#)를 참조하세요.

4. CodeCatalyst 블루프린트 탭에서 블루프린트를 선택하거나 스페이스 블루프린트 탭에서 사용자 지정 블루프린트를 선택한 뒤 다음을 선택합니다.

5. 블루프린트 세부 정보의 대상 버전 드롭다운 메뉴에서 블루프린트 버전을 선택합니다. 최신 카탈로그 버전이 자동으로 선택됩니다.
6. (선택 사항) 기본적으로 블루프린트에서 생성된 소스 코드는 CodeCatalyst 리포지토리에 저장됩니다. 또는 블루프린트의 소스 코드를 타사 리포지토리에 저장하도록 선택할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 확장 프로그램이 있는 프로젝트에 기능 추가](#) 섹션을 참조하세요.

사용하려는 타사 리포지토리 공급자에 따라 다음 중 하나를 수행합니다.

- GitHub 리포지토리: GitHub 계정을 연결합니다.

고급 드롭다운 메뉴를 선택하고 GitHub를 리포지토리 공급자로 선택한 다음 블루프린트에서 생성된 소스 코드를 저장할 GitHub 계정을 선택합니다.

Note

GitHub 계정에 대한 연결을 사용하는 경우 개인 연결을 생성하여 CodeCatalyst ID와 GitHub ID 간에 ID 매핑을 설정해야 합니다. 자세한 내용은 [개인 연결](#) 및 [개인 연결을 사용하여 GitHub 리소스에 액세스](#) 섹션을 참조하세요.

- Bitbucket 리포지토리: Bitbucket 작업 영역을 연결합니다.

고급 드롭다운 메뉴를 선택하고 Bitbucket을 리포지토리 공급자로 선택한 다음, 블루프린트에서 생성된 소스 코드를 저장할 Bitbucket 작업 영역을 선택합니다.

- GitLab 리포지토리: GitLab 사용자를 연결합니다.

고급 드롭다운 메뉴를 선택하고 GitLab을 리포지토리 공급자로 선택한 다음 블루프린트에서 생성된 소스 코드를 저장할 GitLab 사용자를 선택합니다.

7. 블루프린트 구성에서 블루프린트 파라미터를 구성합니다. 블루프린트에 따라 소스 리포지토리의 이름을 지정할 수 있는 옵션이 있을 수 있습니다.
8. 현재 블루프린트 버전과 업데이트된 버전의 차이점을 검토합니다. 풀 요청에 표시되는 차이는 풀 요청이 생성된 시점의 원하는 버전인 현재 버전과 최신 버전 간의 변경 사항을 보여줍니다. 변경 사항이 표시되지 않으면 버전이 동일하거나 현재 버전과 원하는 버전 모두에 대해 동일한 버전을 선택했을 수 있습니다.
9. 풀 요청에 검토하려는 코드와 변경 사항이 포함되어 있다고 생각되면 블루프린트 추가를 선택합니다. 풀 요청을 생성한 후 설명을 추가할 수 있습니다. 풀 요청 또는 파일의 개별 줄 및 풀 요청 전반에 설명을 추가할 수 있습니다. @ 기호와 파일 이름을 사용하여 파일과 같은 리소스에 대한 연결을 추가할 수 있습니다.

Note

풀 요청이 승인되고 병합될 때까지 블루프린트가 적용되지 않습니다. 자세한 내용은 [풀 요청 검토](#) 및 [풀 요청 병합](#) 섹션을 참조하세요.

블루프린트 작성자는 새 프로젝트를 생성하거나 기존 프로젝트에 추가할 수 있는 블루프린트가 없는 지정된 스페이스의 프로젝트에 사용자 지정 블루프린트를 추가할 수도 있습니다. 자세한 내용은 [지정된 스페이스 및 프로젝트에 사용자 지정 블루프린트 게시 및 추가](#) 섹션을 참조하세요.

블루프린트에 대한 업데이트를 더 이상 수신하지 않으려면 프로젝트에서 블루프린트를 연결 해제할 수 있습니다. 자세한 내용은 [업데이트 중지를 위해 프로젝트에서 블루프린트 연결 해제](#) 섹션을 참조하세요.

업데이트 중지를 위해 프로젝트에서 블루프린트 연결 해제

블루프린트의 새 업데이트를 원하지 않는 경우 프로젝트에서 블루프린트를 연결 해제할 수 있습니다. 블루프린트에서 프로젝트에 추가된 리소스 및 기능 소프트웨어 구성 요소는 프로젝트에 남아 있습니다.

Important

CodeCatalyst 프로젝트에서 블루프린트를 연결 해제하려면 스페이스에 스페이스 관리자, 파워 사용자 또는 프로젝트 관리자 역할이 있는 계정으로 로그인해야 합니다.

프로젝트에서 블루프린트 연결을 해제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 스페이스로 이동한 다음 블루프린트의 연결을 해제하려는 프로젝트를 선택합니다.
3. 탐색 창에서 [블루프린트(Blueprints)]를 선택합니다.
4. 연결 해제하려는 리소스가 있는 블루프린트를 선택하고 작업 드롭다운 메뉴를 선택한 다음 블루프린트 연결 해제를 선택합니다.
5. confirm을 입력하여 연결 해제를 확인합니다.
6. 확인(Confirm)을 선택합니다.

프로젝트의 블루프린트 버전 변경

블루프린트가 포함된 프로젝트를 생성하거나 기존 프로젝트에 블루프린트를 추가한 경우 블루프린트의 새 버전에 대한 알림을 받게 됩니다. 승인된 풀 요청을 통해 블루프린트 버전을 업데이트하기 전에 코드 변경 사항과 영향을 받는 환경을 볼 수 있습니다. 라이프사이클 관리를 사용하면 프로젝트에 적용된 하나 이상의 블루프린트 버전을 변경할 수 있으므로 프로젝트의 다른 영역에 영향을 주지 않고 각 블루프린트 버전을 변경할 수 있습니다. 블루프린트 업데이트를 재정의할 수도 있습니다. 자세한 내용은 [블루프린트 사용자로서 수명 주기 관리 작업](#) 섹션을 참조하세요.

Important

CodeCatalyst 프로젝트에서 청사진의 버전을 변경하려면 스페이스 관리자, 파워 유저 또는 프로젝트 관리자 역할이 있는 계정으로 스페이스에 로그인해야 합니다.

Tip

프로젝트에 블루프린트를 추가한 후 이메일 및 Slack 알림을 구성하여 블루프린트의 최신 변경 사항에 대한 업데이트를 제공할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 알림 전송](#) 섹션을 참조하세요.

블루프린트를 최신 버전으로 업데이트

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 블루프린트 버전을 업데이트하려는 스페이스로 이동합니다.
3. 스페이스 대시보드에서 업데이트하려는 블루프린트가 있는 프로젝트를 선택합니다.
4. 탐색 창에서 [블루프린트(Blueprints)]를 선택합니다.
5. 상태 열에서 링크를 선택하여 카탈로그 버전을 변경합니다(예: 카탈로그 버전 0.3.109 변경).
6. 작업 드롭다운 메뉴를 선택한 다음 버전 업데이트를 선택합니다. 최신 버전이 자동으로 선택됩니다.
7. (선택 사항) 블루프린트 구성에서 블루프린트 파라미터를 구성합니다.
8. (선택 사항) 코드 변경 탭에서 현재 블루프린트 버전과 업데이트된 버전의 차이점을 검토합니다. 풀 요청에 표시되는 차이점은 풀 요청이 생성될 때 원하는 버전인 현재 버전과 최신 버전 간의 변경 사항입니다. 변경 사항이 표시되지 않으면 버전이 동일하거나 현재 버전과 원하는 버전 모두에 대해 동일한 버전을 선택했을 수 있습니다.

9. 풀 요청에 검토하려는 코드와 변경 사항이 포함되어 있다고 생각되면 업데이트 적용을 선택합니다. 풀 요청을 생성한 후 설명을 추가할 수 있습니다. 풀 요청 또는 파일의 개별 줄 및 풀 요청 전반에 설명을 추가할 수 있습니다. @ 기호와 파일 이름을 사용하여 파일과 같은 리소스에 대한 연결을 추가할 수 있습니다.

Note

풀 요청이 승인되고 병합될 때까지 블루프린트가 업데이트되지 않습니다. 자세한 내용은 [풀 요청 검토](#) 및 [풀 요청 병합](#) 섹션을 참조하세요.

Note

블루프린트 버전을 업데이트하기 위해 기존 풀 요청이 열려 있는 경우 새 풀 요청을 생성하기 전에 이전 풀 요청을 닫으세요. 버전 업데이트를 선택하면 블루프린트에 대해 보류 중인 풀 요청 목록으로 이동합니다. 프로젝트 설정 및 프로젝트 요약 페이지의 블루프린트 탭에서 보류 중인 풀 요청을 볼 수도 있습니다. 자세한 내용은 [풀 요청 보기](#) 섹션을 참조하세요.

블루프린트 버전 변경

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 블루프린트 버전을 업데이트하려는 스페이스로 이동합니다.
3. 스페이스 대시보드에서 업데이트하려는 블루프린트가 있는 프로젝트를 선택합니다.
4. 탐색 창에서 블루프린트를 선택한 다음 업데이트하려는 블루프린트의 라디오 버튼을 선택합니다.
5. 작업 드롭다운을 선택한 다음, 블루프린트 구성을 선택합니다.
6. 타겟 버전 드롭다운에서 사용하려는 버전을 선택합니다. 최신 버전이 자동으로 선택됩니다.
7. (선택 사항) 블루프린트 구성에서 블루프린트 파라미터를 구성합니다.
8. (선택 사항) 코드 변경 탭에서 현재 블루프린트 버전과 업데이트된 버전의 차이점을 검토합니다. 풀 요청에 표시되는 차이점은 풀 요청이 생성될 때 원하는 버전인 현재 버전과 최신 버전 간의 변경 사항입니다. 변경 사항이 표시되지 않으면 버전이 동일하거나 현재 버전과 원하는 버전 모두에 대해 동일한 버전을 선택했을 수 있습니다.
9. 풀 요청에 검토하려는 코드와 변경 사항이 포함되어 있다고 생각되면 업데이트 적용을 선택합니다. 풀 요청을 생성한 후 설명을 추가할 수 있습니다. 풀 요청 또는 파일의 개별 줄 및 풀 요청 전반

에 설명을 추가할 수 있습니다. @ 기호와 파일 이름을 사용하여 파일과 같은 리소스에 대한 연결을 추가할 수 있습니다.

Note

풀 요청이 승인되고 병합될 때까지 블루프린트가 업데이트되지 않습니다. 자세한 내용은 [풀 요청 검토](#) 및 [풀 요청 병합](#) 섹션을 참조하세요.

Note

블루프린트 버전을 업데이트하기 위해 기존 풀 요청이 열려 있는 경우 새 풀 요청을 생성하기 전에 이전 풀 요청을 닫으세요. 버전 업데이트를 선택하면 블루프린트에 대해 보류 중인 풀 요청 목록으로 이동합니다. 프로젝트 설정 및 프로젝트 요약 페이지의 블루프린트 탭에서 보류 중인 풀 요청을 볼 수도 있습니다. 자세한 내용은 [풀 요청 보기](#) 섹션을 참조하세요.

프로젝트의 블루프린트에 대한 설명 편집

프로젝트를 만들 때 사용했거나 프로젝트가 생성된 후 적용한 블루프린트의 설명을 편집할 수 있습니다. 블루프린트는 프로젝트에서 두 번 이상 사용할 수 있습니다. 프로젝트에서 블루프린트의 목적을 구분하기 위해 해당 블루프린트에 대한 설명을 사용할 수 있습니다. 설명은 특정 블루프린트에서 추가하는 구성 요소를 식별하는 데에도 사용할 수 있습니다.

Important

스페이스에서 사용자 지정 블루프린트의 설명을 편집하려면 스페이스에서 스페이스 관리자, 파워 유저 또는 프로젝트 관리자 역할이 있는 계정으로 로그인해야 합니다.

프로젝트에서 블루프린트의 설명 편집

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 스페이스로 이동한 다음 업데이트하려는 블루프린트 설정이 있는 프로젝트를 선택합니다.
3. 탐색 창에서 [블루프린트(Blueprints)]를 선택합니다.

4. 업데이트하려는 설명이 있는 블루프린트를 선택하고 작업 드롭다운 메뉴를 선택한 다음 설정을 선택합니다.
5. 블루프린트 설명 텍스트 입력 필드에 프로젝트의 블루프린트를 식별할 수 있는 설명을 입력합니다.
6. 저장을 선택합니다.

블루프린트 사용자로서 수명 주기 관리 작업

수명 주기 관리는 업데이트된 옵션 또는 블루프린트 버전에서 코드베이스를 재생성하는 기능입니다. 이를 통해 블루프린트 작성자는 특정 블루프린트가 포함된 모든 프로젝트의 소프트웨어 개발 수명 주기를 중앙에서 관리할 수 있습니다. 예를 들어, 웹 애플리케이션 블루프린트에 보안 수정을 푸시하면 웹 애플리케이션 블루프린트를 포함하거나 웹 애플리케이션 블루프린트에서 생성된 모든 프로젝트가 해당 수정 사항을 자동으로 픽업할 수 있습니다. 또한 이 동일한 관리 프레임워크를 사용하면 블루프린트 사용자가 블루프린트 옵션을 선택하고 나서 변경할 수 있습니다.

주제

- [기존 프로젝트에서 수명 주기 관리 사용](#)
- [프로젝트의 여러 블루프린트에서 수명 주기 관리 사용](#)
- [수명 주기 풀 요청의 충돌 처리](#)
- [수명 주기 관리 변경 사항 옵트아웃](#)
- [프로젝트에서 블루프린트의 수명 주기 관리 재정의](#)

기존 프로젝트에서 수명 주기 관리 사용

블루프린트에서 생성된 프로젝트 또는 블루프린트와 연결되지 않은 기존 프로젝트에 수명 주기 관리를 사용할 수 있습니다. 예를 들어, 블루프린트에서 생성되지 않은 5년 된 Java 애플리케이션에 표준 보안 관행 블루프린트를 추가할 수 있습니다. 해당 블루프린트는 보안 스캔 워크플로 및 기타 관련 코드를 생성합니다. 이제 Java 애플리케이션의 코드베이스 중 해당 부분은 블루프린트에 변경 사항이 생길 때마다 팀의 모범 사례에 따라 자동으로 최신 상태로 유지됩니다.

프로젝트의 여러 블루프린트에서 수명 주기 관리 사용

블루프린트는 아키텍처 구성 요소를 나타내기 때문에 동일한 프로젝트에서 여러 블루프린트를 함께 사용할 수 있습니다. 예를 들어, 프로젝트는 회사 플랫폼 엔지니어가 구축한 중앙 웹 API 블루프린트와

앱 보안 팀이 구축한 릴리스 확인 블루프린트로 구성될 수 있습니다. 이러한 각 블루프린트는 독립적으로 업데이트할 수 있으며 과거에 적용된 병합 해결책을 기억합니다.

Note

임의의 아키텍처 구성 요소로서 블루프린트가 서로 병합하려고 하더라도 모든 블루프린트가 합당하거나 논리적인 방식으로 함께 작동하는 것은 아닙니다.

수명 주기 풀 요청의 충돌 처리

경우에 따라 수명 주기 풀 요청으로 인해 병합 충돌이 발생할 수 있습니다. 이러한 문제를 수동으로 해결할 수 있습니다. 후속 블루프린트 업데이트 시 해결책이 기억됩니다.

수명 주기 관리 변경 사항 옵트아웃

사용자는 프로젝트에서 블루프린트를 제거하여 블루프린트에 대한 모든 참조를 연결 해제하고 수명 주기 업데이트를 옵트아웃할 수 있습니다. 안전상의 이유로, 블루프린트에서 추가된 것을 포함하여 프로젝트의 코드 또는 리소스는 제거되거나 영향을 받지 않습니다. 자세한 내용은 [업데이트 중지를 위해 프로젝트에서 블루프린트 연결 해제](#) 단원을 참조하십시오.

프로젝트에서 블루프린트의 수명 주기 관리 재정의

프로젝트의 특정 파일에 대한 블루프린트 업데이트를 재정의하려면 리포지토리에 소유권 파일을 포함할 수 있습니다. [GitLab's Code Owners](#) 사양은 권장 지침입니다. 블루프린트는 항상 다른 모든 파일보다 Code Owners 파일을 존중하며 다음과 같은 샘플을 생성할 수 있습니다.

```
new BlueprintOwnershipFile(sourceRepo, {
  resynthesis: {
    strategies: [
      {
        identifier: 'dont-override-sample-code',
        description: 'This strategy is applied accross all sample code. The
        blueprint will create sample code, but skip attempting to update it.',
        strategy: MergeStrategies.neverUpdate,
        globs: [
          '**/src/**',
          '**/css/**',
        ],
      },
    ],
  },
})
```

```
    ],
  },
});
```

이렇게 하면 다음 내용이 포함된 `.ownership-file`이 생성됩니다.

```
[dont-override-sample-code] @amazon-codecatalyst/blueprints.import-from-git
# This strategy is applied accross all sample code. The blueprint will create sample
  code, but skip attempting to update it.
# Internal merge strategy: neverUpdate
**/src/**
**/css/**
```

CodeCatalyst 블루프린트를 사용하여 포괄적인 프로젝트 생성

블루프린트를 사용하여 프로젝트를 생성할 때 CodeCatalyst는 소스 리포지토리, 샘플 소스 코드, CI/CD 워크플로, 빌드 및 테스트 보고서, 통합 문제 추적 도구를 사용하여 전체 프로젝트를 생성합니다. 프로젝트 블루프린트는 코드를 사용하여 다양한 유형의 애플리케이션 및 프레임워크에 클라우드 인프라, 리소스 및 샘플 소스 아티팩트를 프로비저닝합니다.

자세한 내용은 [프로젝트 생성](#) 단원을 참조하십시오. 프로젝트를 생성하려면 스페이스 관리자여야 합니다.

주제

- [사용 수 있는 블루프린트](#)
- [프로젝트 블루프린트 정보 찾기](#)

사용 수 있는 블루프린트

블루프린트 이름	블루프린트 설명
ASP.NET 코어 웹 API	이 블루프린트는 .NET 6 ASP.NET Core 웹 API 애플리케이션을 생성합니다. 블루프린트는 .NET용 AWS 배포 도구를 사용하고 Amazon Elastic Container Service AWS App Runner또는 AWS Elastic Beanstalk 배포 대상으로 구성하는 옵션을 제공합니다.

블루프린트 이름	블루프린트 설명
AWS Glue ETL	이 블루프린트는 AWS CDK, AWS Glue, AWS Lambda 및 Amazon Athena를 사용하여 샘플 추출 변환 로드(ETL) 참조 구현을 생성하여 심표로 구분된 값(CSVs)을 Apache Parquet으로 변환합니다.
DevOps 배포 파이프라인	이 블루프린트는 여러 단계에 AWS 걸쳐 참조 애플리케이션을 배포하는 AWS 배포 파이프라인 참조 아키텍처를 사용하여 배포 파이프라인을 생성합니다.
를 사용한 Java API AWS Fargate	이 블루프린트는 컨테이너화된 웹 서비스 프로젝트를 생성합니다. 이 프로젝트는 AWS Copilot CLI 를 사용하여, Amazon ECS의 Amazon DynamoDB에서 지원하는 컨테이너화된 Spring Boot Java 웹 서비스를 구축하고 배포합니다. 프로젝트는 컨테이너화된 앱을 AWS Fargate 서버리스 컴퓨팅의 Amazon ECS 클러스터에 배포합니다. 해당 앱은 DynamoDB의 테이블에 데이터를 저장합니다. 워크플로가 성공적으로 실행되면 Application Load Balancer를 통해 샘플 웹 서비스를 공개적으로 사용할 수 있습니다.
최신 3계층 웹 애플리케이션	이 블루프린트는 잘 설계된 3계층 최신 웹 애플리케이션을 구축하고 배포하기 위해 애플리케이션 계층 및 Vue 프론트엔드 프레임워크에 대한 코드를 Python에서 생성합니다.
.NET 서버리스 애플리케이션	이 블루프린트는 .NET CLI Lambda 도구를 사용하여 AWS Lambda 함수를 생성합니다. 블루프린트는 C# 또는 F# 선택을 포함하여 AWS Lambda 함수에 대한 옵션을 제공합니다.

블루프린트 이름	블루프린트 설명
를 사용한 Node.js API AWS Fargate	이 블루프린트는 컨테이너화된 웹 서비스 프로젝트를 생성합니다. 프로젝트는 AWS Copilot CLI 를 사용하여, Amazon Elastic Container Service에서 컨테이너화된 Express/Node.js 웹 서비스를 빌드하고 배포합니다. 프로젝트는 컨테이너화된 앱을 AWS Fargate 서버리스 컴퓨팅의 Amazon ECS 클러스터에 배포합니다. 워크플로가 성공적으로 실행되면 Application Load Balancer를 통해 샘플 웹 서비스를 공개적으로 사용할 수 있습니다.
서버리스 애플리케이션 모델(SAM)	이 블루프린트는 서버리스 애플리케이션 모델(SAM)을 사용하여 API를 생성하고 배포하는 프로젝트를 생성합니다. 프로그래밍 언어로 SDK for Java, TypeScript 또는 SDK for Python을 선택할 수 있습니다.
서버리스 RESTful 마이크로서비스	이 블루프린트는 할 일 서비스 참조와 Amazon API Gateway 함께 AWS Lambda 및를 사용하는 REST API를 생성합니다. 프로그래밍 언어로 SDK for Java, TypeScript 또는 SDK for Python을 선택할 수 있습니다.
단일 페이지 애플리케이션	이 블루프린트는 React, Vue 및 Angular 프레임워크를 사용하는 단일 페이지 애플리케이션(SPA)을 생성합니다. 호스팅의 경우 AWS Amplify 호스팅 또는 Amazon CloudFront 및 Amazon S3 중에서 선택합니다.

블루프린트 이름	블루프린트 설명
정적 웹 사이트	이 블루프린트는 Hugo 또는 Jekyll 정적 사이트 생성기를 사용하여 정적 웹 사이트를 생성합니다. 정적 사이트 생성기는 텍스트 입력 파일(예: 마크다운)을 사용하여 정적 웹 페이지를 생성합니다. 제품 페이지, 설명서, 블로그 등 거의 변경되지 않는 정보성 콘텐츠에 적합합니다. 블루프린트는 AWS CDK 를 사용하여 정적 웹 페이지를 AWS Amplify 또는 Amazon S3 + CloudFront에 배포합니다.
To Do 웹 애플리케이션	이 블루프린트는 프론트엔드 및 백엔드 구성 요소가 포함된 To Do 서버리스 웹 애플리케이션을 생성합니다. 프로그래밍 언어로 SDK for Java, TypeScript 또는 SDK for Python을 선택할 수 있습니다.
외부 블루프린트 구독	이 블루프린트는 가져온 각 패키지에 대한 워크플로를 생성합니다. 이러한 워크플로는 하루에 한 번 실행되어 NPM에서 패키지의 새 버전을 확인합니다. 새 버전이 있는 경우 워크플로는 CodeCatalyst 스페이스에 새 버전을 사용자 지정 블루프린트로 추가하려고 시도합니다. 패키지를 찾을 수 없거나 블루프린트가 아닌 경우 작업이 실패합니다. 대상 패키지는 NPM에 있어야 하며 패키지는 블루프린트이어야 합니다. 스페이스는 사용자 지정 블루프린트를 지원하는 계층에서 구독해야 합니다.
Bedrock GenAI 챗봇	이 블루프린트는 Amazon Bedrock 및 Anthropic's Claude 를 사용하여 생성형 AI 챗봇을 생성합니다. 이 블루프린트를 사용하면 데이터에 사용자 지정할 수 있는 안전하고 로그인으로 보호되는 LLM 플레이그라운드를 구축하고 배포할 수 있습니다. 자세한 내용은 Bedrock GenAI Chatbot documentation 을 참조하세요.

프로젝트 블루프린트 정보 찾기

CodeCatalyst에서 여러 프로젝트 블루프린트를 사용할 수 있습니다. 각 블루프린트에는 함께 제공되는 요약 및 README 파일이 있습니다. 이 요약에서는 블루프린트에 의해 설치된 리소스에 대해 설명하고, README 파일은 블루프린트에 대해 자세히 설명하면서 사용 방법에 대한 지침을 제공합니다.

CodeCatalyst에서 사용자 지정 블루프린트를 사용하여 프로젝트 표준화

사용자 지정 블루프린트를 사용하여 CodeCatalyst 스페이스 프로젝트의 개발 및 모범 사례를 표준화할 수 있습니다. 사용자 지정 블루프린트를 사용하여 워크플로 정의 및 애플리케이션 코드와 같은 CodeCatalyst 프로젝트의 다양한 측면을 정의할 수 있습니다. 사용자 지정 블루프린트를 사용하여 새 프로젝트를 생성하거나 기존 프로젝트에 적용한 후 해당 프로젝트에 대한 블루프린트 변경 사항을 풀 요청 업데이트로 사용할 수 있습니다. 블루프린트 작성자는 스페이스 전체에서 블루프린트를 사용하는 프로젝트에 대한 세부 정보를 볼 수 있으므로 프로젝트 간에 표준이 어떻게 적용되는지 확인할 수 있습니다. 블루프린트의 수명 주기 관리를 사용하면 모든 프로젝트의 소프트웨어 개발 수명 주기를 중앙에서 관리할 수 있으므로 스페이스 내 프로젝트가 최신 변경 사항 또는 수정 사항을 사용하여 모범 사례를 계속 따르도록 할 수 있습니다. 자세한 내용은 [블루프린트 작성자로서 수명 주기 관리 작업](#) 섹션을 참조하세요.

사용자 지정 블루프린트는 재합성을 통해 이전 프로젝트에 대해 블루프린트 버전을 업데이트할 수 있는 기능을 제공합니다. 재합성은 업데이트된 버전으로 블루프린트 합성을 재실행하거나 수정 및 변경 사항을 기존 프로젝트에 통합하는 프로세스입니다. 자세한 내용은 [사용자 지정 블루프린트 개념](#) 섹션을 참조하세요.

이미 표준화 및 모범 사례가 마련되어 있을 수 있습니다. 사용자 지정 블루프린트를 처음부터 생성하고 개발하는 대신 소스 코드가 있는 기존 소스 리포지토리를 사용자 지정 블루프린트로 변환하도록 선택할 수 있습니다. 자세한 내용은 [소스 리포지토리를 사용자 지정 블루프린트로 변환](#) 섹션을 참조하세요.

블루프린트 SDK 및 샘플 블루프린트를 보려면 [오픈 소스 GitHub 리포지토리](#)를 참조하세요.

주제

- [사용자 지정 블루프린트 개념](#)
- [사용자 지정 블루프린트 시작하기](#)
- [자습서: React 애플리케이션 생성 및 업데이트](#)
- [소스 리포지토리를 사용자 지정 블루프린트로 변환](#)

- [블루프린트 작성자로서 수명 주기 관리 작업](#)
- [프로젝트 요구 사항을 충족하기 위한 사용자 지정 블루프린트 개발](#)
- [사용자 지정 블루프린트를 스페이스에 게시](#)
- [사용자 지정 블루프린트에 대한 게시 권한 설정](#)
- [스페이스 블루프린트 카탈로그에 사용자 지정 블루프린트 추가](#)
- [사용자 지정 블루프린트의 카탈로그 버전 변경](#)
- [사용자 지정 블루프린트의 세부 정보, 버전 및 프로젝트 보기](#)
- [스페이스 블루프린트 카탈로그에서 사용자 지정 블루프린트 제거](#)
- [게시된 사용자 지정 블루프린트 또는 버전 삭제](#)
- [종속성, 불일치 및 도구 처리](#)
- [기여](#)

사용자 지정 블루프린트 개념

다음은 CodeCatalyst 에서 사용자 지정 블루프린트를 사용할 때 알아야 할 몇 가지 개념과 용어입니다.

주제

- [블루프린트 프로젝트](#)
- [스페이스 블루프린트](#)
- [스페이스스 블루프린트 카탈로그](#)
- [합성](#)
- [재합성](#)
- [부분 옵션](#)
- [Projen](#)

블루프린트 프로젝트

블루프린트 프로젝트를 통해 스페이스에 블루프린트를 개발하고 게시할 수 있습니다. 소스 리포지토리는 프로젝트 생성 프로세스 중에 생성되며, 리포지토리 이름은 프로젝트 리소스 세부 정보를 입력할 때 선택한 리포지토리의 이름입니다. 블루프린트 생성 프로세스 중에 워크플로 릴리스를 생성하도록 선택하면 블루프린트 빌더 블루프린트를 사용하여 블루프린트에 게시 워크플로가 생성됩니다. 워크플로는 최신 버전을 자동으로 게시합니다.

스페이스 블루프린트

스페이스의 블루프린트 섹션으로 이동할 때 스페이스 블루프린트 테이블에서 모든 블루프린트를 보고 관리할 수 있습니다. 블루프린트가 스페이스에 게시되면 스페이스 블루프린트 카탈로그에서 추가 및 제거할 수 있는 스페이스 블루프린트로 사용할 수 있습니다. 스페이스의 블루프린트 섹션에서 게시 권한을 관리하고 에서 블루프린트를 삭제할 수도 있습니다. 자세한 내용은 [사용자 지정 블루프린트의 세부 정보, 버전 및 프로젝트 보기](#) 섹션을 참조하세요.

스페이스스 블루프린트 카탈로그

스페이스의 블루프린트 카탈로그에서 추가된 모든 사용자 지정 블루프린트를 볼 수 있습니다. 여기에서 스페이스 멤버는 사용자 지정 블루프린트를 선택하여 새 프로젝트를 생성할 수 있습니다. 이 카탈로그는 모든 스페이스 멤버에 대해 이미 사용 가능한 블루프린트가 있는 CodeCatalyst 카탈로그와 다릅니다. 자세한 내용은 [CodeCatalyst 블루프린트를 사용하여 포괄적인 프로젝트 생성](#) 섹션을 참조하세요.

합성

합성은 프로젝트의 소스 코드, 구성 및 리소스를 나타내는 CodeCatalyst 프로젝트 번들을 생성하는 프로세스입니다. 그런 다음 CodeCatalyst 배포 API 작업에서 번들을 사용하여 프로젝트에 배포합니다. CodeCatalyst 에서 프로젝트를 생성할 필요 없이 프로젝트 생성을 에뮬레이션하기 위해 사용자 지정 블루프린트를 개발하는 동안 프로세스를 로컬에서 실행할 수 있습니다. 다음 명령을 사용하여 합성을 수행할 수 있습니다.

```
yarn blueprint:synth          # fast mode
yarn blueprint:synth --cache  # wizard emulation mode
```

블루프린트는 defaults.json에 병합된 옵션과 함께 기본 blueprint.ts 클래스를 직접적으로 호출하는 것으로 시작됩니다. synth/synth.**[options-name]**/proposed-bundle/ 폴더 아래 새 프로젝트 번들이 생성됩니다. 출력에는 구성했을 수 있는 [부분 옵션](#)을 포함하여 사용자가 설정한 옵션을 고려하여 사용자 지정 블루프린트가 생성하는 프로젝트 번들이 포함됩니다.

재합성

재합성은 기존 프로젝트의 다양한 블루프린트 옵션 또는 블루프린트 버전으로 블루프린트를 재생성하는 프로세스입니다. 블루프린트 작성자는 사용자 지정 블루프린트 코드에서 사용자 지정 병합 전략을 정의할 수 있습니다. .ownership-file에서 소유권 경계를 정의하여 블루프린트를 업데이트할 수 있는 코드베이스 부분을 지정할 수도 있습니다. 사용자 지정 블루프린트는 에 대한 업데이트를 제안할

수 있지만 .ownership-file 사용자 지정 블루프린트를 사용하는 프로젝트 개발자는 프로젝트의 소유권 경계를 결정할 수 있습니다. 사용자 지정 블루프린트를 게시하기 전에 로컬에서 재합성을 실행하고 테스트 및 업데이트할 수 있습니다. 다음 명령을 사용하여 재합성을 수행합니다.

```
yarn blueprint:resynth          # fast mode
yarn blueprint:resynth --cache  # wizard emulation mode
```

블루프린트는 defaults.json에 병합된 옵션과 함께 기본 blueprint.ts 클래스를 직접적으로 호출하는 것으로 시작됩니다. synth/resynth.[\[options-name\]](#)/ 폴더 아래에 새 프로젝트 번들이 생성됩니다. 출력에는 구성했을 수 있는 [부분 옵션](#)을 포함하여 사용자가 설정한 옵션을 고려하여 사용자 지정 블루프린트가 생성하는 프로젝트 번들이 포함됩니다.

합성 및 재합성 프로세스 후에 다음 내용이 생성됩니다.

- proposed-bundle - 대상 블루프린트 버전에 대한 새 옵션으로 실행될 때의 합성 출력입니다.
- existing-bundle - 기존 프로젝트의 모형입니다. 이 폴더에 아무것도 없으면 proposed-bundle과 동일한 출력으로 생성됩니다.
- ancestor-bundle - 이전 버전, 이전 옵션 또는 조합으로 실행할 때 블루프린트가 생성하는 모의입니다. 이 폴더에 아무것도 없으면 proposed-bundle과 동일한 출력으로 생성됩니다.
- resolved-bundle - 번들은 항상 재생성되며 기본적으로 proposed-bundle, existing-bundle 및 ancestor-bundle 간의 3방향 병합으로 설정됩니다. 이 번들은 재합성이 로컬에서 출력할 에뮬레이션을 제공합니다.

블루프린트 출력 번들에 대한 자세한 내용은 [재합성을 사용하여 파일 생성](#) 섹션을 참조하세요.

부분 옵션

Options 인터페이스 전체를 열거할 필요가 없는 src/wizard-configuration/에서 옵션 변형을 추가할 수 있으며, 옵션은 defaults.json 파일 위에 병합됩니다. 이를 통해 특정 옵션에서 테스트 사례를 조정할 수 있습니다.

예시:

Options 인터페이스:

```
{
  language: "Python" | "Java" | "Typescript",
```

```
repositoryName: string
...
}
```

defaults.json 파일:

```
{
  language: "Python",
  repositoryName: "Myrepo"
  ...
}
```

추가 구성 테스트:

- #wizard-config-typescript-test.json

```
{
  language: "Typescript",
}
```
- #wizard-config-java-test.json

```
{
  language: "Java",
}
```

Projen

Projen은 사용자 지정 블루프린트가 자체 업데이트 및 일관성을 유지하는 데 사용하는 오픈 소스 도구입니다. 이 프레임워크는 프로젝트를 빌드, 번들링 및 게시할 수 있는 기능을 제공하며 인터페이스를 사용하여 프로젝트의 구성 및 설정을 관리할 수 있기 때문에 블루프린트는 Projen 패키지로 제공됩니다.

Projen을 사용하여 블루프린트가 생성된 후에도 블루프린트를 대규모로 업데이트할 수 있습니다. Projen 도구는 프로젝트 번들을 생성하는 블루프린트 합성의 기본 기술입니다. Projen은 프로젝트의 구성을 소유하며, 블루프린트 작성자로서 사용자에게 영향을 주지 않습니다. 종속성을 추가한 후 `yarn projen`를 실행하여 프로젝트 구성을 재생성하거나 `projenrc.ts` 파일에서 옵션을 변경할 수 있습니다. Projen은 프로젝트를 합성하기 위한 사용자 지정 블루프린트의 기본 생성 도구이기도 합니다. 자세한 정보는 [Projen GitHub 페이지](#)를 참조하세요. Projen에 대해 자세히 알아보려면 [Projen 설명서](#)를 참조하세요.

사용자 지정 블루프린트 시작하기

블루프린트를 생성하는 과정에서 블루프린트를 구성하고 프로젝트 리소스의 미리 보기를 생성할 수 있습니다. 각 사용자 지정 블루프린트는 스페이스의 블루프린트 카탈로그에 게시하기 위한 워크플로가 기본적으로 포함된 CodeCatalyst 프로젝트에서 관리합니다.

사용자 지정 블루프린트의 세부 정보를 구성하는 동안 블루프린트의 소스 코드를 타사 리포지토리에 저장하도록 선택할 수도 있습니다. 이 리포지토리에서는 사용자 지정 블루프린트를 여전히 관리하고 수명 주기 관리 기능을 활용하여 사용자 지정 블루프린트가 수정될 때 스페이스 프로젝트를 동기화된 상태로 유지할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 확장 프로그램이 있는 프로젝트에 기능 추가](#) 및 [블루프린트 작성자로서 수명 주기 관리 작업](#) 섹션을 참조하세요.

표준화 및 모범 사례를 갖춘 소스 리포지토리가 이미 있는 경우 해당 소스 리포지토리를 사용자 지정 블루프린트로 변환하도록 선택할 수 있습니다. 자세한 내용은 [소스 리포지토리를 사용자 지정 블루프린트로 변환](#) 섹션을 참조하세요.

주제

- [사전 조건](#)
- [1단계: CodeCatalyst에서 사용자 지정 블루프린트 생성](#)
- [2단계: 구성 요소를 사용하여 사용자 지정 블루프린트 개발](#)
- [3단계: 사용자 지정 블루프린트 미리 보기](#)
- [\(선택 사항\) 4단계: 사용자 지정 블루프린트 미리 보기 버전 게시](#)

사전 조건

사용자 지정 블루프린트를 생성하기 전에 다음 요구 사항을 고려하세요.

- CodeCatalyst 스페이스는 엔터프라이즈 계층이어야 합니다. 자세한 내용은 Amazon CodeCatalyst 관리자 안내서의 [Managing billing](#)을 참조하세요.
- 사용자 지정 블루프린트를 생성하려면 스페이스 관리자 또는 파워 유저 역할이 있어야 합니다. 자세한 내용은 [사용자 역할로 액세스 권한 부여](#) 섹션을 참조하세요.

1단계: CodeCatalyst에서 사용자 지정 블루프린트 생성

스페이스 설정에서 사용자 지정 블루프린트를 생성하면 리포지토리가 생성됩니다. 리포지토리에는 블루프린트를 스페이스의 블루프린트 카탈로그에 게시하기 전에 개발해야 하는 모든 필수 리소스가 포함되어 있습니다.

사용자 지정 블루프린트를 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 사용자 지정 블루프린트를 생성할 스페이스로 이동합니다.
3. 스페이스 대시보드에서 설정 탭을 선택한 다음 블루프린트를 선택합니다.
4. 블루프린트 생성을 선택합니다.
5. 블루프린트 이름에 프로젝트에 할당하려는 이름과 관련 리소스 이름을 입력합니다. 이름은 스페이스 내에서 고유해야 합니다.
6. (선택 사항) 기본적으로 블루프린트에서 생성된 소스 코드는 CodeCatalyst 리포지토리에 저장됩니다. 또는 블루프린트의 소스 코드를 타사 리포지토리에 저장하도록 선택할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 확장 프로그램이 있는 프로젝트에 기능 추가](#) 섹션을 참조하세요.

사용하려는 타사 리포지토리 공급자에 따라 다음 중 하나를 수행합니다.

- GitHub 리포지토리: GitHub 계정을 연결합니다.

고급 드롭다운 메뉴를 선택하고 GitHub를 리포지토리 공급자로 선택한 다음 블루프린트에서 생성된 소스 코드를 저장할 GitHub 계정을 선택합니다.

Note

GitHub 계정에 대한 연결을 사용하는 경우 개인 연결을 생성하여 CodeCatalyst ID와 GitHub ID 간에 ID 매핑을 설정해야 합니다. 자세한 내용은 [개인 연결](#) 및 [개인 연결을 사용하여 GitHub 리소스에 액세스](#) 섹션을 참조하세요.

- Bitbucket 리포지토리: Bitbucket 작업 영역을 연결합니다.

고급 드롭다운 메뉴를 선택하고 Bitbucket을 리포지토리 공급자로 선택한 다음, 블루프린트에서 생성된 소스 코드를 저장할 Bitbucket 작업 영역을 선택합니다.

- GitLab 리포지토리: GitLab 사용자를 연결합니다.

고급 드롭다운 메뉴를 선택하고 GitLab을 리포지토리 공급자로 선택한 다음 블루프린트에서 생성된 소스 코드를 저장할 GitLab 사용자를 선택합니다.

7. 블루프린트 세부 정보에서 다음을 수행합니다.
 - a. 블루프린트 표시 이름 텍스트 입력 필드에 스페이스의 블루프린트 카탈로그에 표시될 이름을 입력합니다.
 - b. 설명 텍스트 입력 필드에 사용자 지정 블루프린트에 대한 설명을 입력합니다.

- c. 작성자 이름 텍스트 입력 필드에 사용자 지정 블루프린트의 작성자 이름을 입력합니다.
- d. (선택 사항) 고급 설정을 선택합니다.
 - i. + 추가를 선택하여 `package.json` 파일에 추가되는 태그를 추가합니다.
 - ii. 라이선스 드롭다운 메뉴를 선택한 다음 사용자 지정 블루프린트에 대한 라이선스를 선택합니다.
 - iii. 블루프린트 패키지 이름 텍스트 입력 필드에 블루프린트 패키지를 식별할 이름을 입력합니다.
 - iv. 기본적으로 릴리스 워크플로는 Blueprint Builder라는 프로젝트 내의 게시 블루프린트를 사용하여 생성됩니다. 릴리스 워크플로에서 게시 권한이 활성화되어 있으므로 변경 사항을 푸시하면 워크플로가 최신 블루프린트 버전을 스페이스에 게시합니다. 워크플로 생성을 끄려면 워크플로 릴리스 확인란을 선택 취소합니다.
- 8. (선택 사항) 블루프린트 프로젝트에는 스페이스의 블루프린트 카탈로그에 블루프린트 게시를 지원하는 사전 정의된 코드가 함께 제공됩니다. 선택한 프로젝트 파라미터에 따라 업데이트가 포함된 정의 파일을 보려면 블루프린트 미리 보기 생성에서 코드 보기 또는 워크플로 보기를 선택합니다.
- 9. 블루프린트 생성을 선택합니다.

사용자 지정 블루프린트에 대한 워크플로 생성을 끄지 않은 경우 블루프린트가 생성될 때 워크플로가 자동으로 실행되기 시작합니다. 워크플로 실행이 완료되면 사용자 지정 블루프린트를 기본적으로 스페이스의 블루프린트 카탈로그에 추가할 수 있습니다. 최신 블루프린트 버전이 스페이스에 자동으로 게시되지 않도록 하려면 게시 권한을 끌 수 있습니다. 자세한 내용은 [사용자 지정 블루프린트에 대한 게시 권한 설정](#) 및 [워크플로 실행](#) 섹션을 참조하세요.

`blueprint-release` 게시 워크플로는 블루프린트를 사용하여 생성되므로 블루프린트는 프로젝트에서 적용된 블루프린트로 찾을 수 있습니다. 자세한 내용은 [프로젝트에 블루프린트를 추가하여 리소스 통합](#) 및 [업데이트 중지를 위해 프로젝트에서 블루프린트 연결 해제](#) 섹션을 참조하세요.

2단계: 구성 요소를 사용하여 사용자 지정 블루프린트 개발

사용자 지정 블루프린트를 생성할 때 블루프린트 마법사가 생성되며, 사용자 지정 블루프린트를 개발할 때 구성 요소로 수정할 수 있습니다. `src/blueprints.js` 및 `src/defaults.json` 파일을 업데이트하여 마법사를 수정할 수 있습니다.

⚠ Important

외부 소스의 블루프린트 패키지를 사용하려면, 해당 패키지로 인해 발생할 수 있는 위험을 고려하세요. 스페이스에 추가하는 사용자 지정 블루프린트와 해당 블루프린트가 생성하는 코드는 사용자의 책임입니다.

블루프린트 코드를 구성하기 전에 지원되는 통합 개발 환경(IDE)을 사용하여 CodeCatalyst 프로젝트에서 개발 환경을 생성합니다. 필요한 도구 및 패키지를 사용하려면 개발 환경이 필요합니다.

개발 환경을 생성하려면

1. 탐색 창에서 다음 중 하나를 수행합니다.
 - a. 개요를 선택한 다음 내 개발 환경 섹션으로 이동합니다.
 - b. 코드를 선택한 후 개발 환경을 선택합니다.
 - c. 코드를 선택하고, 소스 리포지토리를 선택한 다음, 블루프린트를 생성할 때 생성한 리포지토리를 선택합니다.
2. 개발 환경 생성을 선택합니다.
3. 드롭다운 메뉴에서 지원되는 IDE를 선택합니다. 자세한 내용은 [개발 환경에서 지원되는 통합 개발 환경을 참조하세요](#).
4. 기존 브랜치에서 작업을 선택하고 기존 브랜치 드롭다운 메뉴에서 생성한 기능 브랜치를 선택합니다.
5. (선택 사항) 별칭 - 선택적 텍스트 입력 필드에 별칭을 입력하여 개발 환경을 식별합니다.
6. 생성(Create)을 선택합니다. 개발 환경이 생성되는 동안 개발 환경 상태 옆에 시작 중이 표시되고, 개발 환경이 생성되면 상태 옆에 실행 중이 표시됩니다.

자세한 내용은 [CodeCatalyst에서 개발 환경으로 코드 작성 및 수정](#) 섹션을 참조하세요.

사용자 지정 블루프린트를 개발하려면

1. 작동 중인 터미널에서 다음 yarn 명령을 사용하여 종속성을 설치합니다.

```
yarn
```

필요한 도구와 패키지는 Yarn을 포함한 CodeCatalyst 개발 환경을 통해 제공됩니다. 개발 환경 없이 사용자 지정 블루프린트를 작업하는 경우 먼저 시스템에 Yarn을 설치합니다. 자세한 내용은 [Yarn 설치 설명서](#)를 참조하세요.

2. 사용자 지정 블루프린트를 개발하여 기본 설정에 맞게 구성합니다. 구성 요소를 추가하여 블루프린트의 마법사를 수정할 수 있습니다. 자세한 내용은 [프로젝트 요구 사항을 충족하기 위한 사용자 지정 블루프린트 개발](#), [프론트엔드 마법사를 사용하여 블루프린트 특성 수정](#), [사용자 지정 블루프린트를 스페이스에 게시](#) 섹션을 참조하세요.

3단계: 사용자 지정 블루프린트 미리 보기

사용자 지정 블루프린트를 설정하고 개발한 후 블루프린트의 미리 보기 버전을 미리 보고 스페이스에 게시할 수 있습니다. 미리 보기 버전을 사용하면 블루프린트가 새 프로젝트를 생성하거나 기존 프로젝트에 적용되기 전에 원하는 것인지 확인할 수 있습니다.

사용자 지정 블루프린트를 미리 보려면

1. 작동 중인 터미널에서 다음 yarn 명령을 사용합니다.

```
yarn blueprint:preview
```

2. 제공된 See this blueprint at: 링크로 이동하여 사용자 지정 블루프린트를 미리 봅니다.
3. 텍스트를 포함한 UI가 구성에 따라 예상대로 표시되는지 확인합니다. 사용자 지정 블루프린트를 변경하려면 blueprint.ts 파일을 편집하고 블루프린트를 재합성한 다음 미리 보기 버전을 다시 게시할 수 있습니다. 자세한 내용은 [재합성](#) 섹션을 참조하세요.

(선택 사항) 4단계: 사용자 지정 블루프린트 미리 보기 버전 게시

사용자 지정 블루프린트의 미리 보기 버전을 스페이스의 블루프린트 카탈로그에 추가하려는 경우 스페이스에 게시할 수 있습니다. 이렇게 하면 미리 보기가 아닌 버전을 카탈로그에 추가하기 전에 사용자로서 블루프린트를 볼 수 있습니다. 미리 보기 버전을 사용하면 실제 버전을 사용하지 않고 게시할 수 있습니다. 예를 들어, 0.0.1 버전에서 작업하는 경우 미리 보기 버전을 게시하고 추가할 수 있으므로 두 번째 버전에 대한 새 업데이트를 게시하고 0.0.2로 추가할 수 있습니다.

사용자 지정 블루프린트의 미리 보기 버전을 게시하려면

제공된 Enable version *[version number]* at: 연결로 이동하여 사용자 지정 블루프린트를 활성화합니다. 이 연결은 [3단계: 사용자 지정 블루프린트 미리 보기](#)에서 yarn 명령을 실행할 때 제공됩니다.

사용자 지정 블루프린트를 생성, 개발, 미리 보기 및 게시한 후 최종 블루프린트 버전을 게시하고 스페이스의 블루프린트 카탈로그에 추가할 수 있습니다. 자세한 내용은 [사용자 지정 블루프린트를 스페이스에 게시](#) 및 [스페이스 블루프린트 카탈로그에 사용자 지정 블루프린트 추가](#) 섹션을 참조하세요.

자습서: React 애플리케이션 생성 및 업데이트

블루프린트 작성자는 사용자 지정 블루프린트를 개발하고 스페이스의 블루프린트 카탈로그에 추가할 수 있습니다. 그런 다음 스페이스 멤버가 이러한 블루프린트를 사용하여 새 프로젝트를 생성하거나 기존 프로젝트에 추가할 수 있습니다. 블루프린트를 계속 변경한 다음 풀 요청을 통해 업데이트로 사용할 수 있습니다.

이 자습서에서는 블루프린트 작성자의 관점과 블루프린트 사용자 관점에 대한 연습을 제공합니다. 이 자습서에서는 React 단일 페이지 웹 애플리케이션 블루프린트를 생성하는 방법을 보여줍니다. 그런 다음 블루프린트를 사용하여 새 프로젝트를 생성합니다. 블루프린트가 변경 사항으로 업데이트되면 블루프린트에서 생성된 프로젝트는 풀 요청을 통해 이러한 변경 사항을 통합합니다.

주제

- [사전 조건](#)
- [1단계: 사용자 지정 블루프린트 생성](#)
- [2단계: 릴리스 워크플로 보기](#)
- [3단계: 카탈로그에 블루프린트 추가](#)
- [4단계: 블루프린트를 사용하여 프로젝트 생성](#)
- [5단계: 블루프린트 업데이트](#)
- [6단계: 블루프린트의 게시된 카탈로그 버전을 새 버전으로 업데이트](#)
- [7단계: 새 블루프린트 버전으로 프로젝트 업데이트](#)
- [8단계: 프로젝트의 변경 사항 보기](#)

사전 조건

사용자 지정 블루프린트를 생성하고 업데이트하려면 다음과 같이 [CodeCatalyst 설정 및 로그인](#)의 작업을 완료해야 합니다.

- CodeCatalyst에 로그인하기 위한 AWS Builder ID가 있어야 합니다.
- 스페이스에 속해 해당 스페이스에서 스페이스 관리자 또는 파워 유저 역할을 할당받습니다. 자세한 내용은 [스페이스 생성](#), [사용자에게 스페이스 권한 부여](#), [스페이스 관리자 역할](#) 섹션을 참조하세요.

1단계: 사용자 지정 블루프린트 생성

사용자 지정 블루프린트를 생성하면 블루프린트 소스 코드와 개발 도구 및 리소스가 포함된 CodeCatalyst 프로젝트가 생성됩니다. 프로젝트는 블루프린트를 개발, 테스트 및 게시하는 곳입니다.

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 블루프린트를 생성할 스페이스로 이동합니다.
3. 설정을 선택하여 스페이스 설정으로 이동합니다.
4. 스페이스 설정 탭에서 블루프린트를 선택하고 블루프린트 생성을 선택합니다.
5. 블루프린트 생성 마법사의 필드를 다음 값으로 업데이트합니다.
 - 블루프린트 이름에 `react-app-blueprint`를 입력합니다.
 - 블루프린트 표시 이름에 `react-app-blueprint`를 입력합니다.
6. 필요에 따라 코드 보기를 선택하여 블루프린트의 블루프린트 소스 코드를 미리 봅니다. 마찬가지로 워크플로 보기를 선택하여 블루프린트를 빌드하고 게시하는 프로젝트에서 생성될 워크플로를 미리 봅니다.
7. 블루프린트 생성을 선택합니다.
8. 블루프린트가 생성되면 블루프린트의 프로젝트로 이동합니다. 이 프로젝트에는 블루프린트 소스 코드와 블루프린트를 개발, 테스트 및 게시하는 데 필요한 도구 및 리소스가 포함되어 있습니다. 릴리스 워크플로가 생성되어 블루프린트가 스페이스에 자동으로 게시되었습니다.
9. 이제 블루프린트 및 블루프린트 프로젝트가 생성되었으므로 다음 단계에서는 소스 코드를 업데이트하여 구성합니다. 개발 환경을 사용하여 브라우저에서 직접 소스 리포지토리를 열고 편집할 수 있습니다.

탐색 창에서 코드를 선택한 다음 개발 환경을 선택합니다.

10. 개발 환경 생성을 선택한 다음 AWS Cloud9(브라우저에서)를 선택합니다.
11. 기본 설정을 유지하고 생성을 선택합니다.
12. AWS Cloud9 터미널에서 다음 명령을 실행하여 블루프린트 프로젝트 디렉터리로 이동합니다.

```
cd react-app-blueprint
```

13. 블루프린트가 생성되면 `static-assets` 폴더가 자동으로 생성되고 채워집니다. 이 자습서에서는 기본 폴더를 삭제하고 react 앱 블루프린트에 대한 새 폴더를 생성합니다.

다음 명령을 실행하여 `static-assets` 폴더를 삭제합니다.

```
rm -r static-assets
```

AWS Cloud9는 Linux 기반 플랫폼을 기반으로 구축되었습니다. Windows 운영 체제를 사용하는 경우 대신 다음 명령을 사용할 수 있습니다.

```
rmdir /s /q static-assets
```

- 이제 기본 폴더가 삭제되었으므로 다음 명령을 실행하여 react-app 블루프린트용 static-assets 폴더를 생성합니다.

```
npx create-react-app static-assets
```

메시지가 표시되면 y를 입력하여 계속 진행합니다.

필요한 패키지가 있는 static-assets 폴더에 새 React 애플리케이션이 생성되었습니다. 변경 사항을 원격 CodeCatalyst 소스 리포지토리로 푸시해야 합니다.

- 최신 변경 사항이 있는지 확인한 뒤 다음 명령을 실행하여 변경 사항을 커밋하고 블루프린트의 CodeCatalyst 소스 리포지토리에 푸시합니다.

```
git pull
```

```
git add .
```

```
git commit -m "Add React app to static-assets"
```

```
git push
```

변경 사항이 블루프린트의 소스 리포지토리로 푸시되면 릴리스 워크플로가 자동으로 시작됩니다. 이 워크플로는 블루프린트 버전을 늘리고 블루프린트를 빌드한 다음 스페이스에 게시합니다. 다음 단계에서는 릴리스 워크플로 실행으로 이동하여 실행 상태를 확인합니다.

2단계: 릴리스 워크플로 보기

- CodeCatalyst 콘솔의 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.

2. blueprint-release 워크플로를 선택합니다.
3. 워크플로에 블루프린트를 빌드하고 게시하는 작업이 있음을 확인할 수 있습니다.
4. 최신 실행에서 워크플로 실행 링크를 선택하여 수행한 코드 변경에서 실행을 확인합니다.
5. 실행이 완료되면 새 블루프린트 버전이 게시됩니다. 게시된 블루프린트 버전은 스페이스 설정에서 볼 수 있지만 스페이스의 블루프린트 카탈로그에 추가될 때까지 프로젝트에서 사용할 수 없습니다. 다음 단계에서는 카탈로그에 블루프린트를 추가합니다.

3단계: 카탈로그에 블루프린트 추가

스페이스의 블루프린트 카탈로그에 블루프린트를 추가하면 스페이스의 모든 프로젝트에서 블루프린트를 사용할 수 있습니다. 스페이스 멤버는 블루프린트를 사용하여 새 프로젝트를 생성하거나 기존 프로젝트에 추가할 수 있습니다.

1. CodeCatalyst 콘솔에서 스페이스로 돌아갑니다.
2. 설정을 선택한 다음 블루프린트를 선택합니다.
3. react-app-blueprint를 선택한 다음 카탈로그에 추가를 선택합니다.
4. 저장을 선택합니다.

4단계: 블루프린트를 사용하여 프로젝트 생성

이제 블루프린트가 카탈로그에 추가되었으므로 프로젝트에 사용할 수 있습니다. 이 단계에서는 방금 생성한 블루프린트를 사용하여 프로젝트를 생성합니다. 이후 단계에서는 블루프린트의 새 버전을 업데이트하고 게시하여 이 프로젝트를 업데이트합니다.

1. 프로젝트 탭을 선택한 다음 프로젝트 생성을 선택합니다.
2. 스페이스 블루프린트를 선택한 다음 react-app-blueprint를 선택합니다.

Note

블루프린트를 선택하면 블루프린트 파일의 내용을 볼 수 있습니다README.md.

3. 다음을 선택합니다.

4.  Note
이 프로젝트 생성 마법사의 내용은 블루프린트에서 구성할 수 있습니다.

프로젝트 이름을 블루프린트 사용자로 입력합니다. 이 자습서에서는 `react-app-project`을 입력합니다. 자세한 내용은 [프로젝트 요구 사항을 충족하기 위한 사용자 지정 블루프린트 개발](#) 섹션을 참조하세요.

다음으로 블루프린트를 업데이트하고 이 프로젝트를 업데이트하는 데 사용할 새 버전을 카탈로그에 추가합니다.

5단계: 블루프린트 업데이트

블루프린트를 사용하여 새 프로젝트를 생성하거나 기존 프로젝트에 적용한 후 블루프린트 작성자로서 계속 업데이트할 수 있습니다. 이 단계에서는 블루프린트를 변경하고 새 버전을 스페이스에 자동으로 게시합니다. 그러면 새 버전을 카탈로그 버전으로 추가할 수 있습니다.

1. [자습서: React 애플리케이션 생성 및 업데이트](#)에서 생성된 `react-app-blueprint` 프로젝트로 이동합니다.
2. [자습서: React 애플리케이션 생성 및 업데이트](#)에서 생성된 개발 환경을 엽니다.
 - a. 탐색 창에서 코드를 선택한 다음 개발 환경을 선택합니다.
 - b. 테이블에서 개발 환경을 찾은 다음 AWS Cloud9에서(브라우저에서) 열기를 선택합니다.
3. 블루프린트 릴리스 워크플로가 실행되면 `package.json` 파일 업데이트를 통해 블루프린트 버전이 증가합니다. AWS Cloud9 터미널에서 다음 명령을 실행하여 에서 변경 사항을 가져옵니다.

```
git pull
```

4. `static-assets` 폴더로 이동하여 다음 명령을 실행합니다.

```
cd /projects/react-app-blueprint/static-assets
```

5. 다음 명령을 실행하여 `static-assets` 폴더에 `hello-world.txt` 파일을 생성합니다.

```
touch hello-world.txt
```

AWS Cloud9는 Linux 기반 플랫폼을 기반으로 구축되었습니다. Windows 운영 체제를 사용하는 경우 대신 다음 명령을 사용할 수 있습니다.

```
echo > hello-world.txt
```


6. 왼쪽 탐색에서 `hello-world.txt` 파일을 두 번 클릭하여 편집기에서 파일을 열고 다음 내용을 추가합니다.

```
Hello, world!
```

파일을 저장합니다.

7. 최신 변경 사항이 있는지 확인한 뒤 다음 명령을 실행하여 변경 사항을 커밋하고 블루프린트의 CodeCatalyst 소스 리포지토리에 푸시합니다.

```
git pull
```

```
git add .
```

```
git commit -m "prettier setup"
```

```
git push
```

변경 사항을 푸시하여 릴리스 워크플로가 시작되었으며, 이 워크플로는 새 버전의 블루프린트를 스페이스에 자동으로 게시합니다.

6단계: 블루프린트의 게시된 카탈로그 버전을 새 버전으로 업데이트

블루프린트를 사용하여 새 프로젝트를 생성하거나 기존 프로젝트에 적용한 후에도 블루프린트를 블루프린트 작성자로 업데이트할 수 있습니다. 이 단계에서는 블루프린트를 변경하고 블루프린트의 카탈로그 버전을 변경합니다.

1. CodeCatalyst 콘솔에서 스페이스로 돌아갑니다.
2. 설정을 선택한 다음 블루프린트를 선택합니다.
3. `react-app-blueprint`를 선택한 다음 카탈로그 버전 관리를 선택합니다.
4. 새 버전을 선택하고 저장을 선택합니다.

7단계: 새 블루프린트 버전으로 프로젝트 업데이트

이제 스페이스의 블루프린트 카탈로그에서 새 버전을 사용할 수 있습니다. 블루프린트 사용자는 [4단계: 블루프린트를 사용하여 프로젝트 생성](#)에서 생성된 프로젝트의 버전을 업데이트할 수 있습니다. 이렇게 하면 모범 사례를 충족하는 데 필요한 최신 변경 사항과 수정 사항이 적용됩니다.

1. CodeCatalyst 콘솔에서 [4단계: 블루프린트를 사용하여 프로젝트 생성](#)에서 생성된 react-app-project 프로젝트로 이동합니다.
2. 탐색 창에서 [블루프린트(Blueprints)]를 선택합니다.
3. 정보 상자에서 블루프린트 업데이트를 선택합니다.
4. 오른쪽 코드 변경 패널에서 hello-world.txt 및 package.json 업데이트를 볼 수 있습니다.
5. 업데이트 적용을 선택합니다.

업데이트 적용을 선택하면 업데이트된 블루프린트 버전의 변경 사항이 포함된 풀 요청이 프로젝트에 생성됩니다. 프로젝트를 업데이트하려면 풀 요청을 병합해야 합니다. 자세한 내용은 [풀 요청 검토](#) 및 [풀 요청 병합](#)(을)를 참조하세요.

1. 블루프린트 테이블에서 블루프린트를 찾습니다. 상태 열에서 보류 중인 풀 요청을 선택한 다음 열린 풀 요청에 대한 링크를 선택합니다.
2. 풀 요청을 검토한 다음 병합을 선택합니다.
3. 빠른 병합을 선택하여 기본값을 유지한 다음 병합을 선택합니다.

8단계: 프로젝트의 변경 사항 보기

이제 [7단계: 새 블루프린트 버전으로 프로젝트 업데이트](#) 이후 프로젝트에서 블루프린트에 대한 변경 사항을 사용할 수 있습니다. 블루프린트 사용자는 소스 리포지토리에서 변경 사항을 볼 수 있습니다.

1. 탐색 창에서 소스 리포지토리를 선택한 다음 프로젝트가 생성될 때 생성된 소스 리포지토리의 이름을 선택합니다.
2. 파일에서 [5단계: 블루프린트 업데이트](#)에서 생성된 hello-world.txt 파일을 볼 수 있습니다.
3. 파일 콘텐츠를 볼 hello-world.txt를 선택합니다.

수명 주기 관리는 블루프린트 작성자가 특정 블루프린트가 포함된 모든 프로젝트의 소프트웨어 개발 수명 주기를 중앙에서 관리할 수 있는 기능을 제공합니다. 이 자습서에서 볼 수 있듯이 블루프린트에

업데이트를 푸시한 다음 블루프린트를 사용하여 새 프로젝트를 생성하거나 기존 프로젝트에 적용할 수 있습니다. 자세한 내용은 [블루프린트 작성자로서 수명 주기 관리 작업](#) 섹션을 참조하세요.

소스 리포지토리를 사용자 지정 블루프린트로 변환

사용자 지정 블루프린트를 사용하면 CodeCatalyst 스페이스의 여러 프로젝트에서 모범 사례 또는 새 옵션을 통합할 수 있습니다. 새로운 사용자 지정 블루프린트를 처음부터 생성하고 개발할 수 있지만 코드와 확립된 모범 사례를 사용하여 기존 CodeCatalyst 또는 타사 소스 리포지토리를 블루프린트 프로젝트로 변환할 수도 있습니다. 기존 리포지토리에서 블루프린트 프로젝트로 관련 아티팩트를 복사하지 않아도 됩니다. 소스 리포지토리를 사용자 지정 블루프린트로 변환한 후 다른 사용자 지정 블루프린트와 마찬가지로 블루프린트를 업데이트, 게시 및 추가할 수 있습니다.

소스 리포지토리를 사용자 지정 블루프린트로 변환한 후 소스 리포지토리가 블루프린트 프로젝트로 재구성되고, 리포지토리의 내용이 리포지토리 내의 `static-assets` 폴더로 이동되며, 블루프린트에 필요한 관련 자산이 리포지토리에 추가됩니다. 사용자 지정 블루프린트를 사용하여 프로젝트를 생성하거나 기존 프로젝트에 추가하면 변환된 소스 리포지토리에 저장된 워크플로 정의도 블루프린트를 통해 프로젝트에 추가됩니다.

Note

소스 리포지토리를 사용자 지정 블루프린트로 변환할 때는 환경 및 보안 암호와 같은 리소스가 포함되지 않습니다. 소스 리포지토리를 사용자 지정 블루프린트로 변환한 후 이러한 리소스를 수동으로 복사하거나 추가해야 합니다.

Important

소스 리포지토리를 사용자 지정 블루프린트로 변환하려면 스페이스에 프로젝트 관리자, 스페이스 관리자 또는 파워 유저 역할이 있는 계정으로 로그인해야 합니다.

소스 리포지토리 목록에서 소스 리포지토리를 사용자 지정 블루프린트로 변환하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 스페이스로 이동한 다음 소스 리포지토리를 사용자 지정 블루프린트로 변환하려는 프로젝트를 선택합니다.
3. 탐색 창에서 코드를 선택하고 소스 리포지토리를 선택한 다음 사용자 지정 블루프린트로 변환하려는 소스 리포지토리의 라디오 버튼을 선택합니다.

4. 블루프린트로 변환을 선택하여 소스 리포지토리를 사용자 지정 블루프린트로 변환합니다.

CodeCatalyst 콘솔의 소스 리포지토리 페이지에서 CodeCatalyst 리포지토리를 사용자 지정 블루프린트로 변환할 수도 있습니다.

소스 리포지토리 페이지에서 소스 리포지토리를 사용자 지정 블루프린트로 변환하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 스페이스로 이동한 다음 소스 리포지토리를 사용자 지정 블루프린트로 변환하려는 프로젝트를 선택합니다.
3. 탐색 창에서 코드를 선택하고 소스 리포지토리를 선택한 다음 사용자 지정 블루프린트로 변환하려는 CodeCatalyst 소스 리포지토리의 이름을 선택합니다.
4. 추가 드롭다운 메뉴를 선택한 다음 블루프린트로 변환을 선택하여 소스 리포지토리를 사용자 지정 블루프린트로 변환합니다.

소스 리포지토리가 사용자 지정 블루프린트로 변환되면 릴리스 워크플로가 자동으로 실행됩니다. 실행이 성공적으로 완료되면 사용자 지정 블루프린트가 스페이스의 사용자 지정 블루프린트 목록에 게시됩니다. 여기에서 변환된 사용자 지정 블루프린트를 스페이스의 카탈로그에 추가하여 새 프로젝트를 생성하거나 기존 프로젝트에 추가할 수 있습니다. 자세한 내용은 [사용자 지정 블루프린트를 스페이스에 게시](#) 및 [스페이스 블루프린트 카탈로그에 사용자 지정 블루프린트 추가](#) 섹션을 참조하세요.

Important

스페이스의 블루프린트 카탈로그에 사용자 지정 블루프린트를 추가하려면 스페이스에 스페이스 관리자 또는 파워 유저 역할이 있는 계정으로 로그인해야 합니다.

블루프린트 작성자로서 수명 주기 관리 작업

수명 주기 관리를 사용하면 단일 공통 소스의 모범 사례에서 대량의 프로젝트를 동기화할 수 있습니다. 이렇게 하면 전체 소프트웨어 개발 수명 주기에 걸쳐 수정 사항 전파와 모든 수의 프로젝트 유지 관리가 확장됩니다. 수명 주기 관리의 내부 캠페인, 보안 수정, 감사, 런타임 업그레이드, 모범 사례의 변경 사항 및 기타 유지 관리 관행을 간소화합니다. 이러한 표준은 한 곳에서 정의되며 새 표준이 게시될 때 자동으로 중앙에서 최신 상태로 유지되기 때문입니다.

블루프린트의 새 버전이 게시되면, 해당 블루프린트가 포함된 모든 프로젝트에 최신 버전으로 업데이트하라는 메시지가 표시됩니다. 블루프린트 작성자는 각 프로젝트에 규정 준수 목적으로 포함된 특정

블루프린트의 버전을 볼 수도 있습니다. 기존 소스 리포지토리에 충돌이 있는 경우, 수명 주기 관리는 풀 요청을 생성합니다. 개발 환경과 같은 다른 모든 리소스의 경우, 모든 수명 주기 관리 업데이트에서 새 리소스를 전적으로 생성합니다. 사용자는 이러한 풀 요청을 자유롭게 병합하거나 병합하지 않을 수 있습니다. 보류 중인 풀 요청이 병합되면, 프로젝트에 사용되며 옵션을 포함한 블루프린트 버전이 업데이트됩니다. 블루프린트 사용자로서 수명 주기 관리 작업을 하는 방법에 대한 자세한 내용은 [기존 프로젝트에서 수명 주기 관리 사용](#) 및 [프로젝트의 여러 블루프린트에서 수명 주기 관리 사용](#) 섹션을 참조하세요.

주제

- [번들 출력 및 병합 충돌에 대한 수명 주기 관리 테스트](#)
- [병합 전략을 사용한 번들 생성 및 파일 지정](#)
- [프로젝트 세부 정보를 위한 컨텍스트 객체 액세스](#)

번들 출력 및 병합 충돌에 대한 수명 주기 관리 테스트

블루프린트의 수명 주기 관리를 로컬에서 테스트하고 충돌 해결을 병합할 수 있습니다. 수명 주기 업데이트의 다양한 단계를 나타내는 synth/ 디렉터리 아래에 일련의 번들이 생성됩니다. 수명 주기 관리를 테스트하려면 블루프린트 yarn blueprint: resynth에서 다음 안 명령을 실행할 수 있습니다. 재합성 및 번들에 대한 자세한 내용은 [재합성](#) 및 [재합성을 사용하여 파일 생성](#) 섹션을 참조하세요.

병합 전략을 사용한 번들 생성 및 파일 지정

병합 전략을 사용하여 재합성으로 번들을 생성하고 사용자 지정 블루프린트의 수명 주기 관리 업데이트를 위한 파일을 지정할 수 있습니다. 재합성 및 병합 전략을 활용하여 업데이트를 관리하고 배포 중에 업데이트되는 파일을 제어할 수 있습니다. 또한 변경 사항이 기존 CodeCatalyst 프로젝트로 병합되는 방식을 제어하는 자신만의 전략을 작성할 수 있습니다.

주제

- [재합성을 사용하여 파일 생성](#)
- [병합 전략 사용](#)
- [수명 주기 관리 업데이트를 위한 파일 지정](#)
- [병합 전략 작성](#)

재합성을 사용하여 파일 생성

재합성은 블루프린트에서 생성된 소스 코드를 이전에 동일한 블루프린트에서 생성된 소스 코드와 병합하여, 블루프린트에 대한 변경 사항을 기존 프로젝트에 전파할 수 있습니다. 병합은 블루프린트 출력

번들의 `resynth()` 함수에서 실행됩니다. 재합성은 먼저 블루프린트와 프로젝트 상태의 다양한 측면을 나타내는 세 개의 번들을 생성합니다. `yarn blueprint:resynth` 명령을 사용하여 로컬에서 수동으로 실행할 수 있으며, 명령이 아직 없는 경우 번들이 생성됩니다. 번들을 수동으로 사용하면 재합성 동작을 로컬에서 모의하고 테스트할 수 있습니다. 기본적으로 블루프린트는 번들의 해당 부분만 일반적으로 소스 제어 하에 있으므로 `src/*`의 리포지토리 전체에서 재합성만 실행합니다. 자세한 내용은 [재합성](#) 섹션을 참조하세요.

- **existing-bundle** - 이 번들은 기존 프로젝트 상태를 나타냅니다. 이는 합성 컴퓨팅에 의해 인위적으로 구성되어 프로젝트에서 배포 중인 항목(있는 경우)에 대한 블루프린트 컨텍스트를 제공합니다. 재합성을 로컬에서 실행할 때 이 위치에 이미 있는 경우 재설정되고 모의로 간주됩니다. 그렇지 않으면 **ancestor-bundle**의 내용으로 설정됩니다.
- **ancestor-bundle** - 일부 이전 옵션 및/또는 버전으로 합성된 경우 블루프린트 출력을 나타내는 번들입니다. 이 블루프린트를 프로젝트에 처음 추가하는 경우 이전 버전이 존재하지 않으므로 **existing-bundle**과 동일한 콘텐츠로 설정됩니다. 로컬에서 이 번들이 이미 이 위치에 있는 경우 모의로 간주됩니다.
- **proposed-bundle** - 이 번들은 블루프린트가 일부 새로운 옵션 및/또는 버전으로 합성된 경우 블루프린트를 모의합니다. `synth()` 함수에서 생성되는 것과 동일한 번들입니다. 로컬에서 이 번들은 항상 재정의됩니다.

각 번들은 `this.context.resynthesisPhase`의 블루프린트 클래스에서 액세스할 수 있는 재합성 단계에서 생성됩니다.

- **resolved-bundle** - CodeCatalyst 프로젝트에 패키징되고 배포되는 대상을 나타내는 최종 번들입니다. 배포 메커니즘으로 전송되는 파일과 차이 볼 수 있습니다. 이는 세 개의 다른 번들 간에 병합을 해결하는 `resynth()` 함수의 출력입니다.

ancestor-bundle와 **proposed-bundle**의 차이를 가져와 **existing-bundle**에 추가하여 **resolved-bundle**를 생성하면 3방향 병합이 적용됩니다. 모든 병합 전략은 파일을 **resolved-bundle**로 처리합니다. 재합성은 `resynth()`를 실행하는 동안 블루프린트의 병합 전략으로 이러한 번들의 도달 범위를 해결하고 결과에서 해결된 번들을 생성합니다.

병합 전략 사용

블루프린트 라이브러리에서 제공한 병합 전략을 사용할 수 있습니다. 이러한 전략은 [재합성을 사용하여 파일 생성](#) 섹션에 언급된 파일에 대한 파일 출력 및 충돌을 해결하는 방법을 제공합니다.

- **alwaysUpdate** - 항상 제안된 파일로 확인되는 전략입니다.

- `neverUpdate` - 항상 기존 파일로 확인되는 전략입니다.
- `onlyAdd` - 기존 파일이 아직 존재하지 않을 때 제안된 파일로 확인되는 전략입니다. 그렇지 않으면 기존 파일로 해결됩니다.
- `threeWayMerge` - 기존, 제안 및 공통 조상 파일 간에 3방향 병합을 수행하는 전략입니다. 파일을 깔끔하게 병합할 수 없는 경우 확인된 파일에 충돌 마커가 포함될 수 있습니다. 전략이 의미 있는 출력을 생성하려면 제공된 파일의 콘텐츠를 UTF-8로 인코딩해야 합니다. 전략은 입력 파일이 바이너리 파일인지 감지하려고 시도합니다. 전략이 바이너리 파일에서 병합 충돌을 감지하면 항상 제안된 파일을 반환합니다.
- `preferProposed` - 기존, 제안 및 공통 조상 파일 간에 3방향 병합을 수행하는 전략입니다. 이 전략은 각 충돌에서 제안된 파일 측을 선택하여 충돌을 해결합니다.
- `preferExisting` - 기존, 제안 및 공통 조상 파일 간에 3방향 병합을 수행하는 전략입니다. 이 전략은 각 충돌에서 기존 파일 측을 선택하여 충돌을 해결합니다.

병합 전략의 소스 코드를 보려면 [open-source GitHub repository](#)를 참조하세요.

수명 주기 관리 업데이트를 위한 파일 지정

재합성 중에 블루프린트는 변경 사항이 기존 소스 리포지토리에 병합되는 방식을 제어합니다. 그러나 블루프린트의 모든 단일 파일에 업데이트를 푸시하지 않을 수도 있습니다. 예를 들어, CSS 스타일시트와 같은 샘플 코드는 프로젝트별 코드입니다. 다른 전략을 지정하지 않으면 3방향 병합 전략이 기본 옵션입니다. 블루프린트는 리포지토리 구문 자체에 대한 병합 전략을 지정하여 소유한 파일과 소유하지 않은 파일을 지정할 수 있습니다. 블루프린트는 병합 전략을 업데이트할 수 있으며, 재합성 중에 최신 전략을 사용할 수 있습니다.

```
const sourceRepo = new SourceRepository(this, {
  title: 'my-repo',
});
sourceRepo.setResynthStrategies([
  {
    identifier: 'dont-override-sample-code',
    description: 'This strategy is applied accross all sample code. The blueprint
will create sample code, but skip attempting to update it.',
    strategy: MergeStrategies.neverUpdate,
    globs: [
      '**/src/**',
      '**/css/**',
    ],
  },
],
```

```
]);
```

여러 병합 전략을 지정할 수 있으며, 마지막 전략이 우선합니다. 검색되지 않은 파일은 기본적으로 Git과 유사한 3방향 병합으로 설정됩니다. MergeStrategies 구문을 통해 제공되는 몇 가지 병합 전략이 있지만 직접 작성할 수도 있습니다. 제공된 전략은 [git 병합 전략](#) 드라이버를 준수합니다.

병합 전략 작성

제공된 빌드 병합 전략 중 하나를 사용하는 것 외에도 자체 전략을 작성할 수도 있습니다. 전략은 표준 전략 인터페이스를 준수해야 합니다. existing-bundle, proposed-bundle, ancestor-bundle에서 파일 버전을 가져와서 하나의 확인된 파일로 병합하는 전략 함수를 작성해야 합니다. 예시:

```
type StrategyFunction = (
  /**
   * file from the ancestor bundle (if it exists)
   */
  commonAncestorFile: ContextFile | undefined,
  /**
   * file from the existing bundle (if it exists)
   */
  existingFile: ContextFile | undefined,
  /**
   * file from the proposed bundle (if it exists)
   */
  proposedFile: ContextFile | undefined,
  options?: {})
  /**
   * Return: file you'd like in the resolved bundle
   * passing undefined will delete the file from the resolved bundle
   */
=> ContextFile | undefined;
```

파일이 존재하지 않는 경우(정의되지 않은 경우), 해당 파일 경로는 해당 특정 위치 번들에 존재하지 않습니다.

예시:

```
strategies: [
  {
    identifier: 'dont-override-sample-code',
```



```

        description: 'This strategy is applied across all sample code. The
        blueprint will create sample code, but skip attempting to update it.',
        strategy: (ancestor, existing, proposed) => {
            const resolvedfile = ...
            ...
            // do something
            ...
            return resolvedfile
        },
        globs: [
            '**/src/**',
            '**/css/**',
        ],
    },
],

```

프로젝트 세부 정보를 위한 컨텍스트 객체 액세스

블루프린트 작성자는 합성 중에 블루프린트의 프로젝트에서 컨텍스트에 액세스하여 스페이스 및 프로젝트 이름 또는 프로젝트의 소스 리포지토리에 있는 기존 파일과 같은 정보를 얻을 수 있습니다. 블루프린트가 생성하는 재합성 단계와 같은 세부 정보를 얻을 수도 있습니다. 예를 들어 컨텍스트에 액세스하여 재합성을 통해 조상 번들 또는 제안된 번들을 생성하고 있는지 확인할 수 있습니다. 그런 다음 기존 코드 컨텍스트를 사용하여 리포지토리에서 코드를 변환할 수 있습니다. 예를 들어 자체 재합성 전략을 작성하여 특정 코드 표준을 설정할 수 있습니다. 작은 블루프린트를 위해 전략에 `blueprint.ts` 파일을 추가하거나 전략을 위해 별도의 파일을 생성할 수 있습니다.

다음 예시는 프로젝트의 컨텍스트에서 파일을 찾고, 워크플로 빌더를 설정하고, 특정 파일에 대한 블루프린트 대치 재합성 전략을 설정하는 방법을 보여줍니다.

```

const contextFiles = this.context.project.src.findAll({
    fileGlobs: ['**/package.json'],
});

// const workflows = this.context.project.src.findAll({
//     fileGlobs: ['**/.codecatalyst/**/*.yaml'],
// });

const security = new WorkflowBuilder(this, {
    Name: 'security-workflow',
});
new Workflow(this, repo, security.getDefinition());
repo.setResynthStrategies([

```

```

    {
      identifier: 'force-security',
      globs: ['**/.codecatalyst/security-workflow.yaml'],
      strategy: MergeStrategies.alwaysUpdate,
    },
  ]);

  for (const contextFile of contextFiles) {
    const packageObject = JSON.parse(contextFile.buffer.toString());
    new SourceFile(internalRepo, contextFile.path, JSON.stringify({
      ...packageObject,
    }, null, 2));
  }
}

```

프로젝트 요구 사항을 충족하기 위한 사용자 지정 블루프린트 개발

사용자 지정 블루프린트를 게시하기 전에 특정 요구 사항을 충족하도록 블루프린트를 개발할 수 있습니다. 사용자 지정 블루프린트를 개발하고 미리 볼 때 프로젝트를 생성하여 블루프린트를 테스트할 수 있습니다. 사용자 지정 블루프린트를 개발하여 특정 소스 코드, 계정 연결, 워크플로, 문제 또는 CodeCatalyst에서 생성할 수 있는 기타 구성 요소와 같은 프로젝트 구성 요소를 포함할 수 있습니다.

Important

외부 소스의 블루프린트 패키지를 사용하려면, 해당 패키지로 인해 발생할 수 있는 위험을 고려하세요. 스페이스에 추가하는 사용자 지정 블루프린트와 해당 블루프린트가 생성하는 코드는 사용자의 책임입니다.

Important

CodeCatalyst 스페이스에서 사용자 지정 블루프린트를 개발하려면 스페이스에 스페이스 관리자 또는 파워 유저 역할이 있는 계정으로 로그인해야 합니다.

사용자 지정 블루프린트를 개발하거나 업데이트하려면

1. 개발 환경을 재개합니다. 자세한 내용은 [개발 환경 재개](#) 섹션을 참조하세요.

개발 환경이 없는 경우 먼저 환경을 생성해야 합니다. 자세한 내용은 [개발 환경 생성](#) 섹션을 참조하세요.

2. 개발 환경에서 작업 중인 터미널을 엽니다.
3. 블루프린트를 생성할 때 릴리스 워크플로를 옵트인하면 최신 블루프린트 버전이 자동으로 게시됩니다. 변경 사항을 가져와 package.json 파일에 증분 버전이 있는지 확인합니다. 다음 명령을 사용합니다.

```
git pull
```

4. src/blueprint.ts 파일에서 사용자 지정 블루프린트의 옵션을 편집합니다. Options 인터페이스는 CodeCatalyst 마법사에 의해 동적으로 해석되어 선택 사용자 인터페이스(UI)를 생성합니다. 구성 요소 및 지원되는 태그를 추가하여 사용자 지정 블루프린트를 개발할 수 있습니다. 자세한 내용은 [프론트엔드 마법사를 사용하여 블루프린트 특성 수정](#), [블루프린트에 환경 구성 요소 추가](#), [블루프린트에 리전 구성 요소 추가](#), [블루프린트에 리포지토리 및 소스 코드 구성 요소 추가](#), [블루프린트에 워크플로 구성 요소 추가](#) 및 [블루프린트에 개발 환경 구성 요소 추가](#) 섹션을 참조하세요.

사용자 지정 블루프린트를 개발할 때 추가 지원을 받기 위해 블루프린트 SDK 및 샘플 블루프린트를 볼 수도 있습니다. 자세한 내용은 [오픈소스 GitHub 리포지토리](#)를 참조하세요.

사용자 지정 블루프린트는 성공적인 합성의 결과로 미리 보기 번들을 제공합니다. 프로젝트 번들은 프로젝트의 소스 코드, 구성 및 리소스를 나타내며 CodeCatalyst 배포 API 작업에서 프로젝트에 배포하는 데 사용됩니다. 사용자 지정 블루프린트를 계속 개발하려면 블루프린트 합성 프로세스를 다시 실행합니다. 자세한 내용은 [사용자 지정 블루프린트 개념](#) 섹션을 참조하세요.

프론트엔드 마법사를 사용하여 블루프린트 특성 수정

CodeCatalyst의 블루프린트 선택 마법사는 blueprint.ts 파일의 Options 인터페이스에서 자동으로 생성됩니다. 프론트엔드 마법사는 [JSDOC 스타일 설명과 태그](#)를 사용하여 블루프린트의 Options 수정 및 특성을 지원합니다. JSDOC 스타일의 설명과 태그를 사용하여 태스크를 수행할 수 있습니다. 예를 들어 옵션 위에 표시된 텍스트를 선택하거나, 입력 검증과 같은 기능을 활성화하거나, 옵션을 축소할 수 있습니다. 마법사는 Options 인터페이스의 TypeScript 유형에서 생성된 추상 구문 트리(AST)를 해석하여 작동합니다. 마법사는 가능한 한 가장 잘 설명된 유형으로 자동으로 구성합니다. 일부 유형은 지원되지 않습니다. 지원되는 다른 유형에는 리전 선택기와 환경 선택기가 포함됩니다.

다음은 블루프린트의 Options와 함께 JSDOC 주석 및 태그를 사용하는 마법사의 예입니다.

```
export interface Options {
  /**
   * What do you want to call your new blueprint?
   * @validationRegex /^[a-zA-Z0-9_]+$ /
   * @validationMessage Must contain only upper and lowercase letters, numbers and
underscores
   */
  blueprintName: string;

  /**
   * Add a description for your new blueprint.
   */
  description?: string;

  /**
   * Tags for your Blueprint:
   * @collapsed true
   */
  tags?: string[];
}
```

Options 인터페이스의 각 옵션 표시 이름은 기본적으로 camelCase에 표시됩니다. JSDOC 스타일 설명의 일반 텍스트는 마법사의 옵션 위에 텍스트로 표시됩니다.

주제

- [지원되는 태그](#)
- [지원되는 TypeScript 유형](#)
- [합성 중 사용자와 통신](#)

지원되는 태그

다음 JSDOC 태그는 프런트 엔드 마법사의 사용자 지정 블루프린트의 Options에서 지원됩니다.

@inlinePolicy ./path/to/policy/file.json

- 필수 - Role 유형으로 선택할 수 있습니다.
- 사용량 - 역할에 필요한 인라인 정책을 전달할 수 있습니다. policy.json 경로는 소스 코드 아래에 있을 것으로 예상됩니다. 역할에 대한 사용자 지정 정책이 필요한 경우 이 태그를 사용합니다.
- 종속성 - blueprint-cli 0.1.12 이상

- 예시 - @inlinePolicy ./deployment-policy.json

```
environment: EnvironmentDefinition{
  awsAccountConnection: AccountConnection{
    /**
     * @inlinePolicy ./path/to/deployment-policy.json
     */
    cdkRole: Role[];
  };
};
```

@trustPolicy ./path/to/policy/file.json

- 필수 - Role 유형으로 선택할 수 있습니다.
- 사용량 - 역할에 필요한 신뢰 정책을 전달할 수 있습니다. policy.json 경로는 소스 코드 아래에 있을 것으로 예상됩니다. 역할에 대한 사용자 지정 정책이 필요한 경우 이 태그를 사용합니다.
- 종속성 - blueprint-cli 0.1.12 이상
- 예시 - @trustPolicy ./trust-policy.json

```
environment: EnvironmentDefinition{
  awsAccountConnection: AccountConnection{
    /**
     * @trustPolicy ./path/to/trust-policy.json
     */
    cdkRole: Role[];
  };
};
```

@validationRegex 정규식

- 필수 - 문자열로 선택할 수 있습니다.
- 사용량 - 지정된 정규식을 사용하여 옵션에 대한 입력 검증을 수행하고 를 표시합니다 @validationMessage.
- 예시 - @validationRegex /^[a-zA-Z0-9_]+\$
- 권장 사항 - @validationMessage와 함께 사용합니다. 검증 메시지는 기본적으로 비어 있습니다.

@validationMessage 문자열

- 필수 - 사용량을 검토할 @validationRegex 또는 기타 오류
- 사용량 - @validation* 실패 시 검증 메시지를 표시합니다.
- 예시: - @validationMessage Must contain only upper and lowercase letters, numbers, and underscores
- 권장 사항 - @validationMessage와 함께 사용합니다. 검증 메시지는 기본적으로 비어 있습니다.

@collapsed boolean(선택 사항)

- 필수 - 해당 없음
- 사용량 - 하위 옵션을 축소할 수 있는 부울입니다. 축소된 주석이 있는 경우 기본값은 true입니다. 값을 @collapsed false로 설정하면 처음에 열려 있는 축소 가능한 섹션이 생성됩니다.
- 예시 - @collapsed true

@displayName 문자열

- 필수 - 해당 없음
- 사용량 - 옵션 표시 이름을 변경합니다. 표시 이름에 camelCase 이외의 형식을 허용합니다.
- 예시 - @displayName Blueprint Name

@displayName 문자열

- 필수 - 해당 없음
- 사용량 - 옵션 표시 이름을 변경합니다. 표시 이름에 [camelCase](#) 이외의 형식을 허용합니다.
- 예시 - @displayName Blueprint Name

@defaultEntropy 번호

- 필수 - 문자열로 선택할 수 있습니다.
- 사용량 - 지정된 길이의 무작위 배정된 영숫자 문자열을 옵션에 추가합니다.
- 예시 - @defaultEntropy 5

@placeholder 문자열(선택 사항)

- 필수 - 해당 없음
- 사용량 - 기본 텍스트 필드 자리 표시자를 변경합니다.
- 예시 - @placeholder type project name here

@textArea 번호(선택 사항)

- 필수 - 해당 없음
- 사용량 - 문자열 입력을 더 큰 텍스트 섹션의 텍스트 영역 구성 요소로 변환합니다. 숫자를 추가하면 행 수가 정의됩니다. 기본값은 모든 행입니다.
- 예시 - @textArea 10

@hidden 부울(선택 사항)

- 필수 - 해당 없음
- 사용량 - 검증 확인에 실패하지 않는 한 사용자로부터 파일을 숨깁니다. 기본값은 true입니다.
- 예시 - @hidden

@button 부울(선택 사항)

- 필수 - 해당 없음
- 사용량 - 설명은 부울 속성에 있어야 합니다. 선택하면 true로 합성되는 버튼을 추가합니다. 토글이 아닙니다.
- 예시 - buttonExample: boolean;

```
/**
 * @button
 */
buttonExample: boolean;
```

@showName 부울(선택 사항)

- 필수 - 해당 없음

- 사용량 - 계정 연결 유형에서만 사용할 수 있습니다. 숨겨진 이름 입력을 표시합니다. 기본값은 default_environment입니다.
- 예시 - @showName true

```
/**
 * @showName true
 */
accountConnection: AccountConnection<{
    ...
}>;
```

@showEnvironmentType 부울(선택 사항)

- 필수 - 해당 없음
- 사용량 - 계정 연결 유형에서만 사용할 수 있습니다. 숨겨진 환경 유형 드롭다운 메뉴를 표시합니다. 모든 연결은 기본적으로 production입니다. 옵션은 비프로덕션 또는 프로덕션입니다.
- 예시 - @showEnvironmentType true

```
/**
 * @showEnvironmentType true
 */
accountConnection: AccountConnection<{
    ...
}>;
```

@forceDefault boolean(선택 사항)

- 필수 - 해당 없음
- 사용량 - 이전에 사용자가 사용한 값 대신 블루프린트 작성자가 제공한 기본값을 사용합니다.
- 예시 - forceDefaultExample: any;

```
/**
 * @forceDefault
 */
forceDefaultExample: any;
```


@requires blueprintName

- 필수 - Options 인터페이스에 주석을 지정합니다.
- 사용량 - 현재 블루프린트의 요구 사항으로 blueprintName 프로젝트에 지정된 를 추가하도록 사용자에게 경고합니다.
- 예시 - @requires '@amazon-codecatalyst/blueprints.blueprint-builder'

```
/*
 * @requires '@amazon-codecatalyst/blueprints.blueprint-builder'
 */
export interface Options extends ParentOptions {
  ...
}
```

@filter 정규식

- 필수 - Selector 또는 MultiSelect 인터페이스에 설명을 지정합니다.
- 사용량 - 마법사의 드롭다운을 지정된 정규식과 일치하는 옵션으로 필터링합니다.
- 예시 - @filter /blueprintPackageName/

```
/**
 * @filter /myPackageName/
 */
blueprintInstantiation?: Selector<BlueprintInstantiation>;
...
}
```

지원되는 TypeScript 유형

다음 TypeScript 유형은 프런트 엔드 마법사의 사용자 지정 블루프린트의 Options에서 지원됩니다.

번호

- 필수 - number 유형으로 선택할 수 있습니다.
- 사용량 - 숫자 입력 필드를 생성합니다.
- 예시 - age: number

```
{
  age: number
}
```

```
...  
}
```

String

- 필수 - string 유형으로 선택할 수 있습니다.
- 사용량 - 문자열 입력 필드를 생성합니다.
- 예시 - name: string

```
{  
  age: string  
  ...  
}
```

문자열 목록

- 필수 - string 유형의 배열로 선택할 수 있습니다.
- 사용량 - 문자열 목록 입력을 생성합니다.
- 예시 - isProduction: boolean

```
{  
  isProduction: boolean  
  ...  
}
```

Checkbox

- 필수 - boolean으로 선택할 수 있습니다.
- 사용량 - 확인란을 생성합니다.
- 예시 - isProduction: boolean

```
{  
  isProduction: boolean  
  ...  
}
```

Radio

- 필수 - 3개 이하의 문자열을 결합할 수 있는 옵션입니다.
- 사용량 - 선택한 라디오를 생성합니다.

Note

4개 이상의 항목이 있는 경우 이 유형은 드롭다운으로 렌더링됩니다.

- 예시 - color: 'red' | 'blue' | 'green'

```
{
  color: 'red' | 'blue' | 'green'
  ...
}
```

Dropdown

- 필수 - 4개 이상의 문자열을 결합할 수 있는 옵션입니다.
- 사용량 - 드롭다운을 생성합니다.
- 예시 - runtimes: 'nodejs' | 'python' | 'java' | 'dotnetcore' | 'ruby'

```
{
  runtimes: 'nodejs' | 'python' | 'java' | 'dotnetcore' | 'ruby'
  ...
}
```

Expandable section

- 필수 - 객체로 선택할 수 있습니다.
- 사용량 - 확장 가능한 섹션을 생성합니다. 객체의 옵션은 마법사의 확장 가능한 섹션 내에 중첩됩니다.
- 예시 -

```
{
  expandableSectionTitle: {
```

```
        nestedString: string;
        nestedNumber: number;
    }
}
```

Tuple

- 필수 - Tuple 유형으로 선택할 수 있습니다.
- 사용량 - 키 값 유로 입력을 생성합니다.
- 예시 - tuple: Tuple[string, string]>

```
{
    tuple: Tuple[string, string]>;
    ...
}
```

Tuple list

- 필수 - Tuple 유형의 배열로 선택할 수 있습니다.
- 사용량 - tuple 목록 입력을 생성합니다.
- 예시 - tupleList: Tuple[string, string]>[]

```
{
    tupleList: Tuple[string, string]>[];
    ...
}
```

Selector

- 필수 - Selector 유형으로 선택할 수 있습니다.
- 사용량 - 프로젝트에 적용된 소스 리포지토리 또는 블루프린트의 드롭다운을 생성합니다.
- 예시 - sourceRepo: Selector<SourceRepository>

```
{
    sourceRepo: Selector<SourceRepository>;
    sourceRepoOrAdd: Selector<SourceRepository | string>;
}
```

```
    blueprintInstantiation: Selector<BlueprintInstantiation>;
    ...
}
```

Multiselect

- 필수 - Selector 유형으로 선택할 수 있습니다.
- 사용량 - 다중 선택 입력을 생성합니다.
- 예시 - multiselect: MultiSelect['A' | 'B' | 'C' | 'D' | 'E']>

```
{
  multiselect: MultiSelect['A' | 'B' | 'C' | 'D' | 'E']>;
  ...
}
```

합성 중 사용자와 통신

블루프린트 작성자는 검증 메시지 외에 사용자에게 다시 통신할 수 있습니다. 예를 들어 스페이스 멤버는 명확하지 않은 블루프린트를 생성하는 옵션의 조합을 볼 수 있습니다. 사용자 지정 블루프린트는 합성을 호출하여 오류 메시지를 사용자에게 다시 전달하는 기능을 지원합니다. 기본 블루프린트는 명확한 오류 메시지를 기대하는 `throwSynthesisError(...)` 함수를 구현합니다. 다음을 사용하여 메시지를 간접 호출할 수 있습니다.

```
//blueprint.ts
this.throwSynthesisError({
  name: BlueprintSynthesisErrorTypes.BlueprintSynthesisError,
  message: 'hello from the blueprint! This is a custom error communicated to the user.'
}))
```

입력 생성 및 프론트엔드 마법사 요소 다시 렌더링

`DynamicKVInput`을 사용하여 마법사 입력을 생성하고 사용자 지정 블루프린트에 대한 프론트엔드 마법사 요소를 동적으로 생성할 수 있습니다.

주제

- [개발 환경 생성](#)
- [동적으로 마법사 요소 생성](#)

개발 환경 생성

DynamicKVInput 유형을 사용하여 사용자 지정 blueprint의 기본값을 사용하여 프론트엔드 마법사 입력을 생성할 수 있습니다. 최신 스키마를 보려면 [DynamicKVInput 정의](#)를 참조하세요.

다음 예시에서는 Options를 사용하여 객체를 형성하는 방법을 보여줍니다.

```
import { DynamicKVInput } from '@amazon-codecatalyst/blueprints.blueprint';

export interface Options extends ParentOptions {

    parameters: DynamicKVInput[];

}
```

다음 예시에서는 여러 속성을 사용하여 기본 파라미터를 설정하는 방법을 보여줍니다.

```
{
  "parameters": [
    {
      "key": "AWS_REGION",
      "value": "us-west-2",
      "displayType": "region",
      "possibleValues": [
        "us-west-1",
        "us-west-2",
        "us-east-1",
        "us-east-2"
      ],
      "displayName": "AWS Region",
      "description": "AWS Region to deploy the solution to."
    },
    {
      "key": "SchedulingActive",
      "value": "Yes",
      "displayType": "dropdown",
      "possibleValues": [
        "Yes",
        "No"
      ],
      "displayName": "Scheduling Active",
      "description": "Activate or deactivate scheduling."
    }
  ],
}
```

```
{
  "key": "ScheduledServices",
  "value": "Both",
  "displayType": "dropdown",
  "possibleValues": [
    "EC2",
    "RDS",
    "Both"
  ],
  "displayName": "Scheduled Services",
  "description": "Services to schedule."
},
{
  "key": "ScheduleRdsClusters",
  "value": "No",
  "displayType": "dropdown",
  "possibleValues": [
    "Yes",
    "No"
  ],
  "displayName": "Schedule RDS Clusters",
  "description": "Enable scheduling of Aurora clusters for RDS service."
},
{
  "key": "CreateRdsSnapshot",
  "value": "No",
  "displayType": "dropdown",
  "possibleValues": [
    "Yes",
    "No"
  ],
  "displayName": "Create RDS Snapshot",
  "description": "Create snapshot before stopping RDS instances (does not
apply to Aurora Clusters)."
},
{
  "key": "MemorySize",
  "value": "128",
  "displayType": "dropdown",
  "possibleValues": [
    "128",
    "384",
    "512",
    "640",
```

```
        "768",
        "896",
        "1024",
        "1152",
        "1280",
        "1408",
        "1536"
    ],
    "displayName": "Memory Size",
    "description": "Size of the Lambda function running the scheduler, increase
size when processing large numbers of instances."
},
{
    "key": "UseCloudWatchMetrics",
    "value": "No",
    "displayType": "dropdown",
    "possibleValues": [
        "Yes",
        "No"
    ],
    "displayName": "Use CloudWatch Metrics",
    "description": "Collect instance scheduling data using CloudWatch metrics."
},
{
    "key": "LogRetentionDays",
    "value": "30",
    "displayType": "dropdown",
    "possibleValues": [
        "1",
        "3",
        "5",
        "7",
        "14",
        "30",
        "60",
        "90",
        "120",
        "150",
        "180",
        "365",
        "400",
        "545",
        "731",
        "1827",
```



```

        "3653"
    ],
    "displayName": "Log Retention Days",
    "description": "Retention days for scheduler logs."
},
{
    "key": "Trace",
    "value": "No",
    "displayType": "dropdown",
    "possibleValues": [
        "Yes",
        "No"
    ],
    "displayName": "Trace",
    "description": "Enable debug-level logging in CloudWatch logs."
},
{
    "key": "EnableSSMMaintenanceWindows",
    "value": "No",
    "displayType": "dropdown",
    "possibleValues": [
        "Yes",
        "No"
    ],
    "displayName": "Enable SSM Maintenance Windows",
    "description": "Enable the solution to load SSM Maintenance Windows, so
that they can be used for EC2 instance Scheduling."
},
{
    "key": "DefaultTimezone",
    "value": "UTC",
    "displayType": "string",
    "displayName": "Default Timezone",
    "description": "Default timezone to use for scheduling."
},
{
    "key": "Regions",
    "value": "us-west-2",
    "displayType": "string",
    "displayName": "Regions",
    "description": "List of regions in which instances should be scheduled,
leave blank for current region only."
},
{

```

```

        "key": "UsingAWSOrganizations",
        "value": "No",
        "displayType": "dropdown",
        "possibleValues": [
            "Yes",
            "No"
        ],
        "displayName": "Using AWS Organizations",
        "description": "Use AWS Organizations to automate spoke account
registration."
    },
    {
        "key": "Principals",
        "displayType": "string",
        "optional": false,
        "displayName": "Principals",
        "description": "(Required) If using AWS Organizations, provide the
Organization ID. Eg. o-xxxxyyy. Else, provide a comma separated list of spoke account
ids to schedule. Eg.: 1111111111, 2222222222 or {param: ssm-param-name}"
    },
    {
        "key": "Namespace",
        "value": "Default",
        "displayType": "string",
        "displayName": "Namespace",
        "description": "Provide unique identifier to differentiate between multiple
solution deployments (No Spaces). Example: Dev"
    },
    {
        "key": "SchedulerFrequency",
        "value": 5,
        "displayType": "number",
        "displayName": "Scheduler Frequency",
        "description": "Scheduler running frequency in minutes."
    }
]
}

```

동적으로 마법사 요소 생성

마법사 입력을 생성하는 것과 동일한 스키마를 사용하여 합성 중에 마법사를 동적으로 다시 렌더링할 수 있습니다. 필요한 경우 사용자의 후속 질문을 처리하는 데 사용할 수 있습니다.

```
//blueprint.ts

export interface Options extends ParentOptions {
  ...
  dynamicOptions: OptionsSchemaDefinition<'optionsIdentifier', KVSchema>;
}
```

그런 다음 Options 구성 요소를 사용하여 합성 기간 중에 마법사를 설정할 수 있습니다.

```
import {
  OptionsSchemaDefinition,
  OptionsSchema,
} from '@amazon-codecatalyst/blueprints.blueprint';

...

// dynamically renders a number in the place where 'optionsIdentifier' was set in the
original options type.
new OptionsSchema<KVSchema>(this, 'optionsIdentifier', [
  {
    "key": "SchedulerFrequency",
    "value": 5,
    "displayType": "number",
    "displayName": "Scheduler Frequency",
    "description": "Scheduler running frequency in minutes."
  }
]);
```

블루프린트에 환경 구성 요소 추가

사용자 지정 블루프린트 마법사는 마법사를 통해 노출된 Options 인터페이스에서 동적으로 생성됩니다. 블루프린트는 노출된 유형에서 사용자 인터페이스(UI) 구성 요소 생성을 지원합니다.

Amazon CodeCatalyst 블루프린트 환경 구성 요소를 가져오려면

다음은 blueprint.ts 파일에 추가합니다.

```
import {...} from '@amazon-codecatalyst/codecatalyst-environments'
```

주제

- [개발 환경 생성](#)

- [환경 목록](#)
- [모의 인터페이스 예시](#)

개발 환경 생성

다음 예시에서는 애플리케이션을 클라우드에 배포하는 방법을 보여줍니다.

```
export interface Options extends ParentOptions {
  ...
  myNewEnvironment: EnvironmentDefinition{
    thisIsMyFirstAccountConnection: AccountConnection{
      thisIsARole: Role['lambda', 's3', 'dynamo'];
    };
  };
}
```

인터페이스는 단일 계정 연결(thisIsMyFirstAccountConnection)을 사용하여 새 환경(myNewEnvironment)을 요청하는 UI 구성 요소를 생성합니다. 계정 연결(thisIsARole)의 역할도 최소 필수 역할 기능으로서 ['lambda', 's3', 'dynamo']를 사용하여 생성됩니다. 모든 사용자에게 계정 연결이 있는 것은 아니므로 사용자가 계정을 연결하지 않거나 계정을 역할에 연결하지 않는 경우를 확인해야 합니다. 역할에는 @inlinePolicies로 주석을 달 수도 있습니다. 자세한 내용은 [@inlinePolicy ./path/to/policy/file.json](#) 섹션을 참조하세요.

환경 구성 요소에는 name 및 environmentType가 필요합니다. 다음 코드는 최소 필수 기본 셰이프입니다.

```
{
  ...
  "myNewEnvironment": {
    "name": "myProductionEnvironment",
    "environmentType": "PRODUCTION"
  },
}
```

그러면 UI 구성 요소에 다양한 필드가 표시됩니다. 필드를 채우면 블루프린트가 완전히 확장된 모양이 됩니다. 테스트 및 개발 목적으로 defaults.json 파일에 전체 모의를 포함하는 것이 도움이 될 수 있습니다.

환경 목록

EnvironmentDefinition 유형 배열을 지정하면 마법사 UI에 환경 목록이 생성됩니다.

```
export interface Options extends ParentOptions {
  ...
  /**
   * @showName readOnly
   */
  myEnvironments: EnvironmentDefinition<{
    thisIsMyFirstAccountConnection: AccountConnection<{
      thisIsARole: Role<['lambda', 's3', 'dynamo']>;
    }>;
  }>[];
}
```

다음 예시에서는 환경 목록의 기본값을 보여줍니다.

```
{
  ...
  "myEnvironments": [
    {
      "name": "myProductionEnvironment",
      "environmentType": "PRODUCTION"
    },
    {
      "name": "myDevelopmentEnvironment",
      "environmentType": "DEVELOPMENT"
    },
  ]
}
```

모의 인터페이스 예시

간단한 모의 인터페이스

```
{
  ...
  "thisIsMyEnvironment": {
    "name": "myProductionEnvironment",
    "environmentType": "PRODUCTION",
  }
}
```

```

    "thisIsMySecondAccountConnection": {
      "id": "12345678910",
      "name": "my-account-connection-name",
      "secondAdminRole": {
        "arn": "arn:aws:iam::12345678910:role/ConnectedQuokkaRole",
        "name": "ConnectedQuokkaRole",
        "capabilities": [
          "lambda",
          "s3",
          "dynamo"
        ]
      }
    }
  }
}

```

복잡한 모의 인터페이스

```

export interface Options extends ParentOptions {
  /**
   * The name of an environment
   * @displayName This is a Environment Name
   * @collapsed
   */
  thisIsMyEnvironment: EnvironmentDefinition{
    /**
     * comments about the account that is being deployed into
     * @displayName This account connection has an overridden name
     * @collapsed
     */
    thisIsMyFirstAccountConnection: AccountConnection{
      /**
       * Blah blah some information about the role that I expect
       * e.g. here's a copy-pastable policy: [to a link]
       * @displayName This role has an overridden name
       */
      adminRole: Role['admin', 'lambda', 's3', 'cloudfront'];
    }
    /**
     * Blah blah some information about the second role that I expect
     * e.g. here's a copy-pastable policy: [to a link]
     */
    lambdaRole: Role['lambda', 's3'];
  };
}

```

```

/**
 * comments about the account that is being deployed into
 */
thisIsMySecondAccountConnection: AccountConnection{
  /**
   * Blah blah some information about the role that I expect
   * e.g. here's a copy-pastable policy: [to a link]
   */
  secondAdminRole: Role['admin', 'lambda', 's3', 'cloudfront'];
  /**
   * Blah blah some information about the second role that I expect
   * e.g. here's a copy-pastable policy: [to a link]
   */
  secondLambdaRole: Role['lambda', 's3'];
};
};
}

```

전체 모의 인터페이스

```

{
  ...
  "thisIsMyEnvironment": {
    "name": "my-production-environment",
    "environmentType": "PRODUCTION",
    "thisIsMySecondAccountConnection": {
      "id": "12345678910",
      "name": "my-connected-account",
      "secondAdminRole": {
        "name": "LambdaQuokkaRole",
        "arn": "arn:aws:iam::12345678910:role/LambdaQuokkaRole",
        "capabilities": [
          "admin",
          "lambda",
          "s3",
          "cloudfront"
        ]
      },
      "secondLambdaRole": {
        "name": "LambdaQuokkaRole",
        "arn": "arn:aws:iam::12345678910:role/LambdaQuokkaRole",
        "capabilities": [
          "lambda",

```

```

        "s3"
      ]
    }
  },
  "thisIsMyFirstAccountConnection": {
    "id": "12345678910",
    "name": "my-connected-account",
    "adminRole": {
      "name": "LambdaQuokkaRole",
      "arn": "arn:aws:iam::12345678910:role/LambdaQuokkaRole",
      "capabilities": [
        "admin",
        "lambda",
        "s3",
        "cloudfront"
      ]
    },
    "lambdaRole": {
      "name": "LambdaQuokkaRole",
      "arn": "arn:aws:iam::12345678910:role/LambdaQuokkaRole",
      "capabilities": [
        "lambda",
        "s3"
      ]
    }
  }
},
}
}

```

블루프린트에 보안 암호 구성 요소 추가

CodeCatalyst에서 보안 암호를 사용하여 워크플로에서 참조할 수 있는 민감한 데이터를 저장할 수 있습니다. 사용자 지정 블루프린트에 보안 암호를 추가하고 워크플로에서 참조할 수 있습니다. 자세한 내용은 [보안 암호를 사용하여 데이터 마스킹](#) 섹션을 참조하세요.

Amazon CodeCatalyst 블루프린트 리전 유형을 가져오려면

다음은 `blueprint.ts` 파일에 추가합니다.

```
import { Secret, SecretDefinition } from '@amazon-codecatalyst/blueprint-component.secrets'
```

주제

- [보안 암호 생성](#)
- [워크플로에서 보안 암호 참조](#)

보안 암호 생성

다음 예시에서는 사용자에게 보안 암호 값과 선택적 설명을 입력하라는 메시지를 표시하는 UI 구성 요소를 생성합니다.

```
export interface Options extends ParentOptions {
    ...
    mySecret: SecretDefinition;
}

export class Blueprint extends ParentBlueprint {
    constructor(options_: Options) {
        new Secret(this, options.secret);
    }
}
```

보안 암호 구성 요소에는 name이 필요합니다. 다음 코드는 최소 필수 기본 셰이프입니다.

```
{
    ...
    "secret": {
        "name": "secretName"
    },
}
```

워크플로에서 보안 암호 참조

다음 예시에서 블루프린트는 보안 암호와 보안 암호 값을 참조하는 워크플로를 생성합니다. 자세한 내용은 [워크플로에서 보안 암호 참조](#) 섹션을 참조하세요.

```
export interface Options extends ParentOptions {
    ...
    /**
     *
     * @validationRegex /^\\w+$/
     */
    username: string;
}
```

```

    password: SecretDefinition;
  }

export class Blueprint extends ParentBlueprint {
  constructor(options_: Options) {
    const password = new Secret(this, options_.password);

    const workflowBuilder = new WorkflowBuilder(this, {
      Name: 'my_workflow',
    });

    workflowBuilder.addAction({
      actionName: 'download_files',
      input: {
        Sources: ['WorkflowSource'],
      },
      output: {
        Artifacts: [{ Name: 'download', Files: ['file1'] }],
      },
      steps: [
        `curl -u ${options_.username}:${password.reference} https://example.com`,
      ],
    });

    new Workflow(
      this,
      repo,
      workflowBuilder.getDefinition(),
    );
  }
}

```

CodeCatalyst에서 보안 암호 사용에 대한 자세한 내용은 [보안 암호를 사용하여 데이터 마스킹](#) 섹션을 참조하세요.

블루프린트에 리전 구성 요소 추가

리전 유형을 사용자 지정 블루프린트의 Options 인터페이스에 추가하여 블루프린트 마법사에서 구성 요소를 생성할 수 있습니다. 블루프린트 마법사에 하나 이상의 AWS 리전을 입력할 수 있습니다. 리

전 유형은 `blueprint.ts` 파일의 기본 블루프린트에서 가져올 수 있습니다. 자세한 내용은 [AWS 리전](#) 섹션을 참조하세요.

Amazon CodeCatalyst 블루프린트 리전 유형을 가져오려면

다음은 `blueprint.ts` 파일에 추가합니다.

```
import { Region } from '@amazon-codecatalyst/blueprints.blueprint'
```

리전 유형 파라미터는 선택할 수 있는 AWS 리전 코드의 배열이거나 지원되는 모든 AWS 리전을 포함하는 데 *를 사용할 수 있습니다.

주제

- [Annotations](#)
- [리전 구성 요소 예시](#)

Annotations

JSDoc 태그를 Options 인터페이스의 각 필드에 추가하여 마법사에서 필드가 나타나고 동작하는 방식을 사용자 지정할 수 있습니다. 리전 유형에서 다음 태그가 지원됩니다.

- `@displayName` 주석을 사용하여 마법사에서 필드의 레이블을 변경할 수 있습니다.

예시: `@displayName AWS Region`

- `@placeholder` 주석을 사용하여 선택/다중 선택 구성 요소의 자리 표시자를 변경할 수 있습니다.

예시: `@placeholder Choose AWS Region`

리전 구성 요소 예시

지정된 목록에서 리전 선택

```
export interface Options extends ParentOptions {  
    ...  
    /**  
     * @displayName Region  
     */  
    region: Region<['us-east-1', 'us-east-2', 'us-west-1', 'us-west-2']>;  
}
```

지정된 목록에서 하나 이상의 리전 선택

```
export interface Options extends ParentOptions {
    ...
    /**
     * @displayName Regions
     */
    multiRegion: Region<['us-east-1', 'us-east-2', 'us-west-1', 'us-west-2']>[];
}
```

AWS 리전 하나 선택

```
export interface Options extends ParentOptions {
    ...
    /**
     * @displayName Region
     */
    region: Region<['*']>;
}
```

지정된 목록에서 하나 이상의 리전 선택

```
export interface Options extends ParentOptions {
    ...
    /**
     * @displayName Regions
     */
    multiRegion: Region<['us-east-1', 'us-east-2', 'us-west-1', 'us-west-2']>[];
}
```

블루프린트에 리포지토리 및 소스 코드 구성 요소 추가

Amazon CodeCatalyst는 리포지토리를 사용하여 코드를 저장합니다. 리포지토리는 이름을 입력으로 사용합니다. 대부분의 구성 요소는 소스 코드 파일, 워크플로 및 관리형 개발 환경(MDE)과 같은 기타 구성 요소와 같은 리포지토리에 저장됩니다. 소스 리포지토리 구성 요소는 파일 및 정적 자산 관리에 사용되는 구성 요소도 내보냅니다. 리포지토리에는 이름 제약이 있습니다. 자세한 내용은 [CodeCatalyst의 소스 리포지토리로 코드 저장 및 협업](#) 섹션을 참조하세요.

```
const repository = new SourceRepository(this, {
    title: 'my-new-repository-title',
});
```

Amazon CodeCatalyst 블루프린트 리포지토리 및 소스 코드 구성 요소를 가져오려면

다음은 `blueprint.ts` 파일에 추가합니다.

```
import {...} from '@caws-blueprint-component/caws-source-repositories'
```

주제

- [파일 추가](#)
- [일반 파일 추가](#)
- [파일 복사](#)
- [여러 파일 대상으로 지정](#)
- [새 리포지토리 생성 및 파일 추가](#)

파일 추가

`SourceFile` 구문을 사용하여 리포지토리에 텍스트 파일을 쓸 수 있습니다. 작업은 가장 일반적인 사용 사례 중 하나이며 리포지토리, 파일 경로 및 텍스트 콘텐츠를 사용합니다. 리포지토리 내에 파일 경로가 없는 경우 구성 요소는 필요한 모든 폴더를 생성합니다.

```
new SourceFile(repository, `path/to/my/file/in/repo/file.txt`, 'my file contents');
```

Note

동일한 리포지토리 내의 동일한 위치에 두 파일을 쓰면 가장 최근의 구현이 이전 파일을 덮어 씁니다. 이 기능을 사용하여 생성된 코드를 계층화할 수 있으며, 사용자 지정 블루프린트가 생성했을 수 있는 코드를 확장하는 데 특히 유용합니다.

일반 파일 추가

리포지토리에 임의의 비트를 쓸 수 있습니다. 버퍼에서 읽고 `File` 구문을 사용할 수 있습니다.

```
new File(repository, `path/to/my/file/in/repo/file.img`, new Buffer(...));

new File(repository, `path/to/my/file/in/repo/new-img.img`, new StaticAsset('path/to/
image.png').content());
```

파일 복사

스타터 코드를 복사하여 붙여넣은 다음 해당 베이스 위에 더 많은 코드를 생성하여 생성된 코드를 시작할 수 있습니다. `static-assets` 디렉터리 내에 코드를 배치한 다음 `StaticAsset` 구문으로 해당 코드를 대상으로 지정합니다. 이 경우 경로는 항상 `static-assets` 디렉터리의 루트에서 시작됩니다.

```
const starterCode = new StaticAsset('path/to/file/file.txt')
const starterCodeText = new StaticAsset('path/to/file/file.txt').toString()
const starterCodeRawContent = new StaticAsset('path/to/image/hello.png').content()

const starterCodePath = new StaticAsset('path/to/image/hello.png').path()
// starterCodePath is equal to 'path/to/image/hello.png'
```

`StaticAsset`의 하위 클래스는 `SubstitutionAsset`입니다. 하위 클래스는 정확히 동일한 함수를 사용하지만 대신 파일에서 콧수염 대체를 실행할 수 있습니다. 복사 및 교체 스타일 생성을 수행하는데 유용할 수 있습니다.

정적 자산 대체는 머스타쉬 템플릿 엔진을 사용하여 생성된 소스 리포지토리에 시드되는 정적 파일을 렌더링합니다. 콧수염 템플릿 규칙은 렌더링 중에 적용되므로 모든 값은 기본적으로 HTML 인코딩됩니다. 이스케이프되지 않은 HTML을 렌더링하려면 `{{{name}}}` 트리플 머스타쉬 구문을 사용합니다. 자세한 내용은 [머스타쉬 템플릿 규칙](#)을 참조하세요.

Note

텍스트 해석이 불가능한 파일 대신 대체 파일을 실행하면 오류가 발생할 수 있습니다.

```
const starterCodeText = new SubstitutionAsset('path/to/file/file.txt').subsitite({
  'my_variable': 'subbed value1',
  'another_variable': 'subbed value2'
})
```

여러 파일 대상으로 지정

정적 자산은 `StaticAsset` 및 그 하위 클래스 `findAll(...)`의 정적 함수를 통해 glob 타겟팅을 지원하며, 경로, 콘텐츠 등이 미리 로드된 정적 자산 목록을 반환합니다. 목록을 `File` 구문과 연결하여 `static-assets` 디렉터리에 콘텐츠를 복사하고 붙여넣을 수 있습니다.

```
new File(repository, `path/to/my/file/in/repo/file.img`, new Buffer(...));
```

```
new File(repository, `path/to/my/file/in/repo/new-img.img`, new StaticAsset('path/to/
image.png').content());
```

새 리포지토리 생성 및 파일 추가

리포지토리 구성 요소를 사용하여 생성된 프로젝트에 새 리포지토리를 생성할 수 있습니다. 그런 다음 생성된 리포지토리에 파일 또는 워크플로를 추가할 수 있습니다.

```
import { SourceRepository } from '@amazon-codecatalyst/codecatalyst-source-
repositories';
...
const repository = new SourceRepository(this, { title: 'myRepo' });
```

다음 예시에서는 기존 리포지토리에 파일 및 워크플로를 추가하는 방법을 보여줍니다.

```
import { SourceFile } from '@amazon-codecatalyst/codecatalyst-source-repositories';
import { Workflow } from '@amazon-codecatalyst/codecatalyst-workflows';
...
new SourceFile(repository, 'README.md', 'This is the content of my readme');
new Workflow(this, repository, {/**...workflowDefinition...**/});
```

두 코드 조각을 결합하면 이름이 myRepo인 단일 리포지토리가 소스 파일 README.md 및 루트의 CodeCatalyst 워크플로와 함께 생성됩니다.

블루프린트에 워크플로 구성 요소 추가

워크플로는 Amazon CodeCatalyst 프로젝트에서 트리거를 기반으로 작업을 실행하는 데 사용됩니다. 워크플로 구성 요소를 사용하여 워크플로 YAML 파일을 빌드하고 통합할 수 있습니다. 자세한 내용은 [워크플로 YAML 정의](#) 섹션을 참조하세요.

Amazon CodeCatalyst 블루프린트 워크플로 구성 요소를 가져오려면

다음을 blueprint.ts 파일에 추가합니다.

```
import { WorkflowBuilder, Workflow } from '@amazon-codecatalyst/codecatalyst-workflows'
```

주제

- [워크플로 구성 요소 예시](#)

- [환경에 연결](#)

워크플로 구성 요소 예시

WorkflowBuilder 구성 요소

클래스를 사용하여 워크플로 정의를 빌드할 수 있습니다. 리포지토리에서 렌더링하기 위해 워크플로 구성 요소에 정의를 부여할 수 있습니다.

```
import { WorkflowBuilder } from '@amazon-codecatalyst/codecatalyst-workflows'

const workflowBuilder = new WorkflowBuilder({} as Blueprint, {
  Name: 'my_workflow',
});

// trigger the workflow on pushes to branch 'main'
workflowBuilder.addBranchTrigger(['main']);

// add a build action
workflowBuilder.addAction({
  // give the action a name
  actionName: 'build_and_do_some_other_stuff',

  // the action pulls from source code
  input: {
    Sources: ['WorkflowSource'],
  },

  // the output attempts to autodiscover test reports, but not in the node modules
  output: {
    AutoDiscoverReports: {
      Enabled: true,
      ReportNamePrefix: AutoDiscovered,
      IncludePaths: ['**/*'],
      ExcludePaths: ['*/node_modules/**/*'],
    },
  },
});

// execute some arbitrary steps
steps: [
  'npm install',
  'npm run myscript',
  'echo hello-world',
],
```



```
// add an account connection to the workflow
environment: convertToWorkflowEnvironment(myEnv),
});
```

Workflow Projen 구성 요소

다음 예시에서는 Projen 구성 요소를 사용하여 워크플로 YAML을 리포지토리에 쓰는 방법을 보여줍니다.

```
import { Workflow } from '@amazon-codecatalyst/codecatalyst-workflows'

...

const repo = new SourceRepository
const blueprint = this;
const workflowDef = workflowBuilder.getDefinition()

// creates a workflow.yaml at .aws/workflows/${workflowDef.name}.yaml
new Workflow(blueprint, repo, workflowDef);

// can also pass in any object and have it rendered as a yaml. This is unsafe and may
  not produce a valid workflow
new Workflow(blueprint, repo, {... some object ...});
```

환경에 연결

많은 워크플로가 AWS 계정 연결에서 실행되어야 합니다. 워크플로는 작업이 계정 및 역할 이름 사양이 있는 환경에 연결되도록 허용하여 이를 처리합니다.

```
import { convertToWorkflowEnvironment } from '@amazon-codecatalyst/codecatalyst-workflows'

const myEnv = new Environment(...);

// can be passed into a workflow constructor
const workflowEnvironment = convertToWorkflowEnvironment(myEnv);

// add a build action
workflowBuilder.addAction({
  ...
```

```
// add an account connection to the workflow
environment: convertToWorkflowEnvironment(myEnv),
});
```

블루프린트에 개발 환경 구성 요소 추가

관리형 개발 환경(MDE)은 CodeCatalyst에서 MDE Workspaces를 생성하고 설정하는 데 사용됩니다. 구성 요소는 `devfile.yaml` 파일을 생성합니다. 자세한 내용은 [Devfile 소개](#) 및 [개발 환경용 리포지토리 devfile 편집](#) 섹션을 참조하세요.

```
new Workspace(this, repository, SampleWorkspaces.default);
```

Amazon CodeCatalyst 블루프린트 작업 영역 구성 요소를 가져오려면

다음은 `blueprint.ts` 파일에 추가합니다.

```
import {...} from '@amazon-codecatalyst/codecatalyst-workspaces'
```

블루프린트에 문제 구성 요소 추가

CodeCatalyst 에서 특성, 작업, 버그 및 프로젝트와 관련된 기타 작업을 모니터링할 수 있습니다. 각 작업은 문제라는 고유한 레코드에 보관되어 있습니다. 각 문제에는 검색, 그룹화 및 필터링할 수 있는 설명, 담당자, 상태 및 기타 속성이 있을 수 있습니다. 기본 보기를 사용하여 문제를 보거나 사용자 지정 필터링, 정렬 또는 그룹화로 고유한 보기를 생성할 수 있습니다. 문제와 관련된 개념에 대한 자세한 내용은 [문제 개념](#) 및 [CodeCatalyst 문제에 대한 할당량](#) 섹션을 참조하세요.

문제 구성 요소는 문제에 대한 JSON 표현을 생성합니다. 구성 요소는 ID 필드와 문제 정의를 입력으로 사용합니다.

Amazon CodeCatalyst 블루프린트를 가져오려면 구성 요소를 발행합니다.

다음은 `blueprint.ts` 파일에 추가합니다.

```
import {...} from '@amazon-codecatalyst/blueprint-component.issues'
```

주제

- [문제 구성 요소 예시](#)

문제 구성 요소 예시

문제 생성

```
import { Issue } from '@amazon-codecatalyst/blueprint-component.issues';
...
new Issue(this, 'myFirstIssue', {
  title: 'myFirstIssue',
  content: 'This is an example issue.',
});
```

우선 순위가 높은 문제 생성

```
import { Workflow } from '@amazon-codecatalyst/codecatalyst-workflows'
...
const repo = new SourceRepository
const blueprint = this;
const workflowDef = workflowBuilder.getDefinition()

// Creates a workflow.yaml at .aws/workflows/${workflowDef.name}.yaml
new Workflow(blueprint, repo, workflowDef);

// Can also pass in any object and have it rendered as a yaml. This is unsafe and may
  not produce a valid workflow
new Workflow(blueprint, repo, {... some object ...});
```

레이블을 사용하여 우선 순위가 낮은 문제 생성

```
import { Issue } from '@amazon-codecatalyst/blueprint-component.issues';
...
new Issue(this, 'myThirdIssue', {
  title: 'myThirdIssue',
  content: 'This is an example of a low priority issue with a label.',
  priority: 'LOW',
  labels: ['exampleLabel'],
});
```

블루프린트 도구 및 CLI 작업

[블루프린트 CLI](#)는 사용자 지정 블루프린트를 관리하고 작업할 수 있는 도구를 제공합니다.

주제

- [블루프린트 도구 작업](#)
- [이미지 업로드 도구](#)

블루프린트 도구 작업

블루프린트 도구로 작업하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 개발 환경을 재개합니다. 자세한 내용은 [개발 환경 재개](#) 섹션을 참조하세요.

개발 환경이 없는 경우 먼저 환경을 생성해야 합니다. 자세한 내용은 [개발 환경 생성](#) 섹션을 참조하세요.

3. 작동 중인 터미널에서 다음 명령을 실행하여 블루프린트 CLI를 설치합니다.

```
npm install -g @amazon-codecatalyst/blueprint-util.cli
```

4. blueprint.ts 파일에서 사용할 도구를 다음 형식으로 가져옵니다.

```
import { <tooling-function-name> } from '@amazon-codecatalyst/blueprint-util.cli/lib/<tooling-folder-name>/<tooling-file-name>;
```

Tip

[CodeCatalyst blueprints GitHub repository](#)에서 사용하려는 도구의 이름을 찾을 수 있습니다.

이미지 업로드 도구를 사용하려면 스크립트에 다음을 추가합니다.

```
import { uploadImagePublicly } from '@amazon-codecatalyst/blueprint-util.cli/lib/image-upload-tool/upload-image-to-aws';
```

예시

- 게시 함수를 사용하려면 스크립트에 다음을 추가합니다.

```
import { publish } from '@amazon-codecatalyst/blueprint-util.cli/lib/publish/publish';
```

- 이미지 업로드 도구를 사용하려면 스크립트에 다음을 추가합니다.

```
import { uploadImagePublicly } from '@amazon-codecatalyst/blueprint-util.cli/lib/
image-upload-tool/upload-image-to-aws';
```

5. 함수를 직접적으로 호출합니다.

예시:

- 게시 함수를 사용하려면 스크립트에 다음을 추가합니다.

```
await publish(logger, config.publishEndpoint, {<your publishing options>});
```

- 이미지 업로드 도구를 사용하려면 스크립트에 다음을 추가합니다.

```
const {imageUrl, imageName} = await uploadImagePublicly(logger, 'path/to/
image');
```

이미지 업로드 도구

이미지 업로드 도구를 사용하면 AWS 계정의 S3 버킷에 이미지를 업로드한 다음 CloudFront 뒤에 해당 이미지를 공개적으로 배포할 수 있습니다. 도구는 로컬 스토리지(및 선택적 버킷 이름)의 이미지 경로를 입력으로 가져와서 공개적으로 사용할 수 있는 이미지에 URL을 반환합니다. 자세한 내용은 [Amazon CloudFront란?](#) 및 [Amazon S3란?](#)을 참조하세요.

이미지 업로드 도구로 작업하려면

1. 블루프린트 SDK 및 샘플 블루프린트에 대한 액세스를 제공하는 [오픈 소스 블루프린트 GitHub 리포지토리](#)를 복제합니다. 작업 중인 터미널에서 다음 명령을 실행합니다.

```
git clone https://github.com/aws/codecatalyst-blueprints.git
```

2. 다음 명령을 실행하여 블루프린트 GitHub 리포지토리로 이동합니다.

```
cd codecatalyst-blueprints
```

3. 다음 명령을 실행하여 종속성을 설치합니다.

```
yarn && yarn build
```

4. 다음 명령을 실행하여 최신 블루프린트 CLI 버전이 설치되어 있는지 확인합니다.

```
yarn upgrade @amazon-codecatalyst/blueprint-util.cli
```

5. 이미지를 업로드하려는 S3 버킷을 사용하여 AWS 계정에 로그인합니다. 자세한 내용은 [AWS CLI 구성하기](#) 및 [AWS 명령줄 인터페이스를 통해 로그인하기](#)를 참조하세요.
6. CodeCatalyst 리포지토리의 루트에서 다음 명령을 실행하여 블루프린트 CLI를 사용하여 디렉터리로 이동합니다.

```
cd packages/utils/blueprint-cli
```

7. 다음 명령을 실행하여 S3 버킷에 이미지를 업로드합니다.

```
yarn blueprint upload-image-public <./path/to/your/image>
<optional:optional-bucket-name>
```

이미지의 URL이 생성됩니다. CloudFront 배포를 배포하는 데 시간이 다소 걸리므로 URL을 즉시 사용할 수 없습니다. 배포 상태를 확인하여 최신 배포 상태를 가져옵니다. 자세한 내용은 [배포 작업](#)을 참조하세요.

스냅샷 테스트를 통한 인터페이스 변경 사항 평가

블루프린트의 여러 구성에서 생성된 스냅샷 테스트가 지원됩니다.

블루프린트는 블루프린트 작성자가 제공한 구성에 대한 [스냅샷 테스트](#)를 지원합니다. 구성은 블루프린트 루트에 있는 defaults.json 파일 위에 병합되는 부분 재정의입니다. 스냅샷 테스트가 활성화되고 구성되면 빌드 및 테스트 프로세스에서 지정된 구성을 합성하고 합성된 출력이 참조 스냅샷에서 변경되지 않았는지 확인합니다. 스냅샷 테스트 코드를 보려면 [CodeCatalyst blueprints GitHub repository](#)를 참조하세요.

스냅샷 테스트 활성화

1. .projenrc.ts 파일에서 스냅샷을 생성하려는 파일로 입력 객체를 ProjenBlueprint로 업데이트합니다. 예:

```
{
  ....
  blueprintSnapshotConfiguration: {
    snapshotGlobs: ['**', '!environments/**', '!aws-account-to-environment/**'],
```

```
  },  
}
```

2. 블루프린트를 재합성하여 블루프린트 프로젝트에서 TypeScript 파일을 생성합니다. 소스 파일은 Projen에서 유지 관리 및 재생성하므로 편집하지 마세요. 다음 명령을 사용합니다.

```
yarn projen
```

3. `src/snapshot-configurations` 디렉터리로 이동하여 빈 객체가 있는 `default-config.json` 파일을 봅니다. 파일을 하나 이상의 자체 테스트 구성으로 업데이트하거나 교체합니다. 런 다음 각 테스트 구성은 프로젝트의 `defaults.json` 파일과 병합되어 합성되고 테스트할 때 스냅샷과 비교됩니다. 다음 명령을 사용하여 테스트합니다.

```
yarn test
```

테스트 명령을 처음 사용하면 다음 메시지가 표시됩니다. Snapshot Summary > NN snapshots written from 1 test suite 후속 테스트 실행은 합성된 출력이 스냅샷에서 변경되지 않았는지 확인하고 다음 메시지를 표시합니다. Snapshots: NN passed, NN total.

의도적으로 블루프린트를 변경하여 다른 출력을 생성한 경우 다음 명령을 실행하여 참조 스냅샷을 업데이트합니다.

```
yarn test:update
```

스냅샷은 각 실행 간에 합성된 출력이 일정할 것으로 기대합니다. 블루프린트가 다양한 파일을 생성하는 경우 스냅샷 테스트에서 해당 파일을 제외해야 합니다. ProjenBlueprint 입력 객체의 `blueprintSnapshotConfiguration` 객체를 업데이트하여 `snapshotGlobs` 속성을 추가합니다. `snapshotGlobs` 속성은 스냅샷 생성에 포함되거나 제외되는 파일을 결정하는 [globs](#) 배열입니다.

Note

`globs`의 기본 목록이 있습니다. 자체 목록을 지정하는 경우 기본 항목을 명시적으로 다시 가져와야 할 수 있습니다.

사용자 지정 블루프린트를 스페이스에 게시

사용자 지정 블루프린트를 스페이스의 블루프린트 카탈로그에 추가하려면 먼저 스페이스에 게시해야 합니다. 게시하기 전에 CodeCatalyst 콘솔에서 블루프린트를 볼 수도 있습니다. 미리 보기 버전 또는 일반 버전의 블루프린트를 게시할 수 있습니다.

Important

외부 소스의 블루프린트 패키지를 사용하려면, 해당 패키지로 인해 발생할 수 있는 위험을 고려하세요. 스페이스에 추가하는 사용자 지정 블루프린트와 해당 블루프린트가 생성하는 코드는 사용자의 책임입니다.

Important

사용자 지정 블루프린트를 CodeCatalyst 스페이스에 게시하려면, 스페이스에 스페이스 관리자 또는 파워 유저 역할이 있는 계정으로 로그인해야 합니다.

주제

- [사용자 지정 블루프린트의 미리 보기 버전 보기 및 게시](#)
- [사용자 지정 블루프린트의 일반 버전 보기 및 게시](#)
- [지정된 스페이스 및 프로젝트에 사용자 지정 블루프린트 게시 및 추가](#)

사용자 지정 블루프린트의 미리 보기 버전 보기 및 게시

사용자 지정 블루프린트의 미리 보기 버전을 스페이스의 블루프린트 카탈로그에 추가하려는 경우 스페이스에 게시할 수 있습니다. 이렇게 하면 미리 보기가 아닌 버전을 카탈로그에 추가하기 전에 사용자로서 블루프린트를 볼 수 있습니다. 미리 보기 버전을 사용하면 실제 버전을 사용하지 않고 게시할 수 있습니다. 예를 들어, 0.0.1 버전에서 작업하는 경우 미리 보기 버전을 게시하고 추가할 수 있으므로 두 번째 버전에 대한 새 업데이트를 게시하고 0.0.2로 추가할 수 있습니다.

변경한 후 package.json 파일을 실행하여 사용자 지정 블루프린트 패키지를 다시 빌드하고 변경 내용을 미리 봅니다.

사용자 지정 블루프린트의 미리 보기 버전을 보고 게시하려면

1. 개발 환경을 재개합니다. 자세한 내용은 [개발 환경 재개](#) 섹션을 참조하세요.
2. 개발 환경에서 작업 중인 터미널을 엽니다.
3. (선택 사항) 아직 설치하지 않은 경우 작동 중인 터미널에서 프로젝트에 필요한 종속성을 설치합니다. 다음 명령을 사용합니다.

```
yarn
```

4. (선택 사항) `.projenrc.ts` 파일을 변경한 경우, 블루프린트를 빌드하고 미리 보기 전에 프로젝트의 구성을 다시 생성합니다. 다음 명령을 사용합니다.

```
yarn projen
```

5. 다음 명령을 사용하여 사용자 지정 블루프린트를 다시 빌드하고 미리 봅니다. 다음 명령을 사용합니다.

```
yarn blueprint:preview
```

제공된 See this blueprint at: 링크로 이동하여 사용자 지정 블루프린트를 미리 봅니다. 텍스트를 포함한 UI가 구성에 따라 예상대로 표시되는지 확인합니다. 사용자 지정 블루프린트를 변경하려면 `blueprint.ts` 파일을 편집하고 블루프린트를 재합성한 다음 미리 보기 버전을 다시 게시할 수 있습니다. 자세한 내용은 [재합성](#) 섹션을 참조하세요.

6. (선택 사항) 사용자 지정 블루프린트의 미리 보기 버전을 게시한 다음 스페이스의 블루프린트 카탈로그에 추가할 수 있습니다. Enable version *[preview version number]* at: 링크로 이동하여 미리 보기 버전을 스페이스에 게시합니다.

CodeCatalyst에서 프로젝트를 생성할 필요 없이 프로젝트 생성을 에뮬레이션할 수 있습니다. 프로젝트를 재합성하려면 다음 명령을 사용합니다.

```
yarn blueprint:synth
```

`synth/synth.[options-name]/proposed-bundle/` 폴더에 블루프린트가 생성됩니다. 자세한 내용은 [합성](#) 섹션을 참조하세요.

사용자 지정 블루프린트를 업데이트하는 경우 대신 다음 명령을 사용하여 프로젝트를 재합성합니다.

```
yarn blueprint:resynth
```

synth/synth.*[options-name]*/proposed-bundle/ 폴더에 블루프린트가 생성됩니다. 자세한 내용은 [재합성](#) 섹션을 참조하세요.

미리 보기 버전을 게시한 후 블루프린트를 추가하여 스페이스 멤버가 해당 블루프린트를 사용하여 새 프로젝트를 생성하거나 기존 프로젝트에 추가할 수 있습니다. 자세한 내용은 [스페이스 블루프린트 카탈로그에 사용자 지정 블루프린트 추가](#) 섹션을 참조하세요.

사용자 지정 블루프린트의 일반 버전 보기 및 게시

사용자 지정 블루프린트를 개발하고 미리 본 후에는 스페이스의 블루프린트 카탈로그에 추가하려는 해당 새 버전을 보고 게시할 수 있습니다. 프로젝트를 생성할 때 생성된 릴리스 워크플로는 푸시된 변경 사항을 자동으로 게시합니다. 블루프린트를 생성할 때 워크플로 생성을 끄면 스페이스의 블루프린트 카탈로그에 해당 블루프린트를 자동으로 추가할 수 없습니다. yarn 명령을 실행한 후에도 사용자 지정 블루프린트를 스페이스에 게시할 수 있습니다.

사용자 지정 블루프린트를 보고 게시하려면

1. 개발 환경을 재개합니다. 자세한 내용은 [개발 환경 재개](#) 섹션을 참조하세요.
2. 개발 환경에서 작업 중인 터미널을 엽니다.
3. • 블루프린트를 생성할 때 릴리스 워크플로 생성을 옵트아웃한 경우 다음 명령을 사용합니다.

```
yarn blueprint:release
```

제공된 See this blueprint at: 링크로 이동하여 사용자 지정 블루프린트를 볼 수 있습니다.

사용자 지정 블루프린트의 업데이트된 버전을 게시하면 스페이스의 블루프린트 카탈로그에 추가할 수 있습니다. Enable version *[release version number]* at: 링크로 이동하여 최신 버전을 스페이스에 게시합니다.

- 블루프린트를 생성할 때 릴리스 워크플로에 옵트인한 경우 변경 사항을 푸시하면 최신 블루프린트 버전이 자동으로 게시됩니다. 다음 명령을 사용합니다.

```
git add .
```

```
git commit -m "commit message"
```

```
git push
```

일반 버전을 게시한 후 블루프린트를 추가하여 스페이스 멤버가 해당 블루프린트를 사용하여 새 프로젝트를 생성하거나 기존 프로젝트에 추가할 수 있습니다. 자세한 내용은 [스페이스 블루프린트 카탈로그에 사용자 지정 블루프린트 추가](#) 섹션을 참조하세요.

지정된 스페이스 및 프로젝트에 사용자 지정 블루프린트 게시 및 추가

기본적으로 `blueprint:preview` 및 `blueprint:release` 명령은 블루프린트를 생성한 CodeCatalyst 스페이스에 게시됩니다. 엔터프라이즈 스페이스가 여러 개 있는 경우, 해당 스페이스에도 동일한 블루프린트를 미리 보고 게시할 수 있습니다. 다른 스페이스의 기존 프로젝트에 블루프린트를 추가할 수도 있습니다.

지정된 스페이스에 사용자 지정 블루프린트를 게시하거나 추가하려면

1. 개발 환경을 재개합니다. 자세한 내용은 [개발 환경 재개](#) 섹션을 참조하세요.
2. 개발 환경에서 작업 중인 터미널을 엽니다.
3. (선택 사항) 아직 설치하지 않은 경우 프로젝트에 필요한 종속성을 설치합니다. 다음 명령을 사용합니다.

```
yarn
```

4. `--space` 태그를 사용하여 미리 보기 또는 일반 버전을 지정된 스페이스에 게시합니다. 예:

- ```
yarn blueprint:preview --space my-awesome-space # publishes under a "preview" version tag to 'my-awesome-space'
```

출력 예시:

```
Enable version 0.0.1-preview.0 at: https://codecatalyst.aws/spaces/my-awesome-space/blueprints
Blueprint applied to [NEW]: https://codecatalyst.aws/spaces/my-awesome-space/blueprints/%40amazon-codecatalyst%2Fmyspace.my-blueprint/publishers/1524817d-a69b-4abe-89a0-0e4a9a6c53b2/versions/0.0.1-preview.0/projects/create
```

- ```
yarn blueprint:release --space my-awesome-space # publishes normal version to 'my-awesome-space'
```

출력 예시:

```
Enable version 0.0.1 at: https://codecatalyst.aws/spaces/my-awesome-space/
blueprints
Blueprint applied to [NEW]: https://codecatalyst.aws/spaces/my-awesome-space/
blueprints/%40amazon-codecatalyst%2Fmyspace.my-blueprint/publishers/1524817d-
a69b-4abe-89a0-0e4a9a6c53b2/versions/0.0.1/projects/create
```

--project를 사용하여 지정된 스페이스의 기존 프로젝트에 사용자 지정 블루프린트의 미리 보기 버전을 추가합니다. 예:

```
yarn blueprint:preview --space my-awesome-space --project my-project # previews
blueprint application to an existing project
```

출력 예시:

```
Enable version 0.0.1-preview.1 at: https://codecatalyst.aws/spaces/my-awesome-
space/blueprints
Blueprint applied to [my-project]: https://codecatalyst.aws/spaces/my-awesome-
space/projects/my-project/blueprints/%40amazon-codecatalyst%2FmySpace.my-blueprint/
publishers/1524817d-a69b-4abe-89a0-0e4a9a6c53b2/versions/0.0.1-preview.1/add
```

사용자 지정 블루프린트에 대한 게시 권한 설정

기본적으로 프로젝트 생성 중에 워크플로 릴리스가 생성된 경우 사용자 지정 블루프린트의 권한이 활성화됩니다. 게시 권한이 활성화되면 스페이스에 블루프린트를 게시할 수 있습니다. 블루프린트를 게시할 수 없도록 권한을 비활성화할 수 있습니다. 권한이 비활성화되면, 블루프린트 생성이 진행되지 않는 동안 릴리스 워크플로가 생성됩니다. 블루프린트의 권한을 활성화하지 않으면 블루프린트에 대한 새 변경 사항을 게시할 수 없습니다.

Important

사용자 지정 블루프린트 프로젝트의 게시 권한을 활성화하거나 비활성화하려면, 스페이스에 스페이스 관리자 또는 파워 유저 역할이 있는 계정으로 로그인해야 합니다.

블루프린트 프로젝트의 게시 권한을 설정하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.

2. CodeCatalyst 콘솔에서 사용자 지정 블루프린트의 게시 권한을 관리할 스페이스로 이동합니다.
3. 스페이스 대시보드에서 설정 탭을 선택한 다음 블루프린트를 선택합니다.
4. 프로젝트 게시 권한 탭을 선택하여 모든 스페이스의 블루프린트에 대한 게시 권한을 봅니다.
5. 관리하려는 블루프린트를 선택한 다음, 활성화 또는 비활성화를 선택하여 게시 권한을 변경합니다. 권한을 활성화하는 경우, 권한 변경 세부 정보를 검토한 다음, 블루프린트 게시 활성화를 선택하여 변경 사항을 확인합니다.

스페이스 블루프린트 카탈로그에 사용자 지정 블루프린트 추가

사용자 지정 블루프린트를 스페이스에 게시한 후 스페이스의 블루프린트 카탈로그에 추가할 수 있습니다. 사용자 지정 블루프린트를 CodeCatalyst 스페이스의 블루프린트 카탈로그에 추가하면 모든 스페이스 멤버가 프로젝트를 생성하거나 기존 프로젝트에 추가할 때 블루프린트를 사용할 수 있습니다. 스페이스의 블루프린트 카탈로그에 사용자 지정 블루프린트를 추가하기 전에 블루프린트의 게시 권한을 활성화해야 합니다. 워크플로 릴리스 생성을 선택한 경우 게시 권한이 기본적으로 활성화됩니다. 자세한 내용은 [사용자 지정 블루프린트에 대한 게시 권한 설정](#) 및 [사용자 지정 블루프린트를 스페이스에 게시](#) 섹션을 참조하세요.

Important

CodeCatalyst 스페이스의 블루프린트 카탈로그에 사용자 지정 블루프린트를 추가하려면 스페이스에 스페이스 관리자 또는 파워 유저 역할이 있는 계정으로 로그인해야 합니다.

스페이스의 블루프린트 카탈로그에 블루프린트를 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 블루프린트는 소스 리포지토리의 기본 브랜치에서만 추가할 수 있습니다. 기능 브랜치에서 블루프린트를 개발한 경우 기능 브랜치를 기본 브랜치의 변경 사항과 병합합니다. 풀 요청을 생성하여 변경 사항을 기본 브랜치에 병합합니다. 자세한 내용은 [Amazon CodeCatalyst에서 풀 요청을 사용하여 코드 검토](#) 섹션을 참조하세요.
3. CodeCatalyst 콘솔에서 사용자 지정 블루프린트를 사용하여 스페이스 대시보드로 이동합니다.
4. 스페이스 대시보드에서 설정 탭을 선택한 다음 블루프린트를 선택합니다.
5. 추가할 블루프린트 이름을 선택한 다음 카탈로그에 추가를 선택합니다. 버전이 두 개 이상인 경우 카탈로그 버전드롭다운 메뉴에서 버전을 선택합니다.
6. 저장을 선택합니다.

사용자 지정 블루프린트의 카탈로그 버전 변경

블루프린트 작성자는 스페이스의 블루프린트 카탈로그에 게시할 버전을 관리할 수 있습니다. 블루프린트의 카탈로그 버전을 변경해도 다른 블루프린트 버전을 사용하는 프로젝트에는 영향을 주지 않습니다.

Important

Amazon CodeCatalyst 스페이스의 블루프린트 카탈로그에서 사용자 지정 블루프린트의 카탈로그 버전을 변경하려면, 스페이스에 스페이스 관리자 또는 파워 유저 역할이 있는 계정으로 로그인해야 합니다.

사용자 지정 블루프린트 버전을 관리하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 사용자 지정 블루프린트 버전을 변경할 스페이스로 이동합니다.
3. 스페이스 대시보드에서 설정 탭을 선택한 다음 블루프린트를 선택합니다.
4. 스페이스 블루프린트 테이블에서 관리하려는 사용자 지정 블루프린트의 라디오 버튼을 선택합니다.
5. 카탈로그 버전 만들기를 선택한 다음 카탈로그 버전 드롭다운 메뉴에서 버전을 선택합니다.
6. 저장을 선택합니다.

사용자 지정 블루프린트의 세부 정보, 버전 및 프로젝트 보기

블루프린트의 세부 정보, 버전, 블루프린트로 만들거나 추가한 프로젝트를 포함하여 스페이스에 게시된 사용자 지정 블루프린트를 볼 수 있습니다.

주제

- [스페이스의 사용자 지정 블루프린트 보기](#)
- [사용자 지정 블루프린트로 생성하거나 추가한 프로젝트 보기](#)

스페이스의 사용자 지정 블루프린트 보기

스페이스의 사용자 지정 블루프린트 보기

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 사용자 지정 블루프린트를 보려는 공간으로 이동합니다.
3. 스페이스 대시보드에서 설정 탭을 선택한 다음 블루프린트를 선택하여 스페이스 블루프린트를 확인합니다. 다음 세부 정보가 테이블에 표시됩니다.
 - 이름 - 사용자 지정 블루프린트의 이름입니다.
 - 카탈로그 상태 - 사용자 지정 블루프린트가 스페이스의 블루프린트 카탈로그에 게시되는지 여부입니다.
 - 최신 버전 - 사용자 지정 블루프린트의 최신 버전입니다.
 - 최신 수정 - 스페이스 블루프린트가 마지막으로 업데이트된 날짜입니다.

사용자 지정 블루프린트로 생성하거나 추가한 프로젝트 보기

사용자 지정 블루프린트로 생성하거나 추가한 프로젝트 보기

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 사용자 지정 블루프린트를 보려는 공간으로 이동합니다.
3. 스페이스 대시보드에서 설정 탭을 선택한 다음 블루프린트를 선택합니다.
4. 스페이스 블루프린트 테이블에서 사용자 지정 블루프린트의 이름을 선택하면 블루프린트를 사용하는 프로젝트와 블루프린트를 사용하지 않는 프로젝트 테이블을 볼 수 있습니다.

스페이스 블루프린트 카탈로그에서 사용자 지정 블루프린트 제거

사용자 지정 블루프린트로 새 프로젝트를 생성하지 않거나 사용자 지정 블루프린트를 기존 프로젝트에 더 이상 적용하지 않는다면, 사용자 지정 블루프린트를 스페이스의 블루프린트 카탈로그에서 제거할 수 있습니다.

Note

스페이스의 블루프린트 카탈로그에서 사용자 지정 블루프린트를 제거해도, 블루프린트에서 생성된 프로젝트나 블루프린트를 적용한 프로젝트에는 영향을 미치지 않습니다. 블루프린트의 리소스는 프로젝트에서 제거되지 않습니다.

⚠ Important

CodeCatalyst 스페이스의 블루프린트 카탈로그에서 사용자 지정 블루프린트를 제거하려면, 스페이스에 스페이스 관리자 또는 파워 유저 역할이 있는 계정으로 로그인해야 합니다.

스페이스 블루프린트 카탈로그에서 사용자 지정 블루프린트를 제거하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 사용자 지정 블루프린트를 사용하여 스페이스 대시보드로 이동합니다.
3. 스페이스 대시보드에서 설정 탭을 선택한 다음 블루프린트를 선택합니다.
4. 제거할 블루프린트 이름을 선택한 다음 카탈로그에서 블루프린트 제거를 선택합니다.

게시된 사용자 지정 블루프린트 또는 버전 삭제

Amazon CodeCatalyst 스페이스에서 사용자 지정 블루프린트 버전 또는 블루프린트 자체를 삭제하면 블루프린트 프로젝트 또는 블루프린트 버전의 리소스에 대한 모든 액세스 권한이 제거됩니다. 블루프린트 버전 또는 블루프린트를 삭제하면 프로젝트 멤버가 프로젝트 리소스에 액세스할 수 없으며 타사 소스 리포지토리에서 프롬프트하는 워크플로가 중지됩니다.

i Note

블루프린트를 삭제해도 블루프린트가 적용된 프로젝트에는 영향을 주지 않습니다. 블루프린트의 리소스는 프로젝트에서 제거되지 않습니다.

블루프린트 버전이 스페이스의 블루프린트 카탈로그에 게시된 경우 게시된 버전을 삭제하기 전에 카탈로그의 새 버전을 선택합니다.

⚠ Important

스페이스에서 게시된 사용자 지정 블루프린트 또는 사용자 지정 블루프린트의 카탈로그 버전을 삭제하려면 스페이스에 스페이스 관리자 또는 파워 유저 역할이 있는 계정으로 로그인해야 합니다.

사용자 지정 블루프린트의 카탈로그 버전을 삭제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 사용자 지정 블루프린트의 카탈로그 버전을 삭제할 스페이스로 이동합니다.
3. 스페이스 대시보드에서 설정 탭을 선택한 다음 블루프린트를 선택합니다.
4. 삭제하려는 카탈로그 버전이 있는 블루프린트의 이름을 선택합니다.
5. 삭제하려는 카탈로그 버전의 라디오 버튼을 선택한 다음 버전 삭제를 선택합니다.
6. 세부 정보를 검토한 다음 새 블루프린트 카탈로그 버전 선택 드롭다운 메뉴에서 다른 블루프린트 버전을 선택합니다.
7. delete를 입력하여 블루프린트 카탈로그 버전의 삭제를 확인합니다.
8. 삭제를 선택합니다.

블루프린트 버전이 스페이스의 블루프린트 카탈로그에 없는 경우 새 버전을 선택하지 않고 버전을 삭제할 수 있습니다.

사용자 지정 블루프린트 버전을 삭제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 사용자 지정 블루프린트 버전을 삭제할 스페이스로 이동합니다.
3. 스페이스 대시보드에서 설정 탭을 선택한 다음 블루프린트를 선택합니다.
4. 삭제하려는 버전이 있는 블루프린트의 이름을 선택합니다.
5. 삭제하려는 버전의 라디오 버튼을 선택한 다음 버전 삭제를 선택합니다.
6. delete를 입력하여 블루프린트 버전 삭제를 확인합니다.
7. 삭제를 선택합니다.

스페이스의 블루프린트 카탈로그에서 블루프린트를 삭제하면 블루프린트의 모든 버전이 삭제됩니다. 블루프린트를 사용하는 스페이스의 프로젝트는 삭제의 영향을 받지 않습니다.

사용자 지정 블루프린트 버전을 삭제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 콘솔에서 사용자 지정 블루프린트를 삭제할 스페이스로 이동합니다.
3. 스페이스 대시보드에서 설정 탭을 선택한 다음 블루프린트를 선택합니다.
4. 스페이스스 블루프린트 테이블에서 삭제할 사용자 지정 블루프린트의 라디오 버튼을 선택한 다음 블루프린트 삭제를 선택합니다.

5. `delete`를 입력하여 사용자 지정 블루프린트 삭제를 확인합니다.
6. 삭제를 선택합니다.

종속성, 불일치 및 도구 처리

잘못된 종속성 관리는 사용자 지정 블루프린트를 사용하는 사용자에게 빌드 실패 및 런타임 문제를 초래할 수 있습니다. 오래된 도구 및 구성 요소는 블루프린트 사용자가 최신 기능 및 버그 수정에 액세스하지 못하게 할 수 있습니다. 종속성을 관리하고, 종속성 불일치를 처리하고, 도구 및 구성 요소를 업그레이드하여 모든 종속성이 동일한 구성 요소 버전에 의존하고 구성 요소가 동기화되도록 할 수 있습니다.

주제

- [종속성 추가](#)
- [종속성 유형 불일치 처리](#)
- [Yarn 및 npm 사용](#)
- [도구 및 구성 요소 업그레이드](#)

종속성 추가

블루프린트 작성자는 `@amazon-codecatalyst/blueprint-component.environments` 같은 패키지를 블루프린트에 추가해야 할 수 있습니다. 해당 패키지로 `projen.ts` 파일을 업데이트한 다음 [Projen](#)을 사용하여 프로젝트 구성을 다시 생성해야 합니다. Projen은 각 블루프린트 코드베이스의 프로젝트 모델 역할을 하며, 모델이 구성 파일을 렌더링하는 방식을 변경하여 이전 버전과 호환되는 도구 업데이트를 푸시할 수 있는 기능을 제공합니다. `package.json` 파일은 Projen 모델이 부분적으로 소유한 파일입니다. Projen은 `package.json` 파일에 포함된 종속성 버전을 인정하지만 다른 옵션은 모델에서 시작해야 합니다.

종속성을 추가하고 **`projenrc.ts`** 파일을 업데이트하려면

1. `projen.ts` 파일에서 `deps` 섹션으로 이동합니다.
2. 블루프린트에 사용할 종속성을 추가합니다.
3. 다음 명령을 사용하여 프로젝트의 구성을 다시 생성합니다.

```
yarn projen && yarn
```

종속성 유형 불일치 처리

[Yarn](#) 업데이트 후 리포지토리 파라미터와 관련하여 다음과 같은 오류가 발생할 수 있습니다.

```
Type 'SourceRepository' is missing the following properties from type
'SourceRepository': synthesisSteps, addSynthesisStep
```

이 오류는 한 구성 요소가 최신 버전의 다른 구성 요소에 의존할 때 발생하는 종속성 불일치로 인해 발생하지만, 종속 구성 요소는 이전 버전에 고정됩니다. 모든 구성 요소가 동일한 버전에 의존하도록 하여 해당 구성 요소 간에 버전이 동기화되도록 하여 오류를 수정할 수 있습니다. 버전을 처리하는 방법이 확실하지 않다면 블루프린트가 포함된 패키지를 동일한 최신 버전(0.0.x)으로 유지하는 것이 가장 좋습니다. 다음 예시에서는 모든 종속 항목이 동일한 버전에 의존하도록 package.json 파일을 구성하는 방법을 보여줍니다.

```
...
"@caws-blueprint-component/caws-environments": "^0.1.12345",
"@caws-blueprint-component/caws-source-repositories": "^0.1.12345",
"@caws-blueprint-component/caws-workflows": "^0.1.12345",
"@caws-blueprint-component/caws-workspaces": "^0.1.12345",
"@caws-blueprint-util/blueprint-utils": "^0.1.12345",
...
"@caws-blueprint/blueprints.blueprint": "*",
```

모든 종속성에 대해 버전을 구성한 후 다음 명령을 사용합니다.

```
yarn install
```

Yarn 및 npm 사용

블루프린트는 도구에 [Yarn](#)을 사용합니다. [npm](#)과 Yarn을 사용하면 종속성 트리가 각각 다르게 해결되므로 도구 문제가 발생합니다. 이러한 문제를 방지하려면 Yarn만 사용하는 것이 가장 좋습니다.

npm을 사용하여 실수로 종속성을 설치한 경우 생성된 package-lock.json 파일을 제거하고 필요한 종속성으로 .projenrc.ts 파일이 업데이트되었는지 확인할 수 있습니다. Projen을 사용하여 프로젝트의 구성을 재생성합니다.

다음을 사용하여 모델에서 재생성합니다.

```
yarn projen
```

.projenrc.ts 파일이 필요한 종속 항목으로 업데이트되었는지 확인한 후 다음 명령을 사용합니다.

```
yarn
```

도구 및 구성 요소 업그레이드

경우에 따라 도구 및 구성 요소를 업그레이드하여 사용 가능한 새 기능을 가져오려고 할 수 있습니다. 버전을 처리하는 방법에 대해 확신이 서지 않는 한 모든 구성 요소를 동일한 버전으로 유지하는 것이 좋습니다. 버전은 구성 요소 간에 동기화되므로 모든 구성 요소에 대해 동일한 버전이 구성 요소 간의 적절한 종속성을 보장합니다.

Yarn 작업 영역 monorepo 사용

다음 명령을 사용하여 사용자 지정 블루프린트 리포지토리의 루트에서 유틸리티 및 구성 요소를 업그레이드합니다.

```
yarn upgrade @amazon-codecatalyst/*
```

monorepo를 사용하지 않는 경우 다음 명령을 사용합니다.

```
yarn upgrade -pattern @amazon-codecatalyst/*
```

도구 및 구성 요소를 업그레이드하는 데 사용할 수 있는 기타 옵션:

- 최신 버전을 확인하려면 @caws-blueprint-component/<some-component>를 사용합니다.
- package.json 파일에서 버전을 설정하고 yarn 명령을 사용하여 수동으로 최신 버전으로 높입니다. 모든 구성 요소와 유틸리티는 버전이 동일해야 합니다.

기여

블루프린트 소프트웨어 개발 키트(SDK)는 기여할 수 있는 오픈 소스 라이브러리입니다. 기여자로서 기여 지침, 피드백 및 결함을 고려합니다. 자세한 내용은 [블루프린트 GitHub 리포지토리](#)를 참조하세요.

CodeCatalyst의 블루프린트 할당량

다음 표에서는 Amazon CodeCatalyst의 블루프린트에 대한 할당량 및 제한을 설명합니다. Amazon CodeCatalyst의 할당량에 대한 자세한 내용은 [CodeCatalyst 할당량](#) 섹션을 참조하세요.

CodeCatalyst 프로젝트당 적용된 최대 블루프린트 수	100

CodeCatalyst의 소스 리포지토리로 코드 저장 및 협업

CodeCatalyst 소스 리포지토리는 Amazon CodeCatalyst에서 호스팅되는 Git 리포지토리입니다. CodeCatalyst의 소스 리포지토리를 사용하여 프로젝트의 자산을 안전하게 저장, 버전 관리 및 관리할 수 있습니다.

CodeCatalyst 리포지토리의 자산에는 다음이 포함될 수 있습니다.

- 문서
- 소스 코드
- 이진 파일

또한 CodeCatalyst는 프로젝트의 소스 리포지토리를 사용하여 워크플로 구성 파일과 같은 프로젝트의 구성 정보를 저장합니다.

CodeCatalyst 프로젝트에 하나 이상의 소스 리포지토리를 가질 수 있습니다. 예를 들어 프론트엔드 소스 코드, 백엔드 소스 코드, 유틸리티 및 문서에 대한 별도의 소스 리포지토리가 필요할 수 있습니다.

다음은 CodeCatalyst에서 소스 리포지토리, 풀 요청 및 개발 환경의 코드 작업을 위한 한 가지 가능한 워크플로입니다.

Mary Major는 블루프린트를 사용하여 CodeCatalyst에서 웹 애플리케이션 프로젝트를 생성합니다. 이 블루프린트는 샘플 코드가 포함된 소스 리포지토리를 생성합니다. 그녀는 친구인 Li Juan, Saanvi Sarkar, Jorge Souza를 초대하여 함께 프로젝트를 진행합니다. Li Juan은 소스 리포지토리의 샘플 코드를 보고 코드에 테스트를 추가하기 위해 몇 가지 빠른 변경을하기로 결정합니다. Li는 개발 환경을 생성하고, IDE AWS Cloud9로 선택하고, 새 브랜치, `### ###` 지정합니다. 개발 환경이 열립니다. Li는 코드를 빠르게 추가한 다음, 변경 사항이 포함된 브랜치를 CodeCatalyst의 소스 리포지토리에 커밋하고 푸시합니다. 그런 다음 Li는 풀 요청을 생성합니다. 풀 요청 작성의 일환으로, Li는 코드 검토를 위해 Jorge Souza와 Saanvi Sarkar를 검토자로 추가했습니다.

코드를 검토하는 동안 Jorge Souza는 자신이 작업 중인 앱의 프로토타입이 포함된 GitHub에 자체 프로젝트 리포지토리가 있음을 기억합니다. 그는 Mary Major에게 GitHub 리포지토리를 프로젝트에 추가 소스 리포지토리로 연결할 수 있는 확장 프로그램을 설치하고 구성하도록 요청합니다. Mary는 GitHub의 리포지토리를 검토하고 Jorge와 협력하여 GitHub 확장 프로그램을 구성하여 GitHub 리포지토리를 프로젝트의 추가 소스 리포지토리로 연결할 수 있도록 합니다.

CodeCatalyst 소스 리포지토리는 Git의 표준 기능을 지원하며 기존 Git 기반 도구에서 작동합니다. Git 클라이언트 또는 통합 개발 환경(IDE)에서 소스 리포지토리를 복제하고 작업할 때 애플리케이션별 암

호로 개인 액세스 토큰(PAT)을 생성하여 사용할 수 있습니다. 이러한 PAT는 CodeCatalyst 사용자 ID와 연결됩니다. 자세한 내용은 [개인 액세스 토큰을 사용하여 사용자 리포지토리 액세스 권한 부여](#) 섹션을 참조하세요.

CodeCatalyst 소스 리포지토리는 풀 요청을 지원합니다. 한 브랜치에서 다른 브랜치로 코드를 병합하기 전에 사용자와 다른 프로젝트 멤버가 코드 변경 사항을 검토하고 설명을 추가할 수 있는 간단한 방법입니다. CodeCatalyst 콘솔에서 변경 사항을 보고 코드 줄에 설명을 추가할 수 있습니다.

CodeCatalyst 소스 리포지토리의 브랜치로 푸시하면 워크플로에서 자동으로 실행을 시작할 수 있으며, 여기에서 변경 사항을 빌드, 테스트 및 배포할 수 있습니다. 프로젝트 템플릿을 사용하여 프로젝트의 일부로 소스 리포지토리를 만든 경우, 프로젝트의 일부로 하나 이상의 워크플로가 구성됩니다. 언제든지 리포지토리에 대한 워크플로를 추가할 수 있습니다. 프로젝트의 워크플로에 대한 YAML 구성 파일은 해당 워크플로에 대한 소스 작업에서 구성된 소스 리포지토리에 저장됩니다. 자세한 내용은 [워크플로 시작하기](#) 섹션을 참조하세요.

주제

- [프라이빗 리포지토리 개념](#)
- [소스 리포지토리 작업 설정](#)
- [CodeCatalyst 소스 리포지토리 및 단일 페이지 애플리케이션 블루프린트 시작하기](#)
- [CodeCatalyst에서 프로젝트의 리포지토리에 소스 코드 저장](#)
- [Amazon CodeCatalyst의 브랜치를 사용하여 소스 코드 작업 구성](#)
- [Amazon CodeCatalyst에서 소스 코드 파일 관리](#)
- [Amazon CodeCatalyst에서 풀 요청을 사용하여 코드 검토](#)
- [Amazon CodeCatalyst에서 커밋을 사용한 소스 코드 변경 사항 이해](#)
- [CodeCatalyst의 소스 리포지토리 할당량](#)

프라이빗 리포지토리 개념

다음은 CodeCatalyst 소스 리포지토리를 사용할 때 알아야 할 몇 가지 개념입니다.

주제

- [Projects](#)
- [소스 리포지토리](#)
- [개발 환경](#)
- [개인 액세스 토큰\(PAT\)](#)

- [브랜치](#)
- [기본 브랜치](#)
- [커밋](#)
- [풀 요청](#)
- [개정](#)
- [워크플로](#)

Projects

프로젝트는 개발 팀과 작업을 지원하는 CodeCatalyst의 협업 스페이스입니다. 프로젝트를 만든 후에는 사용자와 리소스를 추가, 업데이트 또는 제거하고, 프로젝트 대시보드를 사용자 지정하고, 팀의 작업 진행 상황을 모니터링할 수 있습니다. 스페이스 내에 여러 프로젝트를 포함할 수 있습니다.

소스 리포지토리는 스페이스에서 생성하거나 연결하는 프로젝트에만 해당됩니다. 프로젝트 간에 리포지토리를 공유할 수 없으며 리포지토리를 한 스페이스의 둘 이상의 프로젝트에 연결할 수 없습니다. 프로젝트에서 기여자 또는 프로젝트 관리자 역할을 가진 사용자는 해당 역할에 부여된 권한에 따라 해당 프로젝트와 연결된 소스 리포지토리와 상호 작용할 수 있습니다. 자세한 내용은 [사용자 역할로 액세스 권한 부여](#) 섹션을 참조하세요.

소스 리포지토리

소스 리포지토리는 프로젝트에 대한 코드 및 파일을 안전하게 저장합니다. 또한 파일의 버전 기록도 저장합니다. 기본적으로 소스 리포지토리는 CodeCatalyst 프로젝트의 다른 사용자와 공유됩니다. 프로젝트에 대한 소스 리포지토리가 두 개 이상 있을 수 있습니다. CodeCatalyst의 프로젝트에 대한 소스 리포지토리를 생성하거나, 설치된 확장 프로그램에서 해당 서비스가 지원되는 경우 다른 서비스에서 호스팅하는 기존 소스 리포지토리를 연결하도록 선택할 수 있습니다. 예를 들어 GitHub 리포지토리 확장 프로그램을 설치한 후 GitHub 리포지토리를 프로젝트에 연결할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 프로젝트의 리포지토리에 소스 코드 저장](#) 및 [빠른 시작: CodeCatalyst에서 확장 프로그램 설치, 공급자 연결 및 리소스 연결](#) 섹션을 참조하세요.

개발 환경

개발 환경은 프로젝트의 소스 리포지토리에 저장된 코드를 빠르게 작업하기 위해 CodeCatalyst에서 사용할 수 있는 클라우드 기반 개발 환경입니다. 개발 환경에 포함된 프로젝트 도구 및 애플리케이션 라이브러리는 프로젝트의 소스 리포지토리에 있는 devfile로 정의됩니다. 소스 리포지토리에 devfile이 없는 경우 기본 devfile이 자동으로 적용됩니다. 기본 devfile에는 가장 자주 사용되는 프로그래밍 언어

및 프레임워크를 위한 도구가 포함되어 있습니다. 기본적으로 개발 환경은 2코어 프로세서, 4GB RAM, 16GiB 영구 저장소를 갖도록 구성됩니다.

소스 리포지토리의 기존 브랜치를 개발 환경으로 복제하거나 개발 환경 생성의 일부로 새 브랜치를 만들도록 선택할 수 있습니다.

개인 액세스 토큰(PAT)

개인 액세스 토큰(PAT)은 암호와 유사합니다. CodeCatalyst의 모든 스페이스와 프로젝트에서 사용할 수 있도록 사용자 ID와 연결됩니다. PAT를 사용하여 통합 개발 환경(IDE) 및 Git 기반 소스 리포지토리가 포함된 CodeCatalyst 리소스에 액세스합니다. PAT는 CodeCatalyst에서 사용자를 나타내며 사용자 설정에서 이를 관리할 수 있습니다. 사용자는 둘 이상의 PAT를 가질 수 있습니다. 개인 액세스 토큰은 한 번만 표시됩니다. 가장 좋은 방법은 로컬 컴퓨터에 안전하게 저장하는 것입니다. 기본적으로 PAT는 1년 후에 만료됩니다.

통합 개발 환경(IDE)으로 작업할 때 PAT는 Git 암호와 동일합니다. Git 리포지토리와 함께 작동하도록 IDE를 설정할 때 암호를 요청하면 PAT를 제공합니다. IDE를 Git 기반 리포지토리에 연결하는 방법에 대한 자세한 내용은 IDE 설명서를 참조하세요.

브랜치

브랜치는 Git 및 CodeCatalyst의 커밋에 대한 포인터 또는 참조입니다. 브랜치를 사용하여 작업을 구성할 수 있습니다. 예를 들어 브랜치를 사용하여 다른 브랜치에 있는 파일에 영향을 주지 않고 새 파일 또는 다른 버전의 파일에서 작업할 수 있습니다. 브랜치를 사용하면 새 기능을 개발하고 프로젝트 버전을 저장하는 등의 작업을 수행할 수 있습니다. 소스 리포지토리에는 브랜치가 하나 이상 있을 수 있습니다. 템플릿을 사용하여 프로젝트를 생성할 때 프로젝트에 대해 생성된 소스 리포지토리에는 main이라는 브랜치의 샘플 파일이 포함됩니다. main 브랜치는 리포지토리의 기본 브랜치입니다.

기본 브랜치

CodeCatalyst의 소스 리포지토리에는 생성 방식에 관계없이 기본 브랜치가 있습니다. 템플릿을 사용하여 프로젝트를 생성하도록 선택하면 해당 프로젝트에 대해 생성된 소스 리포지토리에는 샘플 코드, 워크플로 정의 및 기타 리소스 외에도 README.md 파일이 포함됩니다. 템플릿을 사용하지 않고 소스 리포지토리를 생성하는 경우 README.md 파일이 첫 번째 커밋으로 추가되고 리포지토리 생성의 일부로 기본 브랜치가 생성됩니다. 이 기본 브랜치의 이름은 기본입니다. 이 기본 브랜치는 사용자가 리포지토리를 복제할 때 로컬 리포지토리에서 기본 브랜치로 사용될 브랜치입니다. 기본 브랜치로 사용되는 브랜치를 변경할 수 있습니다. 자세한 내용은 [리포지토리의 기본 브랜치 관리](#) 섹션을 참조하세요.

소스 리포지토리의 기본 브랜치는 삭제할 수 없습니다. 검색 결과에는 기본 브랜치의 결과만 포함됩니다.

커밋

커밋은 파일 또는 파일 집합의 변경입니다. Amazon CodeCatalyst 콘솔에서 커밋은 변경 사항을 저장하고 소스 리포지토리로 푸시합니다. 커밋에는 변경한 사용자의 ID, 변경 시간 및 날짜, 커밋 제목, 변경에 포함된 모든 메시지를 포함한 변경에 대한 정보가 포함됩니다. 자세한 내용은 [Amazon CodeCatalyst에서 커밋을 사용한 소스 코드 변경 사항 이해](#) 섹션을 참조하세요.

CodeCatalyst의 소스 리포지토리 맥락에서 커밋은 콘텐츠의 스냅샷과 리포지토리의 콘텐츠에 대한 변경 사항입니다. 특정 커밋을 식별하는 데 도움이 되도록 커밋에 Git 태그를 추가할 수도 있습니다.

풀 요청

풀 요청은 사용자와 다른 사용자가 소스 리포지토리에서 한 브랜치에서 다른 브랜치로 코드 변경 사항을 검토하고, 설명하고, 병합하는 주요 방법입니다. 풀 요청을 사용하여 중요하지 않은 변경 내용이나 수정 사항, 중요 기능 추가 또는 릴리스된 소프트웨어의 새 버전에 대한 코드 변경 내용을 공동으로 검토할 수 있습니다. 풀 요청에서 소스 브랜치와 대상 브랜치 간의 변경 사항 또는 해당 브랜치의 개정 간의 차이점을 검토할 수 있습니다. 개별 코드 변경 행에 주석을 추가하고 풀 요청 전체에 대한 주석을 추가할 수 있습니다.

Tip

풀 요청을 생성하는 동안 나타나는 차이는 소스 브랜치의 최신 커밋과 대상 브랜치의 최신 커밋의 차이입니다. 풀 요청이 생성되면, 표시된 차이는 선택한 풀 요청의 수정본과 풀 요청을 생성할 때 대상 브랜치의 가장 최신 커밋 간에 발생합니다. Git의 차이점 및 병합 기반에 대한 자세한 내용은 Git 설명서의 [git-merge-base](#)를 참조하세요.

개정

개정은 풀 요청의 업데이트된 버전입니다. 풀 요청의 소스 브랜치로 푸시할 때마다 해당 푸시에 포함된 커밋에 적용된 변경 사항이 포함된 개정이 생성됩니다. 소스 브랜치와 대상 브랜치의 차이점 외에도 풀 요청의 개정 간의 차이점을 볼 수 있습니다. 자세한 내용은 [Amazon CodeCatalyst에서 풀 요청을 사용하여 코드 검토](#) 섹션을 참조하세요.

워크플로

워크플로는 지속적 통합 및 지속적 전송(CI/CD) 시스템의 일부로 코드를 빌드, 테스트 및 배포하는 방법을 설명하는 자동화된 절차입니다. 워크플로는 워크플로 실행 중에 수행할 일련의 단계 또는 작업을

정의합니다. 또한 워크플로는 워크플로를 시작하게 하는 이벤트 또는 트리거를 정의합니다. 워크플로를 설정하려면 CodeCatalyst 콘솔의 [시각적 또는 YAML 편집기](#)를 사용하여 워크플로 정의 파일을 생성합니다.

Tip

프로젝트에서 워크플로를 사용하는 방법을 간략하게 알아보려면 [블루프린트가 있는 프로젝트를 생성](#)합니다. 각 블루프린트는 검토, 실행 및 실험할 수 있는 기능 워크플로를 배포합니다.

소스 리포지토리는 프로젝트의 워크플로, 알림, 문제 및 기타 구성 정보에 대한 구성 파일 및 기타 정보를 저장할 수도 있습니다. 구성 파일이 필요한 리소스를 생성하거나 리포지토리를 워크플로의 소스 작업으로 지정할 때 구성 파일이 생성되어 소스 리포지토리에 저장됩니다. 블루프린트에서 프로젝트를 생성하는 경우 프로젝트의 일부로 생성된 소스 리포지토리에 구성 파일이 이미 저장됩니다. 이 구성 정보는 리포지토리의 기본 브랜치에 있는 `.codecatalyst` 폴더에 저장됩니다. 기본 브랜치의 브랜치를 생성할 때마다 해당 브랜치의 다른 모든 파일 및 폴더 외에도 이 폴더 및 해당 구성의 사본을 생성합니다.

소스 리포지토리 작업 설정

로컬 컴퓨터에서 Amazon CodeCatalyst의 소스 리포지토리로 작업하는 경우, 자체적으로 또는 지원되는 통합 개발 환경(IDE)에서 Git을 사용하여 코드를 변경하고 코드를 푸시 및 풀링할 수 있습니다. 가장 좋은 방법은 최신 버전의 Git 및 기타 소프트웨어를 사용하는 것입니다.

Note

개발 환경을 사용하는 경우 Git을 설치할 필요가 없습니다. 최신 버전의 Git이 개발 환경에 포함되어 있습니다.

CodeCatalyst에 대한 버전 호환성 정보

구성 요소	버전
Git	최신

Git 설치

IDE 없이 Git 클라이언트에서 소스 리포지토리의 파일, 커밋, 브랜치 및 기타 정보로 작업하려면 로컬 머신에 Git을 설치하세요.

Git 설치를 위해서는 [Git 다운로드](#)와 같은 웹 사이트를 권장합니다.

개인용 액세스 토큰 생성

소스 리포지토리를 로컬 컴퓨터 또는 선호하는 IDE에 복제하려면 개인 액세스 토큰(PAT)을 만들어야 합니다.

개인용 액세스 토큰(PAT)을 만들려면

1. 상단 메뉴 바에서 프로파일 배지를 선택한 다음 내 설정을 선택합니다.

Tip

프로젝트 또는 스페이스의 멤버 페이지로 이동하고 멤버 목록에서 이름을 선택하여 자신의 사용자 프로파일을 찾을 수도 있습니다.

2. PAT 이름에 PAT를 설명하는 이름을 입력합니다.
3. 만료 날짜에 기본 날짜를 유지하거나 달력 아이콘을 선택하여 사용자 지정 날짜를 선택합니다. 만료 날짜는 기본적으로 현재 날짜로부터 1년 후입니다.
4. 생성을 선택합니다.

소스 리포지토리의 복제 리포지토리를 선택할 때도 이 토큰을 생성할 수 있습니다.

5. PAT 보안 암호를 안전한 위치에 저장합니다.

Important

PAT 보안 암호는 한 번만 표시됩니다. 창을 닫은 후에는 검색할 수 없습니다.

CodeCatalyst 소스 리포지토리 및 단일 페이지 애플리케이션 블루프린트 시작하기

이 자습서의 단계에 따라 Amazon CodeCatalyst에서 소스 리포지토리를 사용하는 방법을 알아봅니다.

Amazon CodeCatalyst에서 소스 리포지토리 작업을 시작하는 가장 빠른 방법은 템플릿을 사용하여 프로젝트를 생성하는 것입니다. 템플릿을 사용하여 프로젝트를 생성하면 샘플 코드가 포함된 소스 리포지토리를 포함하여 리소스가 생성됩니다. 이 리포지토리 및 코드 예시를 사용하여 다음 방법을 배울 수 있습니다.

- 프로젝트의 소스 리포지토리 보기 및 콘텐츠 검색
- 코드 작업을 수행할 수 있는 새 브랜치를 사용하여 개발 환경 생성
- 파일 변경, 변경 사항 커밋 및 푸시
- 풀 요청 생성, 다른 프로젝트 멤버와 함께 코드 변경 사항 검토
- 프로젝트의 워크플로 확인, 풀 요청의 소스 브랜치에서 변경 사항을 자동 구축 및 테스트
- 소스 브랜치의 변경 사항을 대상 브랜치로 병합 및 풀 요청 닫기
- 자동으로 빌드 및 배포된 병합된 변경 사항 보기

이 자습서를 최대한 활용하려면 풀 요청을 함께 수행할 수 있도록 프로젝트에 다른 사람을 초대하세요. CodeCatalyst에서 문제 생성 및 풀 요청과 연결, 알림 구성 및 연결된 워크플로가 실행될 때 알림 받기와 같은 추가 기능을 탐색할 수도 있습니다. CodeCatalyst에 대한 전체 내용은 [시작하기 자습서](#) 섹션을 참조하세요.

블루프린트를 사용하여 프로젝트 생성

프로젝트 생성은 함께 작업할 수 있는 첫 번째 단계입니다. 블루프린트를 사용하여 프로젝트를 생성할 수 있으며, 이 경우 샘플 코드가 포함된 소스 리포지토리와 변경 시 코드를 자동으로 빌드하고 배포하는 워크플로도 생성됩니다. 이 자습서에서는 단일 페이지 애플리케이션 블루프린트로 생성된 프로젝트를 안내하지만 소스 리포지토리가 있는 모든 프로젝트의 절차를 따를 수 있습니다. 프로젝트 생성의 일부로 IAM 역할이 없는 경우 IAM 역할을 선택하거나 IAM 역할을 추가해야 합니다. 이 프로젝트에는 CodeCatalystWorkflowDevelopmentRole-**spaceName** 서비스 역할을 사용하는 것이 좋습니다.

이미 클러스터가 있는 경우 [프로젝트의 리포지토리 보기](#) 단계로 건너뛸 수 있습니다.

Note

스페이스 관리자 또는 파워 유저 역할이 있는 사용자만 CodeCatalyst에서 프로젝트를 생성할 수 있습니다. 이 역할이 없고 이 자습서에서 작업할 프로젝트가 필요한 경우 이러한 역할 중 하나를 가진 사람에게 프로젝트를 생성하도록 요청하고 생성된 프로젝트에 사용자를 추가합니다. 자세한 내용은 [사용자 역할로 액세스 권한 부여](#) 섹션을 참조하세요.

블루프린트를 사용하여 프로젝트를 생성하려면

1. CodeCatalyst 콘솔에서 프로젝트를 생성하려는 스페이스로 이동합니다.
2. 스페이스 대시보드에서 프로젝트 생성을 선택합니다.
3. 블루프린트로 시작을 선택합니다.

Tip

Amazon Q가 블루프린트를 제안하도록 프로젝트 요구 사항을 Amazon Q에 제공하여 블루프린트를 추가하도록 선택할 수 있습니다. 자세한 내용은 [Amazon Q를 사용하여 프로젝트를 생성하거나 기능을 추가할 때 블루프린트 선택](#) 및 [Amazon Q를 사용하여 프로젝트를 생성하거나 블루프린트를 사용하여 기능을 추가할 때의 모범 사례](#) 섹션을 참조하세요. 이 기능은 미국 서부(오리건) 리전에서만 사용할 수 있습니다.

이 기능을 사용하려면 스페이스에 생성형 AI 기능을 활성화해야 합니다. 자세한 내용은 [생성형 AI 기능 관리](#)를 참조하세요.

4. CodeCatalyst 블루프린트 또는 스페이스 블루프린트 탭에서 블루프린트를 선택한 다음 다음을 선택합니다.
5. 프로젝트 이름 지정에 프로젝트에 할당할 이름과 관련 리소스 이름을 입력합니다. 이름은 스페이스 내에서 고유해야 합니다.
6. (선택 사항) 기본적으로 블루프린트에서 생성된 소스 코드는 CodeCatalyst 리포지토리에 저장됩니다. 또는 블루프린트의 소스 코드를 타사 리포지토리에 저장하도록 선택할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 확장 프로그램이 있는 프로젝트에 기능 추가](#) 섹션을 참조하세요.

Important

CodeCatalyst는 연결된 리포지토리에 대한 기본 브랜치의 변경 사항 감지를 지원하지 않습니다. 연결된 리포지토리의 기본 브랜치를 변경하려면 먼저 CodeCatalyst에서 연결을 해제하고 기본 브랜치를 변경한 다음 다시 연결해야 합니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결](#) 섹션을 참조하세요.

리포지토리를 연결하기 전에 항상 최신 버전의 확장 프로그램을 사용하는 것이 좋습니다.

사용하려는 타사 리포지토리 공급자에 따라 다음 중 하나를 수행합니다.

- GitHub 리포지토리: GitHub 계정을 연결합니다.

고급 드롭다운 메뉴를 선택하고 GitHub를 리포지토리 공급자로 선택한 다음 블루프린트에서 생성된 소스 코드를 저장할 GitHub 계정을 선택합니다.

Note

GitHub 계정을 연결하는 경우 CodeCatalyst ID와 GitHub ID 간에 ID 매핑을 설정하려면 개인 연결을 생성해야 합니다. 자세한 내용은 [개인 연결](#) 및 [개인 연결을 사용하여 GitHub 리소스에 액세스](#) 섹션을 참조하세요.

- Bitbucket 리포지토리: Bitbucket 작업 영역을 연결합니다.

고급 드롭다운 메뉴를 선택하고 Bitbucket을 리포지토리 공급자로 선택한 다음, 블루프린트에서 생성된 소스 코드를 저장할 Bitbucket 작업 영역을 선택합니다.

- GitLab 리포지토리: GitLab 사용자를 연결합니다.

고급 드롭다운 메뉴를 선택하고 GitLab을 리포지토리 공급자로 선택한 다음 블루프린트에서 생성된 소스 코드를 저장할 GitLab 사용자를 선택합니다.

7. 프로젝트 리소스에서 블루프린트 파라미터를 구성합니다. 블루프린트에 따라 소스 리포지토리의 이름을 지정할 수 있는 옵션이 있을 수 있습니다.
8. (선택 사항) 구성된 프로젝트 파라미터에 따라 업데이트가 포함된 정의 파일을 보려면 프로젝트 미리 보기 생성에서 코드 보기 또는 워크플로 보기를 선택합니다.
9. (선택 사항) 블루프린트의 아키텍처 개요, 필수 연결 및 권한, 블루프린트가 생성하는 리소스 유형 등 블루프린트에 대한 특정 세부 정보를 보려면 블루프린트 카드에서 세부 정보 보기를 선택합니다.
10. 프로젝트 생성을 선택합니다.

프로젝트를 생성하거나 프로젝트 초대를 수락하고 로그인 프로세스를 완료하면 프로젝트 개요 페이지가 열립니다. 새 프로젝트의 프로젝트 개요 페이지에는 미해결 문제나 풀 요청이 포함되어 있지 않습니다. 선택적으로 문제를 생성하고 자신에게 할당하도록 선택할 수 있습니다. 프로젝트에 다른 사람을 초대하도록 선택할 수도 있습니다. 자세한 내용은 [CodeCatalyst에서 문제 생성](#) 및 [프로젝트에 사용자 초대](#) 섹션을 참조하세요.

프로젝트의 리포지토리 보기

프로젝트의 멤버는 프로젝트의 소스 리포지토리를 볼 수 있습니다. 추가 리포지토리를 생성하도록 선택할 수도 있습니다. 스페이스 관리자 역할을 가진 사람이 GitHub 리포지토리, Bitbucket 리포지토리

또는 GitLab 리포지토리 확장 프로그램을 설치 및 구성한 경우 확장 프로그램을 위해 구성된 GitHub 계정, Bitbucket 작업 영역 또는 GitLab 사용자의 타사 리포지토리에 대한 링크를 추가할 수도 있습니다. 자세한 내용은 [소스 리포지토리 생성 및 빠른 시작: CodeCatalyst에서 확장 프로그램 설치, 공급자 연결 및 리소스 연결](#) 섹션을 참조하세요.

Note

단일 페이지 애플리케이션 블루프린트로 생성된 프로젝트의 경우 샘플 코드가 포함된 소스 리포지토리의 기본 이름은 **spa-app**입니다.

프로젝트의 소스 리포지토리로 이동하려면

1. 프로젝트로 이동하여 다음 중 하나를 수행합니다.
 - 프로젝트의 요약 페이지에서 목록에서 원하는 리포지토리를 선택한 다음 리포지토리 보기를 선택합니다.
 - 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다. 소스 리포지토리의 경우 목록에서 리포지토리의 이름을 선택합니다. 필터 표시줄에 리포지토리 이름의 일부를 입력하여 리포지토리 목록을 필터링할 수 있습니다.
2. 리포지토리의 홈 페이지에서 리포지토리의 내용과 풀 요청 수 및 워크플로와 같은 관련 리소스에 대한 정보를 확인합니다. 기본적으로 기본 브랜치의 내용이 표시됩니다. 드롭다운 목록에서 다른 브랜치를 선택하여 보기를 변경할 수 있습니다.

리포지토리의 개요 페이지에는 이 리포지토리의 브랜치 및 해당 파일에 대해 구성된 워크플로 및 풀 요청에 대한 정보가 포함되어 있습니다. 방금 프로젝트를 생성한 경우 코드를 빌드, 테스트 및 배포하는 초기 워크플로는 완료하는 데 몇 분 정도 걸리므로 계속 실행됩니다. 관련 워크플로에서 번호를 선택하면 관련 워크플로와 해당 상태를 볼 수 있지만, 이렇게 하면 CI/CD의 워크플로 페이지가 열립니다. 이 자습서의 경우 개요 페이지에 남아 리포지토리의 코드를 탐색합니다. README.md 파일의 내용은 리포지토리 파일 아래의 이 페이지에서 렌더링됩니다. 파일에 기본 브랜치의 내용이 표시됩니다. 다른 브랜치가 있는 경우 다른 브랜치의 내용을 표시하도록 파일 보기를 변경할 수 있습니다. .codecatalyst 폴더에는 워크플로 YAML 파일과 같은 프로젝트의 다른 부분에 사용되는 코드가 포함되어 있습니다.

폴더의 콘텐츠를 보려면 폴더 이름 옆에 있는 화살표를 선택하여 확장합니다. 예를 들어 src 옆의 화살표를 선택하여 해당 폴더에 포함된 단일 페이지 웹 애플리케이션의 파일을 봅니다. 파일의 콘텐츠를 보려면 목록에서 선택합니다. 이렇게 하면 여러 파일의 내용을 찾아볼 수 있는 파일 보기가 열립니다. 콘솔에서도 단일 파일을 편집할 수 있지만 여러 파일을 편집하려면 개발 환경을 생성해야 합니다.

개발 환경 생성

Amazon CodeCatalyst 콘솔의 소스 리포지토리에서 파일을 추가하고 변경할 수 있습니다. 그러나 여러 파일 및 브랜치를 효과적으로 사용하려면 개발 환경을 사용하거나 리포지토리를 로컬 컴퓨터에 복제하는 것이 좋습니다. 이 자습서에서는 라는 브랜치를 사용하여 AWS Cloud9 개발 환경을 생성합니다. **develop**. 다른 브랜치 이름을 선택할 수 있지만 **develop** 브랜치 이름을 지정하면 이 자습서의 뒷 부분에서 풀 요청을 생성할 때 코드를 빌드하고 테스트하기 위한 워크플로가 자동으로 실행됩니다.

Tip

개발 환경을 사용하는 대신 또는 사용하는 것 외에 로컬에서 리포지토리를 복제하기로 결정하는 경우 로컬 컴퓨터에 Git이 있는지 또는 IDE에 Git이 포함되어 있는지 확인합니다. 자세한 내용은 [소스 리포지토리 작업 설정](#) 섹션을 참조하세요.

새 브랜치로 개발 환경을 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 개발 환경을 생성하려는 프로젝트로 이동합니다.
3. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다. 또는 탐색 창에서 코드를 선택하고 소스 리포지토리를 선택한 다음 개발 환경을 생성할 리포지토리를 선택합니다.
4. 리포지토리 홈 페이지에서 개발 환경 생성을 선택합니다.
5. 드롭다운 메뉴에서 지원되는 IDE를 선택합니다. 자세한 정보는 [개발 환경에 지원되는 통합 개발 환경](#)을 참조하세요.
6. 복제할 리포지토리를 선택하고, 새 브랜치에서 작업을 선택하고, 브랜치 이름 필드에 브랜치 이름을 입력하고, 다음에서 브랜치 생성 드롭다운 메뉴에서 새 브랜치를 만들 브랜치를 선택합니다.
7. 원하는 경우 개발 환경의 별칭을 추가할 수 있습니다.
8. 선택적으로 개발 환경 구성 편집 버튼을 선택하여 개발 환경의 컴퓨팅, 스토리지 또는 제한 시간 구성을 편집합니다.
9. 생성(Create)을 선택합니다. 개발 환경이 생성되는 동안 개발 환경 상태 옆에 시작 중이 표시되고, 개발 환경이 생성되면 상태 옆에 실행 중이 표시됩니다. 선택한 IDE의 개발 환경과 함께 새 탭이 열립니다. 코드를 편집하고 변경 사항을 커밋하고 푸시할 수 있습니다.

개발 환경을 생성한 후에는 파일을 편집하고, 변경 사항을 커밋하고, 변경 사항을 **test** 브랜치로 푸시할 수 있습니다. 이 자습서에서는 src 폴더의 App.tsx 파일에 있는 <p> 태그 간에 콘텐츠를 편집하여

웹 페이지에 표시되는 텍스트를 변경합니다. 변경 사항을 커밋하고 푸시한 다음 CodeCatalyst 탭으로 돌아갑니다.

AWS Cloud9 개발 환경에서 변경하고 푸시하려면

1. 에서 측면 탐색 메뉴를 AWS Cloud9 확장하여 파일을 찾습니다. `src`를 확장하고 `App.tsx`를 엽니다.
2. `<p>` 태그 내부의 텍스트를 변경합니다.
3. 파일을 저장한 다음 Git 메뉴를 사용하여 변경 사항을 커밋하고 푸시합니다. 또는 터미널 창에서 `git commit` 및 `git push` 명령을 사용하여 변경 사항을 커밋하고 푸시합니다.

```
git commit -am "Making an example change"
git push
```

Tip

Git 명령을 성공적으로 실행하기 전에 터미널의 디렉터리를 Git 리포지토리 디렉터리로 변경해야 할 수 있습니다.

풀 요청 생성

풀 요청을 사용하여 중요하지 않은 변경 내용이나 수정 사항, 중요 기능 추가 또는 릴리스된 소프트웨어의 새 버전에 대한 코드 변경 내용을 공동으로 검토할 수 있습니다. 이 자습서에서는 풀 요청을 생성하여 `test` 브랜치에 대한 변경 사항을 `main` 브랜치와 비교 검토합니다. 템플릿으로 생성된 프로젝트에서 풀 요청을 생성하면 해당 관련 워크플로의 실행도 시작됩니다.

풀 요청을 생성하려면

1. 프로젝트로 이동합니다.
2. 다음 중 하나를 수행하세요.
 - 탐색 창에서 코드를 선택하고 풀 요청을 선택한 다음 풀 요청 생성을 선택합니다.
 - 리포지토리 홈 페이지에서 추가를 선택한 다음 풀 요청 생성을 선택합니다.
 - 프로젝트 페이지에서 풀 요청 생성을 선택합니다.
3. 소스 리포지토리에서 지정된 소스 리포지토리가 커밋된 코드가 포함된 리포지토리인지 확인합니다. 이 옵션은 리포지토리의 메인 페이지에서 풀 요청을 생성하지 않은 경우에만 나타납니다.
4. 대상 브랜치에서 코드를 검토한 후 병합할 브랜치를 선택합니다.

5. 소스 브랜치에서 커밋된 코드가 포함된 브랜치를 선택합니다.
6. 풀 요청 제목에 다른 사용자가 검토해야 할 사항과 이유를 이해하는 데 도움이 되는 제목을 입력합니다.
7. (선택 사항) 풀 요청 설명에서 문제에 대한 연결 또는 변경 사항에 대한 설명과 같은 정보를 제공합니다.

 Tip

CodeCatalyst가 풀 요청에 포함된 변경 사항에 대한 설명을 자동으로 생성하도록 설명 쓰기를 선택할 수 있습니다. 자동으로 생성된 설명은 풀 요청에 추가한 후에 변경할 수 있습니다.

이 기능을 사용하려면 스페이스에 생성형 AI 기능을 활성화해야 하며 연결된 리포지토리의 풀 요청에 사용할 수 없습니다. 자세한 내용은 [생성형 AI 기능 관리](#)를 참조하세요.

8. (선택 사항) 문제에서 문제 연결을 선택한 다음 목록에서 문제를 선택하거나 해당 ID를 입력합니다. 문제를 연결 해제하려면 연결 해제 아이콘을 선택합니다.
9. (선택 사항) 필수 검토자에서 필수 검토자 추가를 선택합니다. 프로젝트 멤버 목록에서 추가할 멤버를 선택합니다. 필수 검토자는 풀 요청을 대상 브랜치에 병합하기 전에 변경 사항을 승인해야 합니다.

 Note

검토자를 추가할 때 필수 검토자이면서 동시에 선택적 검토자로 추가할 수 없습니다. 자신을 검토자로 추가할 수 없습니다.

10. (선택 사항) 선택적 검토자에서 선택적 검토자 추가를 선택합니다. 프로젝트 멤버 목록에서 추가할 멤버를 선택합니다. 선택적 검토자는 풀 요청을 대상 브랜치에 병합하기 전에 변경 사항을 요구 사항으로 승인할 필요가 없습니다.
11. 브랜치 간의 차이점을 검토합니다. 풀 요청에 표시되는 차이점은 소스 브랜치의 수정본과 풀 요청이 생성된 시점의 대상 브랜치의 헤드 커밋인 병합 기반 간의 변경 사항입니다. 변경 사항이 표시되지 않으면 브랜치가 동일하거나 소스와 대상 모두에 대해 동일한 브랜치를 선택했을 수 있습니다.
12. 풀 요청에 검토하려는 코드와 변경 사항이 포함되어 있다고 판단되면 생성을 선택합니다.

Note

풀 요청을 생성한 후 설명을 추가할 수 있습니다. 풀 요청 또는 파일의 개별 줄 및 풀 요청 전반에 설명을 추가할 수 있습니다. @ 기호와 파일 이름을 사용하여 파일과 같은 리소스에 대한 연결을 추가할 수 있습니다.

개요를 선택한 다음 워크플로 실행 아래의 풀 요청 세부 정보 영역에서 정보를 검토하여 이 풀 요청을 생성하여 시작된 관련 워크플로에 대한 정보를 볼 수 있습니다. 워크플로 실행을 보려면 실행을 선택합니다.

Tip

브랜치에 **develop**이 아닌 다른 이름을 지정하면 워크플로가 자동으로 실행되지 않아 변경 사항이 빌드 및 테스트되지 않습니다. 이를 구성하려면 onPullRequestBuildAndTest 워크플로에 대한 YAML 파일을 편집합니다. 자세한 내용은 [워크플로 생성](#) 섹션을 참조하세요.

이 풀 요청에 주석을 달고 다른 프로젝트 멤버에게 주석을 달라고 요청할 수 있습니다. 선택적 또는 필수 검토자를 추가하거나 변경하도록 선택할 수도 있습니다. 리포지토리의 소스 브랜치를 더 많이 변경하도록 선택하고, 이러한 커밋된 변경으로 인해 풀 요청에 대한 개정이 어떻게 생성되는지 확인할 수 있습니다. 자세한 내용은 [풀 요청 검토](#), [풀 요청 업데이트](#), [Amazon CodeCatalyst에서 풀 요청을 사용하여 코드 검토](#) 및 [워크플로 실행 상태 및 세부 정보 보기](#) 부분을 참조하세요.

풀 요청 병합

풀 요청이 검토되고 필수 검토자로부터 승인을 받은 후에는 해당 소스 브랜치를 CodeCatalyst 콘솔의 대상 브랜치에 병합할 수 있습니다. 풀 요청을 병합하면 대상 브랜치와 연결된 워크플로를 통해 변경 사항 실행도 시작됩니다. 이 자습서에서는 테스트 브랜치를 기본 로 병합하여 onPushToMainDeployPipeline 워크플로의 실행을 시작합니다.

풀 요청 병합(콘솔)

1. 풀 요청에서 이전 단계에서 생성한 풀 요청을 선택합니다. 풀 요청에서 병합을 선택합니다.
2. 풀 요청에서 사용 가능한 병합 전략 중에서 선택합니다. 선택적으로 풀 요청을 병합한 후 소스 브랜치를 삭제하는 옵션을 선택하거나 선택 취소한 다음 병합을 선택합니다. 병합이 완료되면 풀 요청 상태가 병합됨으로 변경되고 풀 요청의 기본 보기에 더 이상 표시되지 않습니다. 기본 보기에는

상태가 미결인 풀 요청이 표시됩니다. 병합된 풀 요청은 계속 볼 수 있지만 승인하거나 상태를 변경할 수는 없습니다.

Note

병합 버튼이 활성화되지 않았거나 병합할 수 없음 레이블이 표시되면 필수 검토자가 아직 풀 요청을 승인하지 않았거나 CodeCatalyst 콘솔에서 풀 요청을 병합할 수 없습니다. 풀 요청을 승인하지 않은 검토자는 풀 요청 세부 정보 영역의 개요에 시계 아이콘으로 표시됩니다. 필요한 모든 검토자가 풀 요청을 승인했지만 병합 버튼이 여전히 활성화되어 있지 않은 경우 병합 충돌이 발생하거나 스페이스의 스토리지 할당량을 초과했을 수 있습니다. 개발 환경에서 대상 브랜치의 병합 충돌을 해결하고, 변경 사항을 푸시한 다음 풀 요청을 병합하거나, 충돌을 해결하고 로컬에서 병합한 다음 병합이 포함된 커밋을 CodeCatalyst에 푸시할 수 있습니다. 자세한 내용은 [풀 요청 병합\(Git\)](#) 섹션 및 Git 설명서를 참조하세요.

배포된 코드 보기

이제 기본 브랜치에 있던 원래 배포된 코드와 병합된 변경 사항이 자동으로 빌드, 테스트 및 배포되면 이를 볼 시간입니다. 이렇게 하려면 리포지토리의 개요 페이지로 돌아가서 관련 워크플로 아이콘 옆의 번호를 선택하거나 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.

배포된 코드 보기

1. 워크플로의 onPushToMainDeployPipeline에서 최근 실행을 확장합니다.

Note

이 이름은 단일 페이지 애플리케이션 블루프린트로 생성된 프로젝트에 대한 워크플로의 기본 이름입니다.

2. 가장 최근의 실행은 병합된 풀 요청이 main 브랜치에 커밋된 것으로 시작되며, 진행 중 상태가 표시될 가능성이 높습니다. 목록에서 성공적으로 완료된 실행을 선택하여 해당 실행의 세부 정보를 엽니다.
3. 변수를 선택합니다. AppURL 값을 복사합니다. 배포된 단일 페이지 웹 애플리케이션의 URL입니다. 새 브라우저 탭을 열고 값을 붙여넣으면 빌드 및 배포된 코드를 볼 수 있습니다. 탭을 열어 봅니다.

4. 워크폴로 실행 목록으로 돌아가서 가장 최근의 실행이 완료될 때까지 기다립니다. 이 경우 열었던 탭으로 돌아가 웹 애플리케이션을 보고 브라우저를 새로 고칩니다. 병합된 풀 요청에서 변경한 내용이 표시됩니다.

리소스 정리

소스 리포지토리 및 풀 요청 작업을 탐색한 후에는 필요하지 않은 리소스를 모두 제거할 수 있습니다. 풀 요청은 삭제할 수 없지만 종료할 수 있습니다. 생성한 모든 브랜치를 삭제할 수 있습니다.

소스 리포지토리 또는 프로젝트가 더 이상 필요하지 않은 경우 해당 리소스를 삭제할 수도 있습니다. 자세한 내용은 [소스 리포지토리 삭제](#) 및 [프로젝트 삭제](#) 섹션을 참조하세요.

CodeCatalyst에서 프로젝트의 리포지토리에 소스 코드 저장

소스 리포지토리는 프로젝트에 대한 코드 및 파일을 안전하게 저장합니다. 또한 첫 번째 커밋부터 마지막 변경 내용까지 소스 기록을 저장합니다. 소스 리포지토리가 포함된 블루프린트를 선택하면 해당 리포지토리에는 프로젝트의 워크폴로 및 알림을 위한 구성 파일과 기타 정보도 포함됩니다. 이 구성 정보는 .codecatalyst 폴더에 저장됩니다.

프로젝트를 만들 때 소스 리포지토리를 만드는 블루프린트가 있는 프로젝트를 만들거나 기존 프로젝트에 소스 리포지토리를 만들어서 CodeCatalyst에서 소스 리포지토리를 만들 수 있습니다. 프로젝트 사용자는 프로젝트에 대해 만든 리포지토리를 자동으로 보고 사용할 수 있습니다. GitHub, Bibucket 또는 GitLab에서 호스팅되는 Git 리포지토리를 프로젝트에 연결하도록 선택할 수도 있습니다. 이렇게 하면 프로젝트 사용자는 프로젝트의 리포지토리 목록에서 연결된 리포지토리를 보고 액세스할 수 있습니다.

Note

리포지토리를 연결하려면 먼저 리포지토리를 호스팅하는 서비스에 대한 확장 프로그램을 설치해야 합니다. 아카이브된 리포지토리는 연결할 수 없습니다. 빈 리포지토리를 연결할 수는 있지만 기본 브랜치를 만드는 초기 커밋으로 초기화하기 전까지는 CodeCatalyst에서 리포지토리를 사용할 수 없습니다. 자세한 내용은 [스페이스에 확장 프로그램 설치](#) 섹션을 참조하세요.

기본적으로 소스 리포지토리는 Amazon CodeCatalyst 프로젝트의 다른 멤버와 공유됩니다. 프로젝트에 대한 추가 소스 리포지토리를 만들거나 리포지토리를 프로젝트에 연결할 수 있습니다. 프로젝트의 모든 멤버는 프로젝트의 소스 리포지토리에서 파일과 폴더를 보고, 추가하고, 편집하고, 삭제할 수 있습니다.

소스 리포지토리에서 코드를 빠르게 작업하려면 지정된 리포지토리를 복제하는 개발 환경을 만들고 이 환경으로 브랜치하여 개발 환경에 대해 선택한 통합 개발 환경(IDE)에서 코드를 작업할 수 있습니다. 로컬 컴퓨터에 소스 리포지토리를 복제하고 로컬 리포지토리와 원격 리포지토리 간에 변경 사항을 풀 앤 푸시하여 CodeCatalyst에서 작업할 수 있습니다. 또한 선호하는 IDE가 자격 증명 관리를 지원하는 한, 선호하는 IDE에서 소스 리포지토리에 대한 액세스를 구성하여 작업할 수도 있습니다.

리포지토리 이름은 CodeCatalyst 프로젝트에서 고유해야 합니다.

주제

- [소스 리포지토리 생성](#)
- [기존 Git 리포지토리를 소스 리포지토리로 복제](#)
- [소스 리포지토리 연결](#)
- [소스 리포지토리 보기](#)
- [소스 리포지토리의 설정 편집](#)
- [소스 리포지토리 복제](#)
- [소스 리포지토리 삭제](#)

소스 리포지토리 생성

Amazon CodeCatalyst에서 블루프린트를 사용하여 프로젝트를 생성하면 CodeCatalyst가 소스 리포지토리를 생성합니다. 해당 소스 리포지토리에는 워크플로에 대한 구성 정보 및 사용자를 위해 만들어진 기타 리소스 외에도 샘플 코드가 포함되어 있습니다. 이는 CodeCatalyst에서 리포지토리를 시작하는 권장 방법입니다. 프로젝트의 리포지토리를 생성하도록 선택할 수 있습니다. 이러한 리포지토리에는 언제든지 편집하거나 삭제할 수 있는 README.md 파일이 포함됩니다. 소스 리포지토리를 만들 때 선택한 사항에 따라 소스 리포지토리에 .gitignore 파일이 포함될 수도 있습니다.

기존 Git 리포지토리를 CodeCatalyst 소스 리포지토리로 복제하려면 대신 빈 리포지토리를 만드는 것을 고려하세요. 이 리포지토리는 몇 가지 간단한 Git 명령으로 콘텐츠를 추가할 때까지 CodeCatalyst에서 사용할 수 없습니다. 또는 CodeCatalyst 콘솔에서 직접 빈 리포지토리에 콘텐츠를 추가할 수도 있습니다. 또는 지원되는 Git 리포지토리 제공업체의 소스 리포지토리를 연결할 수도 있습니다. 자세한 내용은 [소스 리포지토리 연결](#) 섹션을 참조하세요.

소스 리포지토리를 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트로 이동합니다.

3. 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
4. 리포지토리 추가를 선택하고 리포지토리 생성을 선택합니다.
5. 리포지토리 이름에 리포지토리 이름을 제공합니다. 이 안내서에서는 *codecatalyst-source-repository*를 사용하지만 다른 이름을 선택할 수 있습니다. 리포지토리 이름은 프로젝트에서 고유해야 합니다. 리포지토리 이름 요구 사항에 대한 자세한 내용은 [CodeCatalyst의 소스 리포지토리 할당량](#) 섹션을 참조하세요.
6. (선택 사항) 설명에 리포지토리에 대한 설명을 추가하여 프로젝트의 다른 사용자가 리포지토리의 용도를 이해하는 데 도움이 되도록 합니다.
7. 리포지토리 생성(기본)을 선택합니다. 이 옵션은 기본 브랜치와 README.md 파일을 포함하는 리포지토리를 생성합니다. 빈 리포지토리보다 달리 이 리포지토리가 생성되는 즉시 사용할 수 있습니다.
8. 기본 브랜치에서 다른 이름을 선택할 이유가 없는 한 이름을 main으로 둡니다. 이 가이드의 예시는 모두 기본 브랜치의 이름으로 main을 사용합니다.
9. (선택 사항) 푸시하려는 코드 유형에 맞는 .gitignore 파일을 추가합니다.
10. 생성(Create)을 선택합니다.

Note

CodeCatalyst는 README.md 파일을 만들 때 리포지토리에 파일을 추가합니다. 또한 CodeCatalyst는 main이라는 기본 브랜치에 리포지토리에 대한 초기 커밋을 생성합니다. README.md 파일을 편집하거나 삭제할 수 있지만 기본 브랜치를 삭제할 수는 없습니다.

빈 소스 리포지토리 생성

1. CodeCatalyst 콘솔에서 빈 리포지토리를 만들려는 프로젝트로 이동합니다.
2. 프로젝트 요약 페이지의 소스 리포지토리에서 리포지토리 추가를 선택한 다음 리포지토리 만들기 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다. 리포지토리 추가를 선택하고 리포지토리 생성을 선택합니다.
3. 리포지토리 이름에 리포지토리 이름을 제공합니다. 이 안내서에서는 *codecatalyst-source-repository*를 사용하지만 다른 이름을 선택할 수 있습니다. 리포지토리 이름은 프로젝트에서 고유해야 합니다. 리포지토리 이름 요구 사항에 대한 자세한 내용은 [CodeCatalyst의 소스 리포지토리 할당량](#) 섹션을 참조하세요.
4. (선택 사항) 설명에 리포지토리에 대한 설명을 추가하여 프로젝트의 다른 사용자가 리포지토리의 용도를 이해하는 데 도움이 되도록 합니다.

5. 빈 리포지토리 생성을 선택한 다음 생성을 선택합니다.

기존 Git 리포지토리를 소스 리포지토리로 복제

기존 Git 리포지토리를 Amazon CodeCatalyst의 빈 소스 리포지토리에 복제할 수 있습니다. 이는 이전에 다른 Git 리포지토리 공급자에서 호스팅된 코드를 사용하여 CodeCatalyst를 시작하는 빠른 방법입니다. 미리 복제본을 생성한 다음 미러를 CodeCatalyst로 푸시하여 리포지토리의 콘텐츠를 복제할 수 있습니다. 또는 CodeCatalyst에 콘텐츠를 추가하려는 리포지토리의 로컬 리포지토리가 있는 경우 CodeCatalyst 소스 리포지토리를 로컬 리포지토리에 다른 원격 리포지토리로 추가한 다음 빈 소스 리포지토리로 푸시할 수 있습니다. 두 접근 방식 모두 동일하게 유효합니다. 미리 복제본을 사용하면 브랜치를 매핑할 뿐만 아니라 모든 참조를 매핑합니다. CodeCatalyst에서 리포지토리의 작업 복사본을 생성하는 간단하고 깨끗한 방법입니다. 빈 CodeCatalyst 소스 리포지토리를 가리키는 로컬 리포지토리에 원격 리포지토리를 추가하면 리포지토리 콘텐츠가 CodeCatalyst에 추가되지만, 로컬 리포지토리에서 CodeCatalyst 소스 리포지토리와 원래 Git 원격 리포지토리로 푸시할 수도 있습니다. 이는 코드를 서로 다른 원격 리포지토리에 유지하려는 경우 유용할 수 있지만 다른 개발자가 원격 리포지토리 중 하나에만 코드를 커밋하는 경우 충돌이 발생할 수 있습니다.

다음 절차에서는 기본 Git 명령을 사용하여 이 작업을 수행합니다. 복제를 포함하여 Git에서 작업을 수행하는 방법은 다양합니다. 자세한 내용은 [Git 설명서](#)를 참조하세요.

Important

콘텐츠를 복제하려면 먼저 CodeCatalyst에서 빈 리포지토리를 생성해야 합니다. 개인 액세스 토큰도 있어야 합니다. 자세한 내용은 [빈 소스 리포지토리 생성](#) 및 [개인용 액세스 토큰 생성](#) 섹션을 참조하세요.

git clone --mirror를 사용하여 기존 Git 리포지토리를 CodeCatalyst에 복제

1. CodeCatalyst 콘솔에서 빈 리포지토리를 생성한 프로젝트로 이동합니다.
2. 프로젝트의 요약 페이지에서 목록에서 빈 리포지토리를 선택한 다음 리포지토리 보기를 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다. 프로젝트의 소스 리포지토리 목록에서 빈 리포지토리의 이름을 선택합니다.
3. 빈 리포지토리의 HTTPS 복제 URL을 복사합니다. 미리 복제본을 푸시하려면 이 기능이 필요합니다. 예를 들어, ExampleCorp 스페이스의 MyExampleProject 프로젝트에서 소스 리포지토리 이름을 MyExampleRepo로 지정하고 사용자 이름이 LiJuan인 경우 복제 URL은 다음과 같이 보일 수 있습니다.

```
https://LiJuan@git.us-west-2.codecatalyst.aws/
v1/ExampleCorp/MyExampleProject/MyExampleRepo
```

4. 명령줄 또는 터미널 창에서 `git clone --mirror` 명령을 사용하여 CodeCatalyst 에 복제하려는 Git 리포지토리의 미러 복제본을 생성합니다. 예를 들어 GitHub에서 `codecatalyst-blueprints` 리포지토리의 미러 복제본을 생성하려면 다음 명령을 입력합니다.

```
git clone --mirror https://github.com/aws/codecatalyst-blueprints.git
```

5. 디렉터리를 복제를 생성한 디렉터리로 변경합니다.

```
cd codecatalyst-blueprints.git
```

6. URL과 대상 CodeCatalyst 소스 리포지토리의 이름, `--all` 옵션을 지정하여 `git push` 명령을 실행합니다. (여기서 URL은 3단계에서 복사해 놓은 것입니다.) 예제:

```
git push https://LiJuan@git.us-west-2.codecatalyst.aws/
v1/ExampleCorp/MyExampleProject/MyExampleRepo --all
```

원격 리포지토리를 추가하고 로컬 리포지토리를 CodeCatalyst로 푸시하려면

1. CodeCatalyst 콘솔에서 빈 리포지토리를 생성한 프로젝트로 이동합니다.
2. 프로젝트의 요약 페이지에서 목록에서 빈 리포지토리를 선택한 다음 리포지토리 보기를 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다. 프로젝트의 소스 리포지토리 목록에서 빈 리포지토리의 이름을 선택합니다.
3. 빈 리포지토리의 HTTPS 복제 URL을 복사합니다. 미러 복제본을 푸시하려면 이 기능이 필요합니다. 예를 들어, ExampleCorp 스페이스의 MyExampleProject 프로젝트에서 소스 리포지토리 이름을 MyExampleRepo로 지정하고 사용자 이름이 LiJuan인 경우 복제 URL은 다음과 같이 보일 수 있습니다.

```
https://LiJuan@git.us-west-2.codecatalyst.aws/
v1/ExampleCorp/MyExampleProject/MyExampleRepo
```

4. 명령줄 또는 터미널 창에서 디렉터를 CodeCatalyst로 푸시하려는 로컬 리포지토리로 변경합니다.

5. `git remote -v` 명령을 실행하여 로컬 리포지토리의 기존 원격지를 확인합니다. 예를 들어 미국 동부 (오하이오) 리전에 이름이 **MyDemoRepo**인 AWS CodeCommit 리포지토리의 로컬 리포지토리를 복제하는 경우 명령 출력은 다음과 같을 수 있습니다.

```
origin https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
origin https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (push)
```

리포지토리를 계속 사용하려면 원격 URL을 복사합니다.

6. `git remote remove` 명령을 사용하여 오리진 가져오기 및 푸시에 대한 CodeCommit 리포지토리 URL을 제거합니다.

```
git remote remove origin
```

7. `git remote add` 명령을 사용하여 CodeCatalyst 소스 리포지토리 URL을 로컬 리포지토리의 가져오기 및 푸시 원격지 로 추가합니다. 예제:

```
git remote add origin https://LiJuan@git.us-west-2.codecatalyst.aws/v1/ExampleCorp/MyExampleProject/MyExampleRepo
```

이는 CodeCommit 리포지토리 푸시 URL을 CodeCatalyst 소스 리포지토리 URL로 대체하지만 가져오기 URL은 변경하지 않습니다. 따라서 `git remote -v` 명령을 다시 실행하면 CodeCommit 원격 리포지토리에서 코드를 가져오고(가져오고) 있지만 로컬 리포지토리의 변경 사항을 CodeCatalyst 소스 리포지토리로 푸시하도록 구성되어 있습니다.

```
origin https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo (fetch)
origin https://LiJuan@git.us-west-2.codecatalyst.aws/v1/ExampleCorp/MyExampleProject/MyExampleRepo (push)
```

`git remote set-url` 명령을 사용하여 두 리포지토리로 푸시하려는 경우 선택적으로 CodeCommit 원격 URL을 다시 추가할 수 있습니다.

```
git remote set-url --add --push origin https://git-codecommit.us-east-2.amazonaws.com/v1/repos/MyDemoRepo
```

8. `git push` 명령을 실행하여 구성된 모든 푸시 원격지에 로컬 리포지토리를 푸시합니다. 또는 로컬 리포지토리를 두 리포지토리로 푸시하는 `--all` 옵션을 지정하여 `git push -u -origin` 명령을 실행합니다. 예제:

```
git push -u -origin --all
```

 Tip

Git 버전에 따라 로컬 리포지토리의 모든 브랜치를 빈 리포지토리로 푸시하는 데 모두 작동하지 않을 수 있습니다. 각 브랜치를 체크아웃하고 별도로 푸시해야 할 수 있습니다.

소스 리포지토리 연결

소스 리포지토리를 프로젝트에 연결할 때 리포지토리를 호스팅하는 서비스에 대해 CodeCatalyst 확장 프로그램이 설치되어 있는 경우 해당 확장 프로그램이 있는 리포지토리를 포함할 수 있습니다(해당 스페이스에 대해 해당 확장 프로그램이 설치되어 있는 경우). 스페이스 관리자 역할을 가진 사용자만 확장 프로그램을 설치할 수 있습니다. 확장 프로그램이 설치되면 해당 확장 프로그램에서 액세스하도록 구성된 리포지토리에 연결할 수 있습니다. 자세한 내용은 [스페이스에 확장 프로그램 설치](#) 섹션을 참조하거나 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결](#)을 따르세요.

 Important

리포지토리 확장 프로그램을 설치한 후 CodeCatalyst에 연결하는 모든 리포지토리의 코드가 CodeCatalyst에 인덱싱 및 저장됩니다. 이렇게 하면 CodeCatalyst에서 코드를 검색할 수 있습니다. CodeCatalyst에서 연결된 리포지토리를 사용할 때 코드에 대한 데이터 보호를 더 잘 이해하려면 Amazon CodeCatalyst 사용 설명서의 [데이터 보호](#)를 참조하세요.

리포지토리를 한 스페이스에 있는 하나의 프로젝트에만 연결할 수 있습니다. 아카이브된 리포지토리는 연결할 수 없습니다. 빈 리포지토리를 연결할 수는 있지만 기본 브랜치를 만드는 초기 커밋으로 초기화하기 전까지는 CodeCatalyst에서 리포지토리를 사용할 수 없습니다. 뿐만 아니라

- GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리는 한 스페이스에서 하나의 CodeCatalyst 프로젝트에만 연결할 수 있습니다.
- 비어있거나 보관된 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 CodeCatalyst 프로젝트와 함께 사용할 수 없습니다.
- CodeCatalyst 프로젝트의 리포지토리와 이름이 동일한 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 연결할 수 없습니다.

- GitHub 리포지토리 확장 프로그램은 GitHub Enterprise Server 리포지토리와 호환되지 않습니다.
- Bitbucket 리포지토리 확장 프로그램은 Bitbucket Data Center 리포지토리와 호환되지 않습니다.
- GitLab 리포지토리 확장 프로그램은 GitLab 자체 관리형 프로젝트 리포지토리와 호환되지 않습니다.
- 연결된 리포지토리에서는 설명 쓰기 또는 설명 요약 기능을 사용할 수 없습니다. 이러한 기능은 CodeCatalyst의 풀 요청에서만 사용할 수 있습니다.

GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 기고자로 연결할 수 있지만 타사 리포지토리를 스페이스 관리자 또는 프로젝트 관리자로서만 연결 해제할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리](#), [Bitbucket 리포지토리](#), [GitLab 프로젝트 리포지토리](#) 및 [Jira 프로젝트 연결 해제](#) 섹션을 참조하세요.

Important

CodeCatalyst는 연결된 리포지토리에 대한 기본 브랜치의 변경 사항 감지를 지원하지 않습니다. 연결된 리포지토리의 기본 브랜치를 변경하려면 먼저 CodeCatalyst에서 연결을 해제하고 기본 브랜치를 변경한 다음 다시 연결해야 합니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리](#), [Bitbucket 리포지토리](#), [GitLab 프로젝트 리포지토리](#) 및 [Jira 프로젝트 연결](#) 섹션을 참조하세요.

리포지토리를 연결하기 전에 항상 최신 버전의 확장 프로그램을 사용하는 것이 좋습니다.

소스 리포지토리 연결

1. 리포지토리를 연결하려는 프로젝트로 이동합니다.

Note

리포지토리를 연결하려면 먼저 스페이스 관리자 역할이 있는 사용자가 리포지토리를 호스팅하는 제공업체의 확장 프로그램을 설치해야 합니다. 자세한 내용은 [스페이스에 확장 프로그램 설치](#) 섹션을 참조하세요.

2. 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
3. 리포지토리 추가를 선택하고 리포지토리 연결을 선택합니다.
4. 리포지토리 공급자 드롭다운 메뉴에서 다음 타사 리포지토리 공급자 중 하나를 선택합니다.
GitHub 또는 Bitbucket
5. 연결하도록 선택한 타사 리포지토리 공급자에 따라 다음 중 하나를 수행합니다.

- GitHub 리포지토리: GitHub 리포지토리를 연결합니다.
 1. GitHub 계정 드롭다운 메뉴에서 연결할 리포지토리가 포함된 GitHub 계정을 선택합니다.
 2. GitHub 리포지토리 드롭다운 메뉴에서 CodeCatalyst 프로젝트를 연결할 GitHub 계정을 선택합니다.
 3. (선택 사항) 리포지토리 목록에 GitHub 리포지토리가 표시되지 않는 경우 GitHub의 Amazon CodeCatalyst 애플리케이션에서 리포지토리 액세스에 대해 구성되지 않았을 수 있습니다. 연결된 계정의 CodeCatalyst에서 사용할 수 있는 GitHub 리포지토리를 구성할 수 있습니다.
 - a. [GitHub](#) 계정으로 이동하여 설정을 선택한 다음 애플리케이션을 선택합니다.
 - b. 설치된 GitHub 앱 탭에서 Amazon CodeCatalyst 애플리케이션에 대해 구성을 선택합니다.
 - c. 다음 중 하나를 수행하여 CodeCatalyst에서 연결하려는 GitHub 리포지토리의 액세스를 구성합니다.
 - 모든 현재 및 향후 리포지토리에 대한 액세스를 제공하려면 모든 리포지토리를 선택합니다.
 - 특정 리포지토리에 대한 액세스 권한을 제공하려면 선택된 리포지토리만을 선택하고 리포지토리 선택 드롭다운을 선택한 다음 CodeCatalyst에서 연결할 수 있도록 허용할 리포지토리를 선택합니다.
- Bitbucket 리포지토리: Bitbucket 리포지토리를 연결합니다.
 1. Bitbucket 작업 영역 드롭다운 메뉴에서 연결하려는 리포지토리가 포함된 Bitbucket 작업 영역을 선택합니다.
 2. Bitbucket 리포지토리 드롭다운 메뉴에서 CodeCatalyst 프로젝트를 연결할 Bitbucket 리포지토리를 선택합니다.

 Tip

리포지토리 이름이 회색으로 표시된 경우 Amazon CodeCatalyst 의 다른 프로젝트에 이미 연결되어 있으므로 해당 리포지토리를 연결할 수 없습니다.

6. 연결을 선택합니다.

CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 더 이상 사용하지 않으려면 CodeCatalyst 프로젝트에서 연결을 해제할 수 있습니다. 리포지토리가 연결 해제되면 해당 리포지토리의 이벤트는 워크플로를 시작하지 않으며 CodeCatalyst 개발 환경에서 해당 리포지토리를 사용할 수 없습니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#) 섹션을 참조하세요.

소스 리포지토리 보기

Amazon CodeCatalyst에서 프로젝트와 연결된 소스 리포지토리를 볼 수 있습니다. CodeCatalyst의 소스 리포지토리의 경우 리포지토리의 개요 페이지에서는 다음을 포함하여 해당 리포지토리의 정보 및 활동에 대한 간략한 개요를 제공합니다.

- 리포지토리에 대한 설명(있을 경우)
- 리포지토리의 브랜치 수
- 리포지토리에 대한 열린 풀 요청 수
- 리포지토리에 대한 관련 워크플로 수
- 기본 브랜치의 파일 및 폴더 또는 선택한 브랜치
- 표시된 브랜치에 대한 마지막 커밋의 제목, 작성자 및 날짜
- README.md 파일이 포함된 경우 마크다운에 렌더링된 README.md 파일의 내용

또한 이 페이지에서는 리포지토리에 대한 커밋, 브랜치 및 풀 요청에 연결하는 링크와 개별 파일을 빠르게 열고 보고 편집할 수 있는 방법을 제공합니다.

Note

CodeCatalyst 콘솔에서는 연결된 리포지토리에 대한 이 정보를 볼 수 없습니다. 연결된 리포지토리에 대한 정보를 보려면, 리포지토리 목록에서 링크를 선택하여 리포지토리를 호스팅하는 서비스에서 해당 리포지토리를 엽니다.

프로젝트의 소스 리포지토리로 이동하려면

1. 프로젝트로 이동하여 다음 중 하나를 수행합니다.
 - 프로젝트의 요약 페이지에서 목록에서 원하는 리포지토리를 선택한 다음 리포지토리 보기를 선택합니다.

- 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다. 소스 리포지토리의 경우 목록에서 리포지토리의 이름을 선택합니다. 필터 표시줄에 리포지토리 이름의 일부를 입력하여 리포지토리 목록을 필터링할 수 있습니다.
2. 리포지토리의 홈 페이지에서 리포지토리의 내용과 풀 요청 수 및 워크플로와 같은 관련 리소스에 대한 정보를 확인합니다. 기본적으로 기본 브랜치의 내용이 표시됩니다. 드롭다운 목록에서 다른 브랜치를 선택하여 보기를 변경할 수 있습니다.

i Tip

프로젝트 요약 페이지에서 프로젝트 코드 보기를 선택하여 프로젝트의 리포지토리로 빠르게 이동할 수도 있습니다.

소스 리포지토리의 설정 편집

리포지토리에 대한 설명 편집, 기본 브랜치 선택, 브랜치 규칙 생성 및 관리, CodeCatalyst에서 풀 요청에 대한 승인 규칙 생성 및 관리 등 리포지토리 설정을 관리할 수 있습니다. 이를 통해 프로젝트 멤버가 리포지토리의 용도를 이해하고 팀에서 사용하는 모범 사례와 프로세스를 적용하는 데 도움이 될 수 있습니다.

i Note

소스 리포지토리의 이름은 편집할 수 없습니다.

연결된 리포지토리의 이름, 설명 또는 기타 정보는 CodeCatalyst에서 편집할 수 없습니다. 연결된 리포지토리에 대한 정보를 수정하려면 연결된 리포지토리를 호스팅하는 제공업체에서 정보를 편집해야 합니다. 자세한 내용은 연결된 리포지토리를 호스팅하는 서비스의 설명서를 참조하세요.

리포지토리의 설정 편집

1. CodeCatalyst 콘솔에서 설정을 편집하려는 소스 리포지토리가 포함된 프로젝트로 이동합니다.
2. 프로젝트의 요약 페이지에서 목록에서 원하는 리포지토리를 선택한 다음 리포지토리 보기를 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다.
3. 리포지토리의 개요 페이지에서 추가를 선택한 다음 설정 관리를 선택합니다.

4. 다음 중 한 개 이상을 수행할 수 있습니다.

- 리포지토리에 대한 설명을 편집한 다음 저장을 선택합니다.
- 리포지토리의 기본 브랜치를 변경하려면 기본 브랜치에서 편집을 선택합니다. 자세한 내용은 [리포지토리의 기본 브랜치 관리](#) 섹션을 참조하세요.
- 브랜치에서 특정 작업을 수행할 수 있는 권한을 가진 프로젝트 역할에 대한 규칙을 추가, 제거 또는 변경하려면 브랜치 규칙에서 편집을 선택합니다. 자세한 내용은 [브랜치 규칙을 사용하여 브랜치에 대한 작업 관리](#) 섹션을 참조하세요.
- 브랜치에 풀 요청을 병합하기 위한 승인 규칙을 추가, 제거 또는 변경하려면 승인 규칙에서 편집을 선택합니다. 자세한 내용은 [풀 요청을 승인 규칙과 병합하기 위한 요구 사항 관리](#) 섹션을 참조하세요.

소스 리포지토리 복제

소스 리포지토리에 있는 여러 파일, 브랜치 및 커밋을 효과적으로 작업하려면 소스 리포지토리를 로컬 컴퓨터에 복제하고 Git 클라이언트 또는 통합 개발 환경(IDE)을 사용하여 변경 작업을 수행합니다. 문제 및 풀 요청과 같은 CodeCatalyst 특성을 사용하려면 소스 리포지토리에 변경 사항을 커밋하고 푸시합니다. 코드 작업을 위한 개발 환경을 생성하도록 선택할 수도 있습니다. 개발 환경을 만들면 지정한 리포지토리와 브랜치가 자동으로 개발 환경으로 복제됩니다.

Note

연결된 리포지토리를 CodeCatalyst 콘솔에서 복제하거나 이에 대한 개발 환경을 생성할 수 없습니다. 연결된 리포지토리를 로컬에서 복제하려면 리포지토리 목록에서 링크를 선택하여 리포지토리를 호스팅하는 서비스에서 해당 리포지토리를 연 다음 복제합니다. 자세한 내용은 연결된 리포지토리를 호스팅하는 서비스의 설명서를 참조하세요.

소스 리포지토리에서 개발 환경 생성

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
3. 코드 작업을 수행할 소스 리포지토리를 선택합니다.
4. 개발 환경 생성을 선택합니다.
5. 드롭다운 메뉴에서 지원되는 IDE를 선택합니다. 자세한 정보는 [개발 환경에 지원되는 통합 개발 환경](#)을 참조하세요.

6. 다음 중 하나를 수행하세요.

- 기존 브랜치에서 작업을 선택한 다음 기존 브랜치 드롭다운 메뉴에서 브랜치를 선택합니다.
- 새 브랜치에서 작업을 선택하고, 브랜치 이름 필드에 브랜치 이름을 입력하고, 다음에서 브랜치 생성 드롭다운 메뉴에서 새 브랜치를 만들 브랜치를 선택합니다.

7. 선택적으로 개발 환경의 이름을 추가하거나 구성을 편집합니다.

8. 생성(Create)을 선택합니다.

소스 리포지토리 생성

1. 프로젝트로 이동합니다.
2. 프로젝트의 요약 페이지에서 목록에서 원하는 리포지토리를 선택한 다음 리포지토리 보기를 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다. 필터 표시줄에 리포지토리 이름의 일부를 입력하여 리포지토리 목록을 필터링할 수 있습니다.
- 3.
4. 리포지토리 복제를 선택합니다. 리포지토리의 복제 URL을 복사합니다.

Note

개인 액세스 토큰(PAT)이 없는 경우 토큰 생성을 선택합니다. 토큰을 복사하여 안전한 위치에 저장합니다. Git 클라이언트 또는 통합 개발 환경(IDE)에서 암호를 입력하라는 메시지가 표시되면 이 PAT를 사용합니다.

5. 다음 중 하나를 수행하세요.

- 로컬 컴퓨터에 리포지토리를 복제하려면 터미널 또는 명령줄을 열고 git clone 명령 뒤에 복제 URL을 사용하여 명령을 실행합니다. 예제:

```
git clone https://LiJuan@git.us-west-2.codecatalyst.aws/v1/ExampleCorp/MyExampleProject/MyExampleRepo
```

암호를 입력하라는 메시지가 표시되면 이전에 저장한 PAT를 붙여넣습니다.

 Note

운영 체제가 자격 증명 관리를 제공하거나 자격 증명 관리 시스템을 설치한 경우 PAT를 한 번만 제공하면 됩니다. 그렇지 않으면 모든 Git 작업에 대해 PAT를 제공해야 할 수 있습니다. 가장 좋은 방법은 자격 증명 관리 시스템이 PAT를 안전하게 저장하는 것입니다. 복제 URL 문자열의 일부로 PAT를 포함하지 마세요.

- IDE를 사용하여 리포지토리를 복제하려면 IDE에 대한 설명서를 따르세요. Git 리포지토리를 복제하고 URL을 제공하는 옵션을 선택합니다. 암호를 입력하라는 메시지가 표시되면 PAT를 제공합니다.

소스 리포지토리 삭제

Amazon CodeCatalyst 프로젝트의 소스 리포지토리가 더 이상 필요하지 않은 경우 삭제할 수 있습니다. 소스 리포지토리를 삭제하면 리포지토리에 저장된 모든 프로젝트 정보도 삭제됩니다. 워크플로가 소스 리포지토리에 종속된 경우 리포지토리가 삭제된 후 해당 워크플로는 프로젝트 워크플로 목록에서 삭제됩니다. 소스 리포지토리를 참조하는 문제는 삭제되거나 변경되지 않지만 리포지토리가 삭제되면 문제에 추가된 소스 리포지토리 링크는 모두 실패합니다.

 Important

소스 리포지토리 삭제는 실행 취소할 수 없습니다. 소스 리포지토리를 삭제한 후에는 더 이상 복제하거나, 리포지토리에서 데이터를 가져오거나, 리포지토리로 데이터를 푸시할 수 없습니다. 소스 리포지토리를 삭제한다고 해서 해당 리포지토리의 모든 로컬 사본(로컬 리포지토리)이 삭제되는 것은 아닙니다. 로컬 리포지토리를 삭제하려면 로컬 컴퓨터의 디렉터리와 파일 관리 도구를 사용하세요.

 Note

연결된 리포지토리는 CodeCatalyst 콘솔에서 삭제할 수 없습니다. 연결된 리포지토리를 삭제하려면 리포지토리 목록에서 링크를 선택하여 해당 리포지토리를 호스팅하는 서비스에서 해당 리포지토리를 연 다음 삭제합니다. 자세한 내용은 연결된 리포지토리를 호스팅하는 서비스의 설명서를 참조하세요.

프로젝트에서 연결된 리포지토리를 제거하려면 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#) 섹션을 참조하세요.

소스 리포지토리 삭제

1. 삭제하려는 소스 리포지토리가 포함된 프로젝트로 이동합니다.
2. 프로젝트의 요약 페이지에서 목록에서 원하는 리포지토리를 선택한 다음 리포지토리 보기를 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다.
3. 리포지토리의 홈 페이지에서 추가를 선택한 다음 설정 관리와 리포지토리 삭제를 차례로 선택합니다.
4. 브랜치, 풀 요청 및 관련 워크플로 정보를 검토하여 아직 사용 중이거나 완료되지 않은 작업이 있는 리포지토리를 삭제하지 않는지 확인합니다. 계속하려면 delete 입력한 다음 삭제를 선택합니다.

Amazon CodeCatalyst의 브랜치를 사용하여 소스 코드 작업 구성

Git에서 브랜치는 커밋에 대한 포인터 또는 참조입니다. 개발 시 작업을 체계적으로 정리할 수 있는 편리한 수단입니다. 브랜치를 사용하면 다른 브랜치의 작업에 영향을 주지 않으면서 파일의 새 버전이나 다른 버전에 대한 작업을 분리할 수 있습니다. 브랜치를 사용하면 새 기능을 개발하고 프로젝트 버전을 저장하는 등의 작업을 수행할 수 있습니다. 소스 리포지토리에서 브랜치에 대한 규칙을 구성하여 브랜치에 대한 특정 작업을 해당 프로젝트의 특정 역할로 제한할 수 있습니다.

Amazon CodeCatalyst의 소스 리포지토리에는 생성 방법에 관계없이 콘텐츠와 기본 브랜치가 있습니다. 연결된 리포지토리에는 기본 브랜치나 콘텐츠가 없을 수도 있지만, 초기화하고 기본 브랜치를 생성하기 전까지는 CodeCatalyst에서 사용할 수 없습니다. 블루프린트를 사용하여 프로젝트를 만들면 CodeCatalyst는 해당 프로젝트의 소스 리포지토리를 생성하며, 여기에는 README.md 파일, 샘플 코드, 워크플로 정의 및 기타 리소스가 포함됩니다. 블루프린트를 사용하지 않고 소스 리포지토리를 만들면 첫 번째 커밋으로 README.md 파일이 추가되고 기본 브랜치가 만들어집니다. 이 기본 브랜치의 이름은 기본입니다. 이 기본 브랜치는 사용자가 리포지토리를 복제할 때 로컬 리포지토리에서 기본 브랜치로 사용될 브랜치입니다.

Note

기본 브랜치는 삭제할 수 없습니다. 소스 리포지토리에 대해 처음 만든 브랜치가 해당 리포지토리의 기본 브랜치입니다. 또한 검색은 기본 브랜치의 결과만 표시합니다. 다른 브랜치에 있는 코드는 검색할 수 없습니다.

CodeCatalyst에서 리포지토리를 만들면 첫 번째 커밋이 생성되며, 이 커밋에는 README.md 파일이 포함된 기본 브랜치가 생성됩니다. 기본 브랜치의 이름은 main입니다. 이 이름이 본 안내서의 예시에 사용된 기본 브랜치 이름입니다.

주제

- [브랜치 생성](#)
- [리포지토리의 기본 브랜치 관리](#)
- [브랜치 규칙을 사용하여 브랜치에 대한 작업 관리](#)
- [브랜치에 대한 Git 명령](#)
- [브랜치 및 세부 정보 보기](#)
- [브랜치 삭제](#)

브랜치 생성

CodeCatalyst 콘솔을 사용하여 CodeCatalyst 리포지토리에 브랜치를 만들 수 있습니다. 생성한 브랜치는 다음에 다른 사용자가 리포지토리에서 변경 내용을 가져올 때 표시됩니다.

Tip

코드에서 작업할 개발 환경을 생성하는 과정에서 브랜치를 생성할 수도 있습니다. 자세한 내용은 [개발 환경 생성](#) 섹션을 참조하세요.

Git을 사용하여 브랜치를 생성할 수도 있습니다. 자세한 내용은 [브랜치에 대한 공통 Git 명령](#) 섹션을 참조하세요.

브랜치를 생성하려면(콘솔)

1. CodeCatalyst 콘솔에서 소스 리포지토리가 있는 프로젝트로 이동합니다.

2. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
3. 브랜치를 생성하려는 리포지토리의 이름을 선택합니다.
4. 리포지토리의 개요 페이지에서 추가를 선택한 다음 브랜치 생성을 선택합니다.
5. 브랜치의 이름을 입력합니다.
6. 브랜치를 생성할 브랜치를 선택한 다음 생성을 선택합니다.

리포지토리의 기본 브랜치 관리

Amazon CodeCatalyst에서 어떤 브랜치를 소스 리포지토리의 기본 브랜치로 할지 지정할 수 있습니다. CodeCatalyst의 모든 소스 리포지토리에는 생성 방식에 관계없이 콘텐츠와 기본 브랜치가 있습니다. 블루프린트를 사용하여 프로젝트를 생성하는 경우, 해당 프로젝트에 대해 생성된 소스 리포지토리의 기본 브랜치 이름이 메인으로 지정됩니다. 기본 브랜치의 내용은 해당 리포지토리의 개요 페이지에 자동으로 표시됩니다.

Important

CodeCatalyst는 연결된 리포지토리에 대한 기본 브랜치의 변경 사항을 감지하지 않습니다. 연결된 리포지토리의 기본 브랜치를 변경하려면 먼저 CodeCatalyst에서 연결을 해제하고 기본 브랜치를 변경한 다음 다시 연결해야 합니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결](#) 섹션을 참조하세요.

리포지토리를 연결하기 전에 항상 최신 버전의 확장 프로그램을 사용하는 것이 좋습니다.

기본 브랜치는 소스 리포지토리의 다른 모든 브랜치와 약간 다르게 처리됩니다. 이름 옆에는 기본이라는 특수 레이블이 있습니다. 이 기본 브랜치는 사용자가 리포지토리를 Git 클라이언트로 로컬 컴퓨터로 복제할 때 로컬 리포지토리에서 기본 브랜치로 사용될 브랜치입니다. 또한 워크플로 YAML 파일을 저장하고 문제에 대한 정보를 저장하는 워크플로를 생성할 때 사용되는 기본값입니다. CodeCatalyst에서 검색을 사용하는 경우 리포지토리의 기본 브랜치만 검색됩니다. 기본 브랜치는 프로젝트의 여러 측면에서 기본이므로, 기본 브랜치로 지정된 브랜치는 삭제할 수 없습니다. 그러나 다른 브랜치를 기본 브랜치로 사용하도록 선택할 수 있습니다. 이렇게 하면 이전 기본 브랜치에 적용된 모든 [브랜치 규칙](#)이 기본 브랜치로 지정한 브랜치에 자동으로 적용됩니다.

Note

CodeCatalyst 프로젝트에서 소스 리포지토리의 기본 브랜치를 변경하려면 프로젝트 관리자 역할이 있어야 합니다. 이는 연결된 리포지토리에는 적용되지 않습니다.

리포지토리의 기본 브랜치를 보고 변경하려면

1. 리포지토리가 있는 프로젝트로 이동합니다.
2. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.

기본 브랜치를 포함하여 설정을 보려는 리포지토리를 선택합니다.
3. 리포지토리의 개요 페이지에서 추가를 선택한 다음 설정 관리를 선택합니다.
4. 기본 브랜치에서 기본 브랜치로 지정된 브랜치의 이름은 이름 옆에 기본이라는 레이블과 함께 표시됩니다. 이 동일한 레이블이 브랜치의 브랜치 목록에서 브랜치 이름 옆에 나타납니다.
5. 기본 브랜치를 변경하려면 편집을 선택합니다.

Note

기본 브랜치를 변경하려면 프로젝트에 프로젝트 관리자 역할이 있어야 합니다.

6. 드롭다운 목록에서 기본 브랜치를 만들 브랜치의 이름을 선택한 다음, 저장을 선택합니다.

브랜치 규칙을 사용하여 브랜치에 대한 작업 관리

브랜치를 생성할 때, 해당 역할에 대한 권한을 기반으로 해당 브랜치에 대한 특정 작업이 허용됩니다. 브랜치 규칙을 구성하여 특정 브랜치에 허용되는 작업을 변경할 수 있습니다. 브랜치 규칙은 사용자가 프로젝트에서 수행하는 역할을 기반으로 합니다. 브랜치에 커밋 푸시와 같은 사전 정의된 일부 작업을 프로젝트에서 특정 역할을 가진 사용자로 제한하도록 선택할 수 있습니다. 이를 통해 특정 작업을 수행할 수 있는 역할을 제한하여 프로젝트의 특정 브랜치를 보호할 수 있습니다. 예를 들어, 프로젝트 관리자 역할이 있는 사용자만 해당 브랜치에 병합하거나 푸시하도록 브랜치 규칙을 구성하면, 프로젝트의 다른 역할이 있는 사용자는 해당 브랜치의 코드를 변경할 수 없습니다.

브랜치에 대한 규칙 생성이 미치는 모든 영향을 신중하게 고려해야 합니다. 예를 들어, 브랜치에 대한 푸시를 프로젝트 관리자 역할이 있는 사용자만 할 수 있도록 제한하면, 기고자 역할이 있는 사용자는 해당 브랜치에서 워크플로를 생성하거나 편집할 수 없습니다. 이는 워크플로 YAML이 해당 브랜치에

저장되고 허용된 역할이 아닌 사용자는 YAML에 대한 변경 사항을 커밋하거나 푸시할 수 없기 때문입니다. 가장 좋은 방법은 브랜치 규칙을 생성한 후 테스트하여 의도하지 않은 영향이 나타나지 않는지 확인하는 것입니다. 풀 요청에 대한 승인 규칙과 함께 브랜치 규칙을 사용할 수도 있습니다. 자세한 내용은 [풀 요청을 승인 규칙과 병합하기 위한 요구 사항 관리](#) 단원을 참조하십시오.

Note

CodeCatalyst 프로젝트의 소스 리포지토리에 대한 브랜치 규칙을 관리하려면 프로젝트 관리자 역할이 있어야 합니다. 연결된 리포지토리에 대한 브랜치 규칙은 생성할 수 없습니다.

역할에 대한 기본 권한보다 더 제한적인 브랜치 규칙만 생성할 수 있습니다. 프로젝트에서 사용자의 역할이 허용하는 것보다 더 허용적인 브랜치 규칙은 생성할 수 없습니다. 예를 들어, 검토자 역할이 있는 사용자가 브랜치로 푸시할 수 있도록 허용하는 브랜치 규칙을 생성할 수 없습니다.

소스 리포지토리의 기본 브랜치에 적용되는 브랜치 규칙은 다른 브랜치에 적용되는 브랜치 규칙과 약간 다르게 작동합니다. 기본 브랜치에 적용되는 모든 규칙은 기본 브랜치로 지정한 모든 브랜치에 자동으로 적용됩니다. 이전에 기본 브랜치로 설정된 브랜치는 삭제 방지 기능이 더 이상 없다는 점을 제외하면 기존 규칙이 적용된 상태로 유지됩니다. 이 보호 기능은 현재 기본 브랜치에만 적용됩니다.

브랜치 규칙에는 표준 및 사용자 지정이라는 두 가지 상태가 있습니다. 표준은 브랜치에서 허용되는 작업이 브랜치 작업에 대해 CodeCatalyst에서 사용자가 가진 역할에 대한 권한과 일치하는 작업임을 나타냅니다. 어떤 역할에 어떤 권한이 있는지에 대한 자세한 내용은 [사용자 역할로 액세스 권한 부여](#) 섹션을 참조하세요. 사용자 지정은 하나 이상의 브랜치 작업에 프로젝트의 사용자 역할이 부여한 기본 권한과 다른 작업을 수행할 수 있는 특정 역할 목록이 있는 작업이 있음을 나타냅니다.

Note

브랜치에 대해 하나 이상의 작업을 제한하는 브랜치 규칙을 생성하면, 프로젝트 관리자 역할이 있는 사용자만 해당 브랜치를 삭제할 수 있도록 브랜치 삭제 작업이 자동으로 설정됩니다.

다음 표에는 브랜치에서 이러한 작업을 수행할 수 있는 역할에 대한 작업과 기본 설정이 나열되어 있습니다.

브랜치 작업 및 역할

브랜치 작업	브랜치 규칙이 적용되지 않을 때 이 작업을 수행할 수 있는 역할
브랜치에 병합(풀 요청을 브랜치에 병합하는 것 포함)	프로젝트 관리자, 기고자
브랜치로 푸시	프로젝트 관리자, 기고자
브랜치 삭제	프로젝트 관리자, 기고자
브랜치 삭제(기본 브랜치)	허용되지 않음

브랜치 규칙은 삭제할 수 없지만, 브랜치에서 이 작업을 수행할 수 있는 모든 역할의 작업을 허용하도록 업데이트하여 규칙을 효과적으로 제거할 수 있습니다.

Note

CodeCatalyst 프로젝트의 소스 리포지토리에 대한 브랜치 규칙을 구성하려면 프로젝트 관리자 역할이 있어야 합니다. 이는 연결된 리포지토리에 적용되지 않습니다. 연결된 리포지토리는 CodeCatalyst의 브랜치 규칙을 지원하지 않습니다.

리포지토리의 브랜치 규칙을 보고 편집하려면

1. 리포지토리가 있는 프로젝트로 이동합니다.
2. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.

브랜치 규칙을 보려는 리포지토리를 선택합니다.

3. 리포지토리의 개요 페이지에서 브랜치를 선택합니다.
4. 브랜치 규칙 열에서 리포지토리의 각 브랜치에 대한 규칙 상태를 확인합니다. 표준은 브랜치 작업 규칙이 소스 리포지토리에서 생성된 모든 브랜치의 기본 규칙이며, 프로젝트의 해당 역할에 부여된 권한과 일치함을 나타냅니다. 사용자 지정은 하나 이상의 브랜치 작업에 해당 브랜치에 대해 허용되는 하나 이상의 작업을 다른 역할 집합으로 제한하는 규칙이 있음을 나타냅니다.

브랜치에 대한 브랜치 규칙의 세부 정보를 보려면, 검토하려는 브랜치 옆에 있는 표준 또는 사용자 지정이라는 단어를 선택합니다.

5. 브랜치 규칙을 생성하거나 변경하려면 설정 관리를 선택합니다. 소스 리포지토리의 설정 페이지에 있는 브랜치 규칙에서 편집을 선택합니다.
6. Branch name의 드롭다운 목록에서 규칙을 구성하려는 브랜치의 이름을 선택합니다. 허용되는 각 작업 유형에 대해, 드롭다운 목록에서 해당 작업을 수행하도록 허용할 역할을 선택한 다음, 저장을 선택합니다.

브랜치에 대한 Git 명령

Git을 사용하여 컴퓨터(로컬 리포지토리) 또는 개발 환경에 있는 소스 리포지토리의 복제본에서 브랜치를 생성, 관리 및 삭제한 다음 변경 사항을 CodeCatalyst 소스 리포지토리(원격 리포지토리)에 커밋하고 푸시할 수 있습니다. 예제:

브랜치에 대한 공통 Git 명령

로컬 리포지토리의 모든 브랜치를 나열하며, 현재 브랜치 옆에 별표(*)를 표시합니다.	<code>git branch</code>
원격 리포지토리에 있는 모든 기존 브랜치에 대한 정보를 로컬 리포지토리로 가져옵니다.	<code>git fetch</code>
로컬 리포지토리의 브랜치와 로컬 리포지토리의 원격 추적 브랜치를 모두 나열합니다.	<code>git branch -a</code>
로컬 리포지토리의 원격 추적 브랜치만 나열합니다.	<code>git branch -r</code>
지정된 브랜치 이름을 사용하여 로컬 리포지토리에 브랜치를 생성합니다. 이 브랜치는 커밋하고 변경 내용을 푸시할 때까지 원격 리포지토리에 나타나지 않습니다.	<code>git branch <i>branch-name</i></code>
지정된 브랜치 이름을 사용하여 로컬 리포지토리에 브랜치를 만든 다음 해당 브랜치로 전환합니다.	<code>git checkout -b <i>branch-name</i></code>

지정된 브랜치 이름을 사용하여 로컬 리포지토리의 다른 브랜치로 전환합니다.

```
git checkout other-branch-name
```

원격 리포지토리에 대해 로컬 리포지토리의 지정된 닉네임과 지정된 브랜치 이름을 사용하여 로컬 리포지토리에서 원격 리포지토리로 브랜치를 푸시합니다. 또한 로컬 리포지토리에 있는 브랜치에 대한 업스트림 추적 정보도 설정합니다.

```
git push -u remote-name branch-name
```

로컬 리포지토리의 다른 브랜치에서 변경한 내용을 로컬 리포지토리의 현재 브랜치에 병합합니다.

```
git merge from-other-branch-name
```

병합되지 않은 작업이 포함되어 있지 않는 한 로컬 리포지토리에서 브랜치를 삭제합니다.

```
git branch -d branch-name
```

로컬 리포지토리가 원격 리포지토리에 대해 지정한 닉네임과 지정된 브랜치 이름을 사용하여 원격 리포지토리의 새 브랜치를 삭제합니다. (콜론(:) 사용에 주의하세요.) 또는 명령의 일부로 `--delete`를 지정합니다.

```
git push remote-name :branch-name
```

```
git push remote-name --delete branch-name
```

자세한 내용은 Git 설명서를 참조하세요.

브랜치 및 세부 정보 보기

특정 브랜치에 대한 파일, 폴더 및 가장 최근 커밋을 비롯한 원격 브랜치에 대한 정보를 Amazon CodeCatalyst 콘솔에서 볼 수 있습니다. Git 명령과 로컬 운영 체제를 사용하여 원격 및 로컬 브랜치에 대한 이 정보를 볼 수도 있습니다.

브랜치 보기(콘솔)

1. CodeCatalyst 콘솔에서 브랜치를 보려는 소스 리포지토리가 포함된 프로젝트로 이동합니다. 코드를 선택하고 소스 리포지토리를 선택한 다음 소스 리포지토리를 선택합니다.
2. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.

브랜치를 보려는 리포지토리를 선택합니다.

3. 리포지토리의 기본 브랜치가 표시됩니다. 브랜치에 있는 파일 및 폴더 목록, 가장 최근 커밋에 대한 정보, README.md 파일의 내용(브랜치에 있는 경우)을 볼 수 있습니다. 다른 브랜치에 대한 정보를 보려면 리포지토리의 브랜치 드롭다운 목록에서 해당 브랜치를 선택합니다.
4. 리포지토리의 모든 브랜치를 보려면 모두 보기를 선택합니다. 브랜치 페이지에는 각 브랜치의 이름, 가장 최근 커밋 및 규칙에 대한 정보가 표시됩니다.

Git 및 운영 체제를 사용하여 브랜치 및 세부 정보를 보는 방법에 대한 자세한 내용은 Git 설명서 및 운영 체제 설명서, [Common Git commands for branches](#)를 참조하세요.

브랜치 삭제

브랜치가 더 이상 필요하지 않으면 삭제할 수 있습니다. 예를 들어, 특성이 변경된 브랜치를 기본 브랜치에 병합했는데 해당 특성이 릴리스된 경우, 변경 사항이 이미 기본 브랜치에 포함되어 있으므로 원래 특성이 있는 브랜치를 삭제하고 싶을 수 있습니다. 브랜치 수를 적게 유지하면 사용자가 작업하려는 변경 사항이 포함된 브랜치를 쉽게 찾을 수 있습니다. 브랜치를 삭제하면 사용자가 변경 내용을 가져와 동기화할 때까지 해당 브랜치의 복사본이 로컬 컴퓨터의 리포지토리 복제본에 남아 있습니다.

브랜치 삭제(콘솔)

1. 리포지토리가 있는 프로젝트로 이동합니다.
2. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.

브랜치를 삭제할 리포지토리를 선택합니다.

3. 리포지토리 개요 페이지에서 브랜치 이름 옆의 드롭다운 선택기를 선택한 다음 모두 보기를 선택합니다.
4. 삭제하려는 브랜치를 선택한 다음 브랜치 삭제를 선택합니다.

Note

리포지토리의 기본 브랜치는 삭제할 수 없습니다.

5. 확인 대화 상자가 표시됩니다. 리포지토리, 진행 중인 풀 리퀘스트 수, 브랜치와 관련된 워크플로 수를 보여줍니다.
6. 브랜치 삭제를 확인하려면 텍스트 상자에 delete를 입력한 다음 삭제를 선택합니다.

Git을 사용하여 브랜치를 삭제할 수도 있습니다. 자세한 내용은 [브랜치에 대한 공통 Git 명령](#) 섹션을 참조하세요.

Amazon CodeCatalyst에서 소스 코드 파일 관리

Amazon CodeCatalyst에서 파일은 버전이 관리되고 독립적으로 작동하는 정보입니다. 본래의 사용자 외에, 해당 파일이 저장된 소스 리포지토리와 브랜치의 다른 사용자들도 이용할 수 있습니다. 사용자는 디렉터리 구조를 활용하여 리포지토리 파일을 정리할 수 있습니다. CodeCatalyst는 파일에 대해 커밋된 모든 변경 내용을 자동으로 추적합니다. 사용자는 서로 다른 버전의 파일을 서로 다른 리포지토리 브랜치에 저장할 수 있습니다.

소스 리포지토리에서 여러 파일을 추가하거나 편집하려면 Git 클라이언트, 개발 환경 또는 통합 개발 환경(IDE)을 사용할 수 있습니다. 단일 파일을 추가하거나 편집하려면 CodeCatalyst 콘솔을 사용할 수 있습니다.

주제

- [파일 생성 또는 추가](#)
- [파일 보기](#)
- [파일 변경 내역 보기](#)
- [파일 편집](#)
- [파일 이름 바꾸기 또는 삭제](#)

파일 생성 또는 추가

Amazon CodeCatalyst 콘솔, 개발 환경, 연결된 통합 개발 환경(IDE) 또는 Git 클라이언트를 사용하여 소스 리포지토리에 파일을 생성하고 추가할 수 있습니다. CodeCatalyst 콘솔에는 파일을 생성하기 위한 코드 편집기가 포함되어 있습니다. 이 편집기는 리포지토리의 브랜치에서 README.md 파일과 같은 간단한 파일을 생성하거나 편집하는 편리한 방법입니다. 둘 이상의 파일에서 작업할 때는 [개발 환경 생성](#)을 고려하세요.

소스 리포지토리에서 개발 환경 생성

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
3. 코드 작업을 수행할 소스 리포지토리를 선택합니다.
4. 개발 환경 생성을 선택합니다.

5. 드롭다운 메뉴에서 지원되는 IDE를 선택합니다. 자세한 정보는 [개발 환경에 지원되는 통합 개발 환경](#)을 참조하세요.
6. 다음 중 하나를 수행하세요.
 - 기존 브랜치에서 작업을 선택한 다음 기존 브랜치 드롭다운 메뉴에서 브랜치를 선택합니다.
 - 새 브랜치에서 작업을 선택하고, 브랜치 이름 필드에 브랜치 이름을 입력하고, 다음에서 브랜치 생성 드롭다운 메뉴에서 새 브랜치를 만들 브랜치를 선택합니다.
7. 선택적으로 개발 환경의 이름을 추가하거나 구성을 편집합니다.
8. 생성(Create)을 선택합니다.

CodeCatalyst 콘솔에서 파일 생성

1. 파일을 생성하려는 프로젝트로 이동합니다. 리포지토리로 이동하는 방법에 대한 자세한 내용은 [소스 리포지토리 보기](#) 섹션을 참조하세요.
2. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.

파일을 생성할 리포지토리를 선택합니다.

3. (선택 사항) 기본 브랜치가 아닌 다른 브랜치에 파일을 만들려면 파일을 만들 브랜치를 선택합니다.
4. 새 파일 생성을 선택합니다.
5. 파일 이름에 파일 이름을 입력합니다. 편집기에 파일 내용을 추가합니다.

Tip

브랜치 루트의 하위 폴더 또는 하위 디렉터리에서 파일을 생성하려면 해당 구조를 파일 이름에 포함시킵니다.

선택 사항에 만족하는 경우, 확인을 선택합니다.

6. 파일 이름에서 파일 이름을 검토하고 원하는 대로 변경합니다. 필요에 따라 브랜치의 사용 가능한 브랜치 목록에서 파일을 만들 브랜치를 선택합니다. 커밋 메시지에 선택 사항으로 이 변경을 수행한 이유에 대한 간략하지만 유익한 설명을 입력합니다. 소스 리포지토리에 파일을 추가하는 커밋에 대한 기본 커밋 정보로 표시됩니다.
7. 커밋을 선택하여 파일을 커밋하고 소스 리포지토리로 푸시합니다.

로컬 컴퓨터에 복제하고 Git 클라이언트 또는 연결된 통합 개발 환경(IDE)을 사용하여 파일과 변경 사항을 푸시하여 소스 리포지토리에 파일을 추가할 수도 있습니다.

Note

Git 하위 모듈을 추가하려면 Git 클라이언트 또는 개발 환경을 사용하고 `git submodule add` 명령을 실행해야 합니다. CodeCatalyst 콘솔에서 Git 하위 모듈을 추가하거나 보거나 풀 요청에서 Git 하위 모듈의 차이를 볼 수 없습니다. Git 하위 모듈에 대한 자세한 내용은 [Git 설명서](#)를 참조하세요.

Git 클라이언트 또는 연결된 통합 개발 환경(IDE)을 사용하여 파일 추가

1. 소스 리포지토리를 로컬 컴퓨터에 복제합니다. 자세한 내용은 [소스 리포지토리 복제](#) 섹션을 참조하세요.
2. 로컬 리포지토리에 파일을 생성하거나 로컬 리포지토리에 파일을 복사합니다.
3. 다음 중 하나를 수행하여 커밋을 생성하고 푸시합니다.
 - Git 클라이언트를 사용하는 경우 터미널 또는 명령줄에서 `git add` 명령을 실행하고 추가하려는 파일의 이름을 지정합니다. 또는 추가되거나 변경된 모든 파일을 추가하려면 `git add` 명령 뒤에 마침표 또는 마침표를 붙여 현재 디렉터리 수준의 모든 변경 사항(마침표 하나)을 포함할지, 현재 디렉터리와 모든 하위 디렉터리의 모든 변경 사항(마침표 둘)을 포함할지 표시합니다. 변경 사항을 커밋하려면 `git commit -m` 명령을 실행하고 커밋 메시지를 제공합니다. 변경 사항을 CodeCatalyst의 소스 리포지토리로 푸시하려면 `git push`를 실행합니다. Git 설명에 대한 자세한 내용은 Git 설명서 및 [브랜치에 대한 Git 명령](#)을 참조하세요.
 - 개발 환경 또는 IDE를 사용하는 경우 파일을 생성하고 IDE에 파일을 추가한 다음 커밋하고 변경 사항을 푸시합니다. 자세한 내용은 [CodeCatalyst에서 개발 환경으로 코드 작성 및 수정](#) 또는 IDE 관련 문서를 참조하세요.

파일 보기

Amazon CodeCatalyst 콘솔에서 소스 리포지토리에 있는 파일을 볼 수 있습니다. 기본 브랜치 및 다른 브랜치에서 파일을 볼 수 있습니다. 파일 내용은 보려는 브랜치에 따라 다를 수 있습니다.

CodeCatalyst 콘솔에서 파일 보기

1. 파일을 보려는 프로젝트로 이동합니다. 자세한 내용은 [소스 리포지토리 보기](#) 섹션을 참조하세요.

2. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.

파일을 보려는 리포지토리를 선택합니다.
3. 기본 브랜치에 대한 파일 및 폴더 목록이 표시됩니다. 파일은 종이 아이콘으로 표시되고 폴더는 폴더 아이콘으로 표시됩니다.
4. 해결 방법:
 - 다른 브랜치에 있는 파일과 폴더를 보려면 브랜치 목록에서 해당 브랜치를 선택합니다.
 - 폴더를 확장하려면 목록에서 폴더를 선택합니다.
5. 특정 파일의 콘텐츠를 보려면 목록에서 해당 파일을 선택합니다. 파일의 내용이 브랜치에 표시됩니다. 다른 브랜치에 있는 파일의 내용을 보려면 브랜치 선택기에서 원하는 브랜치를 선택합니다.

i Tip

파일의 내용을 볼 때 파일 보기에서 볼 추가 파일을 선택할 수 있습니다. 파일을 편집하려면 편집을 선택합니다.

콘솔에서 여러 파일을 볼 수 있습니다. Git 클라이언트 또는 통합 개발 환경(IDE)을 사용하여 로컬 컴퓨터에 복제한 파일을 볼 수도 있습니다. 자세한 내용은 Git 클라이언트나 IDE 관련 설명서를 참조하세요.

i Note

CodeCatalyst 콘솔에서는 Git 하위 모듈을 볼 수 없습니다. Git 하위 모듈에 대한 자세한 내용은 [Git 설명서](#)를 참조하세요.

파일 변경 내역 보기

소스 리포지토리의 파일 변경 내역은 Amazon CodeCatalyst 콘솔에서 볼 수 있습니다. 이렇게 하면 파일 이력을 보도록 선택한 브랜치에 대한 다양한 커밋에 의해 파일이 변경된 내용을 이해하는 데 도움이 될 수 있습니다. 예를 들어, 소스 리포지토리의 **main** 브랜치에서 **readme.md** 파일의 변경 내역을 보면 해당 브랜치에서 해당 파일에 대한 변경 사항이 포함된 커밋 목록이 표시됩니다.

Note

CodeCatalyst 콘솔에서 연결된 리포지토리의 파일 기록을 볼 수 없습니다.

CodeCatalyst 콘솔에서 파일 기록 보기

1. 파일 기록을 보려는 프로젝트로 이동합니다. 자세한 내용은 [소스 리포지토리 보기](#) 섹션을 참조하세요.
2. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
3. 파일 기록을 보려는 리포지토리를 선택합니다. 파일 기록을 보려는 브랜치를 선택한 다음 목록에서 파일을 선택합니다. 내역 보기를 선택합니다.
4. 지정한 브랜치에서 이 파일에 대한 변경 사항이 포함된 커밋 목록을 검토합니다. 특정 커밋에 포함된 변경 사항의 세부 정보를 보려면 목록에서 해당 커밋에 대한 커밋 메시지를 선택합니다. 해당 커밋과 상위 커밋의 차이점이 표시됩니다.
5. 다른 브랜치에 있는 파일의 변경 기록을 검토하려면 브랜치 선택기를 사용하여 해당 브랜치로 보기를 변경하고 파일 목록에서 파일을 선택한 다음 기록 보기를 선택합니다.

Note

CodeCatalyst 콘솔에서는 Git 하위 모듈에 대한 변경 기록을 볼 수 없습니다. Git 하위 모듈에 대한 자세한 내용은 [Git 설명서](#)를 참조하세요.

파일 편집

Amazon CodeCatalyst 콘솔에서 개별 파일을 편집할 수 있습니다. 한 번에 여러 파일을 편집하려면 개발 환경을 만들거나 리포지토리를 복제하고 Git 클라이언트 또는 통합 개발 환경(IDE)을 사용하여 변경하세요. 자세한 내용은 [CodeCatalyst에서 개발 환경으로 코드 작성 및 수정](#) 또는 [소스 리포지토리 복제](#)를 참조하세요.

CodeCatalyst 콘솔에서 파일 편집 보기

1. 파일을 편집하려는 프로젝트로 이동합니다. 리포지토리로 이동하는 방법에 대한 자세한 내용은 [소스 리포지토리 보기](#) 섹션을 참조하세요.

2. 파일을 편집할 리포지토리를 선택합니다. 브랜치 보기를 선택한 다음 작업할 브랜치를 선택합니다. 해당 브랜치의 파일 및 폴더 목록에서 파일을 선택합니다.

파일의 내용이 표시됩니다.

3. 편집을 선택합니다.
4. 편집기에서 파일의 내용을 편집한 다음 커밋을 선택합니다. 선택 사항으로 변경 사항 커밋에서 커밋 메시지에 변경 사항에 대한 자세한 정보를 추가합니다. 선택 사항에 만족하는 경우, 확인을 선택합니다.

파일 이름 바꾸기 또는 삭제

개발 환경에서, 컴퓨터의 로컬 환경에서 또는 통합 개발 환경(IDE)에서 파일 이름을 바꾸거나 삭제할 수 있습니다. 파일 이름을 변경하거나 삭제한 후, 그 변경 사항을 소스 리포지토리에 커밋하고 푸시합니다. Amazon CodeCatalyst 콘솔에서는 파일 이름을 변경하거나 삭제할 수 없습니다.

Amazon CodeCatalyst에서 풀 요청을 사용하여 코드 검토

풀 요청은 나와 프로젝트 멤버가 검토하고, 주석을 추가하며, 한 브랜치에서 다른 브랜치로 코드를 변경하기 위한 기본적인 방법입니다. 풀 요청을 사용하여 중요하지 않은 변경 내용이나 수정 사항, 중요 기능 추가 또는 릴리스된 소프트웨어의 새 버전에 대한 코드 변경 내용을 공동으로 검토할 수 있습니다. 문제를 사용하여 프로젝트의 작업을 추적하는 경우 특정 이슈를 풀 요청에 연결하여 풀 요청의 코드 변경으로 어떤 문제가 해결되고 있는지 추적할 수 있습니다. 풀 요청을 생성, 업데이트, 설명, 병합 또는 종료하면 풀 요청의 작성자는 물론 풀 요청의 필수 또는 선택 검토자에게 자동으로 이메일이 전송됩니다.

Tip

프로파일의 일부로서 이메일을 수신할 풀 요청 이벤트를 구성할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 Slack 및 이메일 알림 전송](#) 섹션을 참조하세요.

풀 요청에는 소스 리포지토리에 검토하려는 코드가 포함된 소스 브랜치와 검토한 코드를 병합하려는 대상 브랜치, 두 개의 브랜치가 필요합니다. 소스 브랜치에는 AFTER 커밋이 포함되며 이 커밋은 대상 브랜치에 병합하려는 변경 내용을 포함합니다. 대상 브랜치에는 BEFORE 커밋이 포함되며, 이 커밋은 풀 요청 브랜치가 대상 브랜치에 병합되기 전 코드의 상태를 나타냅니다.

Note

풀 요청을 생성하는 동안 나타나는 차이는 소스 브랜치의 최신 커밋과 대상 브랜치의 최신 커밋의 차이입니다. 풀 요청을 만들면 선택한 풀 요청의 수정본과 풀 요청을 만들 때 대상 브랜치의 끝이었던 커밋의 차이점이 표시됩니다. Git의 차이점 및 병합 기반에 대한 자세한 내용은 Git 설명서의 [git-merge-base](#)를 참조하세요.

특정 소스 리포지토리 및 브랜치에 대한 풀 요청이 생성되는 동안 프로젝트 작업의 일부로 풀 요청을 만들고, 보고, 검토하고, 닫을 수 있습니다. 풀 요청을 보고 작업하기 위해 소스 리포지토리를 볼 필요는 없습니다. 풀 요청 상태는 생성할 때 열기로 설정됩니다. 풀 요청은 상태가 병합됨으로 변경되는 CodeCatalyst 콘솔에서 병합하거나 상태가 종결됨으로 변경될 때까지 열려 있습니다.

코드를 검토했으면 다음 방법 중 하나를 사용하여 풀 요청 상태를 변경할 수 있습니다.

- CodeCatalyst 콘솔에서 풀 요청을 병합합니다. 풀 요청의 소스 브랜치에 있는 코드는 대상 브랜치로 병합됩니다. 풀 요청 상태가 병합됨으로 변경됩니다. 다시 열기로 변경할 수 없습니다.
- 브랜치를 로컬로 병합하고 변경 사항을 푸시한 다음 CodeCatalyst 콘솔에서 풀 요청을 닫습니다.
- 병합하지 않고 풀 요청을 닫으려면 CodeCatalyst 콘솔을 사용하세요. 이렇게 하면 상태가 종료됨으로 변경되고 소스 브랜치의 코드가 대상 브랜치로 병합되지 않습니다.

풀 요청을 생성하려면 먼저 다음을 수행해야 합니다.

- 검토하려는 코드 변경 사항을 브랜치(소스 브랜치)에 커밋하고 푸시합니다.
- 프로젝트에 대한 알림을 설정하여 풀 리퀘스트를 만들 때 실행되는 워크플로에 대해 다른 사용자가 알림을 받을 수 있도록 하세요. (이 단계는 선택 사항이며, 권장 사항은 아닙니다.)

주제

- [풀 요청 생성](#)
- [풀 요청 보기](#)
- [풀 요청을 승인 규칙과 병합하기 위한 요구 사항 관리](#)
- [풀 요청 검토](#)
- [풀 요청 업데이트](#)
- [풀 요청 병합](#)
- [풀 요청 닫기](#)

풀 요청 생성

풀 요청을 생성하면 변경 내용을 다른 브랜치에 병합하기 전에 다른 사용자가 코드 변경을 보고 검토할 수 있습니다. 먼저 코드 변경을 위한 브랜치를 생성합니다. 이 브랜치를 풀 요청의 소스 브랜치라고도 합니다. 리포지토리에 변경 사항을 커밋하고 푸시한 후, 소스 브랜치의 내용을 대상 브랜치의 내용과 비교하는 풀 요청을 생성할 수 있습니다.

Amazon CodeCatalyst 콘솔의 특정 브랜치, 풀 요청 페이지 또는 프로젝트 개요에서 풀 요청을 생성할 수 있습니다. 특정 브랜치에서 풀 요청을 생성하면 풀 요청 생성 페이지에 리포지토리 이름과 소스 브랜치가 자동으로 제공됩니다. 풀 요청을 생성하면 풀 요청에 대한 업데이트와 풀 요청이 병합되거나 종료되는 시기를 알려주는 이메일이 자동으로 수신됩니다.

Note

풀 요청을 생성하는 동안 나타나는 차이는 소스 브랜치의 최신 커밋과 대상 브랜치의 최신 커밋의 차이입니다. 풀 요청이 생성되면, 표시된 차이는 선택한 풀 요청의 수정본과 풀 요청을 생성할 때 대상 브랜치의 가장 최신 커밋 간에 발생합니다. Git의 차이점 및 병합 기반에 대한 자세한 내용은 Git 설명서의 [git-merge-base](#)를 참조하세요.

풀 요청을 생성할 때 설명 쓰기 기능을 사용하여 Amazon Q가 풀 요청에 포함된 변경 사항에 대한 설명을 자동으로 생성하도록 할 수 있습니다. 이 옵션을 선택하면 Amazon Q는 코드 변경 사항이 포함된 소스 브랜치와 이러한 변경 사항을 병합할 대상 브랜치 간의 차이를 분석합니다. 그런 다음 이러한 변경 사항에 대한 요약과 이러한 변경의 의도 및 효과에 대한 최상의 해석을 생성합니다. 이 기능은 CodeCatalyst 풀 요청에 대해 미국 서부(오리건) 리전에서만 사용할 수 있습니다. 연결된 리포지토리의 풀 요청에는 설명 쓰기 기능을 사용할 수 없습니다.

Note

Note

Amazon Bedrock:implements 자동 침해 탐지로 구동됩니다. AWS <https://docs.aws.amazon.com//bedrock/latest/userguide/abuse-detection.html> 소프트웨어 개발용 Amazon Q Developer Agent의 설명 쓰기, 콘텐츠 요약 생성, 권장 작업, Amazon Q를 사용하여 프로젝트에 기능을 생성하거나 추가, Amazon Q에 문제 할당 기능은

Amazon Bedrock에 구축되므로, 사용자는 Amazon Bedrock에서 구현된 제어 기능을 최대한 활용하여 안전, 보안 및 인공 지능(AI)을 책임 있게 사용할 수 있습니다.

풀 요청을 생성하려면

1. 프로젝트로 이동합니다.
2. 다음 중 하나를 수행하세요.
 - 탐색 창에서 코드를 선택하고 풀 요청을 선택한 다음 풀 요청 생성을 선택합니다.
 - 리포지토리 홈 페이지에서 추가를 선택한 다음 풀 요청 생성을 선택합니다.
 - 프로젝트 페이지에서 풀 요청 생성을 선택합니다.
3. 소스 리포지토리에서 지정된 소스 리포지토리가 커밋된 코드가 포함된 리포지토리인지 확인합니다. 이 옵션은 리포지토리의 메인 페이지에서 풀 요청을 생성하지 않은 경우에만 나타납니다.
4. 대상 브랜치에서 코드를 검토한 후 병합할 브랜치를 선택합니다.
5. 소스 브랜치에서 커밋된 코드가 포함된 브랜치를 선택합니다.
6. 풀 요청 제목에 다른 사용자가 검토해야 할 사항과 이유를 이해하는 데 도움이 되는 제목을 입력합니다.
7. (선택 사항) 풀 요청 설명에서 문제에 대한 연결 또는 변경 사항에 대한 설명과 같은 정보를 제공합니다.

Tip

CodeCatalyst가 풀 요청에 포함된 변경 사항에 대한 설명을 자동으로 생성하도록 설명 쓰기를 선택할 수 있습니다. 자동으로 생성된 설명은 풀 요청에 추가한 후에 변경할 수 있습니다.

이 기능을 사용하려면 스페이스에 생성형 AI 기능을 활성화해야 하며 연결된 리포지토리의 풀 요청에 사용할 수 없습니다. 자세한 내용은 [생성형 AI 기능 관리](#)를 참조하세요.

8. (선택 사항) 문제에서 문제 연결을 선택한 다음 목록에서 문제를 선택하거나 해당 ID를 입력합니다. 문제를 연결 해제하려면 연결 해제 아이콘을 선택합니다.
9. (선택 사항) 필수 검토자에서 필수 검토자 추가를 선택합니다. 프로젝트 멤버 목록에서 추가할 멤버를 선택합니다. 필수 검토자는 풀 요청을 대상 브랜치에 병합하기 전에 변경 사항을 승인해야 합니다.

Note

검토자를 추가할 때 필수 검토자이면서 동시에 선택적 검토자로 추가할 수 없습니다. 자신을 검토자로 추가할 수 없습니다.

10. (선택 사항) 선택적 검토자에서 선택적 검토자 추가를 선택합니다. 프로젝트 멤버 목록에서 추가할 멤버를 선택합니다. 선택적 검토자는 풀 요청을 대상 브랜치에 병합하기 전에 변경 사항을 요구 사항으로 승인할 필요가 없습니다.
11. 브랜치 간의 차이점을 검토합니다. 풀 요청에 표시되는 차이점은 소스 브랜치의 수정본과 풀 요청이 생성된 시점의 대상 브랜치의 헤드 커밋인 병합 기반 간의 변경 사항입니다. 변경 사항이 표시되지 않으면 브랜치가 동일하거나 소스와 대상 모두에 대해 동일한 브랜치를 선택했을 수 있습니다.
12. 풀 요청에 검토하려는 코드와 변경 사항이 포함되어 있다고 판단되면 생성을 선택합니다.

Note

풀 요청을 생성한 후 설명을 추가할 수 있습니다. 풀 요청 또는 파일의 개별 줄 및 풀 요청 전반에 설명을 추가할 수 있습니다. @ 기호와 파일 이름을 사용하여 파일과 같은 리소스에 대한 연결을 추가할 수 있습니다.

브랜치에서 풀 요청을 생성하려면

1. 풀 요청을 생성하려는 프로젝트로 이동합니다.
2. 탐색 창에서 소스 리포지토리를 선택한 다음, 검토할 코드 변경 사항이 있는 브랜치를 포함하는 리포지토리를 선택합니다.
3. 기본 브랜치 이름 옆의 드롭다운 화살표를 선택한 다음, 목록에서 원하는 브랜치를 선택합니다. 리포지토리의 모든 브랜치를 보려면 모두 보기를 선택합니다.
4. 추가를 선택한 다음 풀 요청 생성을 선택합니다.
5. 리포지토리 및 소스 브랜치는 미리 선택되어 있습니다. 대상 브랜치에서 코드를 검토한 후 병합할 브랜치를 선택합니다. 풀 요청 제목에 다른 프로젝트 사용자가 검토해야 하는 내용과 이유를 이해하는 데 도움이 되는 제목을 입력합니다. 선택적으로 CodeCatalyst의 관련 문제에 대한 링크를 붙여넣거나 변경 사항에 대한 설명을 추가하는 등 풀 요청 설명에 자세한 정보를 제공합니다.

Note

풀 요청 생성 이벤트를 위해 실행하도록 구성된 워크플로는 풀 요청의 대상 브랜치가 워크플로에 지정된 브랜치 중 하나와 일치하는 경우 풀 요청이 생성된 후 실행됩니다.

6. 브랜치 간의 차이점을 검토합니다. 변경 사항이 표시되지 않으면, 브랜치가 동일하거나, 소스와 대상 모두에 대해 동일한 브랜치를 선택했을 수 있습니다.
7. (선택 사항) 문제에서 문제 연결을 선택한 다음 목록에서 문제를 선택하거나 해당 ID를 입력합니다. 문제를 연결 해제하려면 연결 해제 아이콘을 선택합니다.
8. (선택 사항) 필수 검토자에서 필수 검토자 추가를 선택합니다. 프로젝트 멤버 목록에서 추가할 멤버를 선택합니다. 필수 검토자는 풀 요청을 대상 브랜치에 병합하기 전에 변경 사항을 승인해야 합니다.

Note

검토자를 필수 검토자이면서 동시에 선택적 검토자로 추가할 수 없습니다. 자신을 검토자로 추가할 수 없습니다.

9. (선택 사항) 선택적 검토자에서 선택적 검토자 추가를 선택합니다. 프로젝트 멤버 목록에서 추가할 멤버를 선택합니다. 선택적 검토자는 풀 요청을 대상 브랜치에 병합하기 전에 변경 사항을 승인할 필요가 없습니다.
10. 풀 요청에 검토하려는 변경 사항이 포함되어 있고 필수 검토자가 포함되어 있다고 판단되면, 생성을 선택합니다.

브랜치가 풀 요청의 대상 브랜치와 일치하는 워크플로를 실행하도록 구성된 경우 풀 요청이 생성된 후 풀 요청 세부 정보 영역의 개요에서 해당 워크플로 실행에 대한 정보를 볼 수 있습니다. 자세한 내용은 [워크플로에 트리거 추가](#) 단원을 참조하십시오.

풀 요청 보기

Amazon CodeCatalyst 콘솔에서 프로젝트의 풀 요청을 볼 수 있습니다. 프로젝트 요약 페이지에는 프로젝트에 대한 모든 열린 풀 요청이 표시됩니다. 상태에 관계없이 모든 풀 요청을 보려면, 프로젝트의 풀 요청 페이지로 이동합니다. 풀 요청을 볼 때 생성된 풀 요청의 변경 사항에 대한 모든 설명의 요약을 남기도록 선택할 수 있습니다.

Note**Note**

Amazon Bedrock:implements 자동 침해 탐지로 구동됩니다. AWS <https://docs.aws.amazon.com//bedrock/latest/userguide/abuse-detection.html> 소프트웨어 개발용 Amazon Q Developer Agent의 설명 쓰기, 콘텐츠 요약 생성, 권장 작업, Amazon Q를 사용하여 프로젝트에 기능을 생성하거나 추가, Amazon Q에 문제 할당 기능은 Amazon Bedrock에 구축되므로, 사용자는 Amazon Bedrock에서 구현된 제어 기능을 최대한 활용하여 안전, 보안 및 인공 지능(AI)을 책임 있게 사용할 수 있습니다.

열린 풀 요청을 보려면

1. 풀 요청을 보려는 프로젝트로 이동합니다.
2. 프로젝트 페이지에 풀 요청을 생성한 사람, 풀 요청에 대한 브랜치를 포함하는 리포지토리, 풀 요청이 생성된 날짜에 대한 정보를 포함하여 열린 풀 요청이 표시됩니다. 소스 리포지토리별로 열린 풀 요청 보기를 필터링할 수 있습니다.
3. 모든 풀 요청을 보려면 모두 보기를 선택합니다. 선택기를 사용하여 옵션 중에서 선택할 수 있습니다. 예를 들어, 모든 풀 요청을 보려면 모든 상태 및 모든 작성자를 선택합니다.

또는 탐색 창에서 코드를 선택한 다음 풀 요청을 선택한 다음, 선택기를 사용하여 보기를 구체화합니다.

4. 풀 요청 페이지에서 ID, 제목, 상태 등을 기준으로 풀 요청을 정렬할 수 있습니다. 풀 요청 페이지에 표시되는 정보와 정보의 양을 사용자 지정하려면, 기어 아이콘을 선택합니다.
5. 특정 풀 요청을 보려면 목록에서 선택합니다.
6. 이 풀 요청과 연결된 워크플로 실행의 상태를 보려면 개요를 선택하고, 워크플로 실행에서 풀 요청의 풀 요청 세부 정보 영역의 정보를 검토합니다.

워크플로가 풀 요청 생성 또는 수정 이벤트로 구성되어 있고 워크플로의 대상 브랜치 요구 사항이 풀 요청에 지정된 대상 브랜치와 일치하는 경우, 워크플로 실행이 발생합니다. 자세한 내용은 [워크플로에 트리거 추가](#) 단원을 참조하십시오.

7. 연결된 문제를 보려면 개요를 선택하고 문제에서 풀 요청 세부 정보의 정보를 검토합니다. 연결된 문제를 보려면 목록에서 해당 ID를 선택합니다.

8. (선택 사항) 풀 요청 수정본의 코드 변경에 남아 있는 설명에 대해 요약을 생성하려면 콘텐츠 요약 생성을 선택합니다. 요약에는 전체 풀 요청에 남아 있는 설명이 포함되지 않습니다.

Note

이 기능을 사용하려면 생성형 AI 기능이 스페이스에 활성화되어 있어야 하고, 연결된 리포지토리에는 사용할 수 없으며, 미국 서부(오리건) 리전에서만 사용할 수 있습니다. 자세한 내용은 [생성형 AI 기능 관리](#)를 참조하세요.

9. 풀 요청에서 코드 변경 사항을 보려면 변경을 선택합니다. 풀 요청에서 변경 사항이 있는 파일의 수와 설명이 있는 파일은 변경된 파일에서 빠르게 볼 수 있습니다. 폴더 옆에 표시되는 설명 수는 해당 폴더의 파일에 대한 설명 수를 나타냅니다. 폴더를 확장하여 폴더의 각 파일에 대한 설명 수를 확인합니다. 특정 코드 줄에 남아 있는 모든 설명을 볼 수도 있습니다.

Note

콘솔에 풀 요청의 모든 변경 사항을 표시할 수 있는 것은 아닙니다. 예를 들어, 콘솔에서 Git 하위 모듈을 볼 수 없으므로 풀 요청에서 하위 모듈의 차이를 볼 수 없습니다. 일부 차이는 표시하기에 너무 클 수 있습니다. 자세한 내용은 [CodeCatalyst의 소스 리포지토리 할당량](#) 및 [파일 보기](#) 섹션을 참조하세요.

10. 이 풀 요청에 대한 품질 보고서를 보려면 보고서를 선택합니다.

Note

풀 요청에 나타나는 보고서를 생성하도록 워크플로를 구성해야 합니다. 자세한 내용은 [워크플로를 사용한 테스트](#) 단원을 참조하십시오.

풀 요청을 승인 규칙과 병합하기 위한 요구 사항 관리

풀 요청을 생성할 때 개별 풀 요청에 필수 또는 선택적 검토자를 추가하도록 선택할 수 있습니다. 그러나 특정 대상 브랜치에 병합할 때 모든 풀 요청이 충족해야 하는 요구 사항을 생성할 수도 있습니다. 이러한 요구 사항을 승인 규칙이라고 합니다. 리포지토리의 브랜치에 대해 승인 규칙이 구성됩니다. 대상 브랜치에 승인 규칙이 구성된 풀 요청을 생성할 때는 풀 요청을 해당 브랜치에 병합하기 전에 필요한 검토자의 승인 외에도 해당 규칙의 요구 사항을 충족해야 합니다. 승인 규칙을 생성하면 기본 브랜치와 같은 브랜치에 병합하기 위한 품질 표준을 유지하는 데 도움이 될 수 있습니다.

소스 리포지토리의 기본 브랜치에 적용되는 승인 규칙은 다른 브랜치에 적용되는 승인 규칙과 약간 다르게 작동합니다. 기본 브랜치에 적용되는 모든 규칙은 기본 브랜치로 지정한 모든 브랜치에 자동으로 적용됩니다. 이전에 기본 브랜치로 설정된 브랜치에는 해당 브랜치에 적용된 규칙이 계속 유지됩니다.

승인 규칙을 만들 때는 현재와 미래의 프로젝트 사용자가 해당 규칙을 어떻게 충족할지 고려해야 합니다. 예를 들어 프로젝트에 6명의 사용자가 있고 대상 브랜치에 병합하기 전에 5명의 승인이 필요한 승인 규칙을 만들면, 풀 리퀘스트를 만든 사람을 제외한 모든 사람이 해당 풀 리퀘스트를 승인해야 병합할 수 있는 규칙을 만든 것이 됩니다.

Note

CodeCatalyst 프로젝트에서 승인 규칙을 만들고 관리하려면 프로젝트 관리자 역할이 있어야 합니다. 연결된 리포지토리에 대한 승인 규칙은 생성할 수 없습니다.

승인 규칙을 삭제할 수는 없지만 승인 횟수가 0이 되도록 업데이트하여 규칙을 효과적으로 제거할 수는 있습니다.

풀 요청에 대한 대상 브랜치의 승인 규칙 보기 및 편집

1. 리포지토리가 있는 프로젝트로 이동합니다.
2. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.

승인 규칙을 보려는 리포지토리를 선택합니다.

3. 리포지토리의 개요 페이지에서 브랜치를 선택합니다.
4. 승인 규칙 열에서 보기를 선택하여 리포지토리의 각 브랜치에 대한 규칙 상태를 확인합니다.

최소 승인 수에서 숫자는 풀 요청을 해당 브랜치에 병합하기 위해 필요한 승인 수에 해당합니다.

5. 승인 규칙을 생성하거나 변경하려면 설정 관리를 선택합니다. 소스 리포지토리 설정 페이지에 있는 승인 규칙에서 편집을 선택합니다.

Note

승인 규칙을 편집하려면 프로젝트 관리자 역할이 있어야 합니다.

6. 브랜치의 드롭다운 목록에서 승인 규칙을 구성할 브랜치의 이름을 선택합니다. 최소 승인 수에서 숫자를 입력한 다음 저장을 선택합니다.

풀 요청 검토

Amazon CodeCatalyst 콘솔을 사용하여 풀 요청에 포함된 변경 사항을 공동으로 검토하고 의견을 제시할 수 있습니다. 소스 브랜치와 대상 브랜치 간의 차이 또는 풀 요청의 개정 간의 차이에서 개별 코드 줄에 설명을 추가할 수 있습니다. 풀 요청의 코드 변경에 남아 있는 설명의 요약 생성하여, 다른 사용자가 남긴 피드백을 빠르게 이해할 수 있도록 할 수 있습니다. 코드 작업을 위한 개발 환경을 생성하도록 선택할 수도 있습니다.

Note

Note

Amazon Bedrock:implements 자동 침해 탐지로 구동됩니다. AWS <https://docs.aws.amazon.com//bedrock/latest/userguide/abuse-detection.html> 소프트웨어 개발용 Amazon Q Developer Agent의 설명 쓰기, 콘텐츠 요약 생성, 권장 작업, Amazon Q를 사용하여 프로젝트에 기능을 생성하거나 추가, Amazon Q에 문제 할당 기능은 Amazon Bedrock에 구축되므로, 사용자는 Amazon Bedrock에서 구현된 제어 기능을 최대한 활용하여 안전, 보안 및 인공 지능(AI)을 책임 있게 사용할 수 있습니다.

Tip

프로파일의 일부로서 이메일을 수신할 풀 요청 이벤트를 구성할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 Slack 및 이메일 알림 전송](#) 단원을 참조하십시오.

풀 요청은 풀 요청의 개정과 풀 요청을 생성할 때 대상 브랜치의 최신 커밋이었던 커밋 간의 차이를 보여줍니다. 이를 병합 기반이라고 합니다. Git의 차이점 및 병합 기반에 대한 자세한 내용은 Git 설명서의 [git-merge-base](#)를 참조하세요.

Tip

콘솔에서 작업할 때, 특히 풀 요청이 한동안 열려있는 경우, 검토를 시작하기 전에 풀 요청에 사용할 수 있는 최신 수정본이 있는지 확인하기 위해 브라우저를 새로 고치는 것이 좋습니다.

CodeCatalyst 콘솔에서 풀 요청을 검토하려면

1. 프로젝트로 이동합니다.
2. 다음 중 하나를 수행하여 풀 요청으로 이동합니다.
 - 풀 요청이 프로젝트 페이지에 나열되면, 목록에서 선택합니다.
 - 풀 요청이 프로젝트 페이지에 나열되지 않은 경우, 모두 보기를 선택합니다. 필터와 정렬을 사용하여 풀 요청을 찾은 다음, 목록에서 선택합니다.
 - 탐색 창에서 코드를 선택한 다음, 풀 요청을 선택합니다.
3. 목록에서 검토하려는 풀 요청을 선택합니다. 필터 표시줄에 이름의 일부를 입력하여 풀 요청 목록을 필터링할 수 있습니다.
4. 개요에서 풀 요청의 이름과 제목을 검토할 수 있습니다. 풀 요청 자체에 남아 있는 설명을 생성하고 볼 수 있습니다. 워크플로 실행, 연결된 문제, 검토자, 풀 요청 작성자, 실행 가능한 병합 전략에 대한 정보를 포함하여 풀 요청의 세부 정보를 볼 수도 있습니다.

Note

특정 코드 줄에 남긴 설명은 변경 사항에 표시됩니다.

5. (선택 사항) 전체 풀 요청에 적용되는 설명을 추가하려면 풀 요청에 대한 설명을 확장한 다음 설명 생성 선택합니다.
6. (선택 사항) 이 풀 요청 수정본의 변경 사항에 대해 남아 있는 모든 설명의 요약を 보려면 설명 요약 생성을 선택합니다.

Note

이 기능을 사용하려면 생성형 AI 기능이 스페이스에 대해 활성화되어 있어야 하며 미국 서부(오리건) 리전에서만 사용할 수 있습니다. 자세한 내용은 [생성형 AI 기능 관리](#)를 참조하세요.

7. 변경에서 대상 브랜치와 풀 요청의 최신 수정본 간의 차이점을 확인할 수 있습니다. 수정본이 두 개 이상 있는 경우, 차이에서 비교할 수정본을 변경할 수 있습니다. 수정에 대한 자세한 내용은 [개정](#) 섹션을 참조하세요.

i Tip

풀 요청에서 변경 사항이 있는 파일의 수와 설명이 있는 파일은 변경된 파일에서 빠르게 볼 수 있습니다. 폴더 옆에 표시되는 설명 수는 해당 폴더의 파일에 대한 설명 수를 나타냅니다. 폴더를 확장하여 폴더의 각 파일에 대한 설명 수를 확인합니다.

8. 차이가 표시되는 방식을 변경하려면, 통합 및 분할 중에서 선택합니다.
9. 풀 요청의 행에 설명을 추가하려면, 설명을 작성할 행으로 이동합니다. 해당 행에 나타나는 설명 아이콘을 선택하고 설명을 입력한 다음 저장을 선택합니다.
10. 풀 요청의 수정본 간 또는 소스 브랜치와 대상 브랜치 간의 변경 사항을 보려면, 비교에서 사용 가능한 옵션을 선택합니다. 수정본에서 행에 대한 설명은 해당 수정본에 보존됩니다.
11. 풀 요청 트리거에 대한 코드 적용 범위 보고서를 생성하도록 워크플로를 구성한 경우, 관련 풀 요청에서 행 및 브랜치 적용 범위 조사 결과를 볼 수 있습니다. 코드 적용 범위 조사 결과를 숨기려면 코드 적용 범위 숨기기를 선택합니다. 자세한 내용은 [코드 적용 범위 보고서](#) 단원을 참조하십시오.
12. 풀 요청에서 코드 변경하려는 경우, 풀 요청에서 개발 환경을 생성할 수 있습니다. 개발 환경 생성을 선택합니다. 선택적으로 개발 환경의 이름을 추가하거나 구성을 편집한 다음, 생성을 선택합니다.
13. 보고서에서 이 풀 요청의 품질 보고서를 볼 수 있습니다. 수정본이 두 개 이상 있는 경우, 차이에서 비교할 수정본을 변경할 수 있습니다. 보고서를 이름, 상태, 워크플로, 작업 및 유형별로 필터링할 수 있습니다.

i Note

풀 요청에 나타나는 보고서를 생성하도록 워크플로를 구성해야 합니다. 자세한 내용은 [작업에서 품질 보고서 구성](#) 단원을 참조하십시오.

14. 특정 보고서를 보려면, 목록에서 선택합니다. 자세한 내용은 [워크플로를 사용한 테스트](#) 단원을 참조하십시오.
15. 이 풀 요청의 검토자로 등록되어 있고 변경 사항을 승인하려면, 최신 수정본을 보고 있는지 확인한 다음 승인을 선택합니다.

i Note

모든 필수 검토자는 풀 요청을 승인해야 병합할 수 있습니다.

풀 요청 업데이트

풀 요청을 업데이트하여 다른 프로젝트 멤버가 코드를 더 쉽게 검토할 수 있습니다. 풀 요청을 업데이트하여 검토자, 문제에 대한 링크, 풀 요청 제목 또는 설명을 변경할 수 있습니다. 예를 들어, 풀 요청에 필수 검토자를 변경하여 휴가 중인 사람을 제거하고 다른 사람을 추가할 수 있습니다. 풀 요청 업데이트는 열린 풀 요청의 소스 브랜치에 커밋을 푸시하여 추가 코드를 변경하는 방식으로 수행할 수 있습니다. CodeCatalyst 소스 리포지토리에서 풀 요청의 소스 브랜치로 푸시할 때마다 수정본이 생성됩니다. 프로젝트 멤버는 풀 요청에서 수정본 간의 차이점을 볼 수 있습니다.

풀 요청에 대한 검토자를 업데이트하려면

1. 풀 요청의 검토자를 업데이트하려는 프로젝트로 이동합니다.
2. 프로젝트 페이지의 풀 요청 열기에서 검토자를 업데이트하려는 풀 요청을 선택합니다. 또는 탐색 창에서 코드를 선택하고 풀 요청을 선택한 다음 업데이트할 풀 요청을 선택합니다.
3. (선택 사항) 개요의 풀 요청 세부 정보 영역에서 더하기 기호를 선택하여 필수 또는 선택적 검토자를 추가합니다. 검토자 옆의 X를 선택하여 선택적 또는 필수 검토자를 제거합니다.
4. (선택 사항) 개요의 풀 요청 세부 정보 영역에서 문제 연결을 선택하여 풀 요청에 문제를 연결한 다음, 목록에서 문제를 선택하거나 해당 ID를 입력합니다. 문제를 연결 해제하려면, 연결 해제하려는 문제 옆에 있는 연결 해제 아이콘을 선택합니다.

풀 요청의 소스 브랜치에서 파일 및 코드를 업데이트하려면

1. 여러 파일을 업데이트하려면, [개발 환경을 생성](#)하거나 리포지토리 및 해당 소스 브랜치를 복제하고, Git 클라이언트 또는 통합 개발 환경(IDE)을 사용하여 소스 브랜치의 파일을 변경합니다. CodeCatalyst 소스 리포지토리의 소스 브랜치에 변경 사항을 커밋하고 푸시하여, 변경 사항으로 풀 요청을 자동으로 업데이트합니다. 자세한 내용은 [소스 리포지토리 복제](#) 및 [Amazon CodeCatalyst에서 커밋을 사용한 소스 코드 변경 사항 이해](#) 섹션을 참조하세요.
2. 소스 브랜치에서 개별 파일을 업데이트하려면, 여러 파일에 대해 업데이트 할 때처럼 Git 클라이언트 또는 IDE를 사용할 수 있습니다. CodeCatalyst 콘솔에서 직접 편집할 수도 있습니다. 자세한 내용은 [파일 편집](#) 단원을 참조하십시오.

풀 요청의 제목 및 설명을 업데이트하려면

1. 풀 요청의 제목 또는 설명을 업데이트하려는 프로젝트로 이동합니다.

2. 프로젝트 페이지에는 풀 요청을 생성한 사람, 풀 요청에 대한 브랜치가 포함된 리포지토리, 풀 요청이 생성된 시기에 대한 정보를 포함한 열린 풀 요청이 표시됩니다. 소스 리포지토리별로 열린 풀 요청 보기를 필터링할 수 있습니다. 목록에서 변경하려는 풀 요청을 선택합니다.
3. 모든 풀 요청을 보려면 모두 보기를 선택합니다. 또는 탐색 창에서 코드를 선택한 다음 풀 요청을 선택합니다. 필터 상자 또는 정렬 함수를 사용하여 변경하려는 풀 요청을 찾은 다음, 해당 풀 요청을 선택합니다.
4. 개요에서 편집을 선택합니다.
5. 제목 또는 설명을 변경한 다음 저장을 선택합니다.

풀 요청 병합

코드를 검토하고 모든 필수 검토자가 승인하면, 패스트 포워드와 같은 지원되는 병합 전략을 사용하여 CodeCatalyst 콘솔에서 풀 요청을 병합할 수 있습니다. CodeCatalyst 콘솔에서 지원되는 모든 병합 전략을 모든 풀 요청에서 선택하여 사용할 수 있는 것은 아닙니다. CodeCatalyst는 병합을 평가하며, 콘솔에서 사용할 수 있고 소스 브랜치를 대상 브랜치로 병합할 수 있는 병합 전략 중에서만 선택할 수 있습니다. 로컬 컴퓨터나 개발 환경에서 `git merge` 명령을 실행해 소스 브랜치를 대상 브랜치로 병합하여, 풀 요청을 선택한 Git 병합 전략과 병합할 수도 있습니다. 그런 다음 대상 브랜치의 변경 사항을 CodeCatalyst의 소스 리포지토리로 푸시할 수 있습니다.

Note

Git에서 브랜치를 병합하고 변경 사항을 푸시해도 풀 요청이 자동으로 닫히지 않습니다.

프로젝트 관리자 역할이 있는 경우, 승인 및 승인 규칙에 대한 모든 요구 사항을 아직 충족하지 않은 풀 요청을 병합하도록 선택할 수도 있습니다.

풀 요청 병합 (콘솔)

소스 브랜치와 대상 브랜치 간에 병합 충돌이 없고 모든 필수 검토자가 풀 요청을 승인한 경우, CodeCatalyst 콘솔에서 풀 요청을 병합할 수 있습니다. 충돌이 있거나 병합을 완료할 수 없는 경우, 병합 버튼이 비활성화되고 병합할 수 없음 레이블이 표시됩니다. 이 경우 임의의 필수 검토자의 승인을 얻고, 필요한 경우 충돌을 로컬에서 해결한 다음, 이러한 변경 사항을 푸시해야 병합할 수 있습니다. 풀 요청을 병합하면 풀 요청 생성자와 필수 또는 선택적 검토자에게 이메일이 자동으로 전송됩니다. 풀 요청에 연결된 문제는 자동으로 닫히거나 상태가 변경되지 않습니다.

i Tip

프로파일의 일부로서 이메일을 수신할 풀 요청 이벤트를 구성할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 Slack 및 이메일 알림 전송](#) 단원을 참조하십시오.

풀 요청 병합을 병합하려면

1. 풀 요청을 병합하려는 프로젝트로 이동합니다.
2. 프로젝트 페이지의 풀 요청 열기에서 병합할 풀 요청을 선택합니다. 풀 요청이 표시되지 않으면 모든 풀 요청 보기를 선택한 다음 목록에서 선택합니다. 또는 탐색 창에서 코드를 선택하고 풀 요청을 선택한 다음 병합할 풀 요청을 선택합니다. 병합을 선택합니다.
3. 풀 요청에서 사용 가능한 병합 전략 중에서 선택합니다. 선택적으로 풀 요청을 병합한 후 소스 브랜치를 삭제하는 옵션을 선택하거나 선택 취소한 다음, 병합을 선택합니다.

i Note

병합 버튼이 비활성 상태이거나 병합할 수 없음 레이블이 표시되는 경우, 필수 검토자가 아직 풀 요청을 승인하지 않았거나 CodeCatalyst 콘솔에서 풀 요청을 병합할 수 없는 것입니다. 풀 요청을 승인하지 않은 검토자는 개요의 풀 요청 세부 정보 영역에 시계 아이콘으로 표시됩니다. 모든 필수 검토자가 풀 요청을 승인했지만 병합 버튼이 여전히 비활성 상태인 경우, 병합 충돌이 발생할 수 있습니다. 밀줄이 그어진 병합할 수 없음 레이블을 선택하여 풀 요청을 병합할 수 없는 이유에 대한 자세한 내용을 확인합니다. 개발 환경 또는 CodeCatalyst 콘솔에서 대상 브랜치의 병합 충돌을 해결한 뒤, 풀 요청을 병합하거나 충돌을 해결하고 로컬에서 병합한 다음, 병합이 포함된 커밋을 CodeCatalyst의 소스 브랜치로 푸시할 수 있습니다. 자세한 내용은 [풀 요청 병합\(Git\)](#) 섹션 및 Git 설명서를 참조하세요.

병합 요구 사항 재정의

프로젝트 관리자 역할이 있는 경우, 필수 승인 및 승인 규칙에 대한 모든 요구 사항을 아직 충족하지 못한 풀 요청을 병합하도록 선택할 수 있습니다. 이를 풀 요청의 요구 사항 재정의라고 합니다. 필수 검토자가 작업할 수 없거나 특정 풀 요청을 신속하게 충족할 수 없는 승인 규칙이 있는 브랜치에 병합해야 하는 긴급한 필요가 발생하는 경우, 이 작업을 수행할 수 있습니다.

풀 요청 병합을 병합하려면

1. 요구 사항을 재정의하고 병합하려는 풀 요청에서 병합 버튼 옆의 드롭다운 화살표를 선택합니다. 승인 요구 사항 재정의 선택합니다.
2. 재정의 이유에서 승인 규칙 및 필수 검토자 요구 사항을 충족하지 않고 이 풀 요청을 병합하는 이유에 대한 세부 정보를 제공합니다. 이는 선택 사항이지만 적극 권장됩니다.
3. 선택 사항으로 병합 전략을 선택하거나 기본값을 수락합니다. 자동 생성된 커밋 메시지와 추가적인 세부 정보를 업데이트하도록 선택할 수도 있습니다.
4. 병합 시 소스 브랜치를 삭제하는 옵션을 선택하거나 선택 취소합니다. 풀 요청을 병합하기 위한 요구 사항을 재정의할 때는 다른 팀원과 결정을 검토할 수 있을 때까지 소스 브랜치를 유지하는 것이 좋습니다.
5. 병합을 선택합니다.

풀 요청 병합(Git)

Git은 브랜치를 병합하고 관리하기 위한 다양한 옵션을 지원합니다. 다음은 명령은 사용할 수 있는 몇 가지 옵션입니다. 자세한 내용은 [Git 웹사이트](#)에서 확인할 수 있는 설명서를 참조하세요. 변경 사항을 병합하고 푸시한 후에는 풀 요청을 수동으로 닫습니다. 자세한 내용은 [풀 요청 닫기](#) 단원을 참조하십시오.

브랜치 병합을 위한 공통 Git 명령

로컬 리포지토리의 소스 브랜치에서 변경한 내용을 로컬 리포지토리의 대상 브랜치에 병합합니다.

```
git checkout destination-branch-name
```

```
git merge source-branch-name
```

소스 브랜치를 대상 브랜치로 병합하여 패스트포워드 병합을 지정합니다. 이렇게 하면 브랜치가 병합되고 대상 브랜치 포인터가 소스 브랜치의 최신 커밋으로 이동합니다.

```
git checkout destination-branch-name
```

```
git merge --ff-only source-branch-name
```

소스 브랜치를 대상 브랜치로 병합하며 스쿼시 병합을 지정합니다. 이렇게 하면 소스 브랜치의 모든 커밋이 대상 브랜치의 단일 병합 커밋으로 결합됩니다.

```
git checkout destination-branch-name
```

```
git merge --squash source-branch-name
```

소스 브랜치를 대상 브랜치로 병합하며 3방향 병합을 지정합니다. 이렇게 하면 병합 커밋이 생성되고 소스 브랜치의 개별 커밋이 대상 브랜치에 추가됩니다.

```
git checkout destination-branch-name

git merge --no-ff source-branch-name
```

로컬 리포지토리에서 소스 브랜치를 삭제합니다. 이는 대상 브랜치로 병합하고 변경 사항을 소스 리포지토리로 푸시한 후, 로컬 리포지토리를 정리하는 데 유용합니다.

```
git branch -d source-branch-name
```

원격 리포지토리에 대해 지정된 로컬 리포지토리 별명을 사용하여 원격 리포지토리(CodeCatalyst의 소스 리포지토리)의 소스 브랜치를 삭제합니다. (콜론(:) 사용에 주의하세요.) 또는 명령의 일부로 --delete를 지정합니다.

```
git push remote-name :source-branch-name

git push remote-name --delete source-branch-name
```

풀 요청 닫기

풀 요청을 종결됨으로 표시할 수 있습니다. 이렇게 하면 풀 요청이 병합되지 않지만 작업이 필요한 풀 요청과 더 이상 관련이 없는 풀 요청을 판단하는 데 도움이 될 수 있습니다. 더 이상 이러한 변경 사항을 병합할 계획이 없거나 변경 사항이 다른 풀 요청에서 병합된 경우, 풀 요청을 닫는 것이 좋습니다.

풀 요청을 닫으면 풀 요청 생성자와 필수 또는 선택적 검토자에게 이메일이 자동으로 전송됩니다. 풀 요청을 닫더라도 풀 요청에 연결된 문제의 상태는 자동으로 변경되지 않습니다.

Note

풀 요청을 닫은 후에는 다시 열 수 없습니다.

풀 요청을 닫으려면

1. 풀 요청을 닫으려는 프로젝트로 이동합니다.
2. 프로젝트 페이지에 열린 풀 요청이 표시됩니다. 닫으려는 풀 요청을 선택합니다.
3. 닫기를 선택하세요.
4. 정보를 검토한 후 닫기를 선택합니다.

Amazon CodeCatalyst에서 커밋을 사용한 소스 코드 변경 사항 이해

커밋은 리포지토리의 콘텐츠 및 변경 사항에 대한 스냅샷입니다. 사용자가 브랜치에 변경 내용을 커밋하고 푸시할 때마다 해당 정보가 저장됩니다. Git 커밋 정보에는 커밋 작성자, 변경 사항을 커밋한 사람, 날짜 및 시간, 변경 사항이 포함됩니다. Amazon CodeCatalyst 콘솔에서 파일을 생성하거나 편집할 때 유사한 정보가 자동으로 포함되지만 작성자 이름은 CodeCatalyst 사용자 이름입니다. 특정 커밋을 식별하는 데 도움이 되도록 커밋에 Git 태그를 추가할 수도 있습니다.

Amazon CodeCatalyst에서 다음을 수행할 수 있습니다.

- 브랜치에 대한 커밋 목록을 봅니다.
- 상위 또는 상위와 비교할 때 커밋의 변경 사항을 포함하여 개별 커밋을 봅니다.

파일과 폴더를 볼 수도 있습니다. 자세한 내용은 [Amazon CodeCatalyst에서 소스 코드 파일 관리](#) 섹션을 참조하세요.

주제

- [브랜치에 대한 커밋 보기](#)
- [커밋 표시 방식 변경\(CodeCatalyst 콘솔\)](#)

브랜치에 대한 커밋 보기

CodeCatalyst 콘솔에서 브랜치 커밋을 검토하여 브랜치 변경 기록을 볼 수 있습니다. 이렇게 하면 누가 언제 브랜치를 변경했는지 이해하는 데 도움이 됩니다. 특정 커밋에서 변경된 내용을 검토할 수도 있습니다.

Tip

특정 파일을 변경한 커밋의 기록을 볼 수도 있습니다. 자세한 내용은 [파일 보기](#)를 참조하세요.

Git 클라이언트를 사용하여 커밋을 볼 수도 있습니다. 자세한 내용은 Git 설명서를 참조하세요.

커밋 보기(콘솔)

1. 커밋을 보려는 소스 리포지토리가 포함된 프로젝트로 이동합니다.

2. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.

브랜치에 커밋을 보려는 리포지토리를 선택합니다.
3. 리포지토리의 기본 브랜치가 표시되며, 브랜치에 대한 가장 최근 커밋에 대한 정보도 포함됩니다. 커밋을 선택합니다. 또는 추가를 선택한 다음 커밋 보기를 선택합니다.
4. 다른 브랜치에 대한 커밋을 보려면 브랜치 선택기를 선택한 다음 브랜치의 이름을 선택합니다.
5. 특정 커밋에 대한 세부 정보를 보려면 커밋 제목에서 해당 제목을 선택합니다. 상위 커밋에 대한 정보와 상위 커밋을 지정된 커밋과 비교하여 코드를 변경한 내용을 포함하여 커밋에 대한 세부 정보가 표시됩니다.

Tip

커밋에 둘 이상의 상위가 있는 경우 상위 커밋 ID 옆의 드롭다운 아이콘을 선택하여 정보를 보고 변경 사항을 표시할 상위 커밋을 선택할 수 있습니다.

커밋 표시 방식 변경(CodeCatalyst 콘솔)

커밋 보기에 표시되는 정보를 변경할 수 있습니다. 작성자 및 커밋 ID와 같은 열을 숨기거나 표시하도록 선택할 수 있습니다.

커밋 표시 방법 변경(콘솔)

1. 커밋을 보려는 소스 리포지토리가 포함된 프로젝트로 이동합니다.
2. 프로젝트의 소스 리포지토리 목록에서 리포지토리 이름을 선택합니다. 아니면 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.

커밋을 보는 방식을 변경하려는 리포지토리를 선택합니다.
3. 리포지토리의 기본 브랜치가 표시되며, 브랜치에 대한 가장 최근 커밋에 대한 정보도 포함됩니다. 커밋을 선택합니다.
4. 기어 모양 아이콘을 선택합니다.
5. 기본 설정에서 표시할 커밋 수를 선택하고 커밋 작성자, 커밋 날짜 및 커밋 ID에 대한 정보를 표시할지 여부를 선택합니다.

Note

정보 표시에서는 커밋 제목을 숨길 수 없습니다.

6. 변경한 후에는 저장을 선택하여 저장하거나 취소를 선택하여 취소합니다.

CodeCatalyst의 소스 리포지토리 할당량

다음 테이블에서는 Amazon CodeCatalyst의 소스 리포지토리에 대한 쿼터 및 제한에 대해 설명합니다. Amazon CodeCatalyst의 할당량에 대한 자세한 내용은 [CodeCatalyst 할당량](#) 섹션을 참조하세요.

Resource	정보
브랜치 이름	허용되는 문자의 길이는 1~256자 사이이며 리포지토리 내에서 고유한 문자의 조합이어야 합니다. 브랜치 이름에는 다음과 같은 제한이 있습니다. - 슬래시(/) 또는 마침표(.)로 시작하거나 끝날 수 없음 - 단일 문자 @로 구성될 수 없음 - 두 개 이상의 연속 마침표(.), 슬래시(/) 또는 다음 문자 조합은 포함할 수 없음: @{ - 공백 또는 허용되지 않는 문자(? ^ * [\ ~ :)의 사용 브랜치 이름은 참조입니다. 브랜치 이름에 대한 대부분의 제한 사항은 Git 참조 표준에 근거합니다. 자세한 내용은 Git Internals 및 git-check-ref-format을 참조하세요.
풀 요청에 대한 설명	풀 요청 시 최대 1,000개.
커밋 메시지	최대 1,024자입니다.

Resource	정보
파일 경로	1~4,096자의 허용되는 문자 조합이면 됩니다. 파일 경로는 파일과 파일의 정확한 위치를 지정하는 명확한 이름이어야 합니다. 파일 경로는 깊이가 디렉터리 20개를 초과할 수 없습니다. 또한 파일 경로는 다음과 같아서는 안 됩니다. - 빈 문자열 포함 - 상대적 파일 경로 - 후행 슬래시나 백슬래시로 끝나는 <code>./</code> <code>../</code> <code>//</code> - 문자 조합 포함 파일 이름 및 경로는 정규화된 이름과 경로여야 합니다. 로컬 컴퓨터의 파일 이름과 경로는 해당 운영 체제에 대한 표준을 따라야 합니다. 리포지토리의 파일에 경로를 지정할 때는 Amazon Linux에 대한 표준을 사용합니다.
파일 크기	CodeCatalyst 콘솔 사용 시 개별 파일의 경우 최대 6MB입니다.
CodeCatalyst 콘솔에서 볼 수 있는 파일 크기	CodeCatalyst 콘솔 사용 시 개별 파일의 경우 최대 6MB입니다.

Resource	정보
Git BLOB 크기	최대 2GB  Note 단일 커밋의 모든 파일 수 또는 총 파일 크기에 대한 제한은 없습니다. 단, 메타데이터는 6MB를 초과할 수 없고 단일 BLOB은 2GB를 초과할 수 없습니다. 그러나 모범 사례로 하나의 대규모 커밋이 아닌 여러 개의 소규모 커밋을 수행하는 것이 좋습니다.
커밋에 대한 메타데이터	커밋에 대한 결합 메타데이터(예시: 작성자 정보, 날짜, 상위 커밋 목록, 커밋 메시지 조합)의 경우 최대 6MB입니다.  Note 단일 커밋의 모든 파일 수 또는 총 파일 크기에 대한 제한은 없습니다. 단, 데이터는 20MB를 초과할 수 없고 개별 파일은 6MB를 초과할 수 없으며 단일 BLOB은 2GB를 초과할 수 없습니다.
풀 요청에 연결할 수 있는 CodeCatalyst 문제 수	50
풀 요청에 연결할 수 있는 Jira 문제 수	50
스페이스에 열려 있는 풀 요청 수	Amazon CodeCatalyst 스페이스의 경우 최대 1,000개입니다.
스페이스의 총 풀 요청 수	Amazon CodeCatalyst 스페이스의 경우 최대 10,000개입니다.

Resource	정보
단일 푸시의 참조 수	생성, 삭제, 업데이트를 포함하여 최대 4,000개. 리포지토리의 최대 참조 수는 제한이 없습니다.
스페이스의 리포지토리 수	Amazon CodeCatalyst 스페이스의 경우 최대 5,000개입니다.
리포지토리 설명	0~1,000자의 문자 조합이면 됩니다. 리포지토리 설명은 선택 사항입니다.
리포지토리 이름	리포지토리 이름은 프로젝트에서 고유해야 합니다. 문자, 숫자, 마침표, 밑줄, 대시를 조합하여 사용할 수 있으며 길이는 1~100자 사이여야 합니다. 이름은 대/소문자를 구분하지 않습니다. 리포지토리 이름은 .git으로 끝날 수 없으며 다음 문자를 포함할 수 없습니다. ! ? @ # \$ % ^ & * () + = { } [] \ / > < ~ ` ' " ; :
리포지토리 크기	리포지토리 크기는 스페이스의 전체 스토리지 제한에 따라 영향을 받습니다. 자세한 내용은 요금 및 소스 리포지토리 문제 해결 섹션을 참조하세요.
풀 요청에 대한 검토자	풀 요청에 대해 최대 총 100명의 검토자(선택 또는 필수 사항).
풀 요청에 대한 서면 요약	풀 요청에 대한 최대 서면 요약 수는 스페이스의 청구 계층에 따라 달라집니다. 자세한 설명은 요금 을 참조하세요.

CodeCatalyst에서 개발 환경으로 코드 작성 및 수정

개발 환경은 클라우드 기반 개발 환경입니다. Amazon CodeCatalyst의 개발 환경을 사용하여 프로젝트의 소스 리포지토리에 저장된 코드 작업을 수행할 수 있습니다. 개발 환경을 생성할 때 몇 가지 옵션이 있습니다.

- 또한 CodeCatalyst에서 개발 환경을 만들어 지원되는 통합 개발 환경(IDE)으로 코드 작업을 수행할 수 있습니다.
- 빈 개발 환경을 만들고, 소스 리포지토리에서 코드를 복제하고, 지원되는 IDE로 해당 코드에서 작업하세요.
- IDE에서 개발 환경을 만들고 소스 리포지토리를 개발 환경에 복제합니다.

devfile은 개발 환경을 표준화하는 개방형 표준 YAML 파일입니다. 즉, 이 파일은 개발 환경에 필요한 개발 도구를 코드화합니다. 따라서 개발 환경을 빠르게 설정하고, 프로젝트 간에 전환하고, 팀 멤버 간에 개발 환경 구성을 복제할 수 있습니다. 개발 환경은 지정된 프로젝트에 대해 코딩, 테스트 및 디버깅하는 데 필요한 모든 도구를 구성하는 devfile을 사용하기 때문에 로컬 개발 환경을 생성하고 유지 관리하는 데 소요되는 시간을 최소화합니다.

Important

개발 환경은 CodeCatalyst 자격 증명에 액세스할 수 있는 devfile을 통해 스크립트를 실행할 수 있습니다. 신뢰할 수 없는 리소스를 열기 전에 리포지토리를 검사하십시오.

개발 환경에 포함된 프로젝트 도구 및 애플리케이션 라이브러리는 프로젝트의 소스 리포지토리에 있는 devfile로 정의됩니다. 소스 리포지토리에 devfile이 없는 경우 CodeCatalyst는 기본 devfile을 자동으로 적용합니다. 이 기본 devfile에는 가장 자주 사용되는 프로그래밍 언어 및 프레임워크를 위한 도구가 포함되어 있습니다. 블루프린트를 사용하여 프로젝트를 생성한 경우 CodeCatalyst에서 devfile을 자동으로 생성합니다. devfile에 대한 자세한 내용은 <https://devfile.io>를 참조하세요.

개발 환경을 생성한 후에는 해당 환경에만 액세스할 수 있습니다. 개발 환경에서는 지원되는 IDE에서 소스 리포지토리의 코드를 보고 작업할 수 있습니다.

기본적으로 개발 환경은 2코어 프로세서, 4GB RAM, 16GB 영구 저장소를 갖도록 구성됩니다. 스페이스 관리자 권한이 있는 경우 다른 개발 환경 구성 옵션을 사용하고 컴퓨팅 및 스토리지 제한을 관리하도록 스페이스의 청구 계층을 변경할 수 있습니다.

주제

- [개발 환경 생성](#)
- [개발 환경 중지](#)
- [개발 환경 재개](#)
- [개발 환경 편집](#)
- [개발 환경 삭제](#)
- [SSH를 사용하여 개발 환경에 연결](#)
- [개발 환경용 devfile 구성](#)
- [VPC 연결을 개발 환경에 연결](#)
- [CodeCatalyst에서의 개발 환경 할당량](#)

개발 환경 생성

개발 환경은 다양한 방법으로 만들 수 있습니다.

- 개요, 개발 환경 또는 소스 리포지토리 페이지에서 CodeCatalyst 소스 리포지토리 또는 [연결된 소스 리포지토리](#)를 사용하여 CodeCatalyst에서 개발 환경을 생성합니다.
- 개발 환경 페이지에서 소스 리포지토리에 연결되지 않은 CodeCatalyst의 빈 개발 환경을 만듭니다.
- 선택한 IDE에서 개발 환경을 만들고 소스 리포지토리를 개발 환경에 복제합니다.

Important

Active Directory가 ID 제공업체로 사용되는 스페이스의 사용자는 개발 환경을 사용할 수 없습니다. 자세한 내용은 [Single Sign-On 계정을 사용하여 CodeCatalyst에 로그인할 때는 개발 환경을 생성할 수 없습니다](#) 섹션을 참조하세요.

각 브랜치 및 리포지토리별로 개발 환경 하나를 만들 수 있습니다. 프로젝트에는 여러 리포지토리가 존재할 수 있습니다. 생성한 개발 환경은 CodeCatalyst 계정으로만 관리할 수 있지만, 지원되는 모든 IDE를 사용하여 개발 환경을 열고 작업할 수 있습니다. IDE에서 개발 환경을 사용하려면 AWS Toolkit 이 설치되어 있어야 합니다. 자세한 내용은 [개발 환경에 지원되는 통합 개발 환경](#) 섹션을 참조하세요. 기본적으로 개발 환경은 2코어 프로세서, 4GB RAM 및 16GB 영구 스토리지로 생성합니다.

Note

소스 리포지토리와 연결된 개발 환경을 생성한 경우 리소스 열에는 이 개발 환경을 생성할 때 지정한 브랜치가 항상 표시됩니다. 이는 다른 브랜치를 생성하거나, 개발 환경 내의 다른 브랜치로 전환하거나, 추가 리포지토리를 복제하는 경우에도 적용됩니다. 빈 개발 환경을 생성한 경우 리소스 열이 비어 있습니다.

개발 환경에 지원되는 통합 개발 환경

다음과 같은 지원되는 통합 개발 환경(IDE)에서 개발 환경을 사용할 수 있습니다.

- [AWS Cloud9](#)
- [JetBrains IDEs](#)
 - [IntelliJ IDEA Ultimate](#)
 - [GoLand](#)
 - [PyCharm Professional](#)
- [Visual Studio Code](#)

CodeCatalyst에서 개발 환경 만들기

CodeCatalyst에서 개발 환경 작업을 시작하려면 [AWS Builder ID](#) 또는 [SSO](#)로 인증하고 로그인합니다.

브랜치에서 개발 환경을 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 개발 환경을 생성하려는 프로젝트로 이동합니다.
3. 탐색 창에서 다음 중 하나를 수행합니다.
 - 개요를 선택한 다음 내 개발 환경 섹션으로 이동합니다.
 - 코드를 선택한 후 개발 환경을 선택합니다.
 - 코드를 선택하고 소스 리포지토리를 선택한 다음 개발 환경을 생성할 리포지토리를 선택합니다.
4. 개발 환경 생성을 선택합니다.
5. 드롭다운 메뉴에서 지원되는 IDE를 선택합니다. 자세한 정보는 [개발 환경에 지원되는 통합 개발 환경](#)을 참조하세요.

6. 리포지토리 복제를 선택합니다.
7. 다음 중 하나를 수행하세요.
 - a. 복제할 리포지토리를 선택하고, 기존 브랜치에서 작업을 선택한 다음 기존 브랜치 드롭다운 메뉴에서 브랜치를 선택합니다.

 Note

타사 리포지토리를 선택하는 경우 기존 브랜치에서 작업해야 합니다.

- b. 복제할 리포지토리를 선택하고, 새 브랜치에서 작업을 선택하고, 브랜치 이름 필드에 브랜치 이름을 입력하고, 다음에서 브랜치 생성 드롭다운 메뉴에서 새 브랜치를 만들 브랜치를 선택합니다.

 Note

소스 리포지토리 페이지 또는 특정 소스 리포지토리에서 개발 환경을 생성하는 경우 리포지토리를 선택할 필요가 없습니다. 개발 환경은 소스 리포지토리 페이지에서 선택한 소스 리포지토리에서 생성됩니다.

8. (선택 사항) 별칭 - 선택 사항에서 개발 환경의 별칭을 입력합니다.
9. (선택 사항) 개발 환경 구성 편집 버튼을 선택하여 개발 환경의 컴퓨팅, 스토리지 또는 제한 시간 구성을 편집합니다.
10. (선택 사항) Amazon Virtual Private Cloud(Amazon VPC) - 선택 사항에서 드롭다운 메뉴에서 개발 환경에 연결할 VPC 연결을 선택합니다.

스페이스에 기본 VPC가 설정된 경우 개발 환경이 해당 VPC에 연결된 상태로 실행됩니다. 다른 VPC 연결을 연결하여 이를 재정의할 수 있습니다. 또한 VPC 연결 개발 환경은 AWS Toolkit을 지원하지 않습니다.

사용하려는 VPC 연결이 나열되지 않은 경우 프로젝트에 허용되지 않는 AWS 계정 연결이 포함되어 있기 때문일 수 있습니다. 자세한 내용은 Amazon CodeCatalyst 관리자 안내서의 [프로젝트 제한 계정 연결 구성](#)을 참조하세요.

Note

VPC 연결로 개발 환경을 생성하면 VPC 내에 새 네트워크 인터페이스가 생성됩니다. CodeCatalyst는 연결된 VPC 역할을 사용하여 이 인터페이스와 상호 작용합니다. 또한 IPv4 CIDR 블록이 172.16.0.0/12 IP 주소 범위로 구성되지 않았는지 확인합니다.

11. 생성(Create)을 선택합니다. 개발 환경이 생성되는 동안 개발 환경 상태 열에 시작 중이 표시되고, 개발 환경이 생성되면 상태 열에 실행 중이 표시됩니다.

빈 개발 환경 생성

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 개발 환경을 생성하려는 프로젝트로 이동합니다.
3. 탐색 창에서 다음 중 하나를 수행합니다.
 - 개요를 선택한 다음 내 개발 환경 섹션으로 이동합니다.
 - 코드를 선택한 후 개발 환경을 선택합니다.
4. 개발 환경 생성을 선택합니다.
5. 드롭다운 메뉴에서 지원되는 IDE를 선택합니다. 자세한 정보는 [개발 환경에 지원되는 통합 개발 환경을](#) 참조하세요.
6. 빈 개발 환경 생성을 선택합니다.
7. (선택 사항) 별칭 - 선택 사항에서 개발 환경의 별칭을 입력합니다.
8. (선택 사항) 개발 환경 구성 편집 버튼을 선택하여 개발 환경의 컴퓨팅, 스토리지 또는 제한 시간 구성을 편집합니다.
9. (선택 사항) Amazon Virtual Private Cloud(Amazon VPC) - 선택 사항에서 드롭다운 메뉴에서 개발 환경에 연결할 VPC 연결을 선택합니다.

스페이스에 기본 VPC가 설정된 경우 개발 환경이 해당 VPC에 연결된 상태로 실행됩니다. 다른 VPC 연결을 연결하여 이를 재정의할 수 있습니다. 또한 VPC 연결 개발 환경은 AWS Toolkit을 지원하지 않습니다.

사용하려는 VPC 연결이 나열되지 않은 경우 프로젝트에 허용되지 않는 AWS 계정 연결이 포함되어 있기 때문일 수 있습니다. 자세한 내용은 Amazon CodeCatalyst 관리자 안내서의 [프로젝트 제한 계정 연결 구성](#)을 참조하세요.

Note

VPC 연결로 개발 환경을 생성하면 VPC 내에 새 네트워크 인터페이스가 생성됩니다. CodeCatalyst는 연결된 VPC 역할을 사용하여 이 인터페이스와 상호 작용합니다. 또한 IPv4 CIDR 블록이 172.16.0.0/12 IP 주소 범위로 구성되지 않았는지 확인합니다.

10. 생성(Create)을 선택합니다. 개발 환경이 생성되는 동안 개발 환경 상태 열에 시작 중이 표시되고, 개발 환경이 생성되면 상태 열에 실행 중이 표시됩니다.

Note

개발 환경을 처음 생성하고 여는 데 1~2분이 걸릴 수 있습니다.

Note

IDE에서 개발 환경이 열리면 코드를 커밋하고 변경 사항을 푸시하기 전에 디렉터리를 소스 리포지토리로 변경해야 할 수 있습니다.

IDE에서 개발 환경 생성

개발 환경을 사용하여 프로젝트의 소스 리포지토리에 저장된 코드 작업을 빠르게 수행할 수 있습니다. 개발 환경은 지원되는 통합 개발 환경(IDE)을 사용하여 프로젝트별 완전 작동 클라우드 개발 환경에서 즉시 코딩을 시작할 수 있으므로 개발 속도를 높입니다.

IDE에서 CodeCatalyst를 사용하는 방법에 대한 자세한 내용은 다음 설명서를 참조하세요.

- [Amazon CodeCatalyst for JetBrains IDEs](#)
- [Amazon CodeCatalyst for VS Code](#)
- [용 Amazon CodeCatalyst AWS Cloud9](#)

개발 환경 중지

개발 환경의 /projects 디렉터리는 소스 리포지토리에서 가져온 파일과 개발 환경을 구성하는 데 사용되는 devfile을 저장합니다. 개발 환경 생성 시 비어 있는 /home 디렉터리는 개발 환경을 사용하는

동안 생성한 파일을 저장합니다. 개발 환경의 /projects 및 /home 디렉터리에 있는 모든 것은 지속적으로 저장되므로 다른 개발 환경, 리포지토리 또는 프로젝트로 전환해야 하는 경우 개발 환경에서 작업을 중지할 수 있습니다.

 Warning

웹 브라우저, 원격 셸 및 IDEs를 포함한 인스턴스가 연결된 상태로 남아 있는 경우 개발 환경은 시간 초과되지 않습니다. 따라서 추가 비용이 발생하지 않도록 연결된 모든 인스턴스를 닫아야 합니다.

개발 환경을 만들 때 Timeout(제한 시간) 필드에 선택한 시간 동안 개발 환경이 유휴 상태가 되면, 개발 환경은 자동으로 중지됩니다. 개발 환경이 유휴 상태가 되기 전에 중지할 수 있습니다. 개발 환경을 생성할 때 제한 시간 없음을 선택하면 개발 환경이 자동으로 중지되지 않습니다. 대신 계속 실행됩니다.

 Warning

삭제된 VPC 연결과 연결된 개발 환경을 중지하면 재개할 수 없습니다.

개발 환경 페이지에서 개발 환경을 중지하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 개발 환경을 중지하려는 프로젝트로 이동합니다.
3. 탐색 창에서 코드를 선택합니다.
4. 개발 환경을 선택합니다.
5. 중지하려는 개발 환경의 라디오 버튼을 선택합니다.
6. 작업 메뉴에서 중지를 선택합니다.

 Note

컴퓨팅 사용은 개발 환경이 실행되는 동안에만 요금이 청구되지만, 스토리지 사용은 개발 환경이 존재하는 전체 시간 동안 요금이 청구됩니다. 컴퓨팅 청구를 중지하는 데 개발 환경을 사용하지 않을 때는 중지합니다.

개발 환경 재개

개발 환경의 /projects 디렉터리는 소스 리포지토리에서 가져온 파일과 개발 환경을 구성하는 데 사용되는 devfile을 저장합니다. 개발 환경 생성 시 비어 있는 /home 디렉터리는 개발 환경을 사용하는 동안 생성한 파일을 저장합니다. 개발 환경의 /projects 및 /home 디렉터리에 있는 모든 것은 지속적으로 저장되므로 다른 개발 환경, 리포지토리 또는 프로젝트로 전환해야 하는 경우 개발 환경에서 작업을 중단하고 나중에 개발 환경에서 작업을 재개할 수 있습니다.

개발 환경을 만들 때 Timeout(제한 시간) 필드에 선택한 시간 동안 개발 환경이 유효 상태가 되면, 개발 환경은 자동으로 중지됩니다. 개발 환경이 유효 상태가 되려면 AWS Cloud9 브라우저 탭을 닫아야 합니다.

Note

개발 환경을 생성한 브랜치를 삭제해도 개발 환경을 계속 사용할 수 있으며 실행됩니다. 브랜치를 삭제한 개발 환경에서 작업을 재개하려면 새 브랜치를 생성하고 변경 사항을 푸시합니다.

개요 페이지에서 개발 환경을 재개하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 개발 환경을 재개하려는 프로젝트로 이동하여 내 개발 환경 섹션으로 이동합니다.
3. (IDE)에서 재개를 선택합니다.
 - JetBrains IDE의 경우, JetBrains 게이트웨이 링크를 열 애플리케이션을 선택하라는 메시지가 표시되면 JetBrains 게이트웨이-EAP를 선택합니다. 확인하라는 메시지가 나타나면 링크 열기를 선택합니다.
 - VS Code IDE의 경우 VS Code 링크를 열 애플리케이션 선택을 묻는 메시지가 표시되면 VS Code를 선택합니다. 링크 열기를 선택하여 확인합니다.

소스 리포지토리에서 개발 환경을 재개하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 개발 환경을 재개하려는 프로젝트로 이동합니다.
3. 탐색 창에서 코드를 선택합니다.
4. 소스 리포지토리를 선택합니다.

5. 재개하려는 개발 환경이 포함된 소스 리포지토리를 선택합니다.
6. 브랜치 이름을 선택하여 브랜치의 드롭다운 메뉴를 확인한 다음 브랜치를 선택합니다.
7. 개발 환경 재개를 선택합니다.
 - JetBrains IDEs 경우 링크 열기를 선택하여 이 사이트가 JetBrains Gateway와 함께 JetBrains 게이트웨이 링크를 열도록 허용하시겠습니까?라는 메시지가 표시되면 확인합니다.
 - VS Code IDE의 경우 링크 열기를 선택하여 이 사이트가 Visual Studio Code와 함께 VS Code 링크를 열도록 허용하시겠습니까?라는 메시지가 표시되면 확인합니다.

개발 환경 페이지에서 개발 환경을 재개하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 개발 환경을 재개하려는 프로젝트로 이동합니다.
3. 탐색 창에서 코드를 선택합니다.
4. 개발 환경을 선택합니다.
5. IDE 열에서 개발 환경의 (IDE)에서 재개를 선택합니다.
 - JetBrains IDEs 경우 링크 열기를 선택하여 이 사이트가 JetBrains Gateway와 함께 JetBrains 게이트웨이 링크를 열도록 허용하시겠습니까?라는 메시지가 표시되면 확인합니다.
 - VS Code IDE의 경우 링크 열기를 선택하여 이 사이트가 Visual Studio Code와 함께 VS Code 링크를 열도록 허용하시겠습니까?라는 메시지가 표시되면 확인합니다.

Note

개발 환경 재개는 몇 분 정도 걸립니다.

개발 환경 편집

IDE가 실행되는 동안 개발 환경을 편집할 수 있습니다. 컴퓨팅 또는 비활성 제한 시간을 편집하면 변경 사항을 저장한 후 개발 환경이 다시 시작됩니다.

개발 환경을 편집하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 개발 환경을 편집하려는 프로젝트로 이동합니다.

3. 탐색 창에서 코드를 선택합니다.
4. 개발 환경을 선택합니다.
5. 편집하려는 개발 환경을 선택합니다.
6. 편집을 선택합니다.
7. 컴퓨팅 또는 비활성 제한 시간을 변경할 수 있습니다.
8. 저장을 선택합니다.

개발 환경 삭제

개발 환경에 저장된 콘텐츠 작업이 끝나면 개발 환경을 삭제할 수 있습니다. 새 콘텐츠에서 작업할 새 개발 환경을 생성합니다. 개발 환경을 삭제하면 지속 콘텐츠가 영구적으로 삭제됩니다. 개발 환경을 삭제하기 전에 코드 변경 사항을 커밋하고 개발 환경의 원본 소스 리포지토리에 푸시해야 합니다. 개발 환경을 삭제하면 개발 환경에 대한 컴퓨팅 및 스토리지 청구가 중단됩니다.

개발 환경을 삭제한 후 스토리지 할당량이 업데이트되는 데 몇 분 정도 걸릴 수 있습니다. 스토리지 할당량에 도달하면 이 기간 동안 새 개발 환경을 생성할 수 없습니다.

Important

개발 환경 삭제는 실행 취소할 수 없습니다. 개발 환경을 삭제한 후에는 더 이상 복구할 수 없습니다.

개발 환경을 삭제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 개발 환경을 삭제하려는 프로젝트로 이동합니다.
3. 탐색 창에서 코드를 선택합니다.
4. 개발 환경을 선택합니다.
5. 삭제하려는 개발 환경을 선택합니다.
6. 삭제를 선택합니다.
7. **delete**를 입력하여 개발 환경 삭제를 확인합니다.
8. 삭제를 선택합니다.

Note

스페이스에서 VPC 연결을 삭제하기 전에 해당 VPC와 연결된 개발 환경을 제거해야 합니다. 개발 환경을 삭제하더라도 VPC에서 네트워크 인터페이스를 삭제하지 않을 수 있습니다. 필요에 따라 리소스를 정리해야 합니다. VPC 연결 개발 환경을 삭제할 때 오류가 발생하면 오래된 연결을 [분리하고](#) 사용되지 않음을 확인한 후 [삭제](#)해야 합니다.

SSH를 사용하여 개발 환경에 연결

SSH를 사용하여 개발 환경에 연결하여 포트 포워딩, 파일 업로드 및 다운로드, 다른 IDE 사용 등의 작업을 제한 없이 수행할 수 있습니다.

Note

IDE 탭 또는 창을 닫은 후 장기간 SSH를 계속 사용하려면 IDE의 비활성으로 인해 SSH가 중지되지 않도록 개발 환경에 대해 높은 제한 시간을 설정해야 합니다.

사전 조건

- 다음 운영 체제 중 하나:
 - Windows 10 이상 및 OpenSSH 활성화
 - macOS 및 Bash 버전 3 이상
 - yum, dpkg 또는 rpm 패키지 관리자 및 Bash 버전 3 이상이 포함된 Linux
- 또한 AWS CLI 버전 2.9.4 이상이 필요합니다.

SSH를 사용하여 개발 환경에 연결하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. SSH를 사용하여 개발 환경에 연결하려는 프로젝트로 이동합니다.
3. 탐색 창에서 코드를 선택합니다.
4. 개발 환경을 선택합니다.
5. SSH를 사용하여 연결하려는 실행 중인 개발 환경을 선택합니다.
6. SSH를 통한 연결을 선택하고 원하는 운영 체제를 선택한 뒤 다음을 수행합니다.

- 아직 수행하지 않은 경우 지정된 터미널에 첫 번째 명령을 붙여넣고 실행합니다. 명령은 스크립트를 다운로드하고 로컬 환경에서 다음 수정 사항을 실행하여 SSH를 사용하여 개발 환경에 연결할 수 있습니다.
- 예 [대한 Session Manager 플러그인을 설치합니다. AWS CLI](#)
- SSO 로그인을 수행할 수 있도록 로컬을 수정 AWS Config 하고 CodeCatalyst 프로필을 추가합니다. 자세한 내용은 [CodeCatalyst AWS CLI 에서를 사용하도록 설정](#) 단원을 참조하십시오.
- 로컬 SSH 구성을 수정하고 SSH를 사용하여 개발 환경에 연결하는 데 필요한 구성을 추가합니다.
- SSH 클라이언트가 개발 환경에 연결하는 데 사용하는 스크립트를 ~/.aws/codecatalyst-dev-env 디렉터리에 추가합니다. 이 스크립트는 [CodeCatalyst StartDevEnvironmentSession API](#)를 호출하고 AWS Systems Manager Session Manager 플러그인을 사용하여 로컬 SSH 클라이언트가 원격 개발 환경에 안전하게 연결하는 데 사용하는 개발 환경과의 AWS Systems Manager 세션을 설정합니다.
- 두 번째 명령을 사용하여 AWS SSO를 사용하여 Amazon CodeCatalyst에 로그인합니다. 이 명령은 ~/.aws/codecatalyst-dev-env 디렉터리의 스크립트가 [CodeCatalyst StartDevEnvironmentSession API](#)를 직접적으로 호출할 수 있도록 자격 증명을 요청하고 검색합니다. 이 명령은 자격 증명 만료될 때마다 실행되어야 합니다. 모달(ssh <destination>)에서 마지막 명령을 실행하면 자격 증명 만료되었거나 이 단계의 지침에 따라 SSO 로그인을 수행하지 않은 경우 오류가 발생합니다.
- 세 번째 명령을 사용하여 SSH를 사용하여 지정된 개발 환경에 연결합니다. 이 명령의 구조는 다음과 같습니다.

```
ssh codecatalyst-dev-env=<space-name>=<project-name>=<dev-environment-id>
```

또한 이 명령을 사용하여 포트 전달 또는 파일 업로드 및 다운로드와 같이 SSH 클라이언트에서 허용하는 다른 작업을 수행할 수 있습니다.

- 포트 전달:

```
ssh -L <local-port>:127.0.0.1:<remote-port> codecatalyst-dev-env=<space-name>=<project-name>=<dev-environment-id>
```

- 개발 환경의 홈 디렉터리에 파일 업로드:

```
scp -0 </path-to-local-file> codecatalyst-dev-env=<space-name>=<project-name>=<dev-environment-id>:</path-to-remote-file-or-directory>
```

개발 환경용 devfile 구성

devfile은 팀 전체에서 개발 환경을 사용자 지정하는 데 도움이 되는 개방형 표준입니다. devfile은 필요한 개발 도구를 코드화하는 YAML 파일입니다. devfile을 구성하면 필요한 프로젝트 도구 및 애플리케이션 라이브러리를 미리 결정할 수 있으며 Amazon CodeCatalyst가 이를 개발 환경에 설치합니다. devfile은 생성된 리포지토리에 따라 다르며 각 리포지토리에 대해 별도의 devfile을 생성할 수 있습니다. 개발 환경은 명령 및 이벤트를 지원하며 기본 범용 devfile 이미지를 제공합니다.

빈 블루프린트를 사용하여 프로젝트를 생성하는 경우 devfile을 수동으로 생성할 수 있습니다. 다른 블루프린트를 사용하여 프로젝트를 생성하는 경우 CodeCatalyst는 devfile을 자동으로 생성합니다. 개발 환경의 /projects 디렉터리는 소스 리포지토리와 devfile에서 가져온 파일을 저장합니다. 개발 환경을 처음 생성할 때 비어 있는 /home 디렉터리는 개발 환경을 사용하는 동안 생성한 파일을 저장합니다. 개발 환경의 /projects 및 /home 디렉터리에 있는 모든 항목은 영구적으로 저장됩니다.

Note

/home 폴더는 devfile 또는 devfile 구성 요소 이름의 이름을 변경하는 경우에만 변경됩니다. devfile 또는 devfile 구성 요소 이름을 변경하면 /home 디렉터리의 내용이 대체되고 이전 /home 디렉터리 데이터를 복구할 수 없습니다.

루트에 devfile이 포함되지 않은 소스 리포지토리로 개발 환경을 생성하거나 소스 리포지토리 없이 개발 환경을 생성하는 경우 기본 범용 devfile이 소스 리포지토리에 자동으로 적용됩니다. 모든 IDE에 동일한 기본 범용 devfile 이미지가 사용됩니다. CodeCatalyst는 현재 devfile 버전 2.0.0을 지원합니다. devfile에 대한 자세한 내용은 [Devfile 스키마 - 버전 2.0.0](#)을 참조하세요.

Note

Devfile에는 퍼블릭 컨테이너 이미지만 포함할 수 있습니다.

VPC 연결 개발 환경은 다음 devfile 이미지만 지원합니다.

- 유니버설 이미지

- 리포지토리가 VPC와 동일한 리전에 있는 경우 프라이빗 Amazon ECR 이미지

주제

- [개발 환경용 리포지토리 devfile 편집](#)
- [CodeCatalyst에서 지원하는 Devfile 기능](#)
- [개발 환경용 devfile의 예](#)
- [복구 모드를 사용하여 리포지토리 devfile 문제 해결](#)
- [개발 환경에 대한 범용 devfile 이미지 지정](#)
- [Devfile 명령](#)
- [Devfile 이벤트](#)
- [Devfile 구성 요소](#)

개발 환경용 리포지토리 devfile 편집

다음 절차를 사용하여 개발 환경용 리포지토리 devfile을 편집합니다.

CodeCatalyst에서 개발 환경용 리포지토리 devfile 편집

리포지토리 devfile을 편집하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. devfile을 편집하려는 소스 리포지토리가 포함된 프로젝트로 이동합니다.
3. 탐색 창에서 코드를 선택합니다.
4. 소스 리포지토리를 선택합니다.
5. 편집하려는 devfile이 포함된 소스 리포지토리를 선택합니다.
6. 파일 목록에서 devfile.yaml 파일을 선택합니다.
7. 편집을 선택합니다.
8. devfile을 편집합니다.
9. 커밋을 선택하거나 풀 요청을 생성하여 팀원이 변경 사항을 검토하고 승인할 수 있도록 합니다.

Note

devfile을 편집하는 경우 변경 사항을 적용하려면 devfile을 다시 시작해야 합니다. 이 작업은 `/aws/mde/mde start --location devfile.yaml`을 실행하여 수행할 수 있습니다. devfile을 시작하는 데 문제가 있는 경우 복구 모드로 전환됩니다. 그러나 VPC에 연결된 개발 환경과 연결된 devfile을 편집하는 경우 변경 사항을 적용하려면 개발 환경을 대신 다시 시작해야 합니다.

`/aws/mde/mde status`를 실행하여 사용 중인 devfile을 검토할 수 있습니다. 위치 필드에는 환경 / `projects` 폴더를 기준으로 데브파일의 경로가 있습니다.

```
{
    "status": "STABLE",
    "location": "devfile.yaml"
}
```

`/projects/devfile.yaml`의 기본 devfile을 소스 코드 리포지토리로 이동할 수도 있습니다. devfile의 위치를 업데이트하려면 다음 명령을 사용합니다. `/aws/mde/mde start --location repository-name/devfile.yaml`

IDE에서 개발 환경용 리포지토리 devfile 편집

개발 환경의 구성을 변경하려면 devfile을 편집해야 합니다. 지원되는 IDE에서 devfile을 편집한 다음 개발 환경을 업데이트하는 것이 좋지만 CodeCatalyst의 소스 리포지토리 루트에서 devfile을 편집할 수도 있습니다. 지원되는 IDE에서 devfile을 편집하는 경우, 변경 사항을 커밋하고 소스 리포지토리에 푸시하거나 풀 요청을 생성하여 팀원이 devfile 편집을 검토 및 승인할 수 있게 해야 합니다.

- [에서 개발 환경의 리포지토리 devfile 편집 AWS Cloud9](#)
- [VS Code에서 개발 환경용 리포지토리 devfile 편집](#)
- [JetBrains에서 개발 환경용 리포지토리 devfile 편집](#)

CodeCatalyst에서 지원하는 Devfile 기능

CodeCatalyst는 버전 2.0.0에서 다음과 같은 devfile 기능을 지원합니다. devfile에 대한 자세한 내용은 [Devfile 스키마 - 버전 2.0.0](#)을 참조하세요.

기능	Type
exec	명령
postStart	Event
container	구성 요소
args	구성 요소 속성
env	구성 요소 속성
mountSources	구성 요소 속성
volumeMounts	구성 요소 속성

개발 환경용 devfile의 예

다음은 간단한 devfile의 예입니다.

```

schemaVersion: 2.0.0
metadata:
  name: al2
components:
  - name: test
    container:
      image: public.ecr.aws/amazonlinux/amazonlinux:2
      mountSources: true
      command: ['sleep', 'infinity']
  - name: dockerstore
commands:
  - id: setupscript
    exec:
      component: test
      commandLine: "chmod +x script.sh"
      workingDir: /projects/devfiles
  - id: executescript
    exec:
      component: test
      commandLine: "/projects/devfiles/script.sh"
  - id: yumupdate

```



```

exec:
  component: test
  commandLine: "yum -y update --security"
events:
  postStart:
    - setupscript
    - executescript
    - yumupdate

```

Devfile 시작, 명령 및 이벤트 로그는 캡처되어 `/aws/mde/logs`에 저장됩니다. devfile 동작을 디버깅하려면 작업 devfile을 사용하여 개발 환경을 시작하고 로그에 액세스합니다.

복구 모드를 사용하여 리포지토리 devfile 문제 해결

devfile을 시작하는 데 문제가 있는 경우 복구 모드로 전환되므로 환경에 계속 연결하고 devfile을 수정할 수 있습니다. 복구 모드에 있는 동안 `/aws/mde/mde status` 실행에는 devfile의 위치가 포함되지 않습니다.

```

{
    "status": "STABLE"
}

```

`/aws/mde/logs`의 로그에서 오류를 확인하고, devfile을 수정하고, 다시 `/aws/mde/mde start` 실행을 시도할 수 있습니다.

개발 환경에 대한 범용 devfile 이미지 지정

기본 범용 이미지에는 IDE에 사용할 수 있는 가장 일반적으로 사용되는 프로그래밍 언어와 관련 도구가 포함되어 있습니다. 이미지가 지정되지 않은 경우 CodeCatalyst는 이 이미지를 제공하고 CodeCatalyst에서 관리하는 도구를 포함합니다. 새 이미지 릴리스에 대한 알림을 유지하려면 [SNS를 통한 범용 이미지 알림 구독](#) 섹션을 참조하세요.

Amazon CodeCatalyst는 다음과 같은 devfile 이미지를 적극적으로 지원합니다.

이미지 버전	이미지 식별자
Universal image 4.0	public.ecr.aws/aws-mde/universal-image:4.0

이미지 버전	이미지 식별자
Universal image 5.0	public.ecr.aws/aws-mde/universal-image:5.0

 Note

public.ecr.aws/aws-mde/universal-image:latest를 사용하여 현재 public.ecr.aws/aws-mde/universal-image:5.0인 최신 이미지를 가져올 수도 있습니다.

CodeCatalyst는 다음 이미지를 더 이상 사용하지 않습니다. 이러한 이미지는 계속 사용할 수 있지만 빌드 호스트에 캐시되지 않으므로 개발 환경 시작 시간이 길어집니다.

이미지 버전	이미지 식별자	사용 중단 날짜
Universal image 1.0	public.ecr.aws/aws-mde/universal-image:1.0	2024년 8월 16일
Universal image 2.0	public.ecr.aws/aws-mde/universal-image:2.0	2024년 8월 16일
Universal image 3.0	public.ecr.aws/aws-mde/universal-image:3.0	2025년 7월 30일

 Note

AWS Cloud9를 사용하는 경우 universal-image:3.0로 업그레이드한 후 PHP, Ruby 및 CSS에서 자동 완성이 작동하지 않습니다.

주제

- [SNS를 통한 범용 이미지 알림 구독](#)
- [범용 이미지 4.0 런타임 버전](#)
- [범용 이미지 5.0 런타임 버전](#)

SNS를 통한 범용 이미지 알림 구독

CodeCatalyst는 범용 이미지 알림 서비스를 제공합니다. 이를 사용하여 CodeCatalyst 범용 이미지 업데이트가 릴리스되었을 때 알리는 Amazon Simple Notification Service(SNS) 주제를 구독할 수 있습니다. SNS 주제에 대한 자세한 내용은 [What is Amazon Simple Notification Service?](#)를 참조하세요.

새 범용 이미지가 릴리스될 때마다 구독자에게 알림을 보냅니다. 이 섹션에서는 CodeCatalyst 범용 이미지 업데이트를 구독하는 방법을 설명합니다.

샘플 메시지

```
{
  "Type": "Notification",
  "MessageId": "123456789",
  "TopicArn": "arn:aws:sns:us-east-1:1234657890:universal-image-updates",
  "Subject": "New Universal Image Release",
  "Message": {
    "v1": {
      "Message": "A new version of the Universal Image has been released. You are now able to launch new DevEnvironments using this image.",
      "image": {
        "release_type": "MAJOR VERSION",
        "image_name": "universal-image",
        "image_version": "2.0",
        "image_uri": "public.ecr.aws/amazonlinux/universal-image:2.0"
      }
    }
  },
  "Timestamp": "2021-09-03T19:05:57.882Z",
  "UnsubscribeURL": "example url"
}
```

Amazon SNS 콘솔을 사용하여 CodeCatalyst 범용 이미지 업데이트를 구독하려면

1. Amazon SNS 콘솔의 [대시보드](#)를 엽니다.
2. 탐색 바에서 AWS 리전을 선택합니다.

3. 탐색 창에서 구독을 선택하고 나서 구독 생성을 선택합니다.
4. 주제 ARN에 `arn:aws:sns:us-east-1:089793673375:universal-image-updates`를 입력합니다.
5. 프로토콜에서 이메일을 선택합니다.
6. 엔드포인트에서 이메일 주소를 입력합니다. 이 이메일 주소는 알림을 받는 데 사용됩니다.
7. 구독 생성을 선택합니다.
8. "AWS 알림 - 구독 신청 확인"이라는 제목의 확인 이메일을 받게 됩니다. 이메일을 열고 구독 확인 링크를 선택합니다.

Amazon SNS 콘솔을 사용하여 CodeCatalyst 범용 이미지 업데이트 구독을 취소하려면

1. Amazon SNS 콘솔의 [대시보드](#)를 엽니다.
2. 탐색 바에서 AWS 리전을 선택합니다.
3. 탐색 창에서 구독을 선택한 다음 구독을 해지하려는 구독을 선택합니다.
4. 그런 다음 작업을 선택하고 구독 삭제를 선택합니다.
5. 삭제를 선택합니다.

범용 이미지 4.0 런타임 버전

다음 표에는 `universal-image:4.0`에 대한 런타임 가용성이 나열되어 있습니다.

universal-image:4.0 런타임 버전

실행 시간 이름	버전	특정 메이저 버전 및 최신 마이너 버전
aws cli	2.11	aws-cli: 2.x
docker compose	2.17	docker-compose: 2.x
dotnet	8.0	dotnet: 8.x
golang	1.22	golang: 1.x
java	corretto21	java: corretto21.x
nodejs	20.6	nodejs: 20.x

실행 시간 이름	버전	특정 메이저 버전 및 최신 마이너 버전
php	8.2	php: 8.x
python	3.9	python: 3.x
	3.12	
ruby	3.3	ruby: 3.x
terraform	1.5	terraform: 1.x

범용 이미지 5.0 런타임 버전

다음 표에는 `universal-image:5.0`에 대한 런타임 가용성이 나열되어 있습니다.

universal-image:5.0 런타임 버전

실행 시간 이름	버전	특정 메이저 버전 및 최신 마이너 버전
aws cli	2.25	aws-cli: 2.x
docker compose	2.34	docker-compose: 2.x
dotnet	8.0	dotnet: 8.x
golang	1.24	golang: 1.x
java	corretto21	java: corretto21.x
nodejs	22.0	nodejs: 22.x
php	8.3.16	php: 8.x
python	3.12	python: 3.x
ruby	3.4.2	ruby: 3.x
terraform	1.10.5	terraform: 1.x

Devfile 명령

현재 CodeCatalyst는 devfile의 exec 명령만 지원합니다. 자세한 내용을 알아보려면 Devfile.io 설명서의 [명령 추가](#)를 참조하세요.

다음 예시에서는 devfile에서 exec 명령을 지정하는 방법을 보여줍니다.

```
commands:
  - id: setupscript
    exec:
      component: test
      commandLine: "chmod +x script.sh"
      workingDir: /projects/devfiles
  - id: executescript
    exec:
      component: test
      commandLine: "./projects/devfiles/script.sh"
  - id: updateyum
    exec:
      component: test
      commandLine: "yum -y update --security"
```

개발 환경에 연결한 후 터미널을 통해 정의된 명령을 실행할 수 있습니다.

```
/aws/mde/mde command <command-id>
/aws/mde/mde command executescript
```

장기 실행 명령의 경우 -s 스트리밍 플래그를 사용하여 명령 실행을 실시간으로 출력할 수 있습니다.

```
/aws/mde/mde -s command <command-id>
```

Note

command-id는 소문자여야 합니다.

CodeCatalyst에서 지원하는 Exec 파라미터

CodeCatalyst는 devfile 버전 2.0.0에서 다음 exec 파라미터를 지원합니다.

- commandLine

- component
- id
- workingDir

Devfile 이벤트

현재 CodeCatalyst는 devfile의 postStart 이벤트만 지원합니다. 자세한 내용은 Devfile.io 설명서에서 [postStartObject](#) 섹션을 참조하세요.

다음 예시에서는 devfile에 postStart 이벤트 바인딩을 추가하는 방법을 보여줍니다.

```
commands:
  - id: executescript
    exec:
      component: test
      commandLine: "./projects/devfiles/script.sh"
  - id: updateyum
    exec:
      component: test
      commandLine: "yum -y update --security"
events:
  postStart:
    - updateyum
    - executescript
```

시작 후 개발 환경은 지정된 postStart 명령을 정의된 순서대로 실행합니다. 명령이 실패하면 개발 환경이 계속 실행되고 실행 출력이 /aws/mde/logs 아래의 로그에 저장됩니다.

Devfile 구성 요소

현재 CodeCatalyst는 devfile의 container 구성 요소만 지원합니다. 자세한 내용은 Devfile.io 설명서의 [구성 요소 추가](#)를 참조하세요.

다음 예시에서는 devfile의 컨테이너에 시작 명령을 추가하는 방법을 보여줍니다.

```
components:
  - name: test
    container:
```

```
image: public.ecr.aws/amazonlinux/amazonlinux:2
command: ['sleep', 'infinity']
```

Note

컨테이너에 짧은 라이브 항목 명령이 있는 경우 컨테이너를 계속 실행하려면 `command: ['sleep', 'infinity']`를 포함해야 합니다.

CodeCatalyst는 컨테이너 구성 요소에서 `args`, `env`, `mountSources`, `volumeMounts` 속성도 지원합니다.

VPC 연결을 개발 환경에 연결

VPC 연결은 워크플로가 VPC에 액세스하는 데 필요한 모든 구성을 포함하는 CodeCatalyst 리소스입니다. 스페이스 관리자는 스페이스 멤버를 대신하여 Amazon CodeCatalyst 콘솔에 자체 VPC 연결을 추가할 수 있습니다. VPC 연결을 추가하면 스페이스 멤버가 워크플로 작업을 실행하고 네트워크 규칙을 준수하고 연결된 VPC의 리소스에 액세스할 수 있는 개발 환경을 생성할 수 있습니다.

Important

VPC 연결이 있는 개발 환경은 [CodeCatalyst에 연결된 타사 소스 리포지토리](#)를 지원하지 않습니다.

개발 환경 생성 시에만 개발 환경을 VPC 연결에 연결할 수 있습니다. 개발 환경을 생성한 후에는 개발 환경과 연결된 VPC 연결을 변경할 수 없습니다. 다른 VPC 연결을 사용하려면 현재 개발 환경을 삭제하고 새 환경을 생성해야 합니다.

Note

개발 환경은 프로젝트에 액세스할 수 있는 AWS 계정과의 VPC 연결에만 연결할 수 있습니다. 자세한 내용은 Amazon CodeCatalyst 관리자 안내서의 [프로젝트 제한 계정 연결 구성](#)을 참조하세요.

개발 환경은 생성 시 여러 AWS 리소스와 서비스를 활용합니다. 즉, 개발 환경은 다음 AWS 서비스에 연결됩니다.

- Amazon CodeCatalyst
- AWS SSM
- AWS KMS
- Amazon ECR
- Amazon CloudWatch
- Amazon ECS

Note

AWS Toolkit 는 연결된 VPC 연결을 사용한 개발 환경 생성을 지원하지 않습니다. 또한 이외의 IDE를 사용하는 경우 약 5분의 로드 시간이 발생할 AWS Cloud9 수 있습니다.

스페이스 수준에서 VPC 연결을 관리하려면 스페이스 관리자 역할 또는 파워 유저 역할이 있어야 합니다. VPC에 대한 자세한 내용은 CodeCatalyst 관리자 안내서의 [CodeCatalyst에서 Amazon VPC 관리](#)를 참조하세요.

CodeCatalyst에서의 개발 환경 할당량

다음 표에서는 Amazon CodeCatalyst 의 개발 환경에 대한 할당량 및 제한에 대해 설명합니다. Amazon CodeCatalyst의 할당량에 대한 자세한 내용은 [CodeCatalyst 할당량](#) 섹션을 참조하세요.

월별 개발 환경 시간 수	개발 환경 시간은 스페이스의 전체 스토리지 제한에 영향을 받습니다. 자세한 내용은 요금 및 개발 환경 문제 해결 섹션을 참조하세요.
스페이스당 개발 환경 스토리지 양	개발 환경 스토리지는 스페이스의 전체 스토리지 제한의 영향을 받습니다. 자세한 내용은 요금 및 개발 환경 문제 해결 섹션을 참조하세요.
개발 환경 컴퓨팅의 양	개발 환경 컴퓨팅은 스페이스의 전체 스토리지 제한에 영향을 받습니다. 자세한 내용은 요금 및 개발 환경 문제 해결 섹션을 참조하세요.

CodeCatalyst에서 소프트웨어 패키지 게시 및 공유

Amazon CodeCatalyst는 완전관리형 패키지 리포지토리 서비스를 포함하며, 이 서비스를 사용하면 개발 팀은 애플리케이션 개발에 사용되는 소프트웨어 패키지를 안전하게 저장하고 공유할 수 있습니다. 이러한 패키지는 패키지 리포지토리에 저장되며, 이 리포지토리는 CodeCatalyst의 프로젝트 내에서 생성되고 구성됩니다.

단일 패키지 리포지토리는 지원되는 모든 패키지 유형의 패키지를 저장할 수 있습니다. CodeCatalyst는 다음 패키지 형식을 지원합니다.

- npm
- Maven
- NuGet
- Python

패키지 리포지토리의 패키지는 리포지토리가 포함된 프로젝트 멤버 간에 검색하고 공유할 수 있습니다.

리포지토리에서 패키지를 게시하고 사용하려면 리포지토리 엔드포인트(URL)를 사용하도록 패키지 관리자를 구성합니다. 그런 다음 패키지 관리자를 사용하여 패키지를 리포지토리에 게시할 수 있습니다. Maven, Gradle, npm, yarn, nuget, dotnet, pip, twine와 같은 패키지 관리자를 사용할 수 있습니다.

CodeCatalyst 패키지 리포지토리를 사용하도록 CodeCatalyst 워크플로를 구성할 수도 있습니다. 워크플로에서 패키지를 사용하는 방법에 대한 자세한 내용은 [워크플로에 패키지 리포지토리 연결](#) 섹션을 참조하세요.

업스트림 리포지토리로 추가하여 한 패키지 리포지토리의 패키지를 동일한 프로젝트의 다른 리포지토리에서 사용할 수 있도록 할 수 있습니다. 업스트림 리포지토리에서 사용할 수 있는 모든 패키지 버전을 다운스트림 리포지토리에서도 사용할 수 있습니다. 자세한 내용은 [업스트림 리포지토리 구성 및 사용](#) 단원을 참조하십시오.

게이트웨이라는 특수 유형의 리포지토리를 생성하여 CodeCatalyst 리포지토리에서 오픈 소스 패키지를 사용할 수 있도록 할 수 있습니다. 게이트웨이 리포지토리로 업스트리밍하면 npmjs.com 및 pypi.org 같이 널리 사용되는 퍼블릭 리포지토리의 패키지를 사용하고 CodeCatalyst 리포지토리에 해당 패키지를 자동으로 캐싱할 수 있습니다. 자세한 내용은 [퍼블릭 외부 리포지토리에 연결](#) 단원을 참조하십시오.

주제

- [패키지 개념](#)
- [패키지 리포지토리 구성 및 사용](#)
- [업스트림 리포지토리 구성 및 사용](#)
- [퍼블릭 외부 리포지토리에 연결](#)
- [패키지 게시 및 수정](#)
- [npm 사용](#)
- [Maven 사용](#)
- [NuGet 사용](#)
- [Python 사용](#)
- [패키지 할당량](#)

패키지 개념

다음은 CodeCatalyst에서 패키지를 관리, 게시 또는 사용할 때 알아야 할 몇 가지 개념 및 용어입니다.

Packages

패키지는 소프트웨어와 모든 종속성을 해결하고 소프트웨어를 설치하는 데 필요한 메타데이터를 포함한 번들입니다. CodeCatalyst는 npm 패키지 형식을 지원합니다.

패키지는 다음으로 구성됩니다.

- 이름(예: webpack는 널리 사용되는 npm 패키지의 이름)
- 선택적 [네임스페이스](#)(예: @types/node의 @types)
- [버전](#) 세트(예: 1.0.0, 1.0.1, 1.0.2)
- 패키지 수준 메타데이터(예: npm dist 태그)

패키지 네임스페이스

일부 패키지 형식은 계층적 패키지 이름을 지원하여 패키지를 논리적 그룹으로 구성할 수 있고 이름 충돌을 방지하는 데 도움이 됩니다. 이름이 같은 패키지는 다른 네임스페이스에 저장할 수 있습니다. 예를 들어 npm은 범위를 지원하며 npm 패키지 @types/node는 @types의 범위 및 node의 이름을 지닙니다. @types 범위에는 다른 패키지 이름이 많습니다. CodeCatalyst에서 범위('유형')는 패키지 네임스페이스라고 하고 이름('노드')은 패키지 이름이라고 합니다. Maven 패키지의 경우 패키지 네임스

페이스는 Maven groupId에 해당합니다. `org.apache.logging.log4j:log4j` Maven 패키지에는 `org.apache.logging.log4j` groupId(패키지 네임스페이스)와 `log4j` artifactID(패키지 이름)가 있습니다. Python과 같은 일부 패키지 형식은 npm 범위 또는 Maven groupId와 유사한 개념의 계층적 이름을 지원하지 않습니다. 패키지 이름을 그룹화하는 방법이 없으면 이름 충돌을 피하기가 더 어려울 수 있습니다.

패키지 버전

패키지 버전은 패키지의 특정 버전(예: `@types/node@12.6.9`)을 식별합니다. 버전 번호 형식과 의미 체계는 패키지 형식에 따라 다릅니다. 예를 들어, npm 패키지 버전은 [의미 체계 버전 관리 사양](#)을 준수해야 합니다. CodeCatalyst 패키지 버전은 버전 식별자, 패키지 버전 수준 메타데이터 및 자산 세트로 구성됩니다.

자산

자산은 패키지 버전(예: npm .tgz 파일, Maven POM 및 JAR 파일)과 관련된 CodeCatalyst에 저장된 개별 파일입니다.

패키지 리포지토리

CodeCatalyst 패키지 리포지토리에는 [패키지](#) 세트가 포함되어 있으며, 이 세트에 포함된 각 [패키지 버전](#)은 [자산](#) 세트에 매핑됩니다. 패키지 리포지토리는 다국어 개체이며, 따라서 지원되는 모든 유형의 패키지가 단일 리포지토리에 포함될 수 있습니다. 각 패키지 리포지토리는 NuGet CLIs(nuget, dotnet), npm CLI, Maven CLI(mvn) 및 Python CLI(pip 및 twine)와 같은 도구를 사용하여 패키지를 가져오고 게시하기 위한 엔드포인트를 노출합니다. 각 스페이스에서 생성할 수 있는 패키지 리포지토리 수를 포함하여 CodeCatalyst 패키지 할당량에 대한 자세한 내용은 [패키지 할당량](#) 섹션을 참조하세요.

패키지 리포지토리를 업스트림으로 설정하여 다른 패키지 리포지토리에 연결할 수 있습니다. 리포지토리가 업스트림으로 설정된 경우 업스트림의 모든 패키지와 체인의 추가 업스트림 리포지토리를 사용할 수 있습니다. 자세한 내용은 [업스트림 리포지토리](#) 단원을 참조하십시오.

게이트웨이 리포지토리는 공식 외부 패키지 권한에서 패키지를 가져와 저장하는 특별한 유형의 패키지 리포지토리입니다. 자세한 내용은 [게이트웨이 리포지토리](#) 단원을 참조하십시오.

업스트림 리포지토리

CodeCatalyst를 사용하여 두 패키지 리포지토리 간에 업스트림 관계를 만들 수 있습니다. 다운스트림 리포지토리의 패키지 리포지토리 엔드포인트에서 패키지 리포지토리의 패키지 버전에 액세스할 수 있

는 경우 해당 패키지 리포지토리는 다른 패키지 리포지토리의 업스트림입니다. 업스트림 관계를 사용하면 두 패키지 리포지토리의 내용이 클라이언트의 관점에서 효과적으로 병합됩니다.

예를 들어 패키지 관리자가 리포지토리에 없는 패키지 버전을 요청하는 경우 CodeCatalyst는 패키지 버전에 대해 구성된 업스트림 리포지토리를 검색합니다. 업스트림 리포지토리는 구성된 순서대로 검색되며 패키지가 발견되면 CodeCatalyst가 검색을 중지합니다.

게이트웨이 리포지토리

게이트웨이 리포지토리는 지원되는 외부 공식 패키지 권한에 연결되는 특수한 유형의 패키지 리포지토리입니다. 게이트웨이 리포지토리를 [업스트림 리포지토리](#)로 추가할 때 해당 공식 패키지 권한의 패키지를 사용할 수 있습니다. 다운스트림 리포지토리는 퍼블릭 리포지토리나 통신하지 않으며, 대신 모든 것이 게이트웨이 리포지토리가 중재합니다. 이러한 방식으로 사용되는 패키지는 게이트웨이 리포지토리나 원래 요청을 받은 다운스트림 리포지토리 모두에 저장됩니다.

게이트웨이 리포지토리는 사전 정의되어 있지만 사용할 각 프로젝트에서 생성해야 합니다. 다음 목록에는 CodeCatalyst에서 생성할 수 있는 모든 게이트웨이 리포지토리와 게이트웨이 리포지토리가 연결된 패키지 권한이 포함되어 있습니다.

- npm-public-registry-gateway는 npmjs.com에서 npm 패키지를 제공합니다.
- maven-central-gateway는 Maven Central 리포지토리에서 Maven 패키지를 제공합니다.
- google-android-gateway는 Google Android에서 Maven 패키지를 제공합니다.
- commonsware-gateway는 CommonsWare에서 Maven 패키지를 제공합니다.
- gradle-plugins-gateway는 Gradle Plugins에서 Maven 패키지를 제공합니다.
- nuget-gallery-gateway는 NuGet Gallery에서 NuGet 패키지를 제공합니다.
- pypi-gateway는 Python Package Index에서 Python 패키지를 제공합니다.

패키지 리포지토리 구성 및 사용

CodeCatalyst에서 패키지는 패키지 리포지토리 내에 저장되고 관리됩니다. 패키지를 CodeCatalyst에 게시하거나 CodeCatalyst(또는 지원되는 모든 퍼블릭 패키지 리포지토리)의 패키지를 사용하려면 패키지 리포지토리를 생성하고 패키지 관리자를 연결해야 합니다.

주제

- [패키지 리포지토리 생성](#)
- [패키지 리포지토리에 연결](#)

- [패키지 리포지토리 삭제](#)

패키지 리포지토리 생성

CodeCatalyst에서 패키지 리포지토리를 생성하려면 다음 단계를 수행합니다.

패키지 리포지토리를 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 패키지 리포지토리를 생성하려는 프로젝트로 이동합니다.
3. 탐색 창에서 패키지를 선택합니다.
4. 패키지 리포지토리 페이지에서 패키지 리포지토리 생성을 선택합니다.
5. 패키지 리포지토리 세부 정보 섹션에서 다음을 추가합니다.
 - a. 리포지토리 이름. 프로젝트 또는 팀 이름 또는 리포지토리 사용 방식과 같은 세부 정보와 함께 설명 이름을 사용하는 것이 좋습니다.
 - b. (선택 사항) 설명. 리포지토리 설명은 프로젝트의 여러 팀에 여러 리포지토리가 있는 경우 특히 유용합니다.
6. 업스트림 리포지토리 섹션에서 업스트림 리포지토리 선택을 선택하여 CodeCatalyst 패키지 리포지토리를 통해 액세스하려는 패키지 리포지토리를 추가합니다. 게이트웨이 리포지토리를 추가하여 외부 패키지 리포지토리 또는 기타 CodeCatalyst 리포지토리에 연결할 수 있습니다.
 - 패키지 리포지토리에서 패키지를 요청하면 업스트림 리포지토리가 이 목록에 표시된 순서대로 검색됩니다. 패키지가 발견되면 CodeCatalyst는 검색을 중지합니다. 업스트림 리포지토리의 순서를 변경하려면 목록에서 리포지토리를 끌어다 놓을 수 있습니다.
7. 생성을 선택하여 패키지 리포지토리를 생성합니다.

패키지 리포지토리에 연결

CodeCatalyst에서 패키지를 게시하거나 사용하려면 패키지 리포지토리 엔드포인트 정보 및 CodeCatalyst 자격 증명 정보로 패키지 관리자를 구성해야 합니다. 리포지토리를 생성하지 않은 경우 [패키지 리포지토리 생성](#)의 지침에 따라 리포지토리를 생성할 수 있습니다.

패키지 관리자를 CodeCatalyst 패키지 리포지토리에 연결하는 방법에 대한 지침은 다음 설명서를 참조하세요.

- [Gradle Groovy 구성 및 사용](#)

- [mvn 구성 및 사용](#)
- [nuget 또는 dotnet CLI 구성 및 사용](#)
- [npm 구성 및 사용](#)
- [pip 구성 및 Python 패키지 설치](#)
- [Twine 구성 및 Python 패키지 게시](#)

패키지 리포지토리 삭제

CodeCatalyst에서 패키지 리포지토리를 삭제하려면 다음 단계를 수행합니다.

패키지 리포지토리를 삭제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 삭제하려는 패키지 리포지토리가 포함된 프로젝트로 이동합니다.
3. 탐색 창에서 패키지를 선택합니다.
4. 패키지 리포지토리 페이지에서 삭제할 리포지토리를 선택합니다.
5. 삭제를 선택합니다.
6. 패키지 리포지토리 삭제의 영향에 대해 제공된 정보를 검토합니다.
7. 입력 필드에 delete를 입력한 다음 삭제를 선택합니다.

업스트림 리포지토리 구성 및 사용

게이트웨이 리포지토리와 기타 CodeCatalyst 패키지 리포지토리를 모두 패키지 리포지토리의 업스트림으로 연결할 수 있습니다. 이를 통해 패키지 관리자 클라이언트는 단일 패키지 리포지토리 엔드포인트를 사용하여 둘 이상의 패키지 리포지토리에 포함된 패키지에 액세스할 수 있습니다. 다음은 업스트림 리포지토리 사용의 주요 이점입니다.

- 여러 소스에서 가져오고자 한다면 패키지 관리자를 단일 리포지토리 엔드포인트로 구성하기만 하면 됩니다.
- 업스트림 리포지토리에서 사용되는 패키지는 다운스트림 리포지토리에 저장되므로 업스트림 리포지토리에서 예기치 않은 중단이 발생하거나 패키지가 삭제되더라도 패키지를 사용할 수 있습니다.

패키지 리포지토리를 생성할 때 업스트림 리포지토리를 추가할 수 있습니다. CodeCatalyst 콘솔의 기존 패키지 리포지토리에서 업스트림 리포지토리를 추가하거나 제거할 수도 있습니다.

게이트웨이 리포지토리를 업스트림 리포지토리로 추가하면, 패키지 리포지토리가 게이트웨이 리포지토리의 해당 퍼블릭 패키지 리포지토리에 연결됩니다. 지원되는 퍼블릭 패키지 리포지토리 목록은 [지원되는 외부 패키지 리포지토리 및 해당 리포지토리의 게이트웨이 리포지토리](#)를 참조하세요.

여러 리포지토리를 업스트림 리포지토리로 함께 연결할 수 있습니다. 예를 들어 팀이 project-repo라는 리포지토리를 생성하는데, 퍼블릭 npm 리포지토리인 npmjs.com에 연결되어 업스트림 리포지토리로 추가된 npm-public-registry-gateway가 있는 team-repo라는 다른 리포지토리를 이미 사용하고 있다고 가정해 보겠습니다. team-repo를 업스트림 리포지토리로 project-repo에 추가할 수 있습니다. 이 경우 project-repo, team-repo, npm-public-registry-gateway 및 npmjs.com에서 패키지를 가져오는데 project-repo를 사용하도록 패키지 관리자만 구성하면 됩니다.

주제

- [업스트림 리포지토리 추가](#)
- [업스트림 리포지토리의 검색 순서 편집](#)
- [업스트림 리포지토리가 포함된 패키지 버전 요청](#)
- [업스트림 리포지토리 제거](#)

업스트림 리포지토리 추가

퍼블릭 패키지 리포지토리 또는 다른 CodeCatalyst 패키지 리포지토리를 다운스트림 리포지토리에 업스트림 리포지토리로 추가하면, 다운스트림 리포지토리에 연결된 패키지 관리자가 업스트림 리포지토리의 모든 패키지를 사용할 수 있습니다.

업스트림 리포지토리를 추가하려면

1. 탐색 창에서 패키지를 선택합니다.
2. 패키지 리포지토리 페이지에서 업스트림 리포지토리를 추가할 패키지 리포지토리를 선택합니다.
3. 패키지 리포지토리 이름에서 업스트림을 선택하고 업스트림 리포지토리 선택을 선택합니다.
4. 업스트림 유형 선택에서 다음 중 하나를 선택합니다.
 - 게이트웨이 리포지토리

사용 가능한 게이트웨이 리포지토리 목록에서 선택할 수 있습니다.

Note

CodeCatalyst는 Maven Central, npmjs.com 또는 Nuget Gallery와 같은 퍼블릭 외부 패키지 권한에 연결하기 위해 게이트웨이 리포지토리를 외부 리포지토리에서 가져온 패키지를 검색하고 저장하는 중개 리포지토리로 사용합니다. 이렇게 하면 프로젝트의 모든 패키지 리포지토리가 게이트웨이 중개 리포지토리의 패키지를 사용하므로 시간과 데이터 전송이 절약됩니다. 자세한 내용은 [퍼블릭 외부 리포지토리에 연결](#) 단원을 참조하십시오.

- CodeCatalyst 리포지토리

프로젝트의 사용 가능한 CodeCatalyst 패키지 리포지토리 목록에서 선택할 수 있습니다.

5. 업스트림 리포지토리로 추가하려는 모든 리포지토리를 선택한 후 선택을 선택한 다음 저장을 선택합니다.

업스트림 리포지토리의 검색 순서 변경에 대한 자세한 내용은 [업스트림 리포지토리의 검색 순서 편집](#) 섹션을 참조하세요.

업스트림 리포지토리를 추가한 경우 로컬 리포지토리에 연결된 패키지 관리자를 사용하여 업스트림 리포지토리에서 패키지를 가져올 수 있습니다. 패키지 관리자 구성을 업데이트할 필요는 없습니다. 업스트림 리포지토리에서 패키지 버전을 요청하는 방법에 대한 자세한 내용은 [업스트림 리포지토리가 포함된 패키지 버전 요청](#) 섹션을 참조하세요.

업스트림 리포지토리의 검색 순서 편집

CodeCatalyst는 구성된 검색 순서로 업스트림 리포지토리를 검색합니다. 패키지가 발견되면 CodeCatalyst는 검색을 중지합니다. 패키지를 찾기 위해 업스트림 리포지토리를 검색하는 순서는 변경할 수 있습니다.

업스트림 리포지토리의 검색 순서를 편집하려면

1. 탐색 창에서 패키지를 선택합니다.
2. 패키지 리포지토리 페이지에서 업스트림 리포지토리 검색 순서를 편집할 패키지 리포지토리를 선택합니다.
3. 패키지 리포지토리 이름에서 업스트림을 선택합니다.
4. 업스트림 리포지토리 섹션에서 업스트림 리포지토리와 해당 검색 순서를 볼 수 있습니다. 검색 순서를 변경하려면 목록에 리포지토리를 끌어다 놓습니다.

5. 업스트림 리포지토리의 검색 순서 편집이 완료되면 저장을 선택합니다.

업스트림 리포지토리가 포함된 패키지 버전 요청

다음 예시는 패키지 관리자가 업스트림 리포지토리가 있는 CodeCatalyst 패키지 리포지토리에서 패키지를 요청하는 경우 발생할 수 있는 시나리오를 보여줍니다.

이 예시에서는 npm과 같은 패키지 관리자가 여러 업스트림 리포지토리가 있는 downstream이라는 패키지 리포지토리에서 패키지 버전을 요청합니다. 패키지를 요청하면 다음이 발생할 수 있습니다.

- 요청된 패키지 버전이 downstream에 포함되어 있는 경우 클라이언트에 반환됩니다.
- 요청된 패키지 버전이 downstream에 포함되어 있지 않은 경우 CodeCatalyst는 downstream 업스트림 리포지토리에서 구성된 검색 순서에 따라 해당 버전을 찾습니다. 패키지를 찾으면 해당 패키지에 대한 참조가 downstream에 복사되고 패키지 버전이 클라이언트에 반환됩니다.
- downstream 및 해당 업스트림 리포지토리 모두에 패키지 버전이 포함되어 있지 않은 경우 HTTP 404 Not Found 응답이 클라이언트에 반환됩니다.

하나의 리포지토리에 허용되는 최대 직접 업스트림 리포지토리 수는 10개입니다. 패키지 버전이 요청될 때 CodeCatalyst가 확인하는 최대 리포지토리 수는 25개입니다.

업스트림 리포지토리의 패키지 보존

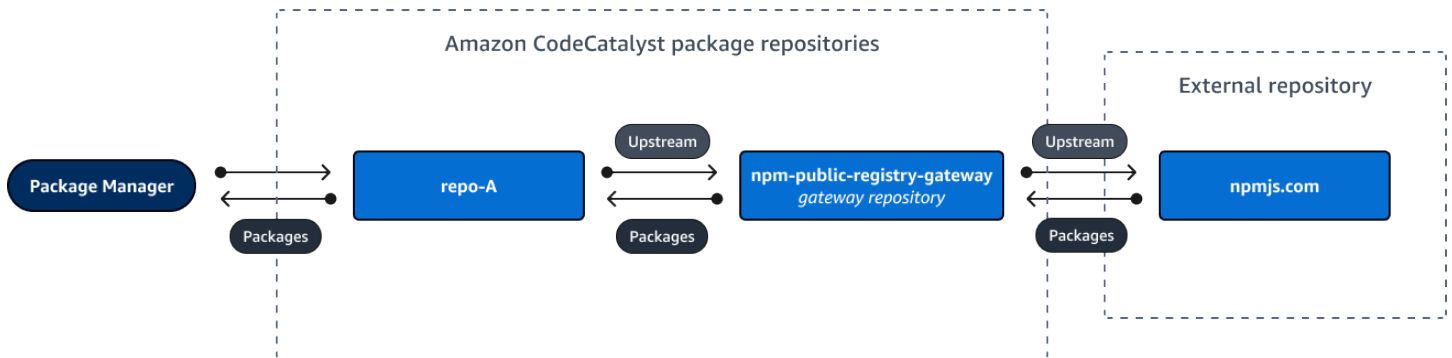
요청된 패키지 버전이 업스트림 리포지토리에서 발견되면 해당 버전에 대한 참조가 유지되며 해당 버전의 요청을 받은 리포지토리에서 항상 사용할 수 있습니다. 이렇게 하면 업스트림 리포지토리가 예기치 않게 중단되는 경우에 패키지에 액세스할 수 있습니다. 유지되는 패키지 버전은 다음 사항에 영향을 받지 않습니다.

- 업스트림 리포지토리 삭제
- 업스트림 리포지토리와 다운스트림 리포지토리 간 연결 해제
- 업스트림 리포지토리에서 패키지 버전 삭제
- 업스트림 리포지토리의 패키지 버전 편집(예: 새 자산 추가)

업스트림 관계를 통해 패키지 가져오기

CodeCatalyst는 업스트림 리포지토리라는 연결된 여러 리포지토리를 통해 패키지를 가져올 수 있습니다. CodeCatalyst 패키지 리포지토리에 게이트웨이 리포지토리에 대한 업스트림 연결이 있는 다른 CodeCatalyst 패키지 리포지토리에 대한 업스트림 연결이 있는 경우, 업스트림 리포지토리에 없는 패

키지에 대한 요청은 외부 리포지토리에서 복사됩니다. 예를 들어, 다음과 같은 구성을 살펴보겠습니다. repo-A라는 리포지토리는 npm-public-registry-gateway라는 게이트웨이 리포지토리에 대한 업스트림 연결이 있습니다. npm-public-registry-gateway는 퍼블릭 패키지 리포지토리인 <https://npmjs.com> 대한 업스트림 연결이 있습니다.



npm이 repo-A 리포지토리를 사용하도록 구성된 경우 `npm install`을 실행하면 <https://npmjs.com>에서 npm-public-registry-gateway로 패키지 복사가 시작됩니다. 설치된 버전도 함께 repo-A로 가져옵니다. 다음 예시에서는 `lodash`를 설치합니다.

```

$ npm config get registry
https://packages.region.codecatalyst.aws/npm/space-name/proj-name/repo-name/
$ npm install lodash
+ lodash@4.17.20
added 1 package from 2 contributors in 6.933s

```

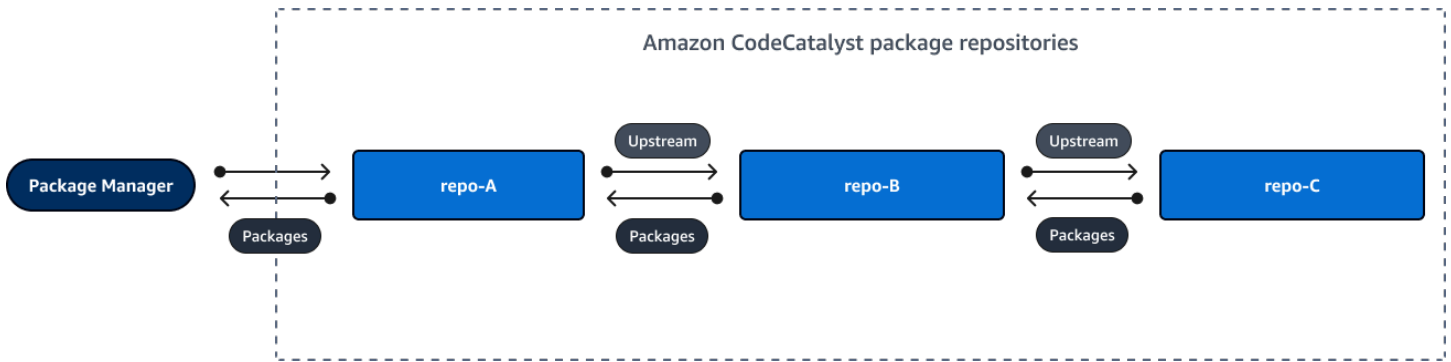
`npm install`을 실행한 후에는 repo-A에는 최신 버전(`lodash 4.17.20`)만 포함됩니다. 해당 버전이 repo-A에서 npm를 통해 가져온 버전이기 때문입니다.

npm-public-registry-gateway에 <https://npmjs.com>에 대한 외부 업스트림 연결이 있기 때문에 <https://npmjs.com>에서 가져온 모든 패키지 버전이 npm-public-registry-gateway에 저장됩니다. 이러한 패키지 버전은 npm-public-registry-gateway로 업스트림 연결이 있는 모든 다운스트림 리포지토리에서 가져올 수 있었을 것입니다.

npm-public-registry-gateway의 내용은 시간이 지남에 따라 <https://npmjs.com>에서 가져온 모든 패키지 및 패키지 버전을 볼 수 있는 방법을 제공합니다.

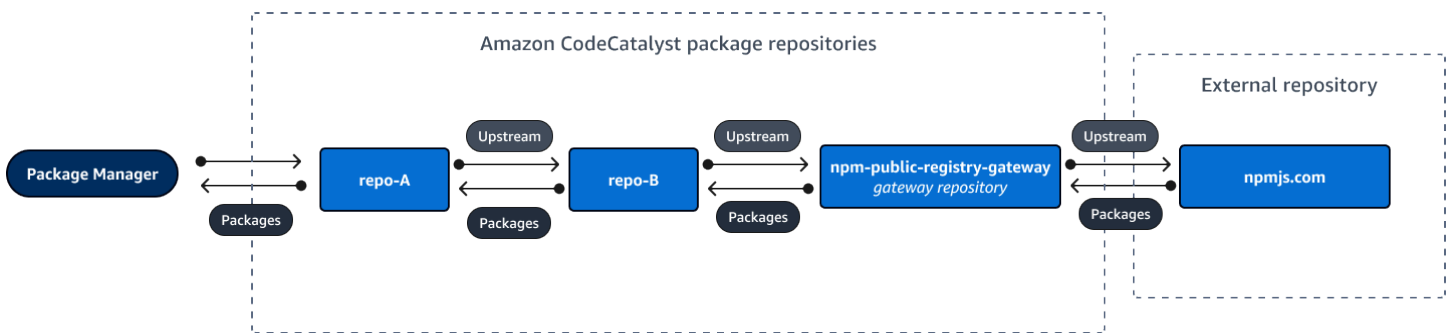
중간 리포지토리에 패키지 보존

CodeCatalyst를 사용하면 업스트림 리포지토리를 연결할 수 있습니다. 예를 들어 repo-B는 repo-A의 업스트림 리포지토리가 되고 repo-C는 repo-B의 업스트림 리포지토리가 됩니다. 이 구성을 통해 repo-A에서 repo-B 및 repo-C 패키지 버전을 사용할 수 있습니다.



패키지 관리자가 repo-A 리포지토리에 연결하고 repo-C 리포지토리에서 패키지 버전을 가져오면 해당 패키지 버전은 repo-B 리포지토리에 유지되지 않습니다. 이 예시에서는 패키지 버전은 최종 repo-A 다운스트림 리포지토리에만 유지됩니다. 중간 리포지토리에는 유지되지 않습니다. 더 긴 체인의 경우에도 마찬가지입니다. 예를 들어, 네 개의 리포지토리 repo-A, repo-B, repo-C, repo-D 및 패키지 관리자가 repo-A에 연결되어 있고 repo-D에서 패키지 버전을 가져온 경우, 해당 패키지 버전은 repo-A에는 유지되지만 repo-B 또는 repo-C에는 유지되지 않습니다.

패키지 보존 동작은 퍼블릭 패키지 리포지토리에서 패키지 버전을 가져올 때와 비슷합니다. 단, 패키지 버전이 퍼블릭 리포지토리에 대한 직접 업스트림 연결이 연결된 게이트웨이 리포지토리에 항상 유지된다는 점이 다릅니다. 예를 들어 repo-B는 repo-A의 업스트림 리포지토리로 사용됩니다. npm-public-registry-gateway는 repo-B의 업스트림 리포지토리로, 퍼블릭 리포지토리 npmjs.com에 대한 업스트림 연결이 연결됩니다. 아래 다이어그램을 참조하세요.



repo-A에 연결된 패키지 관리자가 특정 패키지 버전(예: lodash 4.17.20)을 요청했으나 해당 패키지 버전이 세 리포지토리 중 어디에도 없는 경우 npmjs.com에서 해당 패키지 버전을 가져옵니다. lodash 4.17.20을 가져오면 이는 최종 다운스트림 리포지토리인 repo-A와 퍼블릭 외부 리포지토리인 npmjs.com에 대한 업스트림 연결이 되어 있는 npm-public-registry-gateway에 유지됩니다. lodash 4.17.20은 중간 저장소인 repo-B에는 유지되지 않습니다.

업스트림 리포지토리 제거

업스트림 리포지토리 내의 패키지에 더 이상 액세스하지 않으려면 패키지 리포지토리에서 업스트림 리포지토리를 제거할 수 있습니다.

Warning

업스트림 리포지토리를 제거하면 업스트림 관계 체인이 중단되어 프로젝트 또는 빌드가 중단될 수 있습니다.

업스트림 리포지토리를 제거하려면

1. 탐색 창에서 패키지를 선택합니다.
2. 패키지 리포지토리 페이지에서 업스트림 리포지토리를 제거할 패키지 리포지토리를 선택합니다.
3. 패키지 리포지토리 이름에서 업스트림을 선택합니다.
4. 업스트림 리포지토리 편집 섹션에서 제거하려는 업스트림 리포지토리를 찾은 다음
를 선택합니다.
5. 업스트림 리포지토리 제거가 완료되면 저장을 선택합니다.

퍼블릭 외부 리포지토리에 연결

해당 게이트웨이 리포지토리를 업스트림 리포지토리로 추가하여, CodeCatalyst 패키지 리포지토리를 지원하는 퍼블릭 외부 리포지토리에 연결할 수 있습니다. 게이트웨이 리포지토리는 외부 리포지토리에서 가져온 패키지를 검색하고 저장하는 중간 리포지토리 역할을 합니다. 이렇게 하면 프로젝트의 모든 패키지 리포지토리가 게이트웨이 리포지토리의 저장된 패키지를 사용할 수 있으므로 시간과 데이터 전송이 절약됩니다.

게이트웨이 리포지토리를 사용하여 퍼블릭 리포지토리에 연결하려면

1. 탐색 창에서 패키지를 선택합니다.
2. 패키지에서 게이트웨이 리포지토리 페이지를 선택합니다. 지원되는 게이트웨이 리포지토리 목록과 그 설명을 볼 수 있습니다.

3. 게이트웨이 리포지토리를 사용하려면 먼저 게이트웨이 리포지토리를 생성해야 합니다. 게이트웨이 리포지토리가 생성된 경우 생성된 날짜와 시간이 표시됩니다. 그렇지 않은 경우 생성을 선택하여 생성합니다.
4. 게이트웨이 리포지토리의 패키지를 사용하려면 CodeCatalyst 리포지토리에서 게이트웨이 리포지토리에 업스트림 연결을 설정해야 합니다. 패키지 리포지토리를 선택하고 연결할 패키지 리포지토리를 선택합니다.
5. 퍼블릭 리포지토리에 연결하려면 업스트림을 선택하고 업스트림 리포지토리 선택을 선택합니다.
6. 게이트웨이 리포지토리를 선택하고, 업스트림 리포지토리로 연결할 퍼블릭 리포지토리에 해당하는 게이트웨이 리포지토리를 선택합니다.
7. 업스트림 리포지토리로 추가하려는 게이트웨이 리포지토리를 모두 선택한 경우 선택을 선택합니다.
8. 업스트림 리포지토리 정렬이 완료되면 저장을 선택합니다.

업스트림 리포지토리에 대한 자세한 내용은 [업스트림 리포지토리 구성 및 사용](#) 섹션을 참조하세요.

게이트웨이 리포지토리를 업스트림 리포지토리로 추가한 경우, 로컬 리포지토리에 연결된 패키지 관리자를 사용하여 해당 리포지토리에 해당하는 퍼블릭 외부 패키지 리포지토리에서 패키지를 가져올 수 있습니다. 패키지 관리자 구성을 업데이트할 필요는 없습니다. 이러한 방식으로 사용되는 패키지는 게이트웨이 리포지토리와 로컬 패키지 리포지토리 모두에 저장됩니다. 업스트림 리포지토리에서 패키지 버전을 요청하는 방법에 대한 자세한 내용은 [업스트림 리포지토리가 포함된 패키지 버전 요청](#) 섹션을 참조하세요.

지원되는 외부 패키지 리포지토리 및 해당 리포지토리의 게이트웨이 리포지토리

CodeCatalyst는 게이트웨이 리포지토리가 있는 다음 공식 패키지 권한에 업스트림 연결을 추가하도록 지원합니다.

리포지토리 패키지 유형	설명	게이트웨이 리포지토리 이름
npm	npm 공용 레지스트리	npm-public-registry-gateway
Python	Python Package Index	pypi-gateway

리포지토리 패키지 유형	설명	게이트웨이 리포지토리 이름
Maven	Maven Central	maven-central-gateway
Maven	Google Android Repository	google-android-gateway
Maven	CommonsWare	commonsware-gateway
Maven	Gradle 플러그인 저장소	gradle-plugins-gateway
NuGet	NuGet Gallery	nuget-gallery-gateway

패키지 게시 및 수정

CodeCatalyst의 패키지는 종속성을 해결하고 소프트웨어를 설치하는 데 필요한 소프트웨어 및 메타데이터 번들입니다. CodeCatalyst에서 지원되는 패키지 형식 목록은 [CodeCatalyst에서 소프트웨어 패키지 게시 및 공유](#) 섹션을 참조하세요. 이 섹션에서는 패키지 게시, 보기, 삭제, 패키지 버전 상태 업데이트에 대한 정보를 제공합니다.

주제

- [CodeCatalyst 패키지 리포지토리에 패키지 게시](#)
- [패키지 버전 세부 정보 보기](#)
- [패키지 버전 삭제](#)
- [패키지 버전의 상태 업데이트](#)
- [패키지 원본 제어 편집](#)

CodeCatalyst 패키지 리포지토리에 패키지 게시

패키지 관리자 도구를 사용하여 지원되는 패키지 유형의 버전을 CodeCatalyst 패키지 리포지토리에 게시할 수 있습니다. 패키지 버전을 게시하는 단계는 다음과 같습니다.

패키지 버전을 CodeCatalyst 패키지 리포지토리에 게시하려면

1. 생성하지 않은 경우 [패키지 리포지토리를 생성합니다](#).
2. 패키지 관리자를 패키지 리포지토리에 연결합니다. npm 패키지 관리자를 CodeCatalyst 패키지 리포지토리에 연결하는 방법에 대한 지침은 [npm 구성 및 사용](#) 섹션을 참조하세요.
3. 연결된 패키지 관리자를 사용하여 패키지 버전을 게시합니다.

목차

- [리포지토리 게시 및 업스트림](#)
- [프라이빗 패키지 및 퍼블릭 리포지토리](#)
- [패키지 자산 덮어쓰기](#)

리포지토리 게시 및 업스트림

CodeCatalyst에서는 연결 가능한 업스트림 리포지토리 또는 퍼블릭 리포지토리에 있는 패키지 버전을 게시할 수 없습니다. 예를 들어 npm 패키지 `lodash@1.0`을 패키지 리포지토리 `myrepo`에 게시하고 `myrepo`가 업스트림 리포지토리로 구성된 게이트웨이 리포지토리를 통해 `npmjs.com`에 연결되어 있다고 가정해 보겠습니다. `lodash@1.0`이 업스트림 리포지토리 또는 `npmjs.com`에 있다면, CodeCatalyst는 `myrepo`에서 이에 게시하려는 모든 시도를 거부하며 409 충돌 오류가 발생합니다. 이렇게 하면 업스트림 리포지토리의 패키지와 이름 및 버전이 동일한 패키지를 실수로 게시하지 못하게 되어 예기치 않은 동작이 발생할 수 있습니다.

업스트림 리포지토리에 있는 패키지 이름의 다른 버전을 게시할 수 있습니다. 예를 들어 `lodash@1.0`이 업스트림 리포지토리에 있지만 `lodash@1.1`이 아닌 경우 `lodash@1.1`을 다운스트림 리포지토리에 게시할 수 있습니다.

프라이빗 패키지 및 퍼블릭 리포지토리

CodeCatalyst는 CodeCatalyst 리포지토리에 저장된 패키지를 `npmjs.com` 또는 Maven Central 같은 퍼블릭 리포지토리에 게시하지 않습니다. CodeCatalyst는 퍼블릭 리포지토리에서 CodeCatalyst 리포지토리로 패키지를 가져오지만, 반대 방향으로 패키지를 이동시키지는 않습니다. CodeCatalyst 리포지토리에 게시하는 패키지는 비공개로 유지되며 리포지토리가 속한 CodeCatalyst 프로젝트에서만 사용할 수 있습니다.

패키지 자산 덮어쓰기

이미 존재하며 콘텐츠가 다른 패키지 자산은 다시 게시할 수 없습니다. 예를 들어 Maven 패키지를 JAR 자산 `mypackage-1.0.jar`과 함께 이미 게시했다고 가정해 보겠습니다. 이 자산은 이전 자산과 새 자산의 체크섬이 동일한 경우에만 다시 게시할 수 있습니다. 새로운 내용이 있는 동일한 자산을 다시 게시하려면 먼저 해당 패키지 버전을 삭제하세요. 콘텐츠가 다른 동일한 자산 이름을 다시 게시하려고 하면 HTTP 409 충돌 오류가 발생합니다.

여러 자산을 지원하는 패키지 형식(Python 및 Maven)의 경우, 필요한 권한이 있다면 언제든지 기존 패키지 버전에 이름이 다른 새 자산을 추가할 수 있습니다. npm와 NuGet은 패키지 버전당 자산 하나만 지원하므로, 게시된 패키지 버전을 수정하려면 먼저 패키지 버전을 삭제해야 합니다.

이미 존재하는 자산(예: `mypackage-1.0.jar`)을 다시 게시하려고 할 때 게시된 자산과 새 자산의 내용이 동일하면, 작업은 멍등성이기 때문에 성공하게 됩니다.

패키지 버전 세부 정보 보기

CodeCatalyst 콘솔을 사용하여 특정 패키지 버전에 대한 세부 정보를 볼 수 있습니다.

패키지 버전 세부 정보 보기

1. 탐색 창에서 패키지를 선택합니다.
2. 패키지 리포지토리 페이지에서 세부 정보를 보려는 패키지 버전이 포함된 리포지토리를 선택합니다.
3. 패키지 테이블에서 패키지 버전을 검색합니다. 검색 창을 사용하여 패키지 이름과 형식 지정에 따라 패키지를 필터링할 수 있습니다. 목록에서 패키지를 선택합니다.
4. 패키지 세부 정보 페이지에서 버전을 선택한 다음 보려는 버전을 선택합니다.

패키지 버전 삭제

CodeCatalyst 콘솔의 패키지 버전 세부 정보 페이지에서 패키지 버전을 삭제할 수 있습니다.

패키지 버전 삭제

1. 탐색 창에서 패키지를 선택합니다.
2. 패키지 리포지토리 페이지에서 삭제하려는 패키지 버전이 포함된 리포지토리를 선택합니다.
3. 테이블에서 패키지를 검색하고 선택합니다.

4. 패키지 세부 정보 페이지에서 버전을 선택하고 삭제할 버전을 선택합니다.
5. 패키지 버전 세부 정보 페이지에서 버전 작업을 선택하고 삭제를 선택합니다.
6. 텍스트 필드에 delete를 입력하고 삭제를 선택합니다.

패키지 버전의 상태 업데이트

CodeCatalyst의 각 패키지 버전에는 패키지 버전의 현재 상태 및 가용성을 설명하는 상태가 적용됩니다. CodeCatalyst 콘솔을 사용하여 패키지 버전 상태를 변경할 수 있습니다. 패키지 버전의 가능한 상태 값과 그 의미에 대한 자세한 내용은 [패키지 버전 상태](#) 섹션을 참조하세요.

패키지 버전의 상태 업데이트

1. 탐색 창에서 패키지를 선택합니다.
2. 패키지 리포지토리 페이지에서 상태를 업데이트하려는 패키지 버전이 포함된 리포지토리를 선택합니다.
3. 테이블에서 패키지를 검색하고 선택합니다.
4. 패키지 세부 정보 페이지에서 버전을 선택한 다음 보려는 버전을 선택합니다.
5. 패키지 버전 세부 정보 페이지에서 작업을 선택한 다음 나열 취소, 아카이브 또는 폐기를 선택합니다. 각 패키지 버전 상태에 대한 자세한 내용은 [패키지 버전 상태](#) 섹션을 참조하세요.
6. 텍스트 필드에 확인 텍스트를 입력한 다음 업데이트 중인 상태에 따라 나열 취소, 아카이브 또는 폐기를 선택합니다.

패키지 버전 상태

다음은 패키지 버전 상태에 적용되는 값입니다. 콘솔을 사용하여 패키지 버전 상태를 변경할 수 있습니다. 자세한 내용은 [패키지 버전의 상태 업데이트](#) 섹션을 참조하세요.

- 게시됨: 패키지 버전이 성공적으로 게시되었으며 패키지 관리자를 이용해 요청할 수 있습니다. 패키지 버전은 패키지 관리자에 반환되는 패키지 버전 목록(예: `npm view <package-name> versions` 출력)에 포함됩니다. 패키지 버전의 모든 자산은 리포지토리에서 사용할 수 있습니다.
- 완료되지 않음: 마지막 게시 시도가 완료되지 않았습니다. 현재 Maven 패키지 버전만 완료되지 않은 상태를 가질 수 있습니다. 클라이언트가 패키지 버전용 자산을 하나 이상 업로드하지만 해당 버전이 포함된 패키지의 `maven-metadata.xml` 파일을 게시하지 않으면 이 문제가 발생할 수 있습니다.
- 미등록 - 패키지 버전의 자산을 리포지토리에서 다운로드할 수 있지만, 패키지 버전이 패키지 관리자에게 반환되는 버전 목록에 포함되지 않습니다. 예를 들어 npm 패키지의 경우 `npm view`

<package-name> versions 출력에 패키지 버전이 포함되지 않습니다. 즉, 사용 가능한 버전 목록에 버전이 표시되지 않기 때문에 npm의 종속성 확인 로직은 패키지 버전을 선택하지 않습니다. 하지만 미등록 패키지 버전이 이미 npm package-lock.json 파일에서 참조된 경우, 예를 들어 npm ci를 실행 중일 때 해당 버전을 다운로드하여 설치할 수 있습니다.

- 아카이브됨 - 패키지 버전 자산을 다운로드할 수 없습니다. 패키지 버전은 패키지 관리자에 반환되는 버전 목록에 포함됩니다. 자산을 사용할 수 없으므로 클라이언트의 패키지 버전 사용이 차단됩니다. 애플리케이션 빌드가 아카이브됨으로 업데이트된 버전을 사용하는 경우, 패키지 버전이 로컬에 캐시되지 않으면 빌드가 실패합니다. 아카이브됨 패키지 버전은 리포지토리에 여전히 존재하므로 패키지 관리자나 빌드 도구를 사용하여 다시 게시할 수 없습니다. 하지만 콘솔에서 패키지 버전 상태를 미등록 또는 게시됨으로 변경할 수 있습니다.
- 폐기됨: 패키지 버전이 목록에 표시되지 않으며 리포지토리에서 자산을 다운로드할 수 없습니다. 폐기됨과 보관됨의 주된 차이점은 폐기됨 상태인 경우 CodeCatalyst가 패키지 버전의 자산을 영구적으로 삭제한다는 것입니다. 따라서 패키지 버전을 폐기됨에서 보관됨, 미등록 또는 게시됨으로 전환할 수 없습니다. 자산이 삭제되었으므로 패키지 버전을 사용할 수 없습니다. 패키지 버전이 폐기됨으로 표시되면 패키지 자산의 스토리지 비용이 청구되지 않습니다.

위 목록의 상태 외에도 패키지 버전을 삭제할 수도 있습니다. 삭제된 패키지 버전은 리포지토리에 없으며 패키지 관리자 또는 빌드 도구를 사용하여 해당 패키지 버전을 자유롭게 다시 게시할 수 있습니다.

패키지 이름, 패키지 버전 및 자산 이름 표준화

CodeCatalyst는 패키지 이름, 패키지 버전과 자산 이름을 저장하기 전에 먼저 표준화합니다. 즉, CodeCatalyst의 이름 또는 버전은 패키지가 게시될 때 제공된 것과 다를 수 있습니다. CodeCatalyst에서 각 패키지 유형의 이름 및 버전을 표준화하는 방법에 대한 자세한 내용은 다음 설명서를 참조하세요.

- [Python 패키지 이름 정규화](#)
- [NuGet 패키지 이름, 버전 및 자산 이름 표준화](#)

CodeCatalyst는 다른 패키지 형식에서는 표준화를 수행하지 않습니다.

패키지 원본 제어 편집

Amazon CodeCatalyst에서는 패키지 버전을 직접 게시하거나, 업스트림 리포지토리에서 가져오거나, 게이트웨이를 통해 외부 퍼블릭 리포지토리에서 수집하여, 패키지 버전을 리포지토리에 추가할 수 있습니다. 직접 게시와 퍼블릭 리포지토리에서 수집하는 방법 모두를 통해 패키지의 버전을 추가하면, 종속성 대체 공격에 취약해집니다. 자세한 내용은 [종속성 대체 공격](#) 단원을 참조하십시오. 종속성 대체

공격을 방어하려면 리포지토리의 패키지에서 패키지 원본 제어를 구성하여 해당 패키지 버전을 리포지토리에 추가하는 방법을 제한합니다.

직접 게시 같은 내부 소스와 퍼블릭 리포지토리 같은 외부 소스 모두에서 서로 다른 패키지의 새 버전을 가져오려면 패키지 원본 제어 구성을 고려해야 합니다. 기본적으로 패키지 원본 제어는 패키지의 첫 번째 버전이 리포지토리에 추가되는 방식을 기반으로 구성됩니다.

패키지 원본 제어 설정

패키지 원본 제어를 사용하여 패키지 버전을 리포지토리에 추가하는 방법을 구성할 수 있습니다. 다음 목록에는 사용 가능한 패키지 원본 제어 설정 및 값이 포함되어 있습니다.

게시

이 설정은 패키지 관리자 또는 이와 유사한 도구를 사용하여 패키지 버전을 리포지토리에 직접 게시할 수 있는지를 구성합니다.

- 허용: 패키지 버전을 직접 게시할 수 있습니다.
- 차단: 패키지 버전을 직접 게시할 수 없습니다.

업스트림

이 설정은 패키지 관리자가 요청하는 경우 패키지 버전을 외부 또는 퍼블릭 리포지토리에서 수집하거나 업스트림 리포지토리에서 유지할 수 있는지를 구성합니다.

- 허용: 모든 패키지 버전은 업스트림 리포지토리로 구성된 다른 CodeCatalyst 리포지토리에서 유지하거나 외부 연결을 통해 공개 소스에서 수집할 수 있습니다.
- 차단: 패키지 버전은 업스트림 리포지토리로 구성된 다른 CodeCatalyst 리포지토리에서 유지하거나 외부 연결을 통해 공개 소스에서 수집할 수 없습니다.

기본 패키지 원본 제어 설정

패키지의 기본 패키지 원본 제어는 해당 패키지의 첫 번째 버전이 패키지 리포지토리에 추가되는 방식을 기반으로 합니다.

- 패키지 관리자가 첫 번째 패키지 버전을 직접 게시하는 경우 설정은 게시: 허용 및 업스트림: 차단이 됩니다.
- 첫 번째 패키지 버전을 공개 소스에서 수집하는 경우 설정은 게시: 차단 및 업스트림: 허용이 됩니다.

일반적인 패키지 액세스 제어 시나리오

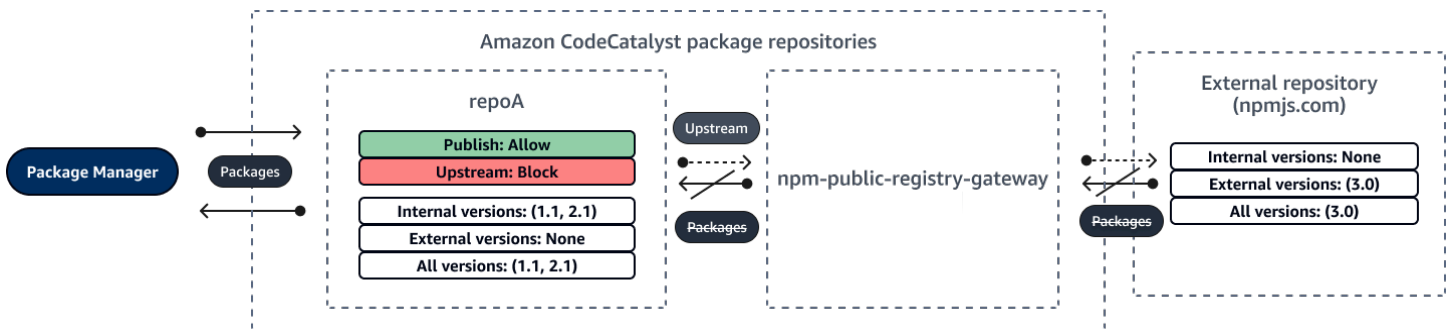
이 섹션에는 CodeCatalyst 리포지토리에 패키지 버전이 추가되는 일반적인 시나리오를 설명합니다. 새 패키지에 대한 패키지 원본 제어 설정은 첫 번째 패키지 버전이 추가되는 방식에 따라 설정됩니다.

다음 시나리오에서 내부 패키지는 패키지 관리자가 리포지토리에 직접 게시하는 패키지(예: 사용자가 유지 관리하는 패키지)입니다. 외부 패키지는 게이트웨이 리포지토리 업스트림을 통해 리포지토리에 수집될 수 있는 퍼블릭 리포지토리에 있는 패키지입니다.

외부 패키지 버전은 기존 내부 패키지를 대상으로 게시됩니다.

이 시나리오에서는 내부 패키지인 packageA를 가정합니다. 팀은 packageA의 첫 번째 패키지 버전을 CodeCatalyst 패키지 리포지토리에 게시합니다. 이 패키지의 첫 번째 패키지 버전이므로, 패키지 원본 제어 설정은 게시: 허용 및 업스트림: 차단으로 자동 설정됩니다. 패키지가 리포지토리에 게시되면, CodeCatalyst 패키지 리포지토리에 연결된 퍼블릭 리포지토리에 같은 이름의 패키지가 게시됩니다. 이는 내부 패키지에 대한 종속성 대체 공격 시도일 수도 있고 우연의 일치일 수도 있습니다. 하지만 패키지 원본 제어는 새로운 외부 버전의 수집을 차단하여 잠재적 공격을 방어하도록 구성되어 있습니다.

다음 이미지에서 repoA는 npm-public-registry-gateway 리포지토리에 대한 업스트림 연결이 있는 CodeCatalyst 패키지 리포지토리입니다. 리포지토리에 packageA 버전 1.1 및 2.1이 있지만 버전 3.0은 퍼블릭 리포지토리에 게시됩니다. 일반적으로 repoA는 패키지 관리자가 패키지를 요청한 후 버전 3.0을 수집합니다. 패키지 수집이 차단으로 설정되어 있기 때문에 버전 3.0은 CodeCatalyst 패키지 리포지토리에 수집되지 않으며 이 리포지토리에 연결된 패키지 관리자가 사용할 수 없습니다.

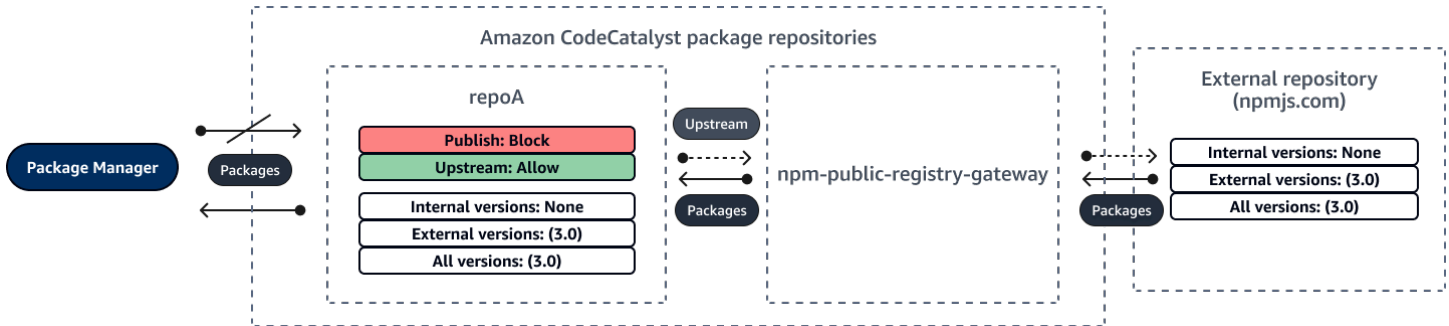


내부 패키지 버전은 기존 외부 패키지를 대상으로 게시됩니다.

이 시나리오에서는 패키지인 packageB가 리포지토리에 연결된 퍼블릭 리포지토리의 외부에 존재합니다. 리포지토리에 연결된 패키지 관리자가 packageB를 요청하면 해당 패키지 버전이 퍼블릭 리포지토리에서 사용자의 리포지토리로 수집됩니다. 이것은 리포지토리에 추가된 packageB의 첫 번째 패키지 버전이므로, 패키지 원본 설정은 게시: 차단 및 업스트림: 허용으로 구성됩니다. 나중에 패키지 이름이 동일한 버전을 리포지토리에 게시합니다. 퍼블릭 패키지를 모르는 상태에서 관련 없는 패키지를 같은 이름으로 게시하려고 하거나, 패치가 적용된 버전을 게시하려고 하거나, 이미 외부에 있는 패키지 버전

을 직접 게시하게 될 수 있습니다. CodeCatalyst는 사용자가 게시하려는 버전을 거부하지만, 사용자가 거부를 명시적으로 무시하고 필요한 경우에만 버전을 게시할 수 있습니다.

다음 이미지에서 repoA는 npm-public-registry-gateway 리포지토리에 대한 업스트림 연결이 있는 CodeCatalyst 패키지 리포지토리입니다. 패키지 리포지토리에는 퍼블릭 리포지토리에서 수집된 버전 3.0이 포함되어 있습니다. 버전 1.2를 리포지토리에 게시하려 합니다. 일반적으로 버전 1.2를 repoA에 게시할 수 있지만 게시가 차단으로 설정되어 있기 때문에 버전 1.2는 게시할 수 없습니다.



기존 외부 패키지의 패치가 적용된 패키지 버전 게시

이 시나리오에서는 패키지인 packageB가 패키지 리포지토리에 연결된 퍼블릭 리포지토리의 외부에 존재합니다. 리포지토리에 연결된 패키지 관리자가 packageB를 요청하면 해당 패키지 버전이 퍼블릭 리포지토리에서 사용자의 리포지토리로 수집됩니다. 이것은 사용자의 리포지토리에 추가된 packageB의 첫 번째 패키지 버전이므로, 패키지 원본 설정은 게시: 차단 및 업스트림: 허용으로 구성됩니다. 팀에서 이 패키지의 패치가 적용된 패키지 버전을 리포지토리에 게시하기로 합니다. 패키지 버전을 직접 게시하기 위해 팀은 패키지 원본 제어 설정을 게시: 허용 및 업스트림: 차단으로 변경합니다. 이제 이 패키지의 버전을 리포지토리에 직접 게시하고 퍼블릭 리포지토리에서 수집할 수 있습니다. 팀에서 패치된 패키지 버전을 게시하면 패키지 원본 설정을 게시: 차단 및 업스트림: 허용으로 되돌립니다.

패키지 원본 제어 편집

패키지 원본 제어는 패키지 첫 번째 패키지 버전이 리포지토리에 추가되는 방식에 따라 자동으로 구성됩니다. 자세한 내용은 [기본 패키지 원본 제어 설정](#) 단원을 참조하십시오. CodeCatalyst 패키지 리포지토리에서 패키지의 패키지 원본 제어를 추가하거나 편집하려면 다음 절차의 단계를 수행하세요.

패키지 원본 제어 추가 또는 편집하려면

1. 탐색 창에서 패키지를 선택합니다.
2. 편집하려는 패키지가 포함된 패키지 리포지토리를 선택합니다.
3. 패키지 표에서 편집할 패키지를 검색하여 선택합니다.
4. 패키지 요약 페이지의 원본 제어를 선택합니다.

5. 원본 제어에서 이 패키지에 설정할 패키지 원본 제어를 선택합니다. 패키지 원본 제어 설정인 게시와 업스트림을 동시에 설정해야 합니다.
 - 패키지 버전을 직접 게시할 수 있도록 허용하려면 게시에서 허용을 선택합니다. 패키지 버전 게시를 차단하려면 차단을 선택합니다.
 - 외부 리포지토리에서 패키지를 수집하고 업스트림 리포지토리에서 패키지를 가져오도록 허용하려면 업스트림 소스에서 허용을 선택합니다. 외부 및 업스트림 리포지토리에서의 패키지 버전 수집과 가져오기를 모두 차단하려면 차단을 선택합니다.
6. 저장을 선택합니다.

리포지토리 게시 및 업스트림

CodeCatalyst에서는 연결 가능한 업스트림 리포지토리 또는 퍼블릭 리포지토리에 있는 패키지 버전을 게시할 수 없습니다. 예를 들어 npm 패키지 `lodash@1.0`을 리포지토리 `myrepo`에 게시하려고 하는데 `myrepo`에 `npmjs.com`과의 외부 연결이 있는 업스트림 리포지토리가 있다고 가정해 보겠습니다. 다음 시나리오를 고려해 보세요.

1. `lodash`에서의 패키지 원본 제어 설정은 게시: 허용과 업스트림: 허용입니다. `lodash@1.0`이 업스트림 리포지토리 또는 `npmjs.com`에 있다면, CodeCatalyst는 `myrepo`에서 이에 게시하려는 모든 시도를 거부하며 409 충돌 오류가 발생합니다. `lodash@1.1` 같은 다른 버전은 게시할 수 있습니다.
2. `lodash`에서의 패키지 원본 제어 설정은 게시: 허용과 업스트림: 차단입니다. `lodash`의 모든 버전은 아직 존재하지 않는 리포지토리에 게시할 수 있습니다. 패키지 버전에 연결할 수 없기 때문입니다.
3. `lodash`에서의 패키지 원본 제어 설정은 게시: 차단과 업스트림: 허용입니다. 패키지 버전은 리포지토리에 직접 게시할 수 없습니다.

종속성 대체 공격

패키지 관리자는 재사용 가능한 코드를 패키징하고 공유하는 프로세스를 단순화합니다. 이러한 패키지는 애플리케이션에서 사용하기 위해 조직에서 개발한 프라이빗 패키지일 수도 있고, 조직 외부에서 개발되어 퍼블릭 패키지 리포지토리에서 배포하는 퍼블릭(대부분의 경우 오픈 소스) 패키지일 수도 있습니다. 패키지를 요청할 때 개발자는 패키지 관리자를 이용해 종속 항목의 새 버전을 가져옵니다. 종속성 혼동 공격이라고도 하는 종속성 대체 공격은 대부분의 패키지 관리자가 패키지의 합법적인 버전과 악성 버전을 구분할 방법이 없다는 사실을 악용합니다.

종속성 대체 공격은 소프트웨어 공급망 공격이라 알려진 공격의 하위 집합에 속합니다. 소프트웨어 공급망 공격은 소프트웨어 공급망에 존재하는 취약성을 악용하는 공격입니다.

종속성 대체 공격은 내부적으로 개발된 패키지와 퍼블릭 리포지토리에서 가져온 패키지를 모두 사용하는 사용자라면 누구나 노릴 수 있습니다. 공격자는 내부 패키지 이름을 식별한 다음 같은 이름의 악성 코드를 전략적으로 퍼블릭 패키지 리포지토리에 배치합니다. 일반적으로 악성 코드는 버전 번호가 높은 패키지에 게시됩니다. 패키지 관리자는 악성 패키지가 패키지 최신 버전이라고 생각하기 때문에 이러한 퍼블릭 피드에서 악성 코드를 가져옵니다. 결과적으로 원하는 패키지와 악성 패키지 간에 ‘혼동’ 또는 ‘대체’가 발생하여 코드가 손상될 수 있습니다.

종속성 대체 공격을 방지하기 위해 Amazon CodeCatalyst는 패키지 원본 제어 기능을 제공합니다. 패키지 원본 제어는 패키지를 리포지토리에 추가하는 방법을 제어하는 설정입니다. 제어는 새 패키지의 첫 번째 패키지 버전이 CodeCatalyst 리포지토리에 추가될 때 자동으로 구성됩니다. 제어를 사용하면 패키지 버전을 리포지토리에 직접 게시하거나 퍼블릭 소스에서 수집할 수 없게 하여 종속성 대체 공격을 방어할 수 있습니다. 패키지 원본 제어 및 이를 변경하는 방법에 대한 자세한 내용은 [패키지 원본 제어 편집](#) 섹션을 참조하세요.

npm 사용

이 항목에서는 Node.js 패키지 관리자인 npm을 CodeCatalyst와 함께 사용하는 방법을 설명합니다.

Note

CodeCatalyst는 node v4.9.1 이상 npm v5.0.0 및 이상을 지원합니다.

주제

- [npm 구성 및 사용](#)
- [npm 태그 처리](#)

npm 구성 및 사용

npm를 CodeCatalyst와 함께 사용하려면 패키지 리포지토리에 npm을 연결하고 인증을 위한 개인 액세스 토큰(PAT)을 제공해야 합니다. CodeCatalyst 콘솔에서 패키지 리포지토리에 npm을 연결하기 위한 지침을 볼 수 있습니다.

목차

- [CodeCatalyst를 사용하여 npm 구성](#)
- [CodeCatalyst 패키지 리포지토리에서 npm 패키지 설치](#)

- [CodeCatalyst를 통해 npmjs에서 npm 패키지 설치](#)
- [CodeCatalyst 패키지 리포지토리에 npm 패키지 게시](#)
- [npm 명령 지원](#)
 - [리포지토리와 상호 작용하는 지원되는 명령](#)
 - [지원되는 클라이언트 측 명령](#)
 - [지원되지 않는 명령](#)

CodeCatalyst를 사용하여 npm 구성

다음 지침은 CodeCatalyst 패키지 리포지토리를 인증하고 npm에 연결하는 방법을 설명합니다. npm에 대한 자세한 내용은 [공식 npm 설명서](#)를 참조하세요.

CodeCatalyst 패키지 리포지토리 **npm**에 연결하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트로 이동합니다.
3. 탐색 창에서 패키지를 선택합니다.
4. 목록에서 패키지 리포지토리를 선택합니다.
5. 리포지토리에 연결을 선택합니다.
6. 구성 세부 정보의 패키지 관리자 클라이언트에서 npm 클라이언트를 선택합니다.
7. 해당하는 구성 단계를 보기 위해 사용 중인 운영 체제를 선택합니다.
8. CodeCatalyst를 사용하여 npm을 인증하려면 개인 액세스 토큰(PAT)이 필요합니다. 토큰이 이미 있으면 해당 토큰을 사용할 수 있습니다. 토큰이 없다면, 다음 단계를 통해 생성할 수 있습니다.
 - a. (선택 사항): PAT 이름 및 만료 날짜를 업데이트합니다.
 - b. 토큰 생성을 선택합니다.
 - c. PAT를 복사하여 안전한 위치에 저장합니다.

Warning

대화 상자를 닫으면 PAT를 다시 보거나 복사할 수 없습니다. 자격 증명은 공격자가 자격 증명을 유용한 후 사용할 수 있는 기간을 최소화하도록 수명이 짧아야 합니다.

9. 프로젝트의 루트 디렉터리에서 다음 명령을 실행하여 패키지 리포지토리로 npm을 구성합니다. 명령은 다음을 수행합니다.

- 프로젝트에 아무 파일도 없는 경우 프로젝트 수준의 `.npmrc` 파일을 생성합니다.
- 패키지 리포지토리 엔드포인트 정보를 프로젝트 수준의 `.npmrc` 파일에 추가합니다.
- 사용자 수준의 `.npmrc` 파일에 자격 증명(PAT)을 추가합니다.

다음 값을 교체합니다.

Note

콘솔 지침에서 복사하는 경우 다음 명령의 값이 업데이트되므로 변경할 필요가 없습니다.

- `username`을 CodeCatalyst 사용자 이름으로 바꿉니다.
- `PAT`를 CodeCatalyst PAT로 바꿉니다.
- `space_name`을 CodeCatalyst 스페이스 이름으로 바꿉니다.
- `proj_name`을 CodeCatalyst 프로젝트 이름으로 바꿉니다.
- `my_repo`를 CodeCatalyst 패키지 리포지토리 이름으로 바꿉니다.

```
npm set registry=https://packages.region.codecatalyst.aws/npm/space-name/proj-name/repo-name/ --location project
npm set //packages.region.codecatalyst.aws/npm/space-name/proj-name/repo-name/:_authToken=username:PAT
```

npm 6 이하의 경우: GET 요청이 있을 경우에도 npm이 항상 CodeCatalyst에 인증 토큰을 전달하도록 하려면 항상 인증 구성 변수를 `npm config set`로 다음과 같이 설정합니다.

```
npm set //packages.region.codecatalyst.aws/npm/space-name/proj-name/repo-name/:always-auth=true --location project
```

CodeCatalyst 패키지 리포지토리에서 npm 패키지 설치

[CodeCatalyst를 사용하여 npm 구성](#) 섹션의 단계에 따라 npm을 리포지토리에 연결한 후 리포지토리에서 npm 명령을 실행할 수 있습니다.

`npm install` 명령을 사용하여 CodeCatalyst 패키지 리포지토리 또는 업스트림 리포지토리 중 하나에 있는 npm 패키지를 설치할 수 있습니다.

```
npm install lodash
```

CodeCatalyst를 통해 npmjs에서 npm 패키지 설치

npmjs.com, npm-public-registry-gateway에 연결된 게이트웨이 리포지토리에 대한 업스트림 연결로 리포지토리를 구성하여 CodeCatalyst 리포지토리를 통해 npmjs.com에서 npm 패키지를 설치할 수 있습니다. npmjs에서 설치된 패키지는 수집되어 게이트웨이 리포지토리와 가장 먼 다운스트림 패키지 리포지토리에 저장됩니다.

npmjs에서 패키지를 설치하려면

1. 아직 구성하지 않은 경우 [CodeCatalyst를 사용하여 npm 구성](#)의 단계에 따라 CodeCatalyst 패키지 리포지토리로 npm을 구성합니다.
2. 리포지토리가 게이트웨이 리포지토리 npm-public-registry-gateway를 업스트림 연결로 추가했는지 확인합니다. [업스트림 리포지토리 추가](#)의 지침에 따라 npm-public-registry-gateway 리포지토리를 선택하여, 어떤 업스트림 소스가 추가되었는지 확인하거나 npm-public-registry-gateway를 업스트림 소스로 추가할 수 있습니다.
3. `npm install` 명령을 사용하여 패키지를 설치합니다.

```
npm install package_name
```

업스트림 리포지토리에서 패키지를 요청하는 방법에 대한 자세한 내용은 [업스트림 리포지토리가 포함된 패키지 버전 요청](#) 섹션을 참조하세요.

CodeCatalyst 패키지 리포지토리에 npm 패키지 게시

[CodeCatalyst를 사용하여 npm 구성](#)를 완료한 후 npm 명령을 실행할 수 있습니다.

`npm publish` 명령을 사용하여 CodeCatalyst 패키지 리포지토리에 npm 패키지를 게시할 수 있습니다.

```
npm publish
```

npm 패키지를 만드는 방법에 관한 자세한 내용은 npm Docs에서 [Node.js 모듈 생성](#) 섹션을 참조하세요.

npm 명령 지원

다음 섹션에는 지원되지 않는 특정 명령 외에도 CodeCatalyst 리포지토리에서 지원하는 npm 명령이 요약되어 있습니다.

주제

- [리포지토리와 상호 작용하는 지원되는 명령](#)
- [지원되는 클라이언트 측 명령](#)
- [지원되지 않는 명령](#)

리포지토리와 상호 작용하는 지원되는 명령

이 섹션에는 npm 클라이언트가 구성될 때 사용된 레지스트리(예: `npm config set registry` 포함)에 하나 이상의 요청을 보내는 npm 명령이 나열되어 있습니다. CodeCatalyst 패키지 리포지토리에 대해 이러한 명령을 간접적으로 호출했을 때 제대로 작동하는 것으로 확인되었습니다.

명령	설명
bugs	패키지의 버그 추적기 URL 위치를 추측한 후 URL 열기를 시도합니다.
ci	프로젝트를 새로 다시 설치합니다.
deprecate	패키지 버전을 더 이상 사용하지 않습니다.
dist-tag	패키지 배포 태그를 수정합니다.
docs	패키지 설명서 URL의 위치를 추측한 다음 --browser config 파라미터를 사용하여 URL 열기를 시도합니다.
doctor	일련의 검사를 실행하여 npm 설치가 JavaScript 패키지를 관리할 수 있는지 검증합니다.
install	패키지를 설치합니다.
install-ci-test	프로젝트를 새로 다시 설치하고 테스트를 실행합니다. 별칭: <code>npm ci</code> . 이 명령은 <code>npm ci</code> 를 실행한 후 즉시 <code>npm test</code> 를 실행합니다.

명령	설명
install-test	패키지를 설치하고 테스트를 실행합니다. npm install을 실행한 후 즉시 npm test를 실행합니다.
outdated	구성된 레지스트리를 검사하여 설치된 패키지가 만료되었는지 확인합니다.
ping	구성되거나 지정된 npm 레지스트리를 ping하고 인증을 확인합니다.
publish	패키지 버전을 레지스트리에 게시합니다.
update	패키지의 리포지토리 URL 위치를 추측한 다음, --browser config 파라미터를 사용하여 URL 열기를 시도합니다.
view	패키지 메타데이터를 표시합니다. 또한 메타데이터 속성을 인쇄하는 데 사용할 수 있습니다.

지원되는 클라이언트 측 명령

이러한 명령은 리포지토리와의 직접 상호 작용할 필요가 없으므로 CodeCatalyst는 명령을 지원하기 위해 아무 것도 할 필요가 없습니다.

명령	설명
bin(legacy)	npm bin 디렉터리를 표시합니다.
build	패키지를 빌드합니다.
cache	패키지 캐시를 조작합니다.
completion	모든 npm 명령에서 탭 완성을 활성화합니다.
config	사용자 및 글로벌 npmrc 파일의 내용을 업데이트합니다.

명령	설명
<u>dedupe</u>	로컬 패키지 트리를 검색하고 종속성을 트리 위로 이동하여 구조를 단순화하려고 합니다. 여기서 종속성을 여러 종속 패키지에서 더 효과적으로 공유할 수 있습니다.
<u>edit</u>	설치된 패키지를 편집합니다. 현재 작업 디렉터리에서 종속성을 선택하고 기본 편집기에서 패키지 폴더를 엽니다.
<u>explore</u>	설치된 패키지를 찾아봅니다. 설치된 특정 패키지의 디렉터리에 서브셀을 생성합니다. 명령이 지정되면 해당 명령은 서브셀에서 실행된 후 즉시 종료됩니다.
<u>help</u>	npm에 관한 도움말을 가져옵니다.
<u>help-search</u>	npm 도움말 설명서를 검색합니다.
<u>init</u>	package.json 파일을 생성합니다.
<u>link</u>	패키지 디렉터리에 심볼릭 링크를 생성합니다.
<u>ls</u>	설치된 패키지를 나열합니다.
<u>pack</u>	패키지에서 tarball을 생성합니다.
<u>prefix</u>	접두사를 표시합니다. -g도 지정되지 않는 한, 이 디렉터리는 package.json 파일을 포함하는 가장 가까운 상위 디렉터리입니다.
<u>prune</u>	상위 패키지의 종속성 목록에 나열되지 않은 패키지를 제거합니다.
<u>rebuild</u>	일치하는 폴더에서 npm build 명령을 실행합니다.
<u>restart</u>	패키지의 중지, 재시작, 시작 스크립트와 관련 사전/사후 스크립트를 실행합니다.

명령	설명
root	유효 node_modules 폴더를 표준 출력으로 출력합니다.
run-script	임의의 패키지 스크립트를 실행합니다.
shrinkwrap	게시할 종속 버전을 잠급니다.
uninstall	패키지를 제거합니다.

지원되지 않는 명령

아래 npm 명령은 CodeCatalyst 패키지 리포지토리에서 지원하지 않습니다.

명령	설명	참고
access	게시된 패키지에서 액세스 수준을 설정합니다.	CodeCatalyst는 퍼블릭 npmjs 리포지토리와는 다른 권한 모델을 사용합니다.
adduser	레지스트리 사용자 계정을 추가합니다.	CodeCatalyst는 퍼블릭 npmjs 리포지토리와는 다른 사용자 모델을 사용합니다.
audit	보안 감사를 실행합니다.	CodeCatalyst는 현재 보안 취약성 데이터를 제공하지 않습니다.
hook	추가, 제거, 나열 및 업데이트를 포함하여 npm 후크를 관리합니다.	CodeCatalyst는 현재 어떠한 변경 알림 메커니즘도 지원하지 않습니다.
login	사용자를 인증합니다. npm adduser에 대한 별칭입니다.	CodeCatalyst는 퍼블릭 npmjs 리포지토리와는 다른 인증 모델을 사용합니다. 자세한 내용은 CodeCatalyst를 사용하여 npm 구성 단원을 참조하세요.

명령	설명	참고
logout	레지스트리에서 로그아웃합니다.	CodeCatalyst는 퍼블릭 npmjs 리포지토리와는 다른 인증 모델을 사용합니다. CodeCatalyst 리포지토리에서 로그아웃할 수 있는 방법은 없지만, 인증 토큰은 구성 가능한 만료 시간이 지나면 만료됩니다. 기본 토큰 지속 시간은 12시간입니다.
owner	패키지 소유자를 관리합니다.	CodeCatalyst는 퍼블릭 npmjs 리포지토리와는 다른 권한 모델을 사용합니다.
profile	레지스트리 프로파일의 설정을 변경합니다.	CodeCatalyst는 퍼블릭 npmjs 리포지토리와는 다른 사용자 모델을 사용합니다.
search	레지스트리에서 검색어와 일치하는 패키지를 검색합니다.	CodeCatalyst는 search 명령을 지원하지 않습니다.
star	좋아하는 패키지를 표시합니다.	CodeCatalyst는 현재 즐겨찾기 메커니즘을 지원하지 않습니다.
stars	즐거찾기로 표시된 패키지를 조회합니다.	CodeCatalyst는 현재 즐겨찾기 메커니즘을 지원하지 않습니다.
team	팀 및 팀 멤버십을 관리합니다.	CodeCatalyst는 퍼블릭 npmjs 리포지토리와는 다른 사용자 및 그룹 멤버 모델을 사용합니다.

명령	설명	참고
token	인증 토큰을 관리합니다.	CodeCatalyst는 인증 토큰을 가져오기 위해 다른 모델을 사용합니다. 자세한 내용은 CodeCatalyst를 사용하여 npm 구성 단원을 참조하세요.
unpublish	레지스트리에서 패키지를 제거합니다.	CodeCatalyst는 npm 클라이언트를 사용하여 리포지토리에서 패키지 버전을 제거하는 것을 지원하지 않습니다. 콘솔을 사용하여 에서 패키지를 삭제할 수 있습니다.
whoami	npm 사용자 이름을 표시합니다.	CodeCatalyst는 퍼블릭 npmjs 리포지토리와는 다른 사용자 모델을 사용합니다.

npm 태그 처리

npm 레지스트리는 패키지 버전의 문자열 별칭인 태그를 지원합니다. 태그를 사용하여 버전 번호를 사용하는 대신 별칭을 제공할 수 있습니다. 예를 들어, 여러 개발 스트림이 있는 프로젝트에서 각 스트림마다 다른 태그(예: stable, beta, dev, canary)를 사용할 수 있습니다. 자세한 내용은 npm Docs에서 [dist-tag](#)를 참조하세요.

기본적으로 npm은 latest 태그를 사용하여 패키지의 현재 버전을 식별합니다. `npm install pkg@version` 또는 `@tag` 지정자가 없는)는 최신 태그를 설치합니다. 일반적으로 프로젝트는 안정적인 릴리스 버전에만 최신 태그를 사용합니다. 그 밖의 태그는 불안정한 버전이나 프리릴리스 버전에 사용됩니다.

npm 클라이언트로 태그를 편집

CodeCatalyst 패키지 리포지토리에서는 세 개의 npm dist-tag 명령(add, rm 및 ls)이 [기본 npm 레지스트리](#)에서와 동일하게 작동합니다.

npm 태그와 업스트림 리포지토리

npm이 패키지에 대한 태그를 요청하고 해당 패키지의 버전이 업스트림 리포지토리에 있을 때 CodeCatalyst는 태그를 병합한 후 클라이언트에 반환합니다. 예를 들어 리포지토리 R에는 업스트림 리포지토리 U가 있습니다. 다음 표에는 두 리포지토리에 모두 있는 web-helper라는 패키지의 태그가 나와 있습니다.

리포지토리	패키지 이름	패키지 태그
R	web-helper	latest(버전 1.0.0의 별칭)
U	web-helper	alpha(버전 1.0.1의 별칭)

이 경우, npm 클라이언트가 리포지토리 R에서 web-helper 패키지의 태그를 가져오면 latest 태그와 alpha 태그를 모두 받습니다. 태그가 가리키는 버전은 변경되지 않습니다.

업스트림 및 로컬 리포지토리의 동일한 패키지에 동일한 태그가 있는 경우 CodeCatalyst는 마지막으로 업데이트된 태그를 사용합니다. 예를 들어, webhelper의 태그가 다음과 같이 수정되었다고 가정해 보겠습니다.

리포지토리	패키지 이름	패키지 태그	최종 업데이트 날짜
R	web-helper	latest(버전 1.0.0의 별칭)	2023년 1월 1일
U	web-helper	latest(버전 1.0.1의 별칭)	2023년 6월 1일

이 경우, npm 클라이언트가 리포지토리 R에서 패키지 web-helper의 태그를 가져오면, 최신 태그는 버전 1.0.1에 별칭을 지정합니다. 이는 가장 최근에 업데이트되었기 때문입니다. 이렇게 하면 npm update를 실행하여 로컬 리포지토리에 아직 없는 업스트림 리포지토리의 새 패키지 버전을 쉽게 사용할 수 있습니다.

Maven 사용

Maven 리포지토리 형식은 Java, Kotlin, Scala, Clojure를 비롯한 다양한 언어에서 사용됩니다. Maven, Gradle, Scala SBT, Apache Ivy, Leiningen을 비롯한 다양한 빌드 도구에서 지원됩니다.

CodeCatalyst와 다음 버전과의 호환성을 테스트하고 확인했습니다.

- 최신 Maven 버전: 3.6.3.
- 최신 Gradle 버전: 6.4.1. 버전 5.5.1도 테스트되었습니다.

주제

- [Gradle Groovy 구성 및 사용](#)
- [mvn 구성 및 사용](#)
- [curl을 사용하여 패키지 게시](#)
- [Maven 체크섬 및 스냅샷 사용](#)

Gradle Groovy 구성 및 사용

CodeCatalyst 에서 Gradle Groovy를 사용하려면 Gradle Groovy를 패키지 리포지토리에 연결하고 인증을 위한 개인 액세스 토큰(PAT)을 제공해야 합니다. CodeCatalyst 콘솔에서 Gradle Groovy를 패키지 리포지토리에 연결하기 위한 지침을 볼 수 있습니다.

목차

- [CodeCatalyst에서 종속성 가져오기](#)
- [CodeCatalyst에서 플러그인 가져오기](#)
- [CodeCatalyst를 통해 외부 패키지 리포지토리에서 패키지 가져오기](#)
- [CodeCatalyst에 패키지 게시](#)
- [IntelliJ IDEA에서 Gradle 빌드 실행](#)
 - [방법 1: PAT를 gradle.properties에 넣기](#)
 - [방법 2: PAT를 별도의 파일에 넣기](#)

CodeCatalyst에서 종속성 가져오기

다음 지침에서는 CodeCatalyst 패키지 리포지토리의 종속성을 가져오도록 Gradle Groovy를 구성하는 방법을 설명합니다.

Gradle Groovy를 사용하여 CodeCatalyst 패키지 리포지토리에서 종속성을 가져오려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트로 이동합니다.

3. 탐색 창에서 패키지를 선택합니다.
4. 패키지 리포지토리 목록에서 패키지 리포지토리를 선택합니다.
5. 리포지토리에 연결을 선택합니다.
6. 리포지토리에 연결 대화 상자의 패키지 관리자 클라이언트 목록에서 Gradle Groovy를 선택합니다.
7. CodeCatalyst로 Gradle Groovy를 인증하려면 개인 액세스 토큰(PAT)이 필요합니다. 이미 토큰이 있으면 그 토큰을 사용하면 됩니다. 없는 경우에는 여기에서 만들 수 있습니다.
 - a. 토큰 생성을 선택합니다.
 - b. PAT를 복사하려면 복사를 선택합니다.

 Warning

대화 상자를 닫으면 PAT를 다시 보거나 복사할 수 없습니다.

8. gradle 속성 파일을 액세스 자격 증명으로 업데이트합니다. *username*을 CodeCatalyst 사용자 이름으로 바꾸고 *PAT*를 자신의 CodeCatalyst 개인 액세스 토큰으로 바꿉니다. 다음 단계에서 동일한 값을 사용하는 한 *spaceUsername* 및 *spacePassword*에는 어떤 값이든 사용할 수 있습니다.

```
spaceUsername=username
spacePassword=PAT
```

9. Gradle 빌드의 CodeCatalyst에서 종속성을 가져오려면 maven 코드 조각을 복사하여 프로젝트 build.gradle 파일의 repositories 섹션에 추가합니다. 다음 값을 교체합니다. 다음 단계에서 동일한 값을 사용하는 한 *spaceName*에는 어떤 값이든 사용할 수 있습니다.

 Note

콘솔 지침에서 복사하는 경우 다음 값을 업데이트해야 하며 변경해서는 안 됩니다.

- *space_name*을 CodeCatalyst 스페이스 이름으로 바꿉니다.
- *proj_name*을 CodeCatalyst 프로젝트 이름으로 바꿉니다.
- *my_repo*를 CodeCatalyst 패키지 리포지토리 이름으로 바꿉니다.

```
maven {
```

```
name = 'spaceName'
url = uri('https://packages.region.codecatalyst.aws/
maven/space_name/proj_name/repo_name/')
credentials(PasswordCredentials)
}
```

10. (선택 사항) - CodeCatalyst 패키지 리포지토리를 프로젝트 종속성의 유일한 소스로 사용하려면 `build.gradle` 파일에서 리포지토리의 다른 섹션을 모두 제거합니다. 리포지토리가 두 개 이상인 경우, Gradle은 나열된 순서대로 각 리포지토리에서 종속성을 검색합니다.

CodeCatalyst에서 플러그인 가져오기

기본적으로 Gradle은 퍼블릭 [Gradle 플러그인 포털](#)에서 플러그인을 확인합니다. 다음 단계에서는 CodeCatalyst 패키지 리포지토리의 플러그인을 확인하도록 Gradle 프로젝트를 구성합니다.

Gradle을 사용하여 CodeCatalyst 패키지 리포지토리에서 플러그인을 가져오려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트로 이동합니다.
3. 탐색 창에서 패키지를 선택합니다.
4. 패키지 리포지토리 목록에서 패키지 리포지토리를 선택합니다.
5. 리포지토리에 연결을 선택합니다.
6. 리포지토리에 연결 대화 상자의 패키지 관리자 클라이언트 목록에서 Gradle을 선택합니다.
7. CodeCatalyst로 Gradle을 인증하려면 개인 액세스 토큰(PAT)이 필요합니다. 이미 토큰이 있으면 그 토큰을 사용하면 됩니다. 없는 경우에는 여기에서 만들 수 있습니다.
 - a. 토큰 생성을 선택합니다.
 - b. PAT를 복사하려면 복사를 선택합니다.

Warning

대화 상자를 닫으면 PAT를 다시 보거나 복사할 수 없습니다.

8. `gradle` 속성 파일을 액세스 자격 증명으로 업데이트합니다. `username`을 CodeCatalyst 사용자 이름으로 바꾸고 `PAT`를 자신의 CodeCatalyst 개인 액세스 토큰으로 바꿉니다. 다음 단계에서 동일한 값을 사용하는 한 `spaceUsername` 및 `spacePassword`에는 어떤 값이든 사용할 수 있습니다.

```
spaceUsername=username
```

spacePassword=PAT

9. settings.gradle 파일에 pluginManagement 블록을 추가합니다. pluginManagement 블록은 settings.gradle에서 명령문 중에 가장 앞에 나타나야 합니다. 다음 값을 교체합니다.

Note

콘솔 지침에서 복사하는 경우 다음 값을 업데이트해야 하며 변경해서는 안 됩니다.

- *spaceName*을 이전 단계에서 사용한 이름 값으로 바꿉니다.
- *space_name*을 CodeCatalyst 스페이스 이름으로 바꿉니다.
- *proj_name*을 CodeCatalyst 프로젝트 이름으로 바꿉니다.
- *my_repo*를 CodeCatalyst 패키지 리포지토리 이름으로 바꿉니다.

```
pluginManagement {
    repositories {
        maven {
            name = 'spaceName'
            url = uri('https://packages.region.codecatalyst.aws/
maven/space_name/proj_name/repo_name/')
            credentials(PasswordCredentials)
        }
    }
}
```

이렇게 하면 Gradle이 지정된 리포지토리의 플러그인을 확인할 수 있습니다. 일반적으로 필요한 Gradle 플러그인을 빌드에서 사용할 수 있도록 이 리포지토리에는 Gradle 플러그인 포털 (gradle-plugins-store)에 대해 구성된 업스트림 연결이 있어야 합니다. 자세한 내용은 [Gradle 설명서](#)를 참조하세요.

CodeCatalyst를 통해 외부 패키지 리포지토리에서 패키지 가져오기

게이트웨이 리포지토리를 나타내는 게이트웨이에 대한 업스트림 연결로 구성하여 CodeCatalyst 리포지토리를 통해 퍼블릭 리포지토리에서 Maven 패키지를 설치할 수 있습니다. 게이트웨이 리포지토리에서 설치된 패키지는 수집되어 CodeCatalyst 리포지토리에 저장됩니다.

CodeCatalyst는 다음과 같은 퍼블릭 Maven 패키지 리포지토리를 지원합니다.

- maven-central-gateway
- google-android-gateway
- gradle-plugins-gateway
- commonsware-gateway

퍼블릭 Maven 패키지 리포지토리에서 패키지를 설치하려면

1. 아직 구성하지 않은 경우, [CodeCatalyst에서 종속성 가져오기](#) 또는 [CodeCatalyst에서 플러그인 가져오기](#)의 단계를 따라 CodeCatalyst 패키지 리포지토리로 Gradle을 구성합니다.
2. 리포지토리에 설치하려는 게이트웨이 리포지토리가 업스트림 연결로 추가되었는지 확인합니다. [업스트림 리포지토리 추가](#)의 지침을 따라 업스트림으로 추가하려는 퍼블릭 패키지 리포지토리를 선택하여 이 작업을 수행할 수 있습니다.

업스트림 리포지토리에서 패키지를 요청하는 방법에 대한 자세한 내용은 [업스트림 리포지토리가 포함된 패키지 버전 요청](#) 섹션을 참조하세요.

CodeCatalyst에 패키지 게시

이 섹션에서는 Gradle Groovy로 빌드한 Java 라이브러리를 CodeCatalyst 리포지토리에 게시하는 방법을 설명합니다.

Gradle Groovy를 사용하여 패키지를 CodeCatalyst 패키지 리포지토리에 게시하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트의 개요 페이지에서 패키지를 선택합니다.
3. 패키지 리포지토리 목록에서 패키지 리포지토리를 선택합니다.
4. 리포지토리에 연결을 선택합니다.
5. 리포지토리에 연결 대화 상자의 패키지 관리자 클라이언트 목록에서 Gradle Groovy를 선택합니다.
6. CodeCatalyst로 Gradle을 인증하려면 개인 액세스 토큰(PAT)이 필요합니다. 이미 토큰이 있으면 그 토큰을 사용하면 됩니다. 없는 경우에는 여기에서 만들 수 있습니다.
 - a. 토큰 생성을 선택합니다.
 - b. PAT를 복사하려면 복사를 선택합니다.

⚠ Warning

대화 상자를 닫으면 PAT를 다시 보거나 복사할 수 없습니다.

7. gradle 속성 파일을 액세스 자격 증명으로 업데이트합니다. *username*을 CodeCatalyst 사용자 이름으로 바꾸고 *PAT*를 자신의 CodeCatalyst 개인 액세스 토큰으로 바꿉니다. 다음 단계에서 동일한 값을 사용하는 한 *spaceUsername* 및 *spacePassword*에는 어떤 값이든 사용할 수 있습니다.

```
spaceUsername=username
spacePassword=PAT
```

8. 프로젝트의 build.gradle 파일 중 plugins 섹션에 maven-publish 플러그인을 추가합니다.

```
plugins {
    id 'java-library'
    id 'maven-publish'
}
```

9. 다음으로 프로젝트 build.gradle 파일에 publishing 섹션을 추가합니다. 다음 값을 교체합니다.

i Note

콘솔 지침에서 복사하는 경우 다음 값을 업데이트해야 하며 변경해서는 안 됩니다.

- *space_name*을 CodeCatalyst 스페이스 이름으로 바꿉니다.
- *proj_name*을 CodeCatalyst 프로젝트 이름으로 바꿉니다.
- *my_repo*를 CodeCatalyst 패키지 리포지토리 이름으로 바꿉니다.

```
publishing {
    publications {
        mavenJava(MavenPublication) {
            groupId = 'group-id'
            artifactId = 'artifact-id'
            version = 'version'
            from components.java
        }
    }
}
```



```

    }
    repositories {
        maven {
            name = 'spaceName'
            url = uri('https://packages.region.codecatalyst.aws/
maven/space_name/proj_name/repo_name/')
            credentials(PasswordCredentials)
        }
    }
}

```

maven-publish 플러그인은 publishing 섹션에 지정된 groupId, artifactId, version을 기반으로 POM 파일을 생성합니다.

10. build.gradle에 대한 이러한 변경이 완료되면 다음 명령을 실행하여 프로젝트를 빌드하여 리포지토리에 업로드합니다.

```
./gradlew publish
```

11. CodeCatalyst 콘솔에서 패키지 리포지토리로 이동하여 패키지가 성공적으로 게시되었는지 확인합니다. 패키지 리포지토리의 패키지 목록에 패키지가 표시되어야 합니다.

자세한 내용은 Gradle 웹 사이트에서 이 주제를 참조하세요.

- [Java 라이브러리 빌드](#)
- [프로젝트를 모듈로 게시](#)

IntelliJ IDEA에서 Gradle 빌드 실행

CodeCatalyst에서 종속성을 가져오는 IntelliJ IDEA에서 Gradle 빌드를 실행할 수 있습니다. CodeCatalyst로 Gradle을 인증하려면 개인 액세스 토큰(PAT)을 사용해야 합니다. gradle.properties 또는 선택한 별도의 파일에 CodeCatalyst PAT를 저장할 수 있습니다.

방법 1: PAT를 **gradle.properties**에 넣기

gradle.properties 파일을 사용하지 않고 PAT로 내용을 덮어쓸 수 있는 경우 이 방법을 사용합니다. gradle.properties를 사용하는 경우 파일의 내용을 덮어쓰는 대신 이 방법을 수정하여 PAT를 추가할 수 있습니다.

Note

예시는 GRADLE_USER_HOME에 있는 `gradle.properties` 파일을 보여줍니다.

먼저 PAT가 없는 경우 PAT를 생성합니다.

개인용 액세스 토큰(PAT)을 만들려면

1. 상단 메뉴 바에서 프로파일 배지를 선택한 다음 내 설정을 선택합니다.

Tip

프로젝트 또는 스페이스의 멤버 페이지로 이동하고 멤버 목록에서 이름을 선택하여 자신의 사용자 프로파일을 찾을 수도 있습니다.

2. PAT 이름에 PAT를 설명하는 이름을 입력합니다.
3. 만료 날짜에 기본 날짜를 유지하거나 달력 아이콘을 선택하여 사용자 지정 날짜를 선택합니다. 만료 날짜는 기본적으로 현재 날짜로부터 1년 후입니다.
4. 생성을 선택합니다.

소스 리포지토리의 복제 리포지토리를 선택할 때도 이 토큰을 생성할 수 있습니다.

5. PAT 보안 암호를 안전한 위치에 저장합니다.

Important

PAT 보안 암호는 한 번만 표시됩니다. 창을 닫은 후에는 검색할 수 없습니다.

다음 코드 조각을 사용하여 `build.gradle` 파일을 업데이트합니다.

```
repositories {
    maven {
        name = 'spaceName'
        url = uri('https://packages.region.codecatalyst.aws/
maven/space_name/proj_name/repo_name/')
        credentials(PasswordCredentials)
    }
}
```

방법 2: PAT를 별도의 파일에 넣기

gradle.properties 파일을 수정하지 않으려면 이 방법을 사용하세요.

먼저 PAT가 없는 경우 PAT를 생성합니다.

개인용 액세스 토큰(PAT)을 만들려면

1. 상단 메뉴 바에서 프로파일 배지를 선택한 다음 내 설정을 선택합니다.

Tip

프로젝트 또는 스페이스의 멤버 페이지로 이동하고 멤버 목록에서 이름을 선택하여 자신의 사용자 프로파일을 찾을 수도 있습니다.

2. PAT 이름에 PAT를 설명하는 이름을 입력합니다.
3. 만료 날짜에 기본 날짜를 유지하거나 달력 아이콘을 선택하여 사용자 지정 날짜를 선택합니다. 만료 날짜는 기본적으로 현재 날짜로부터 1년 후입니다.
4. 생성을 선택합니다.

소스 리포지토리의 복제 리포지토리를 선택할 때도 이 토큰을 생성할 수 있습니다.

5. PAT 보안 암호를 안전한 위치에 저장합니다.

Important

PAT 보안 암호는 한 번만 표시됩니다. 창을 닫은 후에는 검색할 수 없습니다.

PAT를 별도의 파일에 배치하려면

1. 다음 코드 조각을 사용하여 build.gradle 파일을 업데이트합니다. *space_name*, *proj_name*, *repo_name*을 자신의 CodeCatalyst 사용자 이름, 스페이스 이름, 프로젝트 이름, 패키지 리포지토리 이름으로 바꿉니다.

```
def props = new Properties()
file("fileName").withInputStream { props.load(it) }

repositories {
    maven {
        name = 'spaceName'
```

```

        url = uri('https://packages.region.codecatalyst.aws/
maven/space_name/proj_name/repo_name/')
        credentials(PasswordCredentials)
    }
}
}

```

2. build.gradle 파일에 지정된 파일에 PAT를 작성합니다.

```
echo "codecatalystArtifactsToken=PAT" > fileName
```

mvn 구성 및 사용

mvn 명령을 사용하여 Maven 빌드를 실행합니다. 패키지 리포지토리를 사용하고 인증을 위한 개인 액세스 토큰(PAT)을 제공하도록 mvn을 구성해야 합니다.

목차

- [CodeCatalyst에서 종속성 가져오기](#)
- [CodeCatalyst를 통해 외부 패키지 리포지토리에서 패키지 가져오기](#)
- [CodeCatalyst에 패키지 게시](#)
- [타사 패키지 게시](#)

CodeCatalyst에서 종속성 가져오기

CodeCatalyst 리포지토리에서 종속성을 가져오도록 mvn을 구성하려면 Maven 구성 파일, settings.xml 및 프로젝트 객체 모델(POM) 파일을 편집해야 합니다. POM 파일에는 종속성, 빌드 디렉터리, 소스 디렉터리, 테스트 소스 디렉터리, 플러그인 및 목표와 같이 Maven이 프로젝트를 빌드하기 위한 프로젝트 및 구성 정보에 대한 정보가 포함되어 있습니다.

mvn을 사용하여 CodeCatalyst 패키지 리포지토리에서 종속성을 가져오는 방법

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트의 개요 페이지에서 패키지를 선택합니다.
3. 패키지 리포지토리 목록에서 패키지 리포지토리를 선택합니다.
4. 리포지토리에 연결을 선택합니다.
5. 리포지토리에 연결 대화 상자의 패키지 관리자 클라이언트 목록에서 mvn을 선택합니다.

6. mvn을 CodeCatalyst 로 인증하려면 개인 액세스 토큰(PAT)이 필요합니다. 이미 토큰이 있으면 그 토큰을 사용하면 됩니다. 없는 경우에는 여기에서 만들 수 있습니다.
 - a. 토큰 생성을 선택합니다.
 - b. PAT를 복사하려면 복사를 선택합니다.

 Warning

대화 상자를 닫으면 PAT를 다시 보거나 복사할 수 없습니다.

7. 리포지토리가 포함된 프로파일을 settings.xml 파일에 추가합니다. 다음 값을 교체합니다.

 Note

콘솔 지침에서 복사하는 경우 다음 값을 업데이트해야 하며 변경해서는 안 됩니다.

- *space_name*을 CodeCatalyst 스페이스 이름으로 바꿉니다.
- *proj_name*을 CodeCatalyst 프로젝트 이름으로 바꿉니다.
- *my_repo*를 CodeCatalyst 패키지 리포지토리 이름으로 바꿉니다.

```
<profiles>
  <profile>
    <id>repo_name</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <repositories>
      <repository>
        <id>repo_name</id>
        <url>https://packages.region.codecatalyst.aws/
maven/space_name/proj_name/repo_name/</url>
      </repository>
    </repositories>
  </profile>
</profiles>
```

8. settings.xml 파일의 서버 목록에 서버를 추가합니다. 다음 값을 교체합니다.

Note

콘솔 지침에서 복사하는 경우 다음 값을 업데이트해야 하며 변경해서는 안 됩니다.

- *my_repo*를 CodeCatalyst 패키지 리포지토리 이름으로 바꿉니다.
- *username*을 CodeCatalyst 사용자 이름으로 바꿉니다.
- *PAT*를 CodeCatalyst PAT로 바꿉니다.

```
<servers>
  <server>
    <id>repo_name</id>
    <username>username</username>
    <password>PAT</password>
  </server>
</servers>
```

9. (선택 사항) 모든 연결을 캡처하여 게이트웨이 리포지토리 대신 리포지토리로 라우팅하는 미러를 settings.xml 파일에 설정합니다.

Note

콘솔 지침에서 복사하는 경우 다음 값을 업데이트해야 하며 변경해서는 안 됩니다.

- *space_name*을 CodeCatalyst 스페이스 이름으로 바꿉니다.
- *proj_name*을 CodeCatalyst 프로젝트 이름으로 바꿉니다.
- *my_repo*를 CodeCatalyst 패키지 리포지토리 이름으로 바꿉니다.

```
<mirrors>
  <mirror>
    <id>repo_name</id>
    <name>repo_name</name>
    <url>https://packages.region.codecatalyst.aws/
maven/space_name/proj_name/repo_name/</url>
    <mirrorOf>*</mirrorOf>
  </mirror>
</mirrors>
```

```
</mirrors>
```

⚠ Important

`<id>` 요소에서 어떤 값이든 사용할 수 있지만 그 값은 `<server>` 및 `<repository>` 요소에서 모두 동일해야 합니다. 이렇게 하면 지정된 자격 증명을 CodeCatalyst에 대한 요청에 포함할 수 있습니다.

이러한 구성을 변경한 후 프로젝트를 빌드할 수 있습니다.

```
mvn compile
```

CodeCatalyst를 통해 외부 패키지 리포지토리에서 패키지 가져오기

게이트웨이 리포지토리를 나타내는 게이트웨이에 대한 업스트림 연결로 구성하여 CodeCatalyst 리포지토리를 통해 퍼블릭 리포지토리에서 Maven 패키지를 설치할 수 있습니다. 게이트웨이 리포지토리에서 설치된 패키지는 수집되어 CodeCatalyst 리포지토리에 저장됩니다.

현재 CodeCatalyst는 다음과 같은 퍼블릭 Maven 패키지 리포지토리를 지원합니다.

- maven-central-gateway
- google-android-gateway
- gradle-plugins-gateway
- commonsware-gateway

퍼블릭 Maven 패키지 리포지토리에서 패키지를 설치하려면

1. 아직 mvn을 구성하지 않았다면 [CodeCatalyst에서 종속성 가져오기](#)의 단계에 따라 CodeCatalyst 패키지 리포지토리로 구성합니다.
2. 리포지토리가 설치하려는 게이트웨이 리포지토리를 업스트림 연결로 추가했는지 확인합니다. 추가되는 업스트림 소스를 확인하거나 게이트웨이 리포지토리를 업스트림 소스로 추가하려면 [업스트림 리포지토리 추가](#)의 지침을 따르세요.

업스트림 리포지토리에서 패키지를 요청하는 방법에 대한 자세한 내용은 [업스트림 리포지토리가 포함된 패키지 버전 요청](#) 섹션을 참조하세요.

CodeCatalyst에 패키지 게시

mvn과 함께 Maven 패키지를 CodeCatalyst 리포지토리에 게시하려면 ~/.m2/settings.xml 및 프로젝트 POM도 편집해야 합니다.

mvn을 사용하여 CodeCatalyst 패키지 리포지토리에 패키지를 게시하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트의 개요 페이지에서 패키지를 선택합니다.
3. 패키지 리포지토리 목록에서 패키지 리포지토리를 선택합니다.
4. 리포지토리에 연결을 선택합니다.
5. 리포지토리에 연결 대화 상자의 패키지 관리자 클라이언트 목록에서 mvn을 선택합니다.
6. mvn을 CodeCatalyst로 인증하려면 개인 액세스 토큰(PAT)이 필요합니다. 이미 토큰이 있으면 그 토큰을 사용하면 됩니다. 없는 경우에는 여기에서 만들 수 있습니다.
 - a. 토큰 생성을 선택합니다.
 - b. PAT를 복사하려면 복사를 선택합니다.



Warning

대화 상자를 닫으면 PAT를 다시 보거나 복사할 수 없습니다.

7. PAT를 사용하여 로컬 시스템에서 환경 변수를 구성합니다. setting.xml 파일에서 이 환경 변수를 사용합니다.

```
export CODECATALYST_ARTIFACTS_TOKEN=your_PAT
```

8. Maven이 HTTP 요청에서 토큰을 전달하도록 <servers> 섹션을 CodeCatalyst_ARTIFACTS_TOKEN 환경 변수에 대한 참조가 있는 settings.xml에 추가합니다.

```
<settings>
...
  <servers>
    <server>
      <id>repo-name</id>
      <username>username</username>
      <password>${env.CodeCatalyst_ARTIFACTS_TOKEN}</password>
    </server>
  </servers>
</settings>
```



```

    </servers>
    ...
</settings>

```

9. 프로젝트의 pom.xml에 <distributionManagement> 섹션을 추가합니다.

```

<project>
...
  <distributionManagement>
    <repository>
      <id>repo_name</id>
      <name>repo_name</name>
      <url>https://packages.region.codecatalyst.aws/
maven/space_name/proj_name/repo_name/</url>
    </repository>
  </distributionManagement>
...
</project>

```

이러한 구성을 변경한 후 프로젝트를 빌드하여 지정된 리포지토리에 게시할 수 있습니다.

```
mvn deploy
```

CodeCatalyst 콘솔에서 패키지 리포지토리로 이동하여 패키지가 성공적으로 게시되었는지 확인할 수 있습니다.

타사 패키지 게시

mvn deploy:deploy-file을 사용하여 타사 Maven 패키지를 CodeCatalyst 리포지토리에 게시할 수 있습니다. 이는 JAR 파일만 가지고 있고 패키지 소스 코드나 POM 파일에는 액세스할 수 없는 사용자가 패키지를 게시하려고 할 때 도움이 될 수 있습니다.

이 mvn deploy:deploy-file 명령을 실행하면 명령줄에 전달된 정보를 기반으로 POM 파일이 생성됩니다.

먼저 PAT가 없는 경우 PAT를 생성합니다.

개인용 액세스 토큰(PAT)을 만들려면

1. 상단 메뉴 바에서 프로파일 배지를 선택한 다음 내 설정을 선택합니다.

i Tip

프로젝트 또는 스페이스의 멤버 페이지로 이동하고 멤버 목록에서 이름을 선택하여 자신의 사용자 프로파일을 찾을 수도 있습니다.

2. PAT 이름에 PAT를 설명하는 이름을 입력합니다.
3. 만료 날짜에 기본 날짜를 유지하거나 달력 아이콘을 선택하여 사용자 지정 날짜를 선택합니다. 만료 날짜는 기본적으로 현재 날짜로부터 1년 후입니다.
4. 생성을 선택합니다.

소스 리포지토리의 복제 리포지토리를 선택할 때도 이 토큰을 생성할 수 있습니다.

5. PAT 보안 암호를 안전한 위치에 저장합니다.

⚠ Important

PAT 보안 암호는 한 번만 표시됩니다. 창을 닫은 후에는 검색할 수 없습니다.

타사 Maven 패키지를 게시하려면

1. 다음 콘텐츠가 포함된 ~/.m2/settings.xml 파일을 생성합니다.

```
<settings>
  <servers>
    <server>
      <id>repo_name</id>
      <username>username</username>
      <password>PAT</password>
    </server>
  </servers>
</settings>
```

2. mvn deploy:deploy-file 명령을 실행합니다.

```
mvn deploy:deploy-file -DgroupId=commons-cli \
-DartifactId=commons-cli \
-Dversion=1.4 \
```

```
-Dfile=./commons-cli-1.4.jar \
-Dpackaging=jar \
-DrepositoryId=repo-name \
-Durl=https://packages.region.codecatalyst.aws/maven/space-name/proj-name/repo-name/
```

Note

앞의 예에서는 commons-cli 1.4를 게시합니다. groupId, artifactID, 버전 및 파일 인수를 수정하여 다른 JAR을 게시합니다.

이 지침은 Apache Maven 설명서의 [타사 JAR을 원격 리포지토리에 배포하기 위한 가이드](#)의 예시를 기반으로 합니다.

자세한 내용은 Apache Maven Project 웹 사이트에서 다음 주제를 참조하세요.

- [여러 리포지토리 설정](#)
- [설정 참조](#)
- [배포 관리](#)
- [프로파일](#)

curl을 사용하여 패키지 게시

이 섹션에서는 HTTP 클라이언트 curl을 사용하여 CodeCatalyst 패키지 리포지토리에 Maven 패키지를 게시하는 방법을 보여줍니다. 환경에 Maven 클라이언트가 없거나 이 클라이언트를 설치하려는 경우, curl을 사용하여 패키지를 게시하는 것이 도움이 될 수 있습니다.

curl을 사용하여 Maven 패키지를 게시하려면

1. CodeCatalyst로 curl을 인증하려면 환경 변수에 개인 액세스 토큰(PAT)을 저장해야 합니다. 이미 토큰이 있으면 그 토큰을 사용하면 됩니다. 그렇지 않은 경우 토큰을 생성하고 환경 변수를 구성할 수 있습니다.
 - a. [개인 액세스 토큰을 사용하여 사용자 리포지토리 액세스 권한 부여](#)의 단계에 따라 PAT를 생성합니다. PAT를 복사하여 환경 변수에 저장합니다.
 - b. 로컬 시스템의 명령줄에서 PAT로 환경 변수를 구성합니다.

```
export CodeCatalyst_ARTIFACTS_TOKEN=your_PAT
```

2. 다음 curl 명령을 사용하여 JAR을 CodeCatalyst 리포지토리에 게시합니다. *username*, *space_name*, *proj_name*, *repo_name*을 CodeCatalyst 사용자 이름, 스페이스 이름, 프로젝트 이름, 패키지 리포지토리 이름으로 바꿉니다.

```
curl --request PUT https://packages.region.codecatalyst.aws/maven/space-name/proj-name/repo-name/com/mycompany/app/my-app/1.0/my-app-1.0.jar \
  --user "username:CodeCatalyst_ARTIFACTS_TOKEN" --header "Content-Type: application/octet-stream" \
  --data-binary @target/path/to/my-app-1.0.jar
```

3. 다음 curl 명령을 사용하여 POM을 CodeCatalyst 리포지토리에 게시합니다. *username*, *space_name*, *proj_name*, *repo_name*을 CodeCatalyst 사용자 이름, 스페이스 이름, 프로젝트 이름, 패키지 리포지토리 이름으로 바꿉니다.

```
curl --request PUT https://packages.region.codecatalyst.aws/maven/space-name/proj-name/repo-name/com/mycompany/app/my-app/1.0/my-app-1.0.pom \
  --user "username:CodeCatalyst_ARTIFACTS_TOKEN" --header "Content-Type: application/octet-stream" \
  --data-binary @target/my-app-1.0.pom
```

4. 이때 Maven 패키지는 상태가 Unfinished인 CodeCatalyst 리포지토리에 저장됩니다. 패키지를 사용하려면 패키지가 Published 상태에 있어야 합니다. 패키지에 maven-metadata.xml 파일을 업로드하거나 CodeCatalyst 콘솔에서 상태를 변경하여 패키지를 Unfinished에서 Published로 이동할 수 있습니다.

- a. 옵션 1: 다음 curl 명령을 사용하여 패키지에 maven-metadata.xml 파일을 추가합니다. *username*, *space_name*, *proj_name*, *repo_name*을 CodeCatalyst 사용자 이름, 스페이스 이름, 프로젝트 이름, 패키지 리포지토리 이름으로 바꿉니다.

```
curl --request PUT https://packages.region.codecatalyst.aws/maven/space-name/proj-name/repo-name/com/mycompany/app/my-app/maven-metadata.xml \
  --user "username:CodeCatalyst_ARTIFACTS_TOKEN" --header "Content-Type: application/octet-stream" \
  --data-binary @target/maven-metadata.xml
```

다음은 maven-metadata.xml 파일의 내용을 보여주는 예입니다.

```
<metadata modelVersion="1.1.0">
  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <versioning>
    <latest>1.0</latest>
    <release>1.0</release>
    <versions>
      <version>1.0</version>
    </versions>
    <lastUpdated>20200731090423</lastUpdated>
  </versioning>
</metadata>
```

- b. 옵션 2: CodeCatalyst 콘솔에서 패키지 상태를 Published로 업데이트합니다. 패키지 버전의 상태를 업데이트하는 방법에 대한 자세한 내용은 [패키지 버전의 상태 업데이트](#) 섹션을 참조하세요.

패키지의 JAR 파일만 있는 경우, mvn을 사용하여 CodeCatalyst 리포지토리에 사용 가능한 패키지 버전을 게시할 수 있습니다. 이는 패키지의 소스 코드나 POM에 액세스할 수 없는 경우에 도움이 될 수 있습니다. 세부 정보는 [타사 패키지 게시](#) 섹션을 참조하세요.

Maven 체크섬 및 스냅샷 사용

다음 섹션에서는 CodeCatalyst에서 Maven 체크섬 및 Maven 스냅샷을 사용하는 방법을 설명합니다.

Maven 체크섬 사용

Maven 패키지가 CodeCatalyst 패키지 리포지토리에 게시되면 패키지의 각 자산 또는 파일과 관련된 체크섬을 사용하여 업로드를 검증합니다. 자산의 예로는 jar, pom, war 파일 등이 있습니다. 각 자산의 경우, md5 또는 sha1 같은 추가 확장자가 있는 자산 이름을 사용하는 여러 개의 체크섬 파일이 Maven 패키지에 포함되어 있습니다. 예를 들어, my-maven-package.jar라는 이름이 지정된 파일의 체크섬 파일은 my-maven-package.jar.md5 및 my-maven-package.jar.sha1일 수 있습니다.

모든 Maven 패키지에는 maven-metadata.xml 파일도 포함되어 있습니다. 게시가 성공하려면 이 파일을 업로드해야 합니다. 패키지 파일을 업로드하는 동안 체크섬 불일치가 감지되면 게시가 중지됩니다. 이렇게 하면 maven-metadata.xml이 업로드되지 않을 수 있습니다. 이 경우 Maven 패키지의 상태가 Unfinished로 설정됩니다. 이 상태의 패키지 일부인 자산은 다운로드할 수 없습니다.

Maven 패키지를 게시할 때 체크섬 불일치가 발생하는 경우 다음 사항에 유의하세요.

- `maven-metadata.xml`이 업로드되기 전에 체크섬 불일치가 발생하면 패키지의 상태가 `Unfinished`로 설정되지 않습니다. 패키지가 표시되지 않으며 해당 자산을 사용할 수 없습니다. 이 경우 다음 중 하나를 시도한 다음 자산을 다시 다운로드해 보세요.
- Maven 패키지를 다시 게시하는 명령을 실행합니다. 다운로드를 하는 동안 네트워크 문제로 인해 체크섬 파일이 손상된 경우, 이러한 명령 실행이 효과적일 수 있습니다. 다시 시도하여 네트워크 문제가 해결되면 체크섬이 일치하여 다운로드가 완료됩니다.
- Maven 패키지를 다시 게시할 수 없는 경우 해당 패키지를 삭제한 다음 다시 게시합니다.
- `maven-metadata.xml`이 업로드된 후 체크섬 불일치가 발생하면 패키지 상태가 `Published`로 설정됩니다. 체크섬 불일치가 있는 자산을 포함하여 패키지의 모든 자산을 사용할 수 있습니다. 자산을 다운로드하면 CodeCatalyst에서 생성한 체크섬이 함께 다운로드됩니다. 다운로드한 파일이 체크섬 불일치와 연결된 경우 다운로드한 체크섬 파일이 패키지가 게시될 때 업로드된 체크섬과 일치하지 않을 수 있습니다.

Maven 스냅샷 사용

Maven 스냅샷은 최신 프로덕션 브랜치 코드를 참조하는 Maven 패키지의 특수 버전입니다. 이 스냅샷은 최종 릴리스 버전보다 앞서는 개발 버전입니다. 패키지 버전에 추가된 접미사 `SNAPSHOT`으로 Maven 패키지의 스냅샷 버전을 식별할 수 있습니다. 예를 들어, 버전 1.1의 스냅샷은 `1.1-SNAPSHOT`입니다. 자세한 내용은 Apache Maven 프로젝트 웹사이트에서 [SNAPSHOT 버전이란 무엇입니까?](#)를 참조하세요.

CodeCatalyst는 Maven 스냅샷 게시 및 사용을 지원합니다. Maven 스냅샷을 CodeCatalyst 리포지토리에 게시하거나, 직접 연결된 경우 업스트림 리포지토리에 게시할 수 있습니다. 그러나 업스트림 리포지토리 중 하나와 패키지 리포지토리에서 모두 스냅샷 버전은 지원되지 않습니다. 예를 들어, 버전이 `1.2-SNAPSHOT`인 Maven 패키지를 패키지 리포지토리에 업로드하는 경우, CodeCatalyst는 동일한 스냅샷 버전이 있는 Maven 패키지를 업스트림 리포지토리 중 하나에 업로드하는 것을 지원하지 않습니다. 이 시나리오는 예측할 수 없는 결과를 반환할 수 있습니다.

Maven 스냅샷이 게시되면 이전 버전이 빌드라는 새 버전에 보존됩니다. Maven 스냅샷이 게시될 때마다 새 빌드 버전이 생성됩니다. 스냅샷의 모든 이전 버전은 빌드 버전에서 유지 관리됩니다. Maven 스냅샷이 게시되면 상태가 `Published`로 설정되고 이전 버전이 포함된 빌드의 상태가 `Unlisted`로 설정됩니다.

스냅샷을 요청하면 상태 `Published`의 버전이 반환됩니다. 이 버전은 항상 Maven 스냅샷의 최신 버전입니다. 스냅샷의 특정 빌드를 요청할 수도 있습니다.

Maven 스냅샷의 모든 빌드 버전을 삭제하려면 CodeCatalyst 콘솔을 사용합니다.

NuGet 사용

이 항목에서는 CodeCatalyst를 사용하여 NuGet 패키지를 사용하고 게시하는 방법을 설명합니다.

Note

CodeCatalyst는 [NuGet 버전 4.8](#) 이상을 지원합니다.

주제

- [Visual Studio에서 CodeCatalyst 사용](#)
- [nuget 또는 dotnet CLI 구성 및 사용](#)
- [NuGet 패키지 이름, 버전 및 자산 이름 표준화](#)
- [NuGet 호환성](#)

Visual Studio에서 CodeCatalyst 사용

Visual Studio에서 직접 CodeCatalyst의 패키지를 사용할 수 있습니다.

NuGet을 dotnet 또는 nuget과 같은 CLI 도구와 함께 구성하고 사용하려면 [nuget 또는 dotnet CLI 구성 및 사용](#) 섹션을 참조하세요.

목차

- [CodeCatalyst로 Visual Studio 구성](#)
 - [Windows](#)
 - [macOS](#)

CodeCatalyst로 Visual Studio 구성

Windows

CodeCatalyst로 Visual Studio를 구성하려면

1. CodeCatalyst로 인증하려면 개인 액세스 토큰(PAT)이 필요합니다. 이미 토큰이 있으면 그 토큰을 사용하면 됩니다. 토큰이 없다면 [개인 액세스 토큰을 사용하여 사용자 리포지토리 액세스 권한 부여](#)의 지침에 따라 토큰을 생성합니다.
2. nuget 또는 dotnet를 사용하여 패키지 리포지토리 및 자격 증명을 구성합니다.

dotnet

Linux 및 macOS 사용자: Windows 이외의 플랫폼에서는 암호화를 지원하지 않으므로 다음 명령에 `--store-password-in-clear-text` 플래그를 추가해야 합니다. 단, 이렇게 하면 비밀번호가 구성 파일에 일반 텍스트로 저장됩니다.

```
dotnet nuget add source https://packages.region.codecatalyst.aws/nuget/space-name/proj-name/repo-name/v3/index.json --name repo_name --password PAT --username user_name
```

nuget

```
nuget sources add -name repo_name -Source https://packages.region.codecatalyst.aws/nuget/space-name/proj-name/repo-name/v3/index.json -password PAT --username user_name
```

출력 예시:

```
Package source with Name: repo_name added successfully.
```

3. 새 패키지 소스를 사용하도록 Visual Studio를 구성합니다. Visual Studio에서 도구를 선택한 다음 옵션을 선택합니다.
4. 옵션 메뉴에서 NuGet 패키지 관리자 섹션을 확장하고 패키지 소스를 선택합니다.
5. 사용 가능한 패키지 소스 목록에서 *repo_name* 소스가 활성화되어 있는지 확인합니다. NuGet Gallery에 대한 업스트림 연결로 패키지 리포지토리를 구성한 경우 nuget.org 소스를 비활성화합니다.

macOS

CodeCatalyst로 Visual Studio를 구성하려면

1. CodeCatalyst로 인증하려면 개인 액세스 토큰(PAT)이 필요합니다. 이미 토큰이 있으면 그 토큰을 사용하면 됩니다. 토큰이 없다면 [개인 액세스 토큰을 사용하여 사용자 리포지토리 액세스 권한 부여](#)의 지침에 따라 토큰을 생성합니다.
2. 메뉴 모음에서 기본 설정을 선택합니다.
3. NuGet 섹션에서 소스를 선택합니다.

4. 추가를 선택하여 리포지토리 정보를 추가합니다.
 - a. 이름에 CodeCatalyst 패키지 리포지토리 이름을 입력합니다.
 - b. 위치에 CodeCatalyst 패키지 리포지토리 엔드포인트를 입력합니다. 다음 코드 조각은 엔드포인트의 예시를 보여줍니다. *space-name*, *proj-name*, *repo-name*을 CodeCatalyst 스페이스 이름, 프로젝트 이름, 리포지토리 이름으로 바꿉니다.

`https://packages.region.codecatalyst.aws/nuget/space-name/proj-name/repo-name/`
 - c. 사용자 이름에 유효한 값을 입력합니다.
 - d. 암호에 PAT를 입력합니다.
5. 소스 추가를 선택합니다.
6. NuGet Gallery에 대한 업스트림 연결로 패키지 리포지토리를 구성한 경우 nuget.org 소스를 비활성화합니다.

구성 후 Visual Studio는 CodeCatalyst 리포지토리, 모든 업스트림 리포지토리 또는 업스트림 소스로 구성된 경우 [NuGet.org](https://www.nuget.org)의 패키지를 사용할 수 있습니다. Visual Studio에서 NuGet 패키지를 검색하고 설치하는 방법에 대한 자세한 내용은 NuGet 설명서의 [NuGet 패키지 관리자를 사용한 Visual Studio의 패키지 설치 및 관리](#)를 참조하세요.

nuget 또는 dotnet CLI 구성 및 사용

NuGet 및 dotnet과 같은 CLI 도구를 사용하여 CodeCatalyst에서 패키지를 게시하고 사용할 수 있습니다. 이 문서에서는 CLI 도구를 구성하고 이를 사용하여 패키지를 게시하거나 사용하는 방법에 관한 정보를 제공합니다.

목차

- [CodeCatalyst를 사용하여 NuGet 구성](#)
- [CodeCatalyst 리포지토리에서 NuGet 패키지 사용](#)
- [CodeCatalyst를 통해 NuGet.org에서 NuGet 패키지 사용](#)
- [CodeCatalyst에 NuGet 패키지 게시](#)

CodeCatalyst를 사용하여 NuGet 구성

CodeCatalyst로 NuGet을 구성하려면 NuGet 구성 파일에 리포지토리 엔드포인트와 개인 액세스 토큰을 추가하여 CodeCatalyst 패키지 리포지토리에 nuget 또는 dotnet을 연결할 수 있도록 합니다.

CodeCatalyst 패키지 리포지토리로 NuGet을 구성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트의 개요 페이지에서 패키지를 선택합니다.
3. 패키지 리포지토리 목록에서 패키지 리포지토리를 선택합니다.
4. 리포지토리에 연결을 선택합니다.
5. 리포지토리에 연결 대화 상자의 패키지 관리자 클라이언트 목록에서 NuGet 또는 dotnet을 선택합니다.
6. CodeCatalyst로 NuGet을 인증하려면 개인 액세스 토큰(PAT)이 필요합니다. 이미 토큰이 있으면 그 토큰을 사용하면 됩니다. 없는 경우에는 여기에서 만들 수 있습니다.
 - a. 토큰 생성을 선택합니다.
 - b. PAT를 복사하려면 복사를 선택합니다.

Warning

대화 상자를 닫으면 PAT를 다시 보거나 복사할 수 없습니다.

7. 리포지토리의 NuGet 엔드포인트 및 CodeCatalyst PAT를 사용하도록 `nuget` 또는 `dotnet`을 구성합니다. 다음 값을 교체합니다.

Note

콘솔 지침에서 복사하는 경우 다음 값을 업데이트해야 하며 변경해서는 안 됩니다.

- `username`을 CodeCatalyst 사용자 이름으로 바꿉니다.
- `PAT`를 CodeCatalyst PAT로 바꿉니다.
- `space_name`을 CodeCatalyst 스페이스 이름으로 바꿉니다.
- `proj_name`을 CodeCatalyst 프로젝트 이름으로 바꿉니다.
- `my_repo`를 CodeCatalyst 패키지 리포지토리 이름으로 바꿉니다.

- a. `nuget`에서 `nuget sources add` 명령을 사용합니다.

```
nuget sources add -name "repo_name" -Source "https://
packages.region.codecatalyst.aws/nuget/space_name/proj_name/repo_name/v3/
index.json" -username "username" -password "PAT"
```

- b. dotnet에서 dotnet nuget add source 명령을 사용합니다.

Linux 및 macOS 사용자: Windows 이외의 플랫폼에서는 암호화를 지원하지 않으므로 다음 명령에 --store-password-in-clear-text 플래그를 추가해야 합니다. 단, 이렇게 하면 비밀번호가 구성 파일에 일반 텍스트로 저장됩니다.

```
dotnet nuget add source "https://packages.region.codecatalyst.aws/
nuget/space_name/proj_name/repo_name/v3/index.json" -n "proj_name/repo_name" -u
"username" -p "PAT" --store-password-in-clear-text
```

CodeCatalyst로 NuGet을 구성한 후에는 CodeCatalyst 리포지토리 또는 업스트림 리포지토리 중 하나에 저장된 [NuGet 패키지를 사용](#)할 수 있고 CodeCatalyst 리포지토리로 [NuGet 패키지를 게시](#)할 수 있습니다.

CodeCatalyst 리포지토리에서 NuGet 패키지 사용

[CodeCatalyst로 NuGet을 구성](#)한 후에는 CodeCatalyst 리포지토리 또는 업스트림 리포지토리 중 하나에 저장된 NuGet 패키지를 사용할 수 있습니다.

CodeCatalyst 리포지토리 또는 nuget 또는 dotnet이 있는 업스트림 리포지토리 중 하나에서 패키지 버전을 사용하려면 다음 명령을 실행합니다. *packageName*을 사용하려는 패키지 이름으로 바꾸고 *packageSourceName*을 NuGet 구성 파일의 CodeCatalyst 패키지 리포지토리 소스 이름으로 바꿉니다. 이 이름은 리포지토리 이름이어야 합니다.

dotnet을 사용하여 패키지를 설치하려면

```
dotnet add packageName --source packageSourceName
```

nuget을 사용하여 패키지를 설치하려면

```
nuget install packageName --source packageSourceName
```

자세한 내용은 Microsoft 설명서의 [nuget.exe CLI를 사용하여 패키지 관리](#) 또는 [dotnet CLI를 사용하여 패키지 설치 및 관리](#)를 참조하세요.

CodeCatalyst를 통해 NuGet.org에서 NuGet 패키지 사용

NuGet.org에 대한 업스트림 연결을 통해 리포지토리를 구성하여 CodeCatalyst 리포지토리를 통해 [NuGet.org](#)의 NuGet 패키지를 사용할 수 있습니다. Nuget.org에서 사용하는 패키지는 CodeCatalyst 리포지토리로 수집하여 저장됩니다.

Nuget.org의 패키지를 사용하려면

1. 아직 구성하지 않았다면 [CodeCatalyst를 사용하여 NuGet 구성](#)의 단계에 따라 CodeCatalyst 패키지 리포지토리로 NuGet 패키지 관리자를 구성합니다.
2. 리포지토리가 NuGet.org를 업스트림 연결로 추가했는지 확인합니다. [업스트림 리포지토리 추가](#)의 지침에 따르면 NuGet 스토어 리포지토리를 선택하여, 어떤 업스트림 소스가 추가되었는지 확인하거나 Nuget.org를 업스트림 소스로 추가할 수 있습니다.

CodeCatalyst에 NuGet 패키지 게시

[CodeCatalyst로 NuGet을 구성](#)한 후에는 nuget 또는 dotnet을 사용하여 CodeCatalyst 리포지토리에 패키지 버전을 게시할 수 있습니다.

패키지 버전을 CodeCatalyst 리포지토리로 불러오려면 NuGet 구성 파일에 .nupkg 파일의 전체 경로와 CodeCatalyst 리포지토리의 소스 이름을 포함하여 다음 명령을 실행합니다.

dotnet를 사용하여 패키지를 게시하려면

```
dotnet nuget push path/to/nupkg/SamplePackage.1.0.0.nupkg --source packageSourceName
```

nuget를 사용하여 패키지를 게시하려면

```
nuget push path/to/nupkg/SamplePackage.1.0.0.nupkg --source packageSourceName
```

NuGet 패키지 이름, 버전 및 자산 이름 표준화

CodeCatalyst는 패키지 및 자산 이름과 패키지 버전을 저장하기 전에 표준화합니다. 즉, CodeCatalyst의 이름 또는 버전은 패키지 또는 자산이 게시될 때 제공된 것과 다를 수 있습니다.

패키지 이름 표준화: CodeCatalyst는 모든 문자를 소문자로 변환하여 NuGet 패키지 이름을 표준화합니다.

패키지 버전 표준화: CodeCatalyst는 NuGet과 동일한 패턴을 사용하여 NuGet 패키지 버전을 표준화합니다. 다음 정보는 NuGet 설명서의 [표준화된 버전 번호](#)에서 가져왔습니다.

- 버전 번호에서 앞에 오는 0은 제거되었습니다.
 - 1.00은 1.0으로 간주합니다.
 - 1.01.1은 1.1.1으로 간주합니다.
 - 1.00.0.1은 1.0.0.1으로 간주합니다.
- 버전 번호의 네 번째 부분에 있는 0은 생략합니다.
 - 1.0.0.0은 1.0.0으로 간주합니다.
 - 1.0.01.0은 1.0.1으로 간주합니다.
- SemVer 2.0.0 빌드 메타데이터가 제거되었습니다.
 - 1.0.7+r3456은 1.0.7으로 간주합니다.

패키지 에셋 이름 표준화: CodeCatalyst는 표준화된 패키지 이름과 패키지 버전을 기반으로 NuGet 패키지 에셋 이름을 생성합니다.

NuGet 호환성

이 안내서에는 CodeCatalyst의 다양한 NuGet 도구 및 버전과의 호환성에 관한 정보가 포함되어 있습니다.

주제

- [일반 NuGet 호환성](#)
- [NuGet 명령줄 지원](#)

일반 NuGet 호환성

CodeCatalyst는 NuGet 4.8 이상을 지원합니다.

CodeCatalyst는 NuGet HTTP 프로토콜의 V3만 지원합니다. 즉, 프로토콜의 V2를 사용하는 일부 CLI 명령은 지원하지 않습니다. 자세한 내용은 다음 [nuget 명령 지원](#) 섹션을 참조하세요.

CodeCatalyst는 PowerShellGet 2.x를 지원하지 않습니다.

NuGet 명령줄 지원

CodeCatalyst는 NuGet(nuget) 및 .NET Core(dotnet) CLI 도구를 지원합니다.

nuget 명령 지원

CodeCatalyst는 NuGet HTTP 프로토콜 V3만 지원하므로 CodeCatalyst 리소스에 대해 다음 명령을 사용해도 작동하지 않습니다.

- `list`: 이 `nuget list` 명령은 지정된 소스의 패키지 목록을 표시합니다. CodeCatalyst 패키지 리포지토리의 패키지 목록을 가져오려면 CodeCatalyst 콘솔에서 리포지토리로 이동합니다.

Python 사용

이 항목에서는 CodeCatalyst와 함께 Python 패키지 관리자인 `pip`, Python 패키지 게시 유틸리티인 `twine`을 사용하는 방법에 대해 설명합니다.

주제

- [pip 구성 및 Python 패키지 설치](#)
- [Twine 구성 및 Python 패키지 게시](#)
- [Python 패키지 이름 정규화](#)
- [Python 호환성](#)

pip 구성 및 Python 패키지 설치

`pip`을 CodeCatalyst와 함께 사용하려면 패키지 리포지토리에 `pip`를 연결하고 인증을 위한 개인 액세스 토큰을 제공해야 합니다. CodeCatalyst 콘솔에서 패키지 리포지토리에 `pip`을 연결하기 위한 지침을 볼 수 있습니다. `pip`을 CodeCatalyst에 인증하고 연결한 후 `pip` 명령을 실행할 수 있습니다.

목차

- [pip를 사용하여 CodeCatalyst에서 Python 패키지 설치](#)
- [CodeCatalyst를 통해 PyPI에서 패키지 사용](#)
- [pip 명령 지원](#)
 - [리포지토리와 상호 작용하는 지원되는 명령](#)
 - [지원되는 클라이언트 측 명령](#)

pip를 사용하여 CodeCatalyst에서 Python 패키지 설치

다음 지침은 CodeCatalyst 패키지 리포지토리 또는 업스트림 리포지토리 중 하나에서 Python 패키지를 설치하도록 pip를 구성하는 방법을 설명합니다.

CodeCatalyst 패키지 리포지토리에서 Python 패키지를 설치하기 위해 **pip**를 구성하고 사용하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트의 개요 페이지에서 패키지를 선택합니다.
3. 패키지 리포지토리 목록에서 패키지 리포지토리를 선택합니다.
4. 리포지토리에 연결을 선택합니다.
5. 리포지토리에 연결 대화 상자의 패키지 관리자 클라이언트 목록에서 pip를 선택합니다.
6. CodeCatalyst로 pip를 인증하려면 개인 액세스 토큰(PAT)이 필요합니다. 이미 토큰이 있으면 그 토큰을 사용하면 됩니다. 없는 경우에는 여기에서 만들 수 있습니다.
 - a. 토큰 생성을 선택합니다.
 - b. PAT를 복사하려면 복사를 선택합니다.

Warning

대화 상자를 닫으면 PAT를 다시 보거나 복사할 수 없습니다.

7. `pip config` 명령 사용하여 CodeCatalyst 레지스트리 URL 및 자격 증명을 설정합니다. 다음 값을 교체합니다.

Note

콘솔 지침에서 복사하는 경우 다음 값을 업데이트해야 하며 변경해서는 안 됩니다.

- **username**을 CodeCatalyst 사용자 이름으로 바꿉니다.
- **PAT**를 CodeCatalyst PAT로 바꿉니다.
- **space_name**을 CodeCatalyst 스페이스 이름으로 바꿉니다.
- **proj_name**을 CodeCatalyst 프로젝트 이름으로 바꿉니다.
- **my_repo**를 CodeCatalyst 패키지 리포지토리 이름으로 바꿉니다.

```
pip config set global.index-url https://username:PAT@https://
packages.region.codecatalyst.aws/pypi/space_name/proj_name/repo_name/simple/
```

8. 패키지가 리포지토리 또는 업스트림 리포지토리 중 하나에 있다고 할 경우, `pip install`을 사용하여 패키지를 설치할 수 있습니다. 예를 들어 다음 명령을 사용하여 `requests` 패키지를 설치할 수 있습니다.

```
pip install requests
```

`-i` 옵션을 사용하여 CodeCatalyst 리포지토리 대신 <https://pypi.org>에서 패키지를 설치하는 것으로 일시적으로 되돌릴 수 있습니다.

```
pip install -i https://pypi.org/simple requests
```

CodeCatalyst를 통해 PyPI에서 패키지 사용

PyPI에 대한 업스트림 연결이 있는 리포지토리를 구성하여 CodeCatalyst 리포지토리를 통해 [Python 패키지 인덱스\(PyPI\)](#)에서 Python 패키지를 사용할 수 있습니다. PyPI에서 사용하는 패키지는 CodeCatalyst 리포지토리로 수집하여 저장됩니다.

PyPI에서 패키지를 사용하려면

1. 아직 구성하지 않은 경우 [pip를 사용하여 CodeCatalyst에서 Python 패키지 설치](#)의 단계에 따라 CodeCatalyst 패키지 리포지토리로 `pip`를 구성합니다.
2. 리포지토리가 PyPI를 업스트림 소스로 추가했는지 확인합니다. [업스트림 리포지토리 추가](#)의 지침에 따르면 PyPI 스토어 리포지토리를 선택하여, 어떤 업스트림 소스가 추가되었는지 확인하거나 PyPI를 업스트림 소스로 추가할 수 있습니다.

업스트림 리포지토리에서 패키지를 요청하는 방법에 대한 자세한 내용은 [업스트림 리포지토리가 포함된 패키지 버전 요청](#) 섹션을 참조하세요.

pip 명령 지원

다음 섹션에는 지원되지 않는 특정 명령 외에도 CodeCatalyst 리포지토리에서 지원하는 `pip` 명령이 요약되어 있습니다.

주제

- [리포지토리와 상호 작용하는 지원되는 명령](#)
- [지원되는 클라이언트 측 명령](#)

리포지토리와 상호 작용하는 지원되는 명령

이 섹션에는 pip 클라이언트가 구성된 레지스트리에 하나 이상의 요청을 보내는 pip 명령이 나열되어 있습니다. CodeCatalyst 패키지 리포지토리에 대해 이러한 명령을 간접적으로 호출했을 때 제대로 작동하는 것으로 확인되었습니다.

명령	설명
install	패키지를 설치합니다.
download	패키지를 다운로드합니다.

CodeCatalyst는 pip search를 구현하지 않습니다. pip를 CodeCatalyst 패키지 리포지토리로 구성한 경우 pip search를 실행하면 [PyPI](#)에서 패키지를 검색하고 표시합니다.

지원되는 클라이언트 측 명령

이러한 명령은 리포지토리와 직접 상호 작용할 필요가 없으므로 CodeCatalyst는 명령을 지원하기 위해 아무 것도 할 필요가 없습니다.

명령	설명
uninstall	패키지를 제거합니다.
freeze	설치된 패키지를 요구 사항 형식으로 출력합니다.
list	설치된 패키지를 나열합니다.
show	설치된 패키지에 대한 정보를 표시합니다.
check	설치된 패키지에 호환되는 종속성이 있는지 확인합니다.

명령	설명
config	로컬 및 글로벌 구성을 관리합니다.
wheel	요구 사항에 맞게 휠을 빌드합니다.
hash	패키지 아카이브의 해시를 계산합니다.
completion	명령 완성을 돕습니다.
debug	디버깅에 유용한 정보를 표시합니다.
help	명령에 대한 도움말을 표시합니다.

Twine 구성 및 Python 패키지 게시

twine을 CodeCatalyst와 함께 사용하려면 패키지 리포지토리에 twine를 연결하고 인증을 위한 개인 액세스 토큰을 제공해야 합니다. CodeCatalyst 콘솔에서 패키지 리포지토리에 twine을 연결하기 위한 지침을 볼 수 있습니다. twine을 CodeCatalyst에 인증하고 연결한 후 twine 명령을 실행할 수 있습니다.

Twine을 사용하여 CodeCatalyst에 패키지 게시

다음 지침은 CodeCatalyst 패키지 리포지토리를 인증하고 twine에 연결하는 방법을 설명합니다.

twine을 구성하고 사용하여 CodeCatalyst 패키지 리포지토리에 패키지를 게시하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트의 개요 페이지에서 패키지를 선택합니다.
3. 패키지 리포지토리 목록에서 패키지 리포지토리를 선택합니다.
4. 리포지토리에 연결을 선택합니다.
5. 리포지토리에 연결 대화 상자의 패키지 관리자 클라이언트 목록에서 Twine을 선택합니다.
6. CodeCatalyst 를 사용하여 twine을 인증하려면 개인 액세스 토큰(PAT)이 필요합니다. 이미 토큰이 있으면 그 토큰을 사용하면 됩니다. 없는 경우에는 여기에서 만들 수 있습니다.
 - a. 토큰 생성을 선택합니다.
 - b. PAT를 복사하려면 복사를 선택합니다.

⚠ Warning

대화 상자를 닫으면 PAT를 다시 보거나 복사할 수 없습니다.

7. `.pypirc` 파일을 사용하거나 환경 변수를 사용하여 twine를 구성할 수 있습니다.

a. `.pypirc` 파일로 구성하려면

원하는 편집기에서 `~/.pypirc` 파일을 엽니다.

이전 단계에서 생성 및 복사한 PAT, 사용자 이름 및 리포지토리를 포함하여 CodeCatalyst용 인덱스 서버를 추가합니다. 다음 값을 교체합니다.

i Note

콘솔 지침에서 복사하는 경우 다음 값을 업데이트해야 하며 변경해서는 안 됩니다.

- `username`을 CodeCatalyst 사용자 이름으로 바꿉니다.
- `PAT`를 CodeCatalyst PAT로 바꿉니다.
- `space_name`을 CodeCatalyst 스페이스 이름으로 바꿉니다.
- `proj_name`을 CodeCatalyst 프로젝트 이름으로 바꿉니다.
- `my_repo`를 CodeCatalyst 패키지 리포지토리 이름으로 바꿉니다.

```
[distutils]
index-servers = proj-name/repo-name

[proj-name/repo-name]
repository = https://packages.region.codecatalyst.aws/
pypi/space_name/proj_name/repo_name/
password = PAT
username = username
```

b. 환경 변수를 구성하려면

다음 환경 변수를 설정합니다. `TWINE_REPOSITORY_URL` 값에서 `space_name`, `proj_name`, `repo_name`을 CodeCatalyst 스페이스, 프로젝트, 패키지 리포지토리 이름으로 업데이트합니다.

```
export TWINE_USERNAME=username
```

```
export TWINE_PASSWORD=PAT
```

```
export TWINE_REPOSITORY_URL="https://packages.region.codecatalyst.aws/  
pypi/space_name/proj_name/repo_name/"
```

8. `twine upload` 명령을 사용하여 Python 배포를 게시합니다.

Python 패키지 이름 정규화

CodeCatalyst는 패키지 이름을 저장하기 전에 정규화합니다. 즉, CodeCatalyst의 패키지 이름은 패키지가 게시될 때 제공된 이름과 다를 수 있습니다.

Python 패키지의 경우, 정규화를 수행할 때 패키지 이름은 소문자와 `.`, `-` 및 `_` 문자의 모든 인스턴스가 단일 `-` 문자로 대체됩니다. 따라서 `pigeon_cli` 및 `pigeon.cli` 패키지 이름은 `pigeon-cli`와 같이 정규화되어 저장됩니다. 정규화되지 않은 이름은 `pip` 및 `twine`에서 사용할 수 있습니다. Python 패키지 이름 정규화에 관한 자세한 내용은 Python 설명서에서 [PEP 503](#)을 참조하세요.

Python 호환성

CodeCatalyst는 `/simple/` API 지원하지 않지만 Legacy API 작업을 지원합니다. CodeCatalyst는 PyPI의 XML-RPC 또는 JSON API 작업을 지원하지 않습니다.

자세한 내용은 Python Packaging Authority의 GitHub 리포지토리에서 다음 섹션을 참조하세요.

- [Legacy API](#)
- [XML-RPC API](#)
- [JSON API](#)

패키지 할당량

다음 표에서는 Amazon CodeCatalyst의 패키지 할당량 및 제한에 대해 설명합니다. Amazon CodeCatalyst의 할당량에 대한 자세한 내용은 [CodeCatalyst 할당량](#) 섹션을 참조하세요.

Resource	기본 할당량
패키지 리포지토리	스페이스당 최대 1,000개
직접 업스트림 리포지토리	패키지 리포지토리당 최대 10개
검색된 업스트림 패키지 리포지토리	요청된 패키지 버전당 최대 25개의 업스트림 리포지토리 검색
패키지 자산 파일 크기	패키지 자산당 최대 5GB
패키지 자산	패키지 버전당 최대 150개

워크플로를 사용하여 빌드, 테스트 및 배포

[CodeCatalyst 개발 환경](#)에 애플리케이션 코드를 쓰고 이를 [CodeCatalyst 소스 리포지토리](#)에 푸시하고 나면 배포할 준비가 되었습니다. 이를 자동으로 수행하는 방법은 워크플로를 통하는 것입니다.

워크플로는 지속적 통합 및 지속적 전송(CI/CD) 시스템의 일부로 코드를 빌드, 테스트 및 배포하는 방법을 설명하는 자동화된 절차입니다. 워크플로는 워크플로 실행 중에 수행할 일련의 단계 또는 작업을 정의합니다. 또한 워크플로는 워크플로를 시작하게 하는 이벤트 또는 트리거를 정의합니다. 워크플로를 설정하려면 CodeCatalyst 콘솔의 [시각적 또는 YAML 편집기](#)를 사용하여 워크플로 정의 파일을 생성합니다.

Tip

프로젝트에서 워크플로를 사용하는 방법을 간략하게 알아보려면 [블루프린트가 있는 프로젝트를 생성](#)합니다. 각 블루프린트는 검토, 실행 및 실험할 수 있는 기능 워크플로를 배포합니다.

워크플로 정의 파일 정보

워크플로 정의 파일은 워크플로를 설명하는 YAML 파일입니다. 기본적으로 파일은 [소스 리포지토리](#)의 루트에 있는 `~/.codecatalyst/workflows/` 폴더에 저장됩니다. 파일은 확장자가 `.yml` 또는 `.yaml`일 수 있으며 확장자는 소문자여야 합니다.

다음은 간단한 워크플로 정의 파일의 예시입니다. 다음 테이블에서 이 예시의 각 줄에 대해 설명합니다.

```
Name: MyWorkflow
SchemaVersion: 1.0
RunMode: QUEUED
Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  Build:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
```

Steps:

```
- Run: docker build -t MyApp:latest .
```

행	설명
Name: MyWorkflow	워크플로의 이름을 지정합니다. Name 속성에 대한 자세한 내용은 최상위 속성 섹션을 참조하세요.
SchemaVersion: 1.0	워크플로 스키마 버전을 지정합니다. SchemaVersion 속성에 대한 자세한 내용은 최상위 속성 섹션을 참조하세요.
RunMode: QUEUED	CodeCatalyst가 여러 실행을 처리하는 방법을 나타냅니다. 실행 모드에 대한 자세한 내용은 실행의 대기열 동작 구성 섹션을 참조하세요.
Triggers:	워크플로 실행을 시작할 로직을 지정합니다. 트리거에 대한 자세한 내용은 트리거를 사용하여 워크플로 실행 자동 시작 주제를 참조하세요.
- Type: PUSH Branches: - main	기본 소스 리포지토리의 main 브랜치에 코드를 푸시할 때마다 워크플로가 시작되어야 함을 나타냅니다. 워크플로 소스에 대한 자세한 내용은 워크플로에 소스 리포지토리 연결 섹션을 참조하세요.
Actions:	워크플로 실행 중에 수행할 작업을 정의합니다. 이 예시에서는 Actions 섹션은 Build라는 단일 작업을 정의합니다. 작업에 대한 자세한 내용은 워크플로 작업 구성 섹션을 참조하세요.
Build:	Build 작업의 속성을 정의합니다. 빌드 작업에 대한 자세한 내용은 워크플로로 빌드하기 섹션을 참조하세요.
Identifier: aws/builddev1	빌드 작업의 하드 코딩된 고유 식별자를 지정합니다.

행	설명
Inputs: Sources: - WorkflowSource	빌드 작업이 처리를 완료하는 데 필요한 파일을 찾기 위해 WorkflowSource 소스 리포지토리에서 확인되어야 함을 나타냅니다. 자세한 내용은 워크플로에 소스 리포지토리 연결 섹션을 참조하세요.
Configuration:	빌드 작업과 관련된 구성 속성을 포함합니다.
Steps: - Run: docker build -t MyApp:latest .	빌드 작업에 MyApp이라는 Docker 이미지를 빌드하고 latest로 태그를 지정하도록 지시합니다.

워크플로 정의 파일에서 사용 가능한 모든 속성의 전체 목록은 [워크플로 YAML 정의](#) 섹션을 참조하세요.

CodeCatalyst 콘솔의 시각적 및 YAML 편집기 사용

워크플로 정의 파일을 생성하고 편집하려면 원하는 편집기를 사용할 수 있지만 CodeCatalyst 콘솔의 시각적 편집기 또는 YAML 편집기를 사용하는 것이 좋습니다. 이러한 편집기는 YAML 속성 이름, 값, 중첩, 간격, 대문자 표기 등이 올바른지 확인하는 데 도움이 되는 파일 검증을 제공합니다.

다음 이미지는 시각적 편집기의 워크플로를 보여줍니다. 시각적 편집기는 워크플로 정의 파일을 생성하고 구성할 수 있는 완전한 사용자 인터페이스를 제공합니다. 시각적 편집기에는 워크플로의 기본 구성 요소를 보여주는 워크플로 다이어그램(1)과 구성 영역(2)이 포함되어 있습니다.

Workflow diagram 1

Configuration area 2 **BuildBackend**

Inputs **Configuration** **Outputs**

Action name
BuildBackend

Compute type
Choose a type of managed compute to balance action start-up speed with flexibility during run time.
EC2
Optimized for flexibility during action runs.

Compute fleet - optional
Choose the machine or fleet that will run this action.
Choose compute

Environment/account/role - optional
You can use environment, connection, and role to access AWS resources.

Environment
You can [create a new environment](#) or view [environments](#) in the current project.
ExampleEnvironment

AWS account connection
Add an [AWS account](#) and [associate](#) it with an environment.
111122223333
Choose an environment first.

또는 다음 이미지에 표시된 YAML 편집기를 사용할 수 있습니다. YAML 편집기를 사용하여 큰 코드 블록(예: 자습서)에 붙여넣거나 시각적 편집기를 통해 제공되지 않는 고급 속성을 추가합니다.

Workflow diagram 1

Configuration area 2 **BuildBackend**

Inputs **Configuration** **Outputs**

Action name
BuildBackend

Compute type
Choose a type of managed compute to balance action start-up speed with flexibility during run time.
EC2
Optimized for flexibility during action runs.

Compute fleet - optional
Choose the machine or fleet that will run this action.
Choose compute

Environment/account/role - optional
You can use environment, connection, and role to access AWS resources.

Environment
You can [create a new environment](#) or view [environments](#) in the current project.
ExampleEnvironment

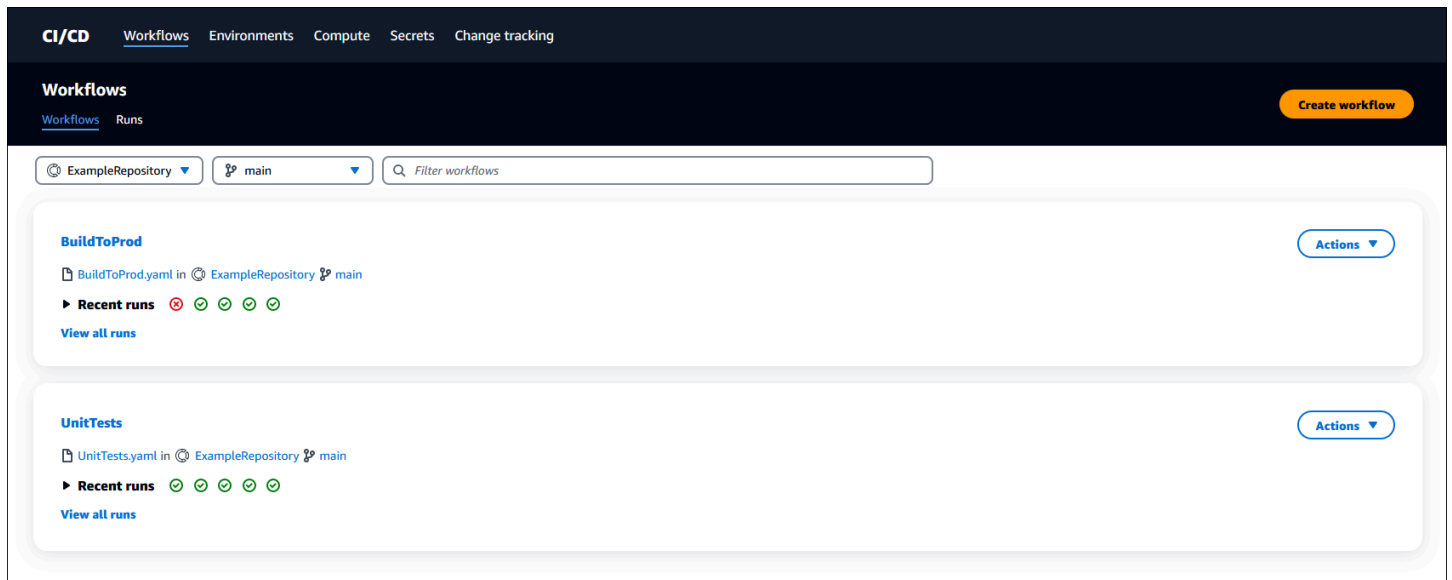
AWS account connection
Add an [AWS account](#) and [associate](#) it with an environment.
111122223333
Choose an environment first.

시각적 편집기에서 YAML 편집기로 전환하여 구성이 기본 YAML 코드에 미치는 영향을 확인할 수 있습니다.

워크플로 검색

워크플로 요약 페이지에서 워크플로를 볼 수 있으며, 동일한 프로젝트에 설정한 다른 워크플로도 볼 수 있습니다.

다음 이미지는 워크플로 요약 페이지를 보여줍니다. BuildToProd와 UnitTests라는 두 가지 워크플로로 채워집니다. 둘 다 몇 번 실행되었음을 알 수 있습니다. 최근 실행을 선택하여 실행 기록을 빠르게 보거나 워크플로 이름을 선택하여 워크플로의 YAML 코드 및 기타 세부 정보를 볼 수 있습니다.



워크플로 실행 세부 정보 보기

워크플로 요약 페이지에서 실행을 선택하여 워크플로 실행의 세부 정보를 볼 수 있습니다.

다음 이미지는 소스 커밋 시 자동으로 시작된 Run-cc11d라는 워크플로 실행의 세부 정보를 보여줍니다. 워크플로 다이어그램은 작업이 실패했음을 나타냅니다(1). 로그(2)로 이동하여 자세한 로그 메시지를 보고 문제를 해결할 수 있습니다. 워크플로 실행에 대한 자세한 내용은 [워크플로 실행](#) 섹션을 참조하세요.

The screenshot displays the Amazon CodeCatalyst console for a workflow named 'Run-cc11d'. The workflow is in a 'Failed' state, indicated by a red circle with a white 'X' icon. The 'BuildBackend' action is highlighted with a red circle and the number '1', indicating it failed. The 'Test' action is shown as successful with a green checkmark. The 'DeployCloudFormationStack' action is also shown as successful. The 'Detailed log messages' panel on the right shows the output of the 'BuildBackend' action, which includes a message about the SAM CLI and an error message: 'Error: Template file not found at /codecatalyst/output/src3862/src/git-codecommit.us'.

다음 단계

워크플로 개념에 대한 자세한 내용은 [워크플로 개념](#) 섹션을 참조하세요.

첫 번째 워크플로를 생성하려면 [워크플로 시작하기](#) 섹션을 참조하세요.

워크플로 개념

CodeCatalyst에서 워크플로를 사용하여 코드를 빌드, 테스트 또는 배포할 때 알아야 할 몇 가지 개념과 용어는 다음과 같습니다.

워크플로

워크플로는 지속적 통합 및 지속적 전송(CI/CD) 시스템의 일부로 코드를 빌드, 테스트 및 배포하는 방법을 설명하는 자동화된 절차입니다. 워크플로는 워크플로 실행 중에 수행할 일련의 단계 또는 작업을 정의합니다. 또한 워크플로는 워크플로를 시작하게 하는 이벤트 또는 트리거를 정의합니다. 워크플로를 설정하려면 CodeCatalyst 콘솔의 [시각적 또는 YAML 편집기](#)를 사용하여 워크플로 정의 파일을 생성합니다.

i Tip

프로젝트에서 워크플로를 사용하는 방법을 간략하게 알아보려면 [블루프린트가 있는 프로젝트를 생성](#)합니다. 각 블루프린트는 검토, 실행 및 실험할 수 있는 기능 워크플로를 배포합니다.

워크플로 정의 파일

워크플로 정의 파일은 워크플로를 설명하는 YAML 파일입니다. 기본적으로 파일은 [소스 리포지토리](#)의 루트에 있는 `~/.codecatalyst/workflows/` 폴더에 저장됩니다. 파일은 확장자가 `.yml` 또는 `.yaml`일 수 있으며 확장자는 소문자여야 합니다.

워크플로 정의 파일에 대한 자세한 내용은 [워크플로 YAML 정의](#) 섹션을 참조하세요.

작업

작업은 워크플로의 기본 구성 요소이며 워크플로 실행 중에 수행할 논리적 작업 단위 또는 태스크를 정의합니다. 일반적으로 워크플로에는 구성 방식에 따라 순차적으로 또는 병렬로 실행되는 여러 작업이 포함됩니다.

작업에 대한 자세한 내용은 [워크플로 작업 구성](#) 섹션을 참조하세요.

작업 그룹

작업 그룹에는 하나 이상의 작업이 포함됩니다. 작업을 작업 그룹으로 그룹화하면 워크플로를 체계적으로 관리할 수 있고, 서로 다른 그룹 간의 종속성을 구성할 수도 있습니다.

작업 그룹에 대한 자세한 내용은 [작업을 작업 그룹으로 그룹화](#) 섹션을 참조하세요.

아티팩트

아티팩트는 워크플로 작업의 출력이며 일반적으로 폴더 또는 파일 아카이브로 구성됩니다. 아티팩트를 사용하면 작업 간에 파일과 정보를 공유할 수 있기 때문에 아티팩트가 중요합니다.

아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

컴퓨팅

컴퓨팅은 워크플로 작업을 실행하기 위해 CodeCatalyst에서 관리 및 유지 관리하는 컴퓨팅 엔진(CPU, 메모리 및 운영 체제)을 의미합니다.

컴퓨팅에 대한 자세한 내용은 [컴퓨팅 및 런타임 이미지 구성](#) 섹션을 참조하세요.

환경

[개발](#) 환경과 혼동해서는 안 되는 CodeCatalyst 환경은 CodeCatalyst [워크플로](#)가 연결되는 대상 AWS 계정 및 선택적 Amazon VPC를 정의합니다. 또한 환경은 워크플로가 대상 계정 내의 AWS 서비스 및 리소스에 액세스하는 데 필요한 [IAM 역할을](#) 정의합니다.

여러 환경을 설정하고 개발, 테스트, 스테이징 및 프로덕션과 같은 이름을 지정할 수 있습니다. 이러한 환경에 배포하면 배포에 대한 정보가 환경의 CodeCatalyst 배포 활동 및 배포 대상 탭에 표시됩니다.

환경에 대한 자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 섹션을 참조하세요.

게이트

게이트는 특정 조건이 충족되지 않으면 워크플로 실행이 진행되지 않도록 하는 데 사용할 수 있는 워크플로 구성 요소입니다. 게이트의 예시로는 사용자가 워크플로 실행을 계속하기 전에 CodeCatalyst 콘솔에서 승인을 제출해야 하는 승인 게이트를 들 수 있습니다.

워크플로의 일련의 작업 사이에 또는 첫 번째 작업(소스 다운로드 직후에 실행되는) 전에 게이트를 추가할 수 있습니다. 필요한 경우 마지막 작업 후 게이트를 추가할 수도 있습니다.

게이트에 대한 자세한 내용은 [워크플로 게이팅](#) 섹션을 참조하세요.

Reports

보고서에는 워크플로 실행 중에 발생하는 테스트에 대한 세부 정보가 포함되어 있습니다. 테스트 보고서, 코드 적용 범위 보고서, 소프트웨어 구성 분석 보고서 및 정적 분석 보고서와 같은 보고서를 생성할 수 있습니다. 보고서를 사용하여 워크플로 중 문제를 해결하는 데 도움을 받을 수 있습니다. 여러 워크플로의 보고서가 많은 경우 보고서를 사용하여 추세와 실패율을 확인하여 애플리케이션 및 배포 구성을 최적화할 수 있습니다.

보고서에 대한 자세한 내용은 [품질 보고서 유형](#)의 내용을 참조하세요.

실행

실행은 워크플로의 단일 반복입니다. 실행하는 동안 CodeCatalyst는 워크플로 구성 파일에 정의된 작업을 수행하고 관련 로그, 아티팩트 및 변수를 출력합니다.

실행에 대한 자세한 내용은 [워크플로 실행](#) 섹션을 참조하세요.

소스

입력 소스라고도 하는 소스는 [워크플로 작업](#)이 작업을 수행하는 데 필요한 파일을 얻기 위해 연결되는 소스 리포지토리입니다. 예를 들어 워크플로 작업은 애플리케이션을 빌드하기 위해 애플리케이션 소스 파일을 얻기 위해 소스 리포지토리에 연결할 수 있습니다.

소스에 대한 자세한 내용은 [워크플로에 소스 리포지토리 연결](#) 섹션을 참조하세요.

변수

변수는 Amazon CodeCatalyst 워크플로에서 참조할 수 있는 정보를 포함하는 키-값 페어입니다. 변수의 값 부분은 워크플로가 실행될 때 실제 값으로 대체됩니다.

변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

워크플로 트리거

워크플로 트리거 또는 간단히 트리거를 사용하면 코드 푸시와 같은 특정 이벤트가 발생하면 워크플로 실행을 자동으로 시작할 수 있습니다. 트리거를 구성하여 소프트웨어 개발자가 CodeCatalyst 콘솔을 통해 워크플로 실행을 수동으로 시작하지 않아도 되도록 할 수 있습니다.

세 가지 유형의 트리거를 사용할 수 있습니다.

- 푸시 - 코드 푸시 트리거는 커밋이 푸시될 때마다 워크플로 실행을 시작하도록 합니다.
- 풀 요청 - 풀 요청 트리거는 풀 요청이 생성, 수정 또는 종료될 때마다 워크플로 실행을 시작하게 합니다.
- 일정 - 일정 트리거는 정의한 일정에 따라 워크플로 실행이 시작되도록 합니다. 소프트웨어 개발자가 다음 날 아침에 작업할 수 있도록 최신 빌드를 준비할 수 있도록 일정 트리거를 사용하여 소프트웨어의 야간 빌드를 실행하는 것을 고려하세요.

푸시, 풀 요청 및 예약 트리거를 단독으로 사용하거나 동일한 워크플로에서 조합하여 사용할 수 있습니다.

트리거는 선택 사항입니다. 구성하지 않으면 워크플로를 수동으로만 시작할 수 있습니다.

트리거에 대한 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 주제를 참조하세요.

워크플로 시작하기

이 자습서에서는 첫 번째 워크플로를 생성하고 구성하는 방법을 알아봅니다.

i Tip

미리 구성된 워크플로로 시작하시겠습니까? 작동하는 워크플로, 샘플 애플리케이션 및 기타 리소스를 사용하여 프로젝트를 설정하는 지침이 포함된 [블루프린트를 사용하여 프로젝트 생성](#) 섹션을 참조하세요.

주제

- [사전 조건](#)
- [1단계: 워크플로 생성 및 구성](#)
- [2단계: 커밋을 사용하여 워크플로 저장](#)
- [3단계: 실행 결과 보기](#)
- [\(선택 사항\) 4단계: 정리](#)

사전 조건

시작하기 전:

- CodeCatalyst 스페이스가 필요합니다. 자세한 내용은 [스페이스 생성](#) 섹션을 참조하세요.
- CodeCatalyst 스페이스에는 다음과 같은 빈 프로젝트가 필요합니다.

```
codecatalyst-project
```

자세한 내용은 [Amazon CodeCatalyst에서 빈 프로젝트 생성](#) 섹션을 참조하세요.

- 프로젝트에는 다음과 같은 CodeCatalyst 리포지토리가 필요합니다.

```
codecatalyst-source-repository
```

자세한 내용은 [소스 리포지토리 생성](#) 섹션을 참조하세요.

i Note

기존 프로젝트와 소스 리포지토리가 있는 경우 이를 사용할 수 있지만, 이 자습서 마지막에 새 프로젝트를 만들면 정리가 더 쉬워집니다.

1단계: 워크플로 생성 및 구성

이 단계에서는 소스 코드가 변경될 때 자동으로 빌드하고 테스트하는 워크플로를 만들고 구성합니다.

워크플로 생성

1. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
2. 워크플로 생성을 선택합니다.

워크플로 정의 파일은 CodeCatalyst 콘솔의 YAML 편집기에 나타납니다.

워크플로 구성

시각적 편집기 또는 YAML 편집기에서 워크플로를 구성할 수 있습니다. YAML 편집기로 시작한 다음 시각적 편집기로 전환해 보겠습니다.

1. + 작업을 선택하면 워크플로에 추가할 수 있는 워크플로 작업 목록을 볼 수 있습니다.
2. 빌드 작업에서 +를 선택하여 작업의 YAML을 워크플로 정의 파일에 추가합니다. 이제 워크플로는 다음과 유사하게 보입니다.

```
Name: Workflow_fe47
SchemaVersion: "1.0"

# Optional - Set automatic triggers.
Triggers:
  - Type: Push
    Branches:
      - main

# Required - Define action configurations.
Actions:
  Build_f0:
    Identifier: aws/build@v1

    Inputs:
      Sources:
        - WorkflowSource # This specifies that the action requires this workflow as
a source

    Outputs:
      AutoDiscoverReports:
```



```

    Enabled: true
    # Use as prefix for the report files
    ReportNamePrefix: rpt

Configuration:
  Steps:
    - Run: echo "Hello, World!"
    - Run: echo "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>" >> report.xml
    - Run: echo "<testsuite tests=\"1\" name=\"TestAgentJunit\" >" >>
report.xml
    - Run: echo "<testcase classname=\"TestAgentJunit\" name=\"Dummy
Test\"/></testsuite>" >> report.xml

```

워크플로는 WorkflowSource 소스 리포지토리의 파일을 Build_f0 작업을 실행하는 컴퓨팅 머신에 복사하고, Hello, World!를 로그에 인쇄하고, 컴퓨팅 머신에서 테스트 보고서를 검색하고, 이를 CodeCatalyst 콘솔의 보고서 페이지에 출력합니다.

3. 시각적 편집기에서 워크플로 정의 파일을 보려면 시각적을 선택합니다. 시각적 편집기의 필드를 사용하면 YAML 편집기에 표시된 YAML 속성을 구성할 수 있습니다.

2단계: 커밋을 사용하여 워크플로 저장

이 단계에서는 변경 사항을 저장합니다. 워크플로는 리포지토리에 .yaml 파일로 저장되므로 커밋과 함께 변경 사항을 저장합니다.

워크플로 변경 사항 커밋

1. (선택 사항) 워크플로의 YAML 코드가 유효한지 확인하려면 검증을 선택합니다.
2. 커밋을 선택합니다.
3. 워크플로 파일 이름에 **my-first-workflow**와 같은 워크플로 구성 파일의 이름을 입력합니다.
4. 커밋 메시지에 **create my-first-workflow.yaml**과 같은 커밋을 식별하는 메시지를 입력합니다.
5. 리포지토리에서 (codecatalyst-repository) 워크플로를 저장할 리포지토리를 선택합니다.
6. 브랜치 이름에서 (main) 워크플로를 저장할 브랜치를 선택합니다.
7. 커밋을 선택합니다.

새 워크플로가 워크플로 목록에 나타납니다. 표시하는 데 몇 초 정도 걸릴 수 있습니다.

워크플로는 커밋과 함께 저장되고 워크플로에 코드 푸시 트리거가 구성되어 있으므로 워크플로를 저장하면 워크플로 실행이 자동으로 시작됩니다.

3단계: 실행 결과 보기

이 단계에서는 커밋에서 시작된 실행으로 이동하여 결과를 봅니다.

실행 결과 보기

1. 워크플로의 이름을 선택합니다. 예를 들어 Workflow_fe47입니다.

소스 리포지토리의 레이블(WorkflowSource)과 빌드 작업(예: Build_f0)을 보여주는 워크플로 다이어그램입니다.

2. 워크플로 실행 다이어그램에서 빌드 작업(예: Build_f0)을 선택합니다.
3. 로그, 보고서, 구성 및 변수 탭의 내용을 검토합니다. 이 탭은 빌드 작업의 결과를 보여줍니다.

자세한 내용은 [빌드 작업 결과 보기](#) 섹션을 참조하세요.

(선택 사항) 4단계: 정리

이 단계에서는 자습서에서 생성한 리소스를 정리합니다.

리소스 삭제

- 이 자습서를 위해 새 프로젝트를 만들었다면 삭제합니다. 지침은 [프로젝트 삭제](#) 섹션을 참조하세요. 프로젝트를 삭제하면 소스 리포지토리도 삭제됩니다.

워크플로로 빌드하기

[CodeCatalyst 워크플로](#)를 사용하여 애플리케이션 및 기타 리소스를 구축할 수 있습니다.

주제

- [애플리케이션을 빌드하려면 어떻게 해야 하나요?](#)
- [빌드 작업의 이점](#)
- [빌드 작업의 대안](#)
- [빌드 작업 추가](#)
- [빌드 작업 결과 보기](#)

- [자습서: Amazon S3에 아티팩트 업로드](#)
- [빌드 및 테스트 작업 YAML](#)

애플리케이션을 빌드하려면 어떻게 해야 하나요?

CodeCatalyst에서 애플리케이션 또는 리소스를 빌드하려면 먼저 워크플로를 생성한 다음 그 안에 빌드 작업을 지정합니다.

빌드 작업은 소스 코드를 컴파일하고 단위 테스트를 실행하며 배포할 준비가 완료된 아티팩트를 생성하는 워크플로 기본 요소입니다.

CodeCatalyst 콘솔의 시각적 편집기 또는 YAML 편집기를 사용하여 워크플로에 빌드 작업을 추가합니다.

애플리케이션 또는 리소스를 빌드하는 상위 단계는 다음과 같습니다.

애플리케이션을 빌드하려면(고급 작업)

1. CodeCatalyst에서 빌드하려는 애플리케이션의 소스 코드를 추가합니다. 자세한 내용은 [CodeCatalyst에서 프로젝트의 리포지토리에 소스 코드 저장](#) 단원을 참조하십시오.
2. CodeCatalyst에서 워크플로를 생성합니다. 워크플로에서는 애플리케이션을 빌드, 테스트 및 배포하는 방법을 정의합니다. 자세한 내용은 [워크플로 시작하기](#) 단원을 참조하십시오.
3. (선택 사항) 워크플로에 워크플로를 자동으로 시작하게 하는 이벤트를 나타내는 트리거를 추가합니다. 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 섹션을 참조하세요.
4. 워크플로에서 애플리케이션 또는 리소스 소스 코드를 컴파일하고 패키징하는 빌드 작업을 추가합니다. 이러한 용도로 테스트를 사용하거나 작업을 배포하지 않으려는 경우 필요에 따라 빌드 작업에서 단위 테스트를 실행하고, 보고서를 생성하고, 애플리케이션을 배포할 수도 있습니다. 테스트 및 배포 작업에 대한 자세한 내용은 [빌드 작업 추가](#) 섹션을 참조하세요.
5. (선택 사항) 워크플로에서 테스트 작업과 배포 작업을 추가하여 애플리케이션 또는 리소스를 테스트하고 배포합니다. 사전 구성된 여러 작업 중에서 선택하여 Amazon ECS와 같은 다양한 대상에 애플리케이션을 배포할 수 있습니다. 자세한 내용은 [워크플로를 사용한 테스트](#) 및 [워크플로를 사용하여 배포](#) 섹션을 참조하세요.
6. 트리거를 통해 워크플로를 수동으로 또는 자동으로 시작합니다. 워크플로는 빌드, 테스트 및 배포 작업을 순서대로 실행하여 애플리케이션 및 리소스를 빌드, 테스트 및 대상에 배포합니다. 자세한 내용은 [워크플로 수동 실행 시작](#) 섹션을 참조하세요.

빌드 작업의 이점

워크플로 내에서 빌드 작업을 사용하면 다음과 같은 이점이 있습니다.

- 완전 관리형 - 빌드 작업에서는 빌드 서버를 직접 설정하고, 패치 및 업데이트를 적용하고, 관리할 필요가 없습니다.
- 온디맨드 - 빌드 작업은 빌드 요구 사항을 충족하기 위해 요구에 따라 규모가 조정됩니다. 사용한 빌드 시간만큼만 요금을 지불합니다. 자세한 내용은 [컴퓨팅 및 런타임 이미지 구성](#) 섹션을 참조하세요.
- 기본 제공 - CodeCatalyst에는 빌드 작업을 포함한 모든 워크플로 작업을 실행하는 데 사용되는 사전 패키징된 런타임 환경 Docker 이미지가 포함되어 있습니다. 이러한 이미지는 AWS CLI 및 Node.js와 같은 애플리케이션을 빌드하는 데 유용한 도구로 미리 구성되어 있습니다. 퍼블릭 또는 프라이빗 레지스트리에서 제공하는 빌드 이미지를 사용하도록 CodeCatalyst를 구성할 수 있습니다. 자세한 내용은 [런타임 환경 이미지 지정](#) 섹션을 참조하세요.

빌드 작업의 대안

빌드 작업을 사용하여 애플리케이션을 배포하는 경우 CodeCatalyst 배포 작업을 대신 사용하는 것이 좋습니다. 배포 작업은 빌드 작업을 사용하는 경우 수동으로 작성해야 하는 백그라운드 구성을 수행합니다. 사용 가능한 배포 작업에 대한 자세한 내용은 [배포 작업 목록](#) 섹션을 참조하세요.

AWS CodeBuild 를 사용하여 애플리케이션을 빌드할 수도 있습니다. 자세한 내용은 [CodeBuild란 무엇입니까?](#)를 참조하세요.

빌드 작업 추가

다음 절차에 따라 CodeCatalyst 워크플로에 빌드 작업을 추가합니다.

Visual

시각적 편집기를 사용하여 빌드 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
3. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
4. 편집을 선택합니다.
5. 비주얼을 선택합니다.

6. 작업을 선택합니다.
7. 작업에서 빌드를 선택합니다.
8. 입력 및 구성 탭에서 필요에 따라 필드를 작성합니다. 각 필드의 설명은 [빌드 및 테스트 작업 YAML](#) 섹션을 참조하세요. 이 참조는 YAML 및 시각적 편집기 모두에 나타나는 각 필드(및 해당 YAML 속성 값)에 대한 자세한 정보를 제공합니다.
9. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
10. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 빌드 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
3. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
4. 편집을 선택합니다.
5. YAML을 선택합니다.
6. 작업을 선택합니다.
7. 작업에서 빌드를 선택합니다.
8. 필요에 따라 YAML 코드의 속성을 수정합니다. 사용 가능한 각 속성에 대한 설명은 [빌드 및 테스트 작업 YAML](#)에서 볼 수 있습니다.
9. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
10. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

작업 정의

빌드 작업은 워크플로 정의 파일 내의 YAML 속성 집합으로 정의됩니다. 이러한 속성에 대한 자세한 내용은 [워크플로 YAML 정의](#)의 [빌드 및 테스트 작업 YAML](#) 섹션을 참조하세요.

빌드 작업 결과 보기

다음 지침을 사용하여 생성된 로그, 보고서 및 변수를 포함한 빌드 작업의 결과를 확인합니다.

빌드 작업의 결과를 보려면

1. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
2. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
3. 워크플로 다이어그램에서 빌드 작업의 이름, 예를 들어 Build를 선택합니다.
4. 빌드 실행에 대한 로그를 보려면 로그를 선택합니다. 다양한 빌드 단계의 로그가 표시됩니다. 필요에 따라 로그를 확장하거나 축소할 수 있습니다.
5. 빌드 작업에서 생성된 테스트 보고서를 보려면 보고서를 선택하거나 탐색 창에서 보고서를 선택합니다. 자세한 내용은 [품질 보고서 유형](#) 단원을 참조하십시오.
6. 빌드 작업에 사용되는 구성을 보려면 구성을 선택합니다. 자세한 내용은 [빌드 작업 추가](#) 단원을 참조하십시오.
7. 빌드 작업에 사용되는 변수를 보려면 변수를 선택합니다. 자세한 내용은 [워크플로에서 변수 사용](#) 단원을 참조하십시오.

자습서: Amazon S3에 아티팩트 업로드

이 자습서에서는 몇 가지 [빌드 작업이](#) 포함된 Amazon CodeCatalyst [워크플로](#)를 사용하여 Amazon S3 버킷에 아티팩트를 업로드하는 방법을 알아봅니다. 이러한 작업은 워크플로가 시작될 때 직렬로 실행됩니다. 첫 번째 빌드 작업은 두 개의 파일 Hello.txt 및 Goodbye.txt를 생성하고 이를 빌드 아티팩트에 번들링합니다. 두 번째 빌드 작업은 아티팩트를 Amazon S3에 업로드합니다. 커밋을 소스 리포지토리로 푸시할 때마다 워크플로가 실행되도록 구성합니다.

주제

- [사전 조건](#)
- [1단계: AWS 역할 생성](#)
- [2단계: Amazon S3 버킷 생성](#)
- [3단계: 소스 리포지토리 생성](#)
- [4단계: 워크플로 생성](#)
- [5단계: 실험 결과 확인](#)
- [정리](#)

사전 조건

시작하려면 다음이 필요합니다.

- 연결된 AWS 계정이 있는 CodeCatalyst 스페이스가 필요합니다. 자세한 내용은 [스페이스 생성](#) 섹션을 참조하세요.
- 스페이스에는 다음과 같은 빈 프로젝트가 필요합니다.

```
codecatalyst-artifact-project
```

처음부터 시작 옵션을 사용하여 이 프로젝트를 생성합니다.

자세한 내용은 [Amazon CodeCatalyst에서 빈 프로젝트 생성](#) 섹션을 참조하세요.

- 프로젝트에는 다음과 같은 CodeCatalyst 환경이 필요합니다.

```
codecatalyst-artifact-environment
```

다음과 같이 이 환경을 구성합니다.

- 개발 등 유형을 선택합니다.
- AWS 계정에 연결합니다.
- 기본 IAM 역할의 경우 아무 역할이나 선택합니다. 나중에 다른 역할을 지정합니다.

자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 단원을 참조하십시오.

1단계: AWS 역할 생성

이 단계에서는 나중에 워크플로의 빌드 작업에 할당할 AWS IAM 역할을 생성합니다. 이 역할은 CodeCatalyst 빌드 작업에 AWS 계정에 액세스하고 아티팩트가 저장될 Amazon S3에 쓸 수 있는 권한을 부여합니다. 이 역할을 빌드 역할이라고 합니다.

Note

다른 자습서를 위해 생성한 빌드 역할이 이미 있는 경우 이 자습서에도 사용할 수 있습니다. 다음 절차에 표시된 권한 및 신뢰 정책이 있는지 확인하세요.

IAM 역할에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 [IAM 역할을 참조](#)하세요.

빌드 역할을 생성하려면

1. 역할에 대한 정책을 다음과 같이 생성합니다.
 - a. 에 로그인합니다 AWS.
 - b. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
 - c. 탐색 창에서 Policies를 선택합니다.
 - d. 정책 생성을 선택합니다.
 - e. JSON 탭을 선택합니다.
 - f. 기존 코드를 삭제합니다.
 - g. 다음 코드를 붙여넣습니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:ListBucket"
      ],
      "Resource": "*"
    }
  ]
}
```

 Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.


```
"Resource": "*"

```

- h. 다음: 태그를 선택합니다.
- i. 다음: 검토를 선택합니다.
- j. 이름에 다음과 같이 입력합니다.

```
codecatalyst-s3-build-policy

```

- k. 정책 생성을 선택합니다.

이제 권한 정책을 생성했습니다.

- 2. 다음과 같이 빌드 역할을 생성합니다.

- a. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
- b. 사용자 지정 신뢰 정책을 선택합니다.
- c. 기존 사용자 지정 신뢰 정책을 삭제합니다.
- d. 다음 사용자 지정 신뢰 정책을 추가합니다.
- e. 다음을 선택합니다.
- f. 권한 정책에서 codecatalyst-s3-build-policy를 검색하고 해당 확인란을 선택합니다.
- g. 다음을 선택합니다.
- h. 역할 이름에 다음과 같이 입력합니다.

```
codecatalyst-s3-build-role

```

- i. 역할 설명에 다음과 같이 입력합니다.

```
CodeCatalyst build role

```

- j. 역할 생성을 선택합니다.

이제 신뢰 정책 및 권한 정책으로 빌드 역할을 생성했습니다.

2단계: Amazon S3 버킷 생성

~~이 단계에서는 Hello.txt 및 Goodbye.txt 아티팩트가 업로드되는 Amazon S3 버킷을 생성합니다.~~

자습서: Amazon S3에 아티팩트 업로드

423

Amazon S3 버킷을 생성하려면

1. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.
2. 기본 창에서 버킷 생성을 선택합니다.
3. 버킷 이름에 다음과 같이 입력합니다.

```
codecatalyst-artifact-bucket
```

4. AWS 리전에서 리전을 선택합니다. 이 자습서에서는 미국 서부(오리건) us-west-2를 선택했다고 가정합니다. Amazon S3에서 지원되는 리전에 대한 자세한 내용은 AWS 일반 참조의 [Amazon Simple Storage Service 엔드포인트 및 할당량](#)을 참조하세요.
5. 페이지 맨 아래에 있는 버킷 생성 버튼을 선택합니다.
6. 방금 생성한 버킷의 이름을 복사합니다. 예시:

```
codecatalyst-artifact-bucket
```

이제 미국 서부(오리건) us-west-2 리전에 **codecatalyst-artifact-bucket** 버킷을 생성했습니다.

3단계: 소스 리포지토리 생성

이 단계에서는 CodeCatalyst에 소스 리포지토리를 생성합니다. 이 리포지토리는 자습서의 워크플로 정의 파일을 저장하는 데 사용됩니다.

소스 리포지토리에 대한 자세한 정보는 [소스 리포지토리 생성](#) 섹션을 참조하세요.

소스 리포지토리를 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. `codecatalyst-artifact-project` 프로젝트로 이동합니다.
3. 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
4. 리포지토리 추가를 선택하고 리포지토리 생성을 선택합니다.
5. 리포지토리 이름에 다음과 같이 입력합니다.

```
codecatalyst-artifact-source-repository
```

6. 생성(Create)을 선택합니다.

이제 codecatalyst-artifact-source-repository 리포지토리를 생성했습니다.

4단계: 워크플로 생성

이 단계에서는 순차적으로 실행되는 다음 구성 요소로 구성된 워크플로를 생성합니다.

- 트리거 - 이 트리거는 소스 리포지토리에 변경 사항을 푸시할 때 워크플로 실행을 자동으로 시작합니다. 트리거에 대한 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 섹션을 참조하세요.
- 빌드 작업 GenerateFiles - 트리거 시 GenerateFiles 작업은 두 개의 파일 Hello.txt 및 Goodbye.txt를 생성하고 출력 아티팩트 codecatalystArtifact에 패키징합니다.
- 다른 빌드 작업 Upload - GenerateFiles 작업이 완료되면 Upload 작업이 AWS CLI 명령 `aws s3 sync`를 실행하여 소스 리포지토리의 codecatalystArtifact 및 소스 리포지토리에 있는 파일을 Amazon S3 버킷에 업로드합니다. 는 CodeCatalyst 컴퓨팅 플랫폼에 사전 설치 및 구성되어 AWS CLI 제공되므로 설치하거나 구성할 필요가 없습니다.

CodeCatalyst 컴퓨팅 플랫폼의 사전 패키징된 소프트웨어에 대한 자세한 내용은 [런타임 환경 이미지 지정](#) 섹션을 참조하세요. AWS CLI명령에 대한 자세한 내용은 `aws s3 sync` 명령 참조의 [동기화](#)를 AWS CLI 참조하세요.

이러한 빌드 작업에 대한 자세한 내용은 [워크플로로 빌드하기](#) 섹션을 참조하세요.

워크플로 생성

1. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
2. 워크플로 생성을 선택합니다.
3. YAML 샘플 코드를 삭제합니다.
4. 다음 YAML 코드를 추가합니다.

Note

다음 YAML 코드에서 원하는 경우 Connections: 섹션을 생략할 수 있습니다. 이 섹션을 생략하는 경우 환경의 기본 IAM 역할 필드에 지정된 역할에 [1단계: AWS 역할 생성](#)에 설명된 권한 및 신뢰 정책이 포함되어 있는지 확인해야 합니다. 기본 IAM 역할이 있는 환경 설정에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.

Name: codecatalyst-artifact-workflow

```

SchemaVersion: 1.0

Triggers:
  - Type: Push
    Branches:
      - main
Actions:
  GenerateFiles:
    Identifier: aws/build@v1
    Configuration:
      Steps:
        # Create the output files.
        - Run: echo "Hello, World!" > "Hello.txt"
        - Run: echo "Goodbye!" > "Goodbye.txt"
    Outputs:
      Artifacts:
        - Name: codecatalystArtifact
          Files:
            - "**/*"
  Upload:
    Identifier: aws/build@v1
    DependsOn:
      - GenerateFiles
    Environment:
      Name: codecatalyst-artifact-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-s3-build-role
    Inputs:
      Artifacts:
        - codecatalystArtifact
    Configuration:
      Steps:
        # Upload the output artifact to the S3 bucket.
        - Run: aws s3 sync . s3://codecatalyst-artifact-bucket

```

위의 코드를 다음과 같이 변경합니다.

- *codecatalyst-artifact-environment*를 [사전 조건](#)에서 생성한 환경 이름으로 변경합니다.
- *codecatalyst-account-connection*을 [사전 조건](#)에서 생성한 계정 연결 이름으로 변경합니다.

- `codecatalyst-s3-build-role`을 [1단계: AWS 역할 생성](#)에서 생성한 빌드 역할 이름으로 변경합니다.
- `codecatalyst-artifact-bucket`을 [2단계: Amazon S3 버킷 생성](#)에서 생성한 Amazon S3 이름으로 변경합니다.

이러한 파일의 속성에 대한 자세한 내용은 [빌드 및 테스트 작업 YAML](#) 섹션을 참조하세요.

5. (선택 사항) 커밋하기 전에 YAML 코드가 유효한지 확인하려면 검증을 선택합니다.
6. 커밋을 선택합니다.
7. 워크플로 커밋 대화 상자에서 다음을 입력합니다.
 - a. 워크플로 파일 이름은 기본값인 `codecatalyst-artifact-workflow`를 그대로 사용합니다.
 - b. 커밋 메시지에 다음을 입력합니다.

add initial workflow file

- c. 리포지토리에서 `codecatalyst-artifact-source-repository`를 선택합니다.
- d. 브랜치 이름에서 기본을 선택합니다.
- e. 커밋을 선택합니다.

이제 워크플로를 생성했습니다. 워크플로 상단에 정의된 트리거로 인해 워크플로 실행이 자동으로 시작됩니다. 특히 `codecatalyst-artifact-workflow.yaml` 파일을 소스 리포지토리에 커밋(및 푸시)할 때 트리거가 워크플로 실행을 시작했습니다.

진행 중인 워크플로 실행을 보려면

1. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
2. 방금 생성한 `codecatalyst-artifact-workflow` 워크플로를 선택합니다.
3. `GenerateFiles`를 선택하여 첫 번째 빌드 작업 진행 상황을 확인합니다.
4. 업로드를 선택하여 두 번째 빌드 작업 진행 상황을 확인합니다.
5. 업로드 작업이 완료되면 다음을 수행합니다.
 - 워크플로 실행에 성공한 경우 다음 절차로 이동합니다.
 - 워크플로 실행에 실패한 경우 로그를 선택하여 문제를 해결합니다.

5단계: 실험 결과 확인

워크플로가 실행된 후 Amazon S3 서비스로 이동하여 *codecatalyst-artifact-bucket* 버킷을 확인합니다. 이제 다음 파일과 폴더가 포함되어야 합니다.

```
.
|- .aws/
|- .git/
|Goodbye.txt
|Hello.txt
|README.md
```

Goodbye.txt 및 Hello.txt 파일은 codecatalystArtifact 아티팩트의 일부이므로 업로드되었습니다. .aws/, .git/ 및 README.md 파일은 소스 리포지토리에 있었기 때문에 업로드되었습니다.

정리

이러한 서비스에 대한 요금이 부과되지 않도록 CodeCatalyst 및 AWS 에서 정리합니다.

CodeCatalyst에서 정리하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. codecatalyst-artifact-source-repository 소스 리포지토리를 삭제합니다.
3. codecatalyst-artifact-workflow 워크플로를 선택합니다.

에서 정리하려면 AWS

1. 다음과 같이 Amazon S3에서 정리합니다.
 - a. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.
 - b. codecatalyst-artifact-bucket 버킷에서 파일을 삭제합니다.
 - c. codecatalyst-artifact-bucket 버킷을 삭제합니다.
2. 다음과 같이 IAM에서 정리합니다.
 - a. IAM 콘솔(<https://console.aws.amazon.com/iam/>)을 엽니다.
 - b. codecatalyst-s3-build-policy를 삭제합니다.
 - c. codecatalyst-s3-build-role를 삭제합니다.

빌드 및 테스트 작업 YAML

다음은 빌드 및 테스트 작업의 YAML 정의입니다. YAML 속성이 매우 유사하기 때문에 두 작업에 대한 참조가 하나 있습니다.

이 작업 정의는 더 광범위한 워크플로 정의 파일 내의 섹션으로 존재합니다. 이 파일에 대한 자세한 내용은 [워크플로 YAML 정의](#)를 참조합니다.

다음 코드에서 YAML 속성을 선택하면 설명이 표시됩니다.

Note

이어지는 대부분의 YAML 속성에는 시각적 편집기에 해당 UI 요소가 있습니다. UI 요소를 찾으려면 Ctrl+F를 사용합니다. 요소가 연결된 YAML 속성과 함께 나열됩니다.

```
# The workflow definition starts here.
# See ### ## for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
action-name:
  Identifier: aws/build@v1 | aws/managed-test@v1
  DependsOn:
    - dependent-action-name-1
  Compute:
    Type: EC2 | Lambda
    Fleet: fleet-name
  Timeout: timeout-minutes
  Environment:
    Name: environment-name
  Connections:
    - Name: account-connection-name
      Role: iam-role-name
  Caching:
    FileCaching:
      key-name-1:
        Path: file1.txt
      RestoreKeys:
```

- *restore-key-1*

Inputs:Sources:

- *source-name-1*
- *source-name-2*

Artifacts:

- *artifact-name*

Variables:

- Name: *variable-name-1*
Value: *variable-value-1*
- Name: *variable-name-2*
Value: *variable-value-2*

Outputs:Artifacts:

- Name: *output-artifact-1*

Files:

- build-output/artifact-1.jar
- "build-output/build*"

- Name: *output-artifact-2*

Files:

- build-output/artifact-2.1.jar
- build-output/artifact-2.2.jar

Variables:

- *variable-name-1*
- *variable-name-2*

AutoDiscoverReports:Enabled: *true | false*ReportNamePrefix: *AutoDiscovered*IncludePaths:

- *"/**/*"*

ExcludePaths:

- *node_modules/cdk/junit.xml*

SuccessCriteria:PassRate: *percent*LineCoverage: *percent*BranchCoverage: *percent*Vulnerabilities:Severity: *CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL*Number: *whole-number*StaticAnalysisBug:Severity: *CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL*Number: *whole-number*StaticAnalysisSecurity:Severity: *CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL*


```

    Number: whole-number
  StaticAnalysisQuality:
    Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
    Number: whole-number
  StaticAnalysisFinding:
    Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
    Number: whole-number
Reports:
  report-name-1:
    Format: format
    IncludePaths:
      - "*.xml"
    ExcludePaths:
      - report2.xml
      - report3.xml
    SuccessCriteria:
      PassRate: percent
      LineCoverage: percent
      BranchCoverage: percent
    Vulnerabilities:
      Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
      Number: whole-number
    StaticAnalysisBug:
      Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
      Number: whole-number
    StaticAnalysisSecurity:
      Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
      Number: whole-number
    StaticAnalysisQuality:
      Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
      Number: whole-number
    StaticAnalysisFinding:
      Severity: CRITICAL | HIGH | MEDIUM | LOW | INFORMATIONAL
      Number: whole-number
Configuration:
  Container:
    Registry: registry
    Image: image
  Steps:
    - Run: "step 1"
    - Run: "step 2"
  Packages:
    NpmConfiguration:
      PackageRegistries:

```

```
- PackagesRepository: package-repository
  Scopes:
    - "@scope"
ExportAuthorizationToken: true | false
```

action-name

(필수)

작업 이름을 지정합니다. 워크플로 내의 모든 작업 이름은 고유해야 합니다. 작업 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 작업 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

해당 UI: 구성 탭/작업 이름

Identifier

(*action-name*/Identifier)

작업을 식별합니다. 버전을 변경하려는 경우가 아니면 이 속성을 변경하지 마세요. 자세한 내용은 [사용할 작업 버전 지정](#) 섹션을 참조하세요.

빌드 작업에 `aws/build@v1`를 사용합니다.

테스트 작업에 `aws/managed-test@v1`를 사용합니다.

해당 UI: 워크플로 다이어그램/Action-name/*aws/build@v1*/*aws/managed-test@v1* 레이블

DependsOn

(*action-name* /DependsOn)

(선택 사항)

이 작업을 실행하기 위해 성공적으로 실행해야 하는 작업, 작업 그룹 또는 게이트를 지정합니다.

'depends on' 함수에 대한 자세한 내용은 [작업 순서 지정](#) 섹션을 참조하세요.

해당 UI: 입력 탭/의존 - 선택 사항

Compute

(*action-name*/Compute)

(선택 사항)

워크플로 작업을 실행하는 데 사용되는 컴퓨팅 엔진입니다. 워크플로 수준 또는 작업 수준에서 컴퓨팅을 지정할 수 있지만 둘 다 지정할 수는 없습니다. 워크플로 수준에서 지정하면 컴퓨팅 구성이 워크플로에 정의된 모든 작업에 적용됩니다. 워크플로 수준에서는 동일한 인스턴스에서 여러 작업을 실행할 수도 있습니다. 자세한 내용은 [작업 간에 컴퓨팅 공유](#) 섹션을 참조하세요.

해당 UI: 없음

Type

(*action-name*/Compute/Type)

([Compute](#) 포함 시 필수)

컴퓨팅 엔진의 유형입니다. 다음 값 중 하나를 사용할 수 있습니다.

- EC2(시각 편집기) 또는 EC2(YAML 편집기)
작업 실행 중 유연성을 위해 최적화되었습니다.
- Lambda(시각 편집기) 또는 Lambda(YAML 편집기)
작업 시작 속도를 최적화했습니다.

컴퓨팅 유형에 대한 자세한 정보는 [컴퓨팅 유형](#)을 참고하세요.

해당 UI: 구성 탭/컴퓨팅 유형

Fleet

(*action-name*/Compute/Fleet)

(선택 사항)

워크플로 또는 워크플로 작업을 실행할 시스템 또는 플릿을 지정합니다. 온디맨드 플릿의 경우 작업이 시작되면 워크플로가 필요한 리소스를 프로비저닝하고 작업이 완료되면 시스템이 파괴됩니다. 온디맨드 플릿의 예시: Linux.x86-64.Large, Linux.x86-64.XLarge. 온디맨드 플릿에 대한 자세한 내용은 [온디맨드 플릿 속성](#) 섹션을 참조하세요.

프로비저닝된 플릿을 사용하면 워크플로 작업을 실행하도록 전용 시스템 세트를 구성할 수 있습니다. 이러한 시스템은 유휴 상태로 유지되므로 작업을 즉시 처리할 수 있습니다. 프로비저닝된 플릿에 대한 자세한 내용은 [프로비저닝된 플릿 속성](#) 섹션을 참조하세요.

Fleet 생략 시 기본값은 `Linux.x86-64.Large`입니다.

해당 UI: 구성 탭/컴퓨팅 플릿

Timeout

(*action-name*/Timeout)

(선택 사항)

CodeCatalyst가 작업을 종료하기 전에 작업을 실행할 수 있는 시간을 분(YAML 편집기) 또는 시간 및 분(시각적 편집기) 단위로 지정합니다. 최소값은 5분이고 최대값은 [CodeCatalyst의 워크플로 할당량](#)에 설명되어 있습니다. 기본 제한 시간은 최대 제한 시간과 동일합니다.

해당 UI: 구성 탭/제한 시간 - 선택 사항

Environment

(*action-name*/Environment)

(선택 사항)

작업에 사용할 CodeCatalyst 환경을 지정합니다. 작업은 선택한 환경에 지정된 AWS 계정 및 선택적 Amazon VPC에 연결됩니다. 작업은 환경에 지정된 기본 IAM 역할을 사용하여 연결하고 [Amazon VPC 연결](#)에 지정된 IAM 역할을 AWS 계정으로 사용하여 Amazon VPC에 연결합니다.

Note

기본 IAM 역할에 작업에 필요한 권한이 없는 경우 다른 역할을 사용하도록 작업을 구성할 수 있습니다. 자세한 내용은 [작업의 IAM 역할 변경](#) 섹션을 참조하세요.

환경에 대한 자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 및 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 구성 탭/환경

Name

(*action-name*/Environment/Name)

(선택 사항)

작업과 연결하려는 기존 환경의 이름을 지정합니다.

해당 UI: 구성 탭/환경

Connections

(*action-name*/Environment/Connections)

(선택 사항)

작업과 연결할 계정 연결을 지정합니다. Environment에서 계정 연결을 최대 1개까지 지정할 수 있습니다.

계정 연결을 지정하지 않는 경우:

- 작업은 CodeCatalyst 콘솔의 환경에 지정된 AWS 계정 연결 및 기본 IAM 역할을 사용합니다. 환경에 계정 연결 및 기본 IAM 역할을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.
- 기본 IAM 역할에는 작업에 필요한 정책 및 권한이 포함되어야 합니다. 이러한 정책 및 권한이 무엇인지 확인하려면 작업의 YAML 정의 설명서에서 역할 속성에 대한 설명을 참조하세요.

계정 연결에 대한 자세한 정보는 [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 섹션을 참조하세요. 환경에 계정 연결을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환

Name

(*action-name*/Environment/Connections/Name)

([Connections](#) 포함 시 필수)

계정 연결의 이름을 지정합니다.

해당 UI: 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환

Role

(*action-name*/Environment/Connections/Role)

([Connections](#) 포함 시 필수)

Amazon S3 및 Amazon ECR과 같은 AWS 서비스에 액세스하고 운영하기 위해 이 작업이 사용하는 IAM 역할의 이름을 지정합니다. 이 역할이 스페이스의 AWS 계정 연결에 추가되었는지 확인합니다. 계정 연결에 IAM 역할을 추가하려면 [계정 연결에 IAM 역할 추가](#) 섹션을 참조하세요.

IAM 역할을 지정하지 않으면 작업은 CodeCatalyst 콘솔의 [환경](#)에 나열된 기본 IAM 역할을 사용합니다. 환경에서 기본 역할을 사용하는 경우 다음 정책이 있는지 확인합니다.

Note

이 작업에서 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 사용할 수 있습니다. 이에 대한 자세한 내용은 [계정 및 스페이스의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 생성](#) 섹션을 참조하세요. CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할에 보안 위험을 초래할 수 있는 전체 액세스 권한이 있음을 이해합니다. 보안에 대한 우려가 적은 자습서 및 시나리오에서만 이 역할을 사용하는 것이 좋습니다.

Warning

빌드 및 테스트 작업에 필요한 권한으로 권한을 제한합니다. 더 광범위한 권한을 가진 역할을 사용하면 보안 위험이 발생할 수 있습니다.

해당 UI: 구성 탭/환경/# ##의 내용/점 3개 메뉴/역할 전환

Caching

(*action-name*/Caching)

(선택 사항)

캐시를 지정하여 디스크 파일을 저장하고 후속 워크플로 실행 시 해당 캐시에서 복원할 수 있는 섹션입니다.

파일 캐싱에 대한 자세한 정보는 [워크플로 실행 간 파일 캐싱](#)을 참조하세요.

해당 UI: 구성 탭/파일 캐싱 - 선택 사항

FileCaching

(*action-name*/Caching/FileCaching)

(선택 사항)

캐시 시퀀스의 구성을 지정하는 섹션입니다.

해당 UI: 구성 탭/파일 캐싱 - 선택적/캐시 추가

key-name-1

(*action-name*/Caching/FileCaching/*key-name-1*)

(선택 사항)

기본 캐시 속성 이름의 이름을 지정합니다. 캐시 속성 이름은 워크플로 내에서 고유해야 합니다. 각 작업은 FileCaching에서 최대 5개의 항목을 가질 수 있습니다.

해당 UI: 구성 탭/파일 캐싱 - 선택 사항/캐시 추가/키

Path

(*action-name*/Caching/FileCaching/*key-name-1*/Path)

(선택 사항)

캐시의 관련 경로를 지정합니다.

해당 UI: 구성 탭/파일 캐싱 - 선택 사항/캐시 추가/경로

RestoreKeys

(*action-name*/Caching/FileCaching/*key-name-1*/RestoreKeys)

(선택 사항)

기본 캐시 속성을 찾을 수 없을 때 대체로 사용할 복원 키를 지정합니다. 복원 키 이름은 워크플로 내에서 고유해야 합니다. 각 캐시는 RestoreKeys에서 최대 5개의 항목을 가질 수 있습니다.

해당 UI: 구성 탭/파일 캐싱 - 선택 사항/캐시 추가/복원 키 - 선택 사항

Inputs

(*action-name*/Inputs)

(선택 사항)

이 Inputs 섹션에서는 워크플로 실행 중에 작업에 필요한 데이터를 정의합니다.

Note

빌드 작업 또는 테스트 작업당 최대 4개의 입력(소스 1개 및 아티팩트 3개)이 허용됩니다. 변수는 이 합계에 포함되지 않습니다.

서로 다른 입력(소스 및 아티팩트)에 있는 파일을 참조해야 하는 경우 소스 입력은 기본 입력이고 아티팩트는 보조 입력입니다. 보조 입력의 파일에 대한 참조는 특수 접두사를 사용하여 기본 입력에서 파일을 구분합니다. 자세한 내용은 [예시: 여러 아티팩트에서 파일 참조](#)를 참조하세요.

해당 UI: 입력 탭

Sources

(*action-name*/Inputs/Sources)

(선택 사항)

작업에 필요한 소스 리포지토리를 나타내는 레이블을 지정합니다. 현재 지원되는 유일한 레이블은 워크플로 정의 파일이 저장되는 소스 리포지토리를 나타내는 WorkflowSource입니다.

소스를 생략하는 경우 *action-name*/Inputs/Artifacts 아래에 하나 이상의 입력 아티팩트를 지정해야 합니다.

소스에 대한 자세한 내용은 [워크플로에 소스 리포지토리 연결](#) 섹션을 참조하세요.

해당 UI: 없음

Artifacts - input

(*action-name*/Inputs/Artifacts)

(선택 사항)

이 작업에 대한 입력으로 제공하려는 이전 작업의 아티팩트를 지정합니다. 이러한 아티팩트는 이전 작업에서 출력 아티팩트로 이미 정의되어 있어야 합니다.

입력 아티팩트를 지정하지 않으면 *action-name*/Inputs/Sources 아래에 소스 리포지토리를 하나 이상 지정해야 합니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

Note

아티팩트 - 선택적 드롭다운 목록을 사용할 수 없거나(시각적 편집기) YAML(YAML 편집기)을 검증할 때 오류가 발생하는 경우 작업이 하나의 입력만 지원하기 때문일 수 있습니다. 이 경우 소스 입력을 제거해 보세요.

해당 UI: 입력 탭/아티팩트 - 선택 사항

Variables - input

(*action-name*/Inputs/Variables)

(선택 사항)

작업에 사용할 수 있도록 하려는 입력 변수를 정의하는 이름/값 페어의 시퀀스를 지정합니다. 변수 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 변수 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

예시를 비롯한 변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

해당 UI: 입력 탭/변수 - 선택 사항

Outputs

(*action-name*/Outputs)

(선택 사항)

워크플로 실행 중에 작업에 의해 출력되는 데이터를 정의합니다.

해당 UI: 출력 탭

Artifacts - output

(*action-name*/Outputs/Artifacts)

(선택 사항)

작업에서 생성된 아티팩트의 이름을 지정합니다. 아티팩트 이름은 워크플로 내에서 고유해야 하며 영숫자 문자(a-z, A-Z, 0-9) 및 밑줄(_)로 제한됩니다. 공백, 하이픈(-) 및 특수 문자는 허용되지 않습니다. 출력 아티팩트 이름에서 공백, 하이픈 및 기타 특수 문자를 활성화하는 데 따옴표를 사용할 수 없습니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

해당 UI: 출력 탭/아티팩트

Name

(*action-name*/Outputs/Artifacts/Name)

([Artifacts - output](#) 포함 시 필수)

작업에서 생성된 아티팩트의 이름을 지정합니다. 아티팩트 이름은 워크플로 내에서 고유해야 하며 영숫자 문자(a-z, A-Z, 0-9) 및 밑줄(_)로 제한됩니다. 공백, 하이픈(-) 및 특수 문자는 허용되지 않습니다. 출력 아티팩트 이름에서 공백, 하이픈 및 기타 특수 문자를 활성화하는 데 따옴표를 사용할 수 없습니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

해당 UI: 출력 탭/아티팩트/새 출력/빌드 아티팩트 이름

Files

(*action-name*/Outputs/Artifacts/Files)

([Artifacts - output](#) 포함 시 필수)

CodeCatalyst가 작업으로 출력되는 아티팩트에 포함하는 파일을 지정합니다. 이러한 파일은 실행 시 워크플로 작업에 의해 생성되며 소스 리포지토리에서도 사용할 수 있습니다. 파일 경로는 소스 리포지토리 또는 이전 작업의 아티팩트에 상주할 수 있으며 소스 리포지토리 또는 아티팩트 루트와 관련이 있습니다. glob 패턴을 사용하여 경로를 지정할 수 있습니다. 예시:

- 빌드 위치 또는 소스 리포지토리 위치의 루트에 있는 단일 파일을 지정하려면 my-file.jar를 사용합니다..
- 하위 디렉터리에 단일 파일을 지정하려면 directory/my-file.jar 또는 directory/subdirectory/my-file.jar를 사용합니다.

- 모든 파일을 지정하려면 "**/*"를 사용합니다. ** glob 패턴은 임의의 수의 하위 디렉터리와 일치함을 나타냅니다.
- directory라는 디렉터리에 있는 모든 파일 및 디렉터를 지정하려면 "directory/**/*"를 사용합니다. ** glob 패턴은 임의의 수의 하위 디렉터리와 일치함을 나타냅니다.
- directory라는 디렉터리의 모든 파일을 지정하되 해당 하위 디렉터리는 지정하지 않으려면 "directory/*"를 사용합니다.

Note

파일 경로에 별표(*) 또는 기타 특수 문자가 하나 이상 포함된 경우 경로를 큰따옴표("")로 묶습니다. 특수 문자에 대한 자세한 내용은 [구문 지침 및 규칙](#) 섹션을 참조하세요.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

Note

파일 경로에 접두사를 추가하여 찾을 아티팩트 또는 소스를 나타내야 할 수 있습니다. 자세한 내용은 [소스 리포지토리 파일 참조](#) 및 [아티팩트의 파일 참조](#) 섹션을 참조하세요.

해당 UI: 출력 탭/아티팩트/아티팩트 추가/빌드에서 생성된 파일

Variables - output

(*action-name*/Outputs/Variables)

(선택 사항)

후속 작업에서 사용할 수 있도록 작업을 내보낼 변수를 지정합니다.

예시를 비롯한 변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

해당 UI: 출력 탭/변수/변수 추가

variable-name-1

(*action-name*/Outputs/Variables/*variable-name-1*)

(선택 사항)

작업을 내보낼 변수의 이름을 지정합니다. 이 변수는 동일한 작업의 Inputs 또는 Steps 섹션에 이미 정의되어 있어야 합니다.

예시를 비롯한 변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

해당 UI: 출력 탭/변수/변수 추가/이름

AutoDiscoverReports

(*action-name*/Outputs/AutoDiscoverReports)

(선택 사항)

자동 검색 기능의 구성을 정의합니다.

자동 검색을 활성화하면 CodeCatalyst는 작업에 전달된 모든 Inputs와 작업 자체에서 생성된 모든 파일을 검색하여 테스트, 코드 적용 범위 및 소프트웨어 구성 분석(SCA) 보고서를 찾습니다. 발견된 각 보고서에 대해 CodeCatalyst는 이를 CodeCatalyst 보고서로 변환합니다. CodeCatalyst 보고서는 CodeCatalyst 서비스에 완전히 통합되고 CodeCatalyst 콘솔을 통해 보고 조작할 수 있는 보고서입니다.

Note

기본적으로 자동 검색 기능은 모든 파일을 검사합니다. [IncludePaths](#) 또는 [ExcludePaths](#) 속성을 사용하여 검사할 파일을 제한할 수 있습니다.

해당 UI: 출력 탭/보고서/자동 검색 보고서

Enabled

(*action-name*/Outputs/AutoDiscoverReports/Enabled)

(선택 사항)

자동 검색 기능을 활성화하거나 비활성화합니다.

유효한 값은 true 또는 false입니다.

Enabled 생략 시 기본값은 true입니다.

해당 UI: 출력 탭/보고서/자동 검색 보고서

ReportNamePrefix

(*action-name*/Outputs/AutoDiscoverReports/ReportNamePrefix)

([AutoDiscoverReports](#) 포함 및 활성화 시 필수)

연결된 CodeCatalyst 보고서의 이름을 지정하기 위해 CodeCatalyst가 찾는 모든 보고서에 우선하는 접두사를 지정합니다. 예를 들어 접두사를 AutoDiscovered로 지정하고 CodeCatalyst가 두 개의 테스트 보고서, TestSuiteOne.xml 및 TestSuiteTwo.xml를 자동으로 검색하면 연결된 CodeCatalyst 보고서의 이름이 AutoDiscoveredTestSuiteOne 및 AutoDiscoveredTestSuiteTwo로 지정됩니다.

해당 UI: 출력 탭/보고서/접두사 이름

IncludePaths

(*action-name*/Outputs/AutoDiscoverReports/IncludePaths)

또는

(*action-name*/Outputs/Reports/*report-name-1*/IncludePaths)

([AutoDiscoverReports](#) 포함 및 활성화 시 또는 [Reports](#) 포함 시 필수)

원시 보고서를 검색할 때 CodeCatalyst에 포함되는 파일 및 파일 경로를 지정합니다. 예를 들어 "/test/report/*" 지정 시 CodeCatalyst는 작업에서 /test/report/* 디렉터리를 찾는 데 사용되는 전체 [빌드 이미지](#)를 검색합니다. 해당 디렉터리를 찾으면 CodeCatalyst는 해당 디렉터리에서 보고서를 찾습니다.

Note

파일 경로에 별표(*) 또는 기타 특수 문자가 하나 이상 포함된 경우 경로를 큰따옴표("")로 묶습니다. 특수 문자에 대한 자세한 내용은 [구문 지침 및 규칙](#) 섹션을 참조하세요.

이 속성이 생략되면 기본값은 "**/*"입니다. 즉, 검색에 모든 경로의 모든 파일이 포함됩니다.

Note

수동으로 구성된 보고서의 경우 IncludePaths는 단일 파일과 일치하는 글로브 패턴이어야 합니다.

해당 UI:

- 출력 탭/보고서/보고서 자동 검색/경로 포함/제외/경로 포함
- 출력 탭/보고서/보고서 수동 구성/*report-name-1*/포함/경로 제외/경로 포함

ExcludePaths

(*action-name*/Outputs/AutoDiscoverReports/ExcludePaths)

또는

(*action-name*/Outputs/Reports/ *report-name-1*/ExcludePaths)

(선택 사항)

원시 보고서를 검색할 때 CodeCatalyst에서 제외하는 파일 및 파일 경로를 지정합니다. 예를 들어 "/test/my-reports/**/*.*" 지정 시 CodeCatalyst는 /test/my-reports/ 디렉터리의 파일을 검색하지 않습니다. 디렉터리의 모든 파일을 무시하려면 */* glob 패턴을 사용합니다.

Note

파일 경로에 별표(*) 또는 기타 특수 문자가 하나 이상 포함된 경우 경로를 큰따옴표("")로 묶습니다. 특수 문자에 대한 자세한 내용은 [구문 지침 및 규칙](#) 섹션을 참조하세요.

해당 UI:

- 출력 탭/보고서/보고서 자동 검색/경로 포함/제외/경로 제외
- 출력 탭/보고서/보고서 수동 구성/*report-name-1*/경로 포함/제외/경로 제외

SuccessCriteria

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria)

또는

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria)

(선택 사항)

테스트, 코드 적용 범위, 소프트웨어 구성 분석(SCA) 및 정적 분석(SA) 보고서의 성공 기준을 지정합니다.

자세한 내용은 [보고서의 성공 기준 구성](#) 섹션을 참조하세요.

해당 UI: 출력 탭/보고서/성공 기준

PassRate

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/PassRate)

또는

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/PassRate)

(선택 사항)

관련 CodeCatalyst 보고서를 통과로 표시하기 위해 통과해야 하는 테스트 보고서의 테스트 비율을 지정합니다. 유효한 값에는 십진수가 포함됩니다. 예시: 50, 60.5. 통과율 기준은 테스트 보고서에만 적용됩니다. 테스트 보고서에 대한 자세한 내용은 [테스트 보고서](#)의 내용을 참조하세요.

해당 UI: 출력 탭/보고서/성공 기준/통과율

LineCoverage

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/LineCoverage)

또는

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/LineCoverage)

(선택 사항)

연결된 CodeCatalyst 보고서를 통과로 표시하기 위해 다루어야 하는 코드 적용 범위 보고서의 줄 비율을 지정합니다. 유효한 값에는 십진수가 포함됩니다. 예시: 50, 60.5. 라인 적용 범위 기준은 코드 적용

범위 보고서에만 적용됩니다. 코드 적용 범위 보고서에 대한 자세한 내용은 [코드 적용 범위 보고서](#) 섹션을 참조하세요.

해당 UI: 출력 탭/보고서/성공 기준/라인 적용 범위

BranchCoverage

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/BranchCoverage)

또는

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/BranchCoverage)

(선택 사항)

연결된 CodeCatalyst 보고서를 통과로 표시하기 위해 다루어야 하는 코드 적용 범위 보고서의 브랜치 비율을 지정합니다. 유효한 값에는 십진수가 포함됩니다. 예시: 50, 60.5. 브랜치 적용 범위 기준은 코드 적용 범위 보고서에만 적용됩니다. 코드 적용 범위 보고서에 대한 자세한 내용은 [코드 적용 범위 보고서](#) 섹션을 참조하세요.

해당 UI: 출력 탭/보고서/성공 기준/지점 적용 범위

Vulnerabilities

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/Vulnerabilities)

또는

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/Vulnerabilities)

(선택 사항)

SCA 보고서에서 허용되는 취약성의 최대 수와 심각도를 지정하여 관련 CodeCatalyst 보고서를 통과로 표시합니다. 취약성을 지정하려면 다음을 지정해야 합니다.

- 개수에 포함하려는 취약성의 최소 심각도입니다. 최대부터 최소 심각도까지 유효한 값은 CRITICAL, HIGH, MEDIUM, LOW 및 INFORMATIONAL입니다.

예를 들어 HIGH 선택 시 HIGH 및 CRITICAL 취약성이 집계됩니다.

- 허용하려는 지정된 심각도의 최대 취약성 수입니다. 이 수를 초과하면 CodeCatalyst 보고서가 실패로 표시됩니다. 유효한 값은 정수입니다.

취약성 기준은 SCA 보고서에만 적용됩니다. SCA 보고서에 대한 자세한 내용은 [소프트웨어 구성 분석 보고서](#)의 내용을 참조하세요.

최소 심각도를 지정하려면 Severity 속성을 사용합니다. 최대 취약성 수를 지정하려면 Number 속성을 사용합니다.

해당 UI: 출력 탭/보고서/성공 기준/취약성

StaticAnalysisBug

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/StaticAnalysisBug)

또는

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/StaticAnalysisBug)

(선택 사항)

관련 CodeCatalyst 보고서에 대해 SA 보고서에 허용되는 최대 버그 수와 심각도를 전달된 것으로 표시합니다. 버그를 지정하려면 다음을 지정해야 합니다.

- 개수에 포함하려는 버그의 최소 심각도입니다. 최대부터 최소 심각도까지 유효한 값은 CRITICAL, HIGH, MEDIUM, LOW 및 INFORMATIONAL입니다.

예를 들어 HIGH 선택 시 HIGH 및 CRITICAL 버그가 집계됩니다.

- 허용하려는 지정된 심각도의 최대 버그 수입니다. 이 수를 초과하면 CodeCatalyst 보고서가 실패로 표시됩니다. 유효한 값은 정수입니다.

버그 기준은 PyLint 및 ESLint SA 보고서에만 적용됩니다. SA 보고서에 대한 자세한 내용은 [정적 분석 보고서](#)의 내용을 참조하세요.

최소 심각도를 지정하려면 Severity 속성을 사용합니다. 최대 취약성 수를 지정하려면 Number 속성을 사용합니다.

해당 UI: 출력 탭/보고서/성공 기준/버그

StaticAnalysisSecurity

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/StaticAnalysisSecurity)

또는

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/StaticAnalysisSecurity)

(선택 사항)

관련 CodeCatalyst 보고서에 대해 SA 보고서에서 허용되는 보안 취약성의 최대 수와 심각도를 전달된 것으로 표시합니다. 보안 취약성을 지정하려면 다음을 지정해야 합니다.

- 개수에 포함하려는 보안 취약성의 최소 심각도입니다. 최대부터 최소 심각도까지 유효한 값은 CRITICAL, HIGH, MEDIUM, LOW 및 INFORMATIONAL입니다.

예를 들어 HIGH 선택 시 HIGH 및 CRITICAL 취약성이 집계됩니다.

- 허용하려는 지정된 심각도의 최대 보안 취약성 수입니다. 이 수를 초과하면 CodeCatalyst 보고서가 실패로 표시됩니다. 유효한 값은 정수입니다.

보안 취약성 기준은 PyLint 및 ESLint SA 보고서에만 적용됩니다. SA 보고서에 대한 자세한 내용은 [정적 분석 보고서](#)의 내용을 참조하세요.

최소 심각도를 지정하려면 Severity 속성을 사용합니다. 최대 취약성 수를 지정하려면 Number 속성을 사용합니다.

해당 UI: 출력 탭/보고서/성공 기준/보안 취약성

StaticAnalysisQuality

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/StaticAnalysisQuality)

또는

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/StaticAnalysisQuality)

(선택 사항)

관련 CodeCatalyst 보고서에 대해 SA 보고서에서 통과로 표시할 수 있는 품질 문제의 최대 수와 심각도를 지정합니다. 품질 문제를 지정하려면 다음을 지정해야 합니다.

- 개수에 포함하려는 품질 문제의 최소 심각도입니다. 최대부터 최소 심각도까지 유효한 값은 CRITICAL, HIGH, MEDIUM, LOW 및 INFORMATIONAL입니다.

예를 들어 HIGH 선택 시 HIGH 및 CRITICAL 품질 문제가 집계됩니다.

- 허용하려는 지정된 심각도의 최대 품질 문제 수입니다. 이 수를 초과하면 CodeCatalyst 보고서가 실패로 표시됩니다. 유효한 값은 정수입니다.

품질 문제 기준은 PyLint 및 ESLint SA 보고서에만 적용됩니다. SA 보고서에 대한 자세한 내용은 [정적 분석 보고서](#)의 내용을 참조하세요.

최소 심각도를 지정하려면 Severity 속성을 사용합니다. 최대 취약성 수를 지정하려면 Number 속성을 사용합니다.

해당 UI: 출력 탭/보고서/성공 기준/품질 문제

StaticAnalysisFinding

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/StaticAnalysisFinding)

또는

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/StaticAnalysisFinding)

(선택 사항)

관련 CodeCatalyst 보고서에 대해 SA 보고서에서 통과로 표시할 수 있는 조사 결과의 최대 수와 심각도를 지정합니다. 조사 결과를 지정하려면 다음을 지정해야 합니다.

- 개수에 포함하려는 조사 결과의 최소 심각도입니다. 최대부터 최소 심각도까지 유효한 값은 CRITICAL, HIGH, MEDIUM, LOW 및 INFORMATIONAL입니다.

예를 들어 HIGH 선택 시 HIGH 및 CRITICAL 조사 결과가 집계됩니다.

- 허용하려는 지정된 심각도의 최대 조사 결과 수입니다. 이 수를 초과하면 CodeCatalyst 보고서가 실패로 표시됩니다. 유효한 값은 정수입니다.

조사 결과는 SARIF SA 보고서에만 적용됩니다. SA 보고서에 대한 자세한 내용은 [정적 분석 보고서](#)의 내용을 참조하세요.

최소 심각도를 지정하려면 Severity 속성을 사용합니다. 최대 취약성 수를 지정하려면 Number 속성을 사용합니다.

해당 UI: 출력 탭/보고서/성공 기준/조사 결과

Reports

(*action-name*/Outputs/Reports)

(선택 사항)

테스트 보고서의 구성을 지정하는 섹션입니다.

해당 UI: 출력 탭/보고서

report-name-1

(*action-name*/Outputs/Reports/report-name-1)

([Reports](#) 포함 시 필수)

원시 보고서에서 생성될 CodeCatalyst 보고서에 부여할 이름입니다.

해당 UI: 출력 탭/보고서 수동 구성/보고서 이름

Format

(*action-name*/Outputs/Reports/*report-name-1*/Format)

([Reports](#) 포함 시 필수)

보고서에 사용할 파일 형식을 지정합니다. 가능한 값은 다음과 같습니다.

- 테스트 보고서의 경우:
 - Cucumber JSON에서 Cucumber(시각 편집기) 또는 CUCUMBERJSON(YAML 편집기)를 지정합니다.
 - JUnit XML에 JUnit(시각 편집기) 또는 JUNITXML(YAML 편집기)를 지정합니다.
 - NUnit XML에 NUnit(시각 편집기) 또는 NUNITXML(YAML 편집기)를 지정합니다.
 - NUnit 3 XML에서 NUnit3(시각 편집기) 또는 NUNIT3XML(YAML 편집기)를 지정합니다.
 - Visual Studio TRX에서 Visual Studio TRX(시각 편집기) 또는 VISUALSTUDIOTRX(YAML 편집기)를 지정합니다.
 - TestNG XML에서 TestNG(시각 편집기) 또는 TESTNGXML(YAML 편집기)를 지정합니다.
- 코드 적용 범위 보고서의 경우:
 - Clover XML에서 Clover(시각 편집기) 또는 CLOVERXML(YAML 편집기)를 지정합니다.
 - Cobertura XML에서 Cobertura(시각 편집기) 또는 COBERTURAXML(YAML 편집기)를 지정합니다.
 - JaCoCo XML의 경우 JaCoCo(시각 편집기) 또는 JACOCOXML(YAML 편집기)를 지정합니다.
 - [simplecov-json](#)이 아닌 [simplecov](#)에서 생성한 SimpleCov JSON의 경우 Simplecov(시각 편집기) 또는 SIMPLECOV(YAML 편집기)를 지정합니다.
- 소프트웨어 구성 분석(SCA) 보고서의 경우:

- SARIF에서 SARIF(시각 편집기) 또는 SARIFSCA(YAML 편집기)를 지정합니다.

해당 UI: 출력 탭/보고서/보고서 수동 구성/보고서 추가/구성/*report-name-1*/보고서 유형 및 보고서 형식

Configuration

(*action-name*/Configuration)

(필수) 작업의 구성 속성을 정의할 수 있는 섹션입니다.

해당 UI: 구성 탭

Container

(*action-name*/Configuration/Container)

(선택 사항)

작업이 처리를 완료하는 데 사용하는 Docker 이미지 또는 컨테이너를 지정합니다. CodeCatalyst 와 함께 제공되는 [활성 이미지](#) 중 하나를 지정하거나 자체 이미지를 사용할 수 있습니다. 자체 이미지를 사용하도록 선택하면 Amazon ECR, Docker Hub 또는 다른 레지스트리에 상주할 수 있습니다. Docker 이미지를 지정하지 않으면 작업이 처리에 활성 이미지 중 하나를 사용합니다. 기본적으로 사용되는 활성 이미지에 대한 자세한 내용은 [활성 이미지](#) 섹션을 참조하세요.

자체 Docker 이미지 지정에 대한 자세한 내용은 [작업에 사용자 지정 런타임 환경 Docker 이미지 할당](#) 섹션을 참조하세요.

해당 UI: 런타임 환경 Docker 이미지 - 선택 사항

Registry

(*action-name*/Configuration/Container/Registry)

(Container 포함 시 필수)

이미지가 저장되는 레지스트리를 지정합니다. 유효한 값으로는 다음이 포함됩니다.

- CODECATALYST (YAML 편집기)

이미지는 CodeCatalyst 레지스트리에 저장됩니다.

- Docker Hub(시각 편집기) 또는 DockerHub(YAML 편집기)

이미지는 Docker Hub 이미지 레지스트리에 저장됩니다.

- 기타 레지스트리(시각 편집기) 또는 Other(YAML 편집기)

이미지는 사용자 지정 이미지 레지스트리에 저장됩니다. 공개적으로 사용 가능한 레지스트리를 사용할 수 있습니다.

- Amazon Elastic Container Registry(시각 편집기) 또는 ECR(YAML 편집기)

이미지는 Amazon Elastic Container Registry 이미지 리포지토리에 저장됩니다. Amazon ECR 리포지토리에서 이미지를 사용하려면 이 작업에서 Amazon ECR에 대한 액세스 권한이 필요합니다. 이 액세스를 활성화하려면 다음 권한과 사용자 지정 신뢰 정책을 포함하는 [IAM 역할](#)을 생성해야 합니다. (원하는 경우 권한 및 정책을 포함하도록 기존 역할을 수정할 수 있습니다.)

IAM 역할에는 역할 정책의 다음 권한이 포함되어야 합니다.

- `ecr:BatchCheckLayerAvailability`
- `ecr:BatchGetImage`
- `ecr:GetAuthorizationToken`
- `ecr:GetDownloadUrlForLayer`

IAM 역할에는 다음과 같은 사용자 지정 신뢰 정책이 포함되어야 합니다.

IAM 역할 생성에 대한 자세한 정보는 IAM 사용자 설명서의 [사용자 지정 신뢰 정책을 사용하여 역할 생성\(콘솔\)](#)을 참조하세요.

역할을 생성한 후에는 환경을 통해 작업에 할당해야 합니다. 자세한 내용은 [작업과 환경 연결](#) 섹션을 참조하세요.

해당 UI: Amazon Elastic Container Registry, Docker Hub 및 기타 레지스트리 옵션

Image

(*action-name*/Configuration/Container/Image)

(Container 포함 시 필수)

다음 중 하나를 지정하세요.

- CODECATALYST 레지스트리를 사용하는 경우 이미지를 다음 [활성 이미지](#) 중 하나로 설정합니다.

- CodeCatalystLinux_x86_64:2024_03
 - CodeCatalystLinux_x86_64:2022_11
 - CodeCatalystLinux_Arm64:2024_03
 - CodeCatalystLinux_Arm64:2022_11
 - CodeCatalystLinuxLambda_x86_64:2024_03
 - CodeCatalystLinuxLambda_x86_64:2022_11
 - CodeCatalystLinuxLambda_Arm64:2024_03
 - CodeCatalystLinuxLambda_Arm64:2022_11
 - CodeCatalystWindows_x86_64:2022_11
- Docker Hub 레지스트리를 사용하는 경우 이미지를 Docker Hub 이미지 이름과 선택적 태그로 설정합니다.

예시: postgres:latest

- Amazon ECR 레지스트리를 사용하는 경우 이미지를 Amazon ECR 레지스트리 URI로 설정합니다.

예시: 111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-ecs-image-repo

- 사용자 지정 레지스트리를 사용하는 경우 이미지를 사용자 지정 레지스트리에서 예상되는 값으로 설정합니다.

해당 UI: 런타임 환경 Docker 이미지(레지스터가 인 경우CODECATALYST), Docker Hub 이미지(레지스터가 Docker Hub인 경우), ECR 이미지 URL(레지스터가 Amazon Elastic Container Registry인 경우) 및 이미지 URL(레지스터가 기타 레지스트리인 경우).

Steps

(*action-name*/Configuration/Steps)

(필수)

빌드 도구를 설치, 구성 및 실행하기 위해 작업 중에 실행할 셸 명령을 지정합니다.

다음은 npm 프로젝트를 구축하는 방법의 예입니다.

```
Steps:
- Run: npm install
```

```
- Run: npm run build
```

다음은 파일 경로를 지정하는 방법의 예입니다.

Steps:

```
- Run: cd $ACTION_BUILD_SOURCE_PATH_WorkflowSource/app && cat file2.txt
- Run: cd $ACTION_BUILD_SOURCE_PATH_MyBuildArtifact/build-output/ && cat file.txt
```

파일 경로 지정에 대한 자세한 내용은 [소스 리포지토리 파일 참조](#) 및 [아티팩트의 파일 참조](#)를 참조하세요.

해당 UI: 구성 탭/셸 명령

Packages

(*action-name*/Configuration/Packages)

(선택 사항)

작업이 종속성을 해결하는 데 사용하는 패키지 리포지토리를 지정할 수 있는 섹션입니다. 패키지를 사용하면 애플리케이션 개발에 사용되는 소프트웨어 패키지를 안전하게 저장하고 공유할 수 있습니다.

패키지에 대한 자세한 내용은 [CodeCatalyst에서 소프트웨어 패키지 게시 및 공유](#) 섹션을 참조하세요.

해당 UI: 구성 탭/패키지

NpmConfiguration

(*action-name*/Configuration/Packages/NpmConfiguration)

([Packages](#) 포함 시 필수)

npm 패키지 형식의 구성을 정의하는 섹션입니다. 이 구성은 워크플로 실행 중 작업에서 사용됩니다.

npm 패키지 구성에 대한 자세한 내용은 [npm 사용](#) 섹션을 참조하세요.

해당 UI: 구성 탭/패키지/구성 추가/npm

PackageRegistries

(*action-name*/Configuration/Packages/NpmConfiguration/PackageRegistries)

([Packages](#) 포함 시 필수)

일련의 패키지 리포지토리의 구성 속성을 정의할 수 있는 섹션입니다.

해당 UI: 구성 탭/패키지/구성 추가/npm/패키지 리포지토리 추가

PackagesRepository

(*action-name*/Configuration/Packages/NpmConfiguration/PackageRegistries/PackagesRepository)

([Packages](#) 포함 시 필수)

작업을 사용할 CodeCatalyst 패키지 리포지토리의 이름을 지정합니다.

여러 기본 리포지토리를 지정하는 경우 마지막 리포지토리가 우선합니다.

패키지 리포지토리에 대한 자세한 내용은 [패키지 리포지토리](#) 섹션을 참조하세요.

해당 UI: 구성 탭/패키지/구성 추가/npm/패키지 리포지토리/패키지 리포지토리 추가

Scopes

(*action-name*/Configuration/Packages/NpmConfiguration/PackageRegistries/Scopes)

(선택 사항)

패키지 레지스트리에서 정의할 범위 시퀀스를 지정합니다. 범위를 정의할 때 지정된 패키지 리포지토리는 나열된 모든 범위에 대한 레지스트리로 구성됩니다. 해당 범위의 패키지가 npm 클라이언트를 통해 요청되면 기본값 대신 해당 리포지토리를 사용합니다. 각 범위 이름 앞에는 '@' 접두사를 붙여야 합니다.

범위 재정의의 포함하면 마지막 리포지토리가 우선합니다.

Scopes 생략 시 지정된 패키지 리포지토리가 작업에 사용되는 모든 패키지의 기본 레지스트리로 구성됩니다.

범위에 대한 자세한 내용은 [패키지 네임스페이스](#) 및 [범위 지정 패키지](#) 섹션을 참조하세요.

해당 UI: 구성 탭/패키지/구성 추가/npm/패키지 리포지토리 추가/범위 - 선택 사항

ExportAuthorizationToken

(*action-name*/Configuration/Packages/ExportAuthorizationToken)

(선택 사항)

권한 부여 토큰 내보내기 기능을 활성화 또는 비활성화합니다. 활성화된 경우 내보낸 권한 부여 토큰을 사용하여 CodeCatalyst 패키지 리포지토리로 인증하도록 패키지 관리자를 수동으로 구성할 수 있습니다. 토큰을 작업에서 참조할 수 있는 환경 변수로 사용할 수 있습니다.

유효한 값은 true 또는 false입니다.

ExportAuthorizationToken 생략 시 기본값은 false입니다.

권한 토큰 내보내기에 대한 자세한 내용은 [워크플로 작업에서 권한 부여 토큰 사용](#) 섹션을 참조하세요.

해당 UI: 구성 탭/패키지/권한 토큰 내보내기

워크플로를 사용한 테스트

CodeCatalyst에서 빌드 및 테스트와 같은 다양한 워크플로 작업의 일부로 테스트를 실행할 수 있습니다. 이러한 워크플로 작업은 모두 품질 보고서를 생성할 수 있습니다. 테스트 작업은 테스트, 코드 적용 범위, 소프트웨어 구성 분석 및 정적 분석 보고서를 생성하는 워크플로 작업입니다. 이러한 보고서는 CodeCatalyst 콘솔에 표시됩니다.

주제

- [품질 보고서 유형](#)
- [테스트 작업 추가](#)
- [테스트 작업 결과 보기](#)
- [작업에서 실패한 테스트 건너뛰기](#)
- [universal-test-runner와 통합](#)
- [작업에서 품질 보고서 구성](#)
- [테스트 모범 사례](#)
- [지원되는 SARIF 속성](#)

품질 보고서 유형

Amazon CodeCatalyst 테스트 작업은 다음과 같은 유형의 품질 보고서를 지원합니다. YAML에서 이러한 보고서를 포맷하는 방법에 대한 예시는 [품질 보고서 YAML 예시](#) 섹션을 참조하세요.

주제

- [테스트 보고서](#)

- [코드 적용 범위 보고서](#)
- [소프트웨어 구성 분석 보고서](#)
- [정적 분석 보고서](#)

테스트 보고서

CodeCatalyst에서 빌드 중에 실행되는 단위 테스트, 통합 테스트 및 시스템 테스트를 구성할 수 있습니다. 그런 다음 CodeCatalyst는 테스트 결과를 포함하는 보고서를 생성할 수 있습니다.

테스트 보고서를 사용하여 테스트 문제를 해결할 수 있습니다. 빌드 프로젝트의 여러 빌드에 많은 테스트 보고서가 있는 경우 테스트 보고서를 사용하여 추세와 테스트 및 실패율을 보고 빌드를 쉽게 최적화할 수 있습니다.

다음 테스트 보고서 파일 형식을 사용할 수 있습니다.

- Cucumber JSON(.json)
- JUnit XML(.xml)
- NUnit XML(.xml)
- NUnit3 XML(.xml)
- TestNG XML(.xml)
- Visual Studio TRX (.trx, .xml)

코드 적용 범위 보고서

CodeCatalyst에서는 테스트에 대한 코드 적용 범위 보고서를 생성할 수 있습니다. CodeCatalyst는 다음과 같은 코드 적용 범위 지표를 제공합니다.

행 범위

행 범위는 테스트에서 다루는 명령문 수를 측정합니다. 문은 설명이 포함되지 않은 단일 명령어입니다.

$$\text{line coverage} = (\text{total lines covered}) / (\text{total number of lines})$$

브랜치 적용 범위

또는 문과 같은 제어 구조의 가능한 모든 브랜치 중 테스트에서 다루는 분기 수(if 또는 case)를 측정합니다.

```
branch coverage = (total branches covered)/(total number of branches)
```

다음 코드 적용 범위 보고서 파일 형식이 지원됩니다.

- JaCoCo XML(.xml)
- SimpleCov JSON([simplecov-json](#), .json이 아닌 [simplecov](#)에서 생성됨)
- Clover XML(버전 3, .xml)
- Cobertura XML(.xml)
- LCOV(.info)

소프트웨어 구성 분석 보고서

CodeCatalyst에서 소프트웨어 구성 분석(SCA) 도구를 사용하여 애플리케이션의 구성 요소를 분석하고 알려진 보안 취약성을 확인할 수 있습니다. 심각도가 다양한 취약성과 이를 수정하는 방법을 자세히 설명하는 SARIF 보고서를 검색하고 구문 분석할 수 있습니다. 최대부터 최소 심각도까지 유효한 심각도 값은 CRITICAL, HIGH, MEDIUM, LOW, INFORMATIONAL입니다.

지원되는 SCA 보고서 파일 형식은 다음과 같습니다.

- SARIF(.sarif, .json)

정적 분석 보고서

정적 분석(SA) 보고서를 사용하여 소스 수준 코드 결함을 식별할 수 있습니다. CodeCatalyst에서 SA 보고서를 생성하여 코드를 배포하기 전에 코드 문제를 해결할 수 있습니다. 이러한 문제에는 버그, 보안 취약성, 품질 문제 및 기타 취약성이 포함됩니다. 최대부터 최소 심각도까지 유효한 심각도 값은 CRITICAL, HIGH, MEDIUM, LOW 및 INFORMATIONAL입니다.

CodeCatalyst는 다음과 같은 SA 지표를 제공합니다.

버그

소스 코드에서 발견된 가능한 버그 수를 식별합니다. 이러한 버그에는 메모리 안전성에 관한 문제가 포함될 수 있습니다. 다음은 버그의 예시입니다.

```
// The while loop will inadvertently index into array x out-of-bounds
int x[64];
```

```
while (int n = 0; n <= 64; n++) {  
    x[n] = 0;  
}
```

보안 취약성

소스 코드에서 발견된 여러 가지 가능한 보안 취약성을 식별합니다. 이러한 보안 취약성에는 보안 암호 토큰을 일반 텍스트로 저장하는 등의 문제가 포함될 수 있습니다.

품질 문제

소스 코드에서 발견된 여러 가지 가능한 품질 문제를 식별합니다. 이러한 품질 문제에는 스타일 규칙과 관련된 문제가 포함될 수 있습니다. 다음은 품질 문제의 예입니다.

```
// The function name doesn't adhere to the style convention of camelCase  
int SUBTRACT(int x, int y) {  
    return x-y  
}
```

기타 취약성

소스 코드에서 발견된 여러 가지 가능한 보안 취약성을 식별합니다.

CodeCatalyst에서 지원되는 코드 적용 범위 보고서 파일 형식은 다음과 같습니다.

- PyLint(.py)
- ESLint(.js, .jsx, .ts, .tsx)
- SARIF(.sarif, .json)

테스트 작업 추가

다음 절차에 따라 CodeCatalyst 워크플로에 테스트 작업을 추가합니다.

Visual

시각적 편집기를 사용하여 테스트 작업 추가

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.

3. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
4. 편집을 선택합니다.
5. 비주얼을 선택합니다.
6. 작업을 선택합니다.
7. 작업에서 테스트를 선택합니다.
8. 입력 및 구성 탭에서 필요에 따라 필드를 작성합니다. 각 필드의 설명은 [빌드 및 테스트 작업 YAML](#) 섹션을 참조하세요. 이 참조는 YAML 및 시각적 편집기 모두에 나타나는 각 필드(및 해당 YAML 속성 값)에 대한 자세한 정보를 제공합니다.
9. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
10. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 빌드 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
3. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
4. 편집을 선택합니다.
5. YAML을 선택합니다.
6. 작업을 선택합니다.
7. 작업에서 테스트를 선택합니다.
8. 필요에 따라 YAML 코드의 속성을 수정합니다. 사용 가능한 각 속성에 대한 설명은 [빌드 및 테스트 작업 YAML](#)에서 볼 수 있습니다.
9. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
10. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

작업 정의 테스트

테스트 작업은 워크플로 정의 파일 내의 YAML 속성 집합으로 정의됩니다. 이러한 속성에 대한 자세한 내용은 [워크플로 YAML 정의의 빌드 및 테스트 작업 YAML](#) 섹션을 참조하세요.

테스트 작업 결과 보기

다음 지침에 따라 생성된 로그, 보고서 및 변수를 포함한 테스트 작업의 결과를 확인합니다.

테스트 작업 결과 보기

1. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
2. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
3. 워크플로 다이어그램에서 테스트 작업의 이름(예시: 테스트)을 선택합니다.
4. 작업에서 생성된 로그를 보려면 로그를 선택합니다. 다양한 작업 단계의 로그가 표시됩니다. 필요에 따라 로그를 확장하거나 축소할 수 있습니다.
5. 테스트 작업에서 생성된 테스트 보고서를 보려면 보고서를 선택하거나 탐색 창에서 보고서를 선택합니다. 자세한 내용은 [품질 보고서 유형](#) 섹션을 참조하세요.
6. 빌드 작업에 사용되는 구성을 보려면 구성을 선택합니다. 자세한 내용은 [테스트 작업 추가](#) 섹션을 참조하세요.
7. 테스트 작업에 사용되는 변수를 보려면 변수를 선택합니다. 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

작업에서 실패한 테스트 건너뛰기

작업에 테스트 명령이 두 개 이상 있는 경우 이전 명령이 실패하더라도 작업의 후속 테스트 명령이 실행되도록 허용할 수 있습니다. 예를 들어, 다음 명령에서는 test1이 실패하더라도 test2가 항상 실행되도록 할 수 있습니다.

Steps:

- Run: npm install
- Run: npm run test1
- Run: npm run test2

일반적으로 단계가 오류를 반환하면 Amazon CodeCatalyst는 워크플로 작업을 중지하고 실패로 표시합니다. 오류 출력을 null로 리디렉션하여 작업 단계가 계속 실행되도록 허용할 수 있습니다. 명령에 `2>/dev/null`을 추가하면 이 작업을 수행할 수 있습니다. 이렇게 수정하면 앞의 예시는 다음과 같이 표시됩니다.

Steps:

- Run: npm install

```
- Run: npm run test1 2>/dev/null
- Run: npm run test2
```

두 번째 코드 조각에서는 `npm install` 명령의 상태가 그대로 유지되지만 `npm run test1` 명령에서 반환된 오류는 무시됩니다. 결과적으로 `npm run test2` 명령이 실행됩니다. 이렇게 하면 오류 발생 여부에 관계없이 두 보고서를 한 번에 볼 수 있습니다.

universal-test-runner와 통합

테스트 작업은 오픈 소스 명령줄 도구인 `universal-test-runner`와 통합됩니다. `universal-test-runner`는 [테스트 실행 프로토콜](#)을 사용하여 주어진 프레임워크에서 모든 언어에 대한 테스트를 실행합니다. `universal-test-runner`는 다음 프레임워크를 지원합니다.

- [Gradle](#)
- [Jest](#)
- [Maven](#)
- [pytest](#)
- [.NET](#)

`universal-test-runner`는 테스트 동작을 위해 큐레이팅된 이미지에만 설치됩니다. 용자 지정 Docker Hub 또는 Amazon ECR을 사용하도록 테스트 동작을 구성하는 경우 고급 테스트 특성을 사용하려면 `universal-test-runner`를 수동으로 설치해야 합니다. 이렇게 하려면 이미지에 Node.js(14 이상)를 설치한 후 다음 셸 명령 - `Run: npm install -g @aws/universal-test-runner`을 사용하여 npm을 통해 `universal-test-runner`를 설치합니다. 셸 명령을 통해 컨테이너에 Node.js를 설치하는 방법에 대한 자세한 내용은 [Installing and Updating Node Version Manager](#)를 참조하세요.

`universal-test-runner`에 대한 자세한 내용은 [What is universal-test-runner?](#) 섹션을 참조하세요.

Visual

시각적 편집기에서 `universal-test-runner` 사용

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
3. 워크플로의 이름을 선택합니다.
4. 편집을 선택합니다.
5. 비주얼을 선택합니다.

6. 작업을 선택합니다.
7. 작업에서 테스트를 선택합니다.
8. 구성 탭에서 지원되는 프레임워크를 선택하여 샘플 코드를 업데이트하여 셸 명령 필드를 완료합니다. 예를 들어 지원되는 프레임워크를 사용하려면 다음과 유사한 Run 명령을 사용합니다.

```
- Run: run-tests <framework>
```

원하는 프레임워크가 지원되지 않는 경우 사용자 지정 어댑터 또는 러너를 제공하는 것이 좋습니다. 셸 명령 필드에 대한 설명은 [Steps](#) 섹션을 참조하세요.

9. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
10. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기에서 universal-test-runner 사용

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
3. 워크플로의 이름을 선택합니다.
4. 편집을 선택합니다.
5. YAML을 선택합니다.
6. 작업을 선택합니다.
7. 작업에서 테스트를 선택합니다.
8. 필요에 따라 YAML 코드를 수정합니다. 예를 들어 지원되는 프레임워크를 사용하려면 다음과 유사한 Run 명령을 사용합니다.

```
Configuration:
Steps:
  - Run: run-tests <framework>
```

원하는 프레임워크가 지원되지 않는 경우 사용자 지정 어댑터 또는 러너를 제공하는 것이 좋습니다. Steps 속성에 대한 설명은 [Steps](#) 섹션을 참조하세요.

9. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
10. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

작업에서 품질 보고서 구성

이 섹션에서는 작업에서 품질 보고서를 구성하는 방법을 설명합니다.

주제

- [자동 검색 및 수동 보고서](#)
- [보고서의 성공 기준 구성](#)
- [품질 보고서 YAML 예시](#)

자동 검색 및 수동 보고서

자동 검색이 활성화되면 CodeCatalyst는 작업에 전달된 모든 입력과 작업 자체에서 생성된 모든 파일을 검색하여 테스트, 코드 적용 범위, 소프트웨어 구성 분석(SCA) 및 정적 분석(SA) 보고서를 찾습니다. CodeCatalyst에서 이러한 각 보고서를 보고 조작할 수 있습니다.

생성되는 보고서를 수동으로 구성할 수도 있습니다. 생성하려는 보고서 유형과 파일 형식을 지정할 수 있습니다. 자세한 내용은 [품질 보고서 유형](#) 섹션을 참조하세요.

보고서의 성공 기준 구성

테스트, 코드 적용 범위, 소프트웨어 구성 분석(SCA) 또는 정적 분석(SA) 보고서의 성공 기준을 결정하는 값을 설정할 수 있습니다.

성공 기준은 보고서가 통과 또는 실패하는지 여부를 결정하는 임계값입니다. CodeCatalyst는 먼저 테스트, 코드 적용 범위, SCA 또는 SA 보고서일 수 있는 보고서를 생성한 다음 생성된 보고서에 성공 기준을 적용합니다. 그런 다음 성공 기준이 충족되었는지 여부와 어느 정도까지 충족되었는지 보여줍니다. 지정된 성공 기준을 충족하지 않는 보고서가 있으면 성공 기준을 지정한 CodeCatalyst 작업이 실패합니다.

예를 들어 SCA 보고서의 성공 기준을 설정할 때 가장 심각함에서 가장 덜 심각함까지의 유효한 취약성 값은 CRITICAL, HIGH, MEDIUM, LOW, INFORMATIONAL입니다. HIGH 심각도에서 하나의 취약성을 스캔하도록 기준을 설정하면 HIGH 심각도에서 하나 이상의 취약성이 있거나 심각도에서 취약성이 없지만 HIGH 심각도에서 하나의 취약성과 같이 CRITICAL 심각도가 더 높은 수준에서 하나 이상의 취약성이 있는 경우 보고서가 실패합니다.

성공 기준을 지정하지 않으면 다음을 수행합니다.

- 원시 보고서를 기반으로 생성된 CodeCatalyst 보고서에는 성공 기준이 표시되지 않습니다.

- 성공 기준은 연결된 워크플로 작업이 통과 또는 실패하는지 여부를 결정하는 데 사용되지 않습니다.

Visual

성공 기준 구성

1. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
2. 보고서를 생성하는 작업이 포함된 워크플로를 선택합니다. 성공 기준을 적용하려는 보고서입니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
3. 편집을 선택합니다.
4. 비주얼을 선택합니다.
5. 워크플로 다이어그램에서 CodeCatalyst 보고서를 생성하도록 구성한 작업을 선택합니다.
6. 출력 탭을 선택합니다.
7. 보고서 자동 검색 또는 보고서 수동 구성에서 성공 기준을 선택합니다.

성공 기준이 나타납니다. 이전 선택에 따라 다음 옵션 중 일부 또는 전부가 표시될 수 있습니다.

통과율

관련 CodeCatalyst 보고서를 통과로 표시하기 위해 통과해야 하는 테스트 보고서의 테스트 비율을 지정합니다. 유효한 값에는 십진수가 포함됩니다. 예시: 50, 60.5. 통과율 기준은 테스트 보고서에만 적용됩니다. 테스트 보고서에 대한 자세한 내용은 [테스트 보고서](#)의 내용을 참조하세요.

행 범위

연결된 CodeCatalyst 보고서를 통과로 표시하기 위해 다루어야 하는 코드 적용 범위 보고서의 줄 비율을 지정합니다. 유효한 값에는 십진수가 포함됩니다. 예시: 50, 60.5. 라인 적용 범위 기준은 코드 적용 범위 보고서에만 적용됩니다. 코드 적용 범위 보고서에 대한 자세한 내용은 [코드 적용 범위 보고서](#) 섹션을 참조하세요.

브랜치 적용 범위

연결된 CodeCatalyst 보고서를 통과로 표시하기 위해 다루어야 하는 코드 적용 범위 보고서의 브랜치 비율을 지정합니다. 유효한 값에는 십진수가 포함됩니다. 예시: 50, 60.5. 브랜치 적용 범위 기준은 코드 적용 범위 보고서에만 적용됩니다. 코드 적용 범위 보고서에 대한 자세한 내용은 [코드 적용 범위 보고서](#) 섹션을 참조하세요.

취약성(SCA)

SCA 보고서에서 허용되는 취약성의 최대 수와 심각도를 지정하여 관련 CodeCatalyst 보고서를 통과로 표시합니다. 취약성을 지정하려면 다음을 지정해야 합니다.

- 개수에 포함하려는 취약성의 최소 심각도입니다. 최대부터 최소 심각도까지 유효한 값은 CRITICAL, HIGH, MEDIUM, LOW 및 INFORMATIONAL입니다.

예를 들어 HIGH 선택 시 HIGH 및 CRITICAL 취약성이 집계됩니다.

- 허용하려는 지정된 심각도의 최대 취약성 수입니다. 이 수를 초과하면 CodeCatalyst 보고서가 실패로 표시됩니다. 유효한 값은 정수입니다.

취약성 기준은 SCA 보고서에만 적용됩니다. SCA 보고서에 대한 자세한 내용은 [소프트웨어 구성 분석 보고서](#)의 내용을 참조하세요.

bugs

관련 CodeCatalyst 보고서에 대해 SA 보고서에 허용되는 최대 버그 수와 심각도를 전달된 것으로 표시합니다. 버그를 지정하려면 다음을 지정해야 합니다.

- 개수에 포함하려는 버그의 최소 심각도입니다. 최대부터 최소 심각도까지 유효한 값은 CRITICAL, HIGH, MEDIUM, LOW 및 INFORMATIONAL입니다.

예를 들어 HIGH 선택 시 HIGH 및 CRITICAL 버그가 집계됩니다.

- 허용하려는 지정된 심각도의 최대 버그 수입니다. 이 수를 초과하면 CodeCatalyst 보고서가 실패로 표시됩니다. 유효한 값은 정수입니다.

버그 기준은 PyLint 및 ESLint SA 보고서에만 적용됩니다. SA 보고서에 대한 자세한 내용은 [정적 분석 보고서](#)의 내용을 참조하세요.

보안 취약성

관련 CodeCatalyst 보고서에 대해 SA 보고서에서 허용되는 보안 취약성의 최대 수와 심각도를 전달된 것으로 표시합니다. 보안 취약성을 지정하려면 다음을 지정해야 합니다.

- 개수에 포함하려는 보안 취약성의 최소 심각도입니다. 최대부터 최소 심각도까지 유효한 값은 CRITICAL, HIGH, MEDIUM, LOW 및 INFORMATIONAL입니다.

예를 들어 HIGH 선택 시 HIGH 및 CRITICAL 취약성이 집계됩니다.

- 허용하려는 지정된 심각도의 최대 보안 취약성 수입니다. 이 수를 초과하면 CodeCatalyst 보고서가 실패로 표시됩니다. 유효한 값은 정수입니다.

보안 취약성 기준은 PyLint 및 ESLint SA 보고서에만 적용됩니다. SA 보고서에 대한 자세한 내용은 [정적 분석 보고서](#)의 내용을 참조하세요.

품질 문제

관련 CodeCatalyst 보고서에 대해 SA 보고서에서 통과로 표시할 수 있는 품질 문제의 최대 수와 심각도를 지정합니다. 품질 문제를 지정하려면 다음을 지정해야 합니다.

- 개수에 포함하려는 품질 문제의 최소 심각도입니다. 최대부터 최소 심각도까지 유효한 값은 CRITICAL, HIGH, MEDIUM, LOW 및 INFORMATIONAL입니다.

예를 들어 HIGH 선택 시 HIGH 및 CRITICAL 품질 문제가 집계됩니다.

- 허용하려는 지정된 심각도의 최대 품질 문제 수입니다. 이 수를 초과하면 CodeCatalyst 보고서가 실패로 표시됩니다. 유효한 값은 정수입니다.

품질 문제 기준은 PyLint 및 ESLint SA 보고서에만 적용됩니다. SA 보고서에 대한 자세한 내용은 [정적 분석 보고서](#)의 내용을 참조하세요.

8. 커밋을 선택합니다.
9. 워크플로를 실행하여 CodeCatalyst가 원시 보고서에 성공 기준을 적용하도록 하고, 성공 기준 정보가 포함된 관련 CodeCatalyst 보고서를 다시 생성합니다. 자세한 내용은 [워크플로 수동 실행 시작](#) 섹션을 참조하세요.

YAML

성공 기준 구성

1. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
2. 보고서를 생성하는 작업이 포함된 워크플로를 선택합니다. 성공 기준을 적용하려는 보고서입니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
3. 편집을 선택합니다.

4. YAML을 선택합니다.
5. 워크플로 다이어그램에서 CodeCatalyst 보고서를 생성하도록 구성된 작업을 선택합니다.
6. 세부 정보 창에서 출력 탭을 선택합니다.
7. 작업의 AutoDiscoverReports 섹션 또는 Reports 섹션에서 SuccessCriteria 속성을 PassRate, LineCoverage, BranchCoverage, Vulnerabilities, StaticAnalysisBug, StaticAnalysisSecurity, StaticAnalysisQuality 속성과 함께 추가합니다.

이러한 각 속성에 대한 설명은 [빌드 및 테스트 작업 YAML](#) 섹션을 참조하세요.

8. 커밋을 선택합니다.
9. 워크플로를 실행하여 CodeCatalyst가 원시 보고서에 성공 기준을 적용하도록 하고 성공 기준 정보가 포함된 관련 CodeCatalyst 보고서를 다시 생성합니다. 배치 워크플로우에 대한 자세한 내용은 [워크플로 수동 실행 시작](#) 섹션을 참조하세요.

품질 보고서 YAML 예시

다음 예시에서는 테스트 보고서, 코드 적용 범위 보고서, 소프트웨어 구성 분석 보고서, 정적 분석 보고서 등 4개의 보고서를 수동으로 구성하는 방법을 보여줍니다.

```
Reports:
  MyTestReport:
    Format: JUNITXML
    IncludePaths:
      - "*.xml"
    ExcludePaths:
      - report1.xml
    SuccessCriteria:
      PassRate: 90
  MyCoverageReport:
    Format: CLOVERXML
    IncludePaths:
      - output/coverage/jest/clover.xml
    SuccessCriteria:
      LineCoverage: 75
      BranchCoverage: 75
  MySCARReport:
    Format: SARIFSCA
    IncludePaths:
      - output/sca/reports.xml
```

```

SuccessCriteria:
  Vulnerabilities:
    Number: 5
    Severity: HIGH
MySAReport:
  Format: ESLINTJSON
  IncludePaths:
    - output/static/eslint.xml
  SuccessCriteria:
    StaticAnalysisBug:
      Number: 10
      Severity: MEDIUM
    StaticAnalysisSecurity:
      Number: 5
      Severity: CRITICAL
    StaticAnalysisQuality:
      Number: 0
      Severity: INFORMATIONAL

```

테스트 모범 사례

CodeCatalyst에서 제공하는 테스트 기능을 사용할 때는 다음 모범 사례를 따르는 것이 좋습니다.

주제

- [자동 검색](#)
- [성공 기준](#)
- [경로 포함/제외](#)

자동 검색

CodeCatalyst에서 작업을 구성할 때 자동 검색을 사용하면 JUnit 테스트 보고서와 같은 다양한 도구의 출력을 자동으로 검색하고 해당 도구에서 관련 CodeCatalyst 보고서를 생성할 수 있습니다. 자동 검색은 검색된 출력의 이름이나 경로가 변경되더라도 보고서가 계속 생성되도록 하는 데 도움이 됩니다. 새 파일이 추가되면 CodeCatalyst는 자동으로 파일을 검색하고 관련 보고서를 생성합니다. 그러나 자동 검색을 사용하는 경우 이 기능의 다음 측면 중 일부를 고려해야 합니다.

- 작업에서 자동 검색을 활성화하면 동일한 유형의 자동으로 검색된 모든 보고서가 동일한 성공 기준을 공유합니다. 예를 들어 최소 통과율과 같은 공유 기준이 모든 자동 검색된 테스트 보고서에 적용됩니다. 동일한 유형의 보고서에 대해 다른 기준이 필요한 경우 이러한 각 보고서를 명시적으로 구성해야 합니다.

- 자동 검색은 종속성에서 생성된 보고서를 찾을 수도 있으며 성공 기준이 구성된 경우 이러한 보고서에 대한 작업에 실패할 수 있습니다. 이 문제는 제외 경로 구성을 업데이트하여 해결할 수 있습니다.
- 자동 검색은 런타임에 작업을 스캔하기 때문에 매번 동일한 보고서 목록을 생성하지는 않습니다. 특정 보고서를 항상 생성하려면 보고서를 명시적으로 구성해야 합니다. 예를 들어 빌드의 일부로 테스트 실행을 중지해야 하는 경우 테스트 프레임워크는 출력을 생성하지 않으며, 결과적으로 테스트 보고서가 생성되지 않고 작업이 성공할 수 있습니다. 작업의 성공이 특정 테스트에 따라 달라지도록 하려면 해당 보고서를 명시적으로 구성해야 합니다.

Tip

새 프로젝트 또는 기존 프로젝트를 시작할 때 전체 프로젝트 디렉터리(**/* 포함)에 대해 자동 검색을 사용합니다. 이렇게 하면 하위 디렉터리 내의 파일을 포함하여 프로젝트의 모든 파일에서 보고서 생성이 간접 호출됩니다.

자세한 내용은 [작업에서 품질 보고서 구성](#) 섹션을 참조하세요.

성공 기준

성공 기준을 구성하여 보고서에 품질 임계값을 적용할 수 있습니다. 예를 들어 두 개의 코드 적용 범위 보고서가 자동으로 검색된 경우, 하나는 80%의 행 적용 범위이고 다른 하나는 60%의 행 적용 범위인 경우 다음 옵션이 있습니다.

- 행 적용 범위에 대한 자동 검색 성공 기준을 80%로 설정합니다. 이로 인해 첫 번째 보고서가 통과하고 두 번째 보고서가 실패하여 전체 작업이 실패합니다. 워크플로의 차단을 해제하려면 두 번째 보고서의 행 적용 범위가 80%를 초과할 때까지 프로젝트에 새 테스트를 추가합니다.
- 행 적용 범위에 대한 자동 검색 성공 기준을 60%로 설정합니다. 이렇게 하면 두 보고서가 모두 전달되어 작업이 성공합니다. 그런 다음 두 번째 보고서에서 코드 적용 범위를 늘리는 작업을 수행할 수 있습니다. 그러나 이 접근 방식을 사용하면 첫 번째 보고서의 적용 범위가 80% 미만으로 떨어지지 않는다고 보장할 수 없습니다.
- 시각적 편집기를 사용하거나 각 보고서에 대해 명시적 YAML 섹션 및 경로를 추가하여 하나 또는 두 개의 보고서를 명시적으로 구성합니다. 이렇게 하면 각 보고서에 대해 별도의 성공 기준과 사용자 지정 이름을 구성할 수 있습니다. 그러나 이 접근 방식을 사용하면 보고서 경로가 변경되면 작업이 실패할 수 있습니다.

자세한 내용은 [보고서의 성공 기준 구성](#) 섹션을 참조하세요.

경로 포함/제외

작업 결과를 검토할 때 IncludePaths 및 ExcludePaths를 구성하여 CodeCatalyst에서 생성한 보고서 목록을 조정할 수 있습니다.

- IncludePaths는 보고서를 검색할 때 CodeCatalyst가 포함할 파일 및 파일 경로를 지정하는 데 사용됩니다. 예를 들어 `"/test/report/*"` 지정 시 CodeCatalyst는 작업에서 `/test/report/` 디렉터리를 찾는 데 사용되는 전체 빌드 이미지를 검색합니다. 해당 디렉터리를 찾으면 CodeCatalyst는 해당 디렉터리에서 보고서를 찾습니다.

Note

수동으로 구성된 보고서의 경우 IncludePaths는 단일 파일과 일치하는 글로브 패턴이어야 합니다.

- ExcludePaths는 보고서를 검색할 때 CodeCatalyst가 제외할 파일 및 파일 경로를 지정하는 데 사용됩니다. 예를 들어 `"/test/reports/**/*"` 지정 시 CodeCatalyst는 `/test/reports/` 디렉터리의 파일을 검색하지 않습니다. 디렉터리의 모든 파일을 무시하려면 `**/*` glob 패턴을 사용합니다.

다음은 가능한 glob 패턴의 몇 가지 예입니다.

패턴	설명
<code>*.*</code>	점을 포함한 모든 객체 이름과 해당합니다.
<code>*.xml</code>	.xml로 끝나는 현재 디렉터리의 모든 객체 이름과 일치
<code>*.{xml,txt}</code>	.xml 또는 .txt로 끝나는 현재 디렉터리의 모든 객체 이름과 일치
<code>**/*.xml</code>	.xml로 끝나는 모든 디렉터리의 객체 이름과 일치
<code>testFolder</code>	파일로 취급하여 testFolder 객체와 일치

패턴	설명
<code>testFolder/*</code>	<code>testFolder/file.xml</code> 와 같은 <code>testFolder</code> 에서 하위 폴더의 한 수준의 객체와 일치
<code>testFolder/*/*</code>	<code>testFolder/reportsFolder/file.xml</code> 와 같은 <code>testFolder</code> 에서 하위 폴더의 두 가지 수준의 객체와 일치
<code>testFolder/**</code>	<code>testFolder</code> 아래 파일뿐만 아니라 하위 폴더 <code>testFolder</code> 도 일치시킵니다(예: <code>testFolder/file.xml</code> 와 <code>testFolder/otherFolder/file.xml</code>).

CodeCatalyst는 다음과 같이 glob 패턴을 해석합니다.

- 슬래시(/) 문자는 파일 경로의 디렉터리를 구분합니다.
- 별표(*) 문자는 폴더의 경계를 넘지 않고 0개 이상의 이름 구성 요소 문자와 해당합니다.
- 이중 별표(**)는 모든 디렉터리에서 이름 구성 요소의 0개 이상 문자와 일치합니다.

Note

`ExcludePaths`가 `IncludePaths`보다 우선합니다. `IncludePaths` 및 `ExcludePaths`에 모두 동일한 폴더가 포함된 경우 해당 폴더는 보고서에 대해 스캔되지 않습니다.

지원되는 SARIF 속성

정적 분석 결과 교환 형식(SARIF)은 Amazon CodeCatalyst의 소프트웨어 구성 분석(SCA) 및 정적 분석 보고서에서 사용할 수 있는 출력 파일 형식입니다. 다음 예시는 정적 분석 보고서에서 SARIF를 수동으로 구성하는 방법을 보여줍니다.

```
Reports:
MySAReport:
Format: SARIFSA
IncludePaths:
```

```
- output/sa_report.json
SuccessCriteria:
  StaticAnalysisFinding:
    Number: 25
    Severity: HIGH
```

CodeCatalyst는 다음과 같은 SARIF 속성을 지원하며, 이를 사용하여 분석 결과가 보고서에 표시되는 방식을 최적화할 수 있습니다.

주제

- [sarifLog 객체](#)
- [run 객체](#)
- [toolComponent 객체](#)
- [reportingDescriptor 객체](#)
- [result 객체](#)
- [location 객체](#)
- [physicalLocation 객체](#)
- [logicalLocation 객체](#)
- [fix 객체](#)

sarifLog 객체

이름	필수	설명
\$schema	예	버전 2.1.0 에 대한 SARIF JSON 스키마의 URI입니다.
version	예	CodeCatalyst는 SARIF 버전 2.1.0만 지원합니다.
runs[]	예	SARIF 파일에는 하나 이상의 실행 배열이 포함되며, 각 실행은 분석 도구의 단일 실행을 나타냅니다.

run 객체

이름	필수	설명
<code>tool.driver</code>	예	분석 도구를 설명하는 <code>toolComponent</code> 객체입니다.
<code>tool.name</code>	아니요	분석을 수행하는 데 사용되는 도구의 이름을 나타내는 속성입니다.
<code>results[]</code>	예	CodeCatalyst에 표시되는 분석 도구의 결과입니다.

toolComponent 객체

이름	필수	설명
<code>name</code>	예	분석 도구의 이름입니다.
<code>properties.artifactScanned</code>	아니요	도구에서 분석한 총 아티팩트 수입니다.
<code>rules[]</code>	예	규칙을 나타내는 <code>reportingDescriptor</code> 객체의 배열입니다. 이러한 규칙을 기반으로 분석 도구는 분석되는 코드에서 문제를 찾습니다.

reportingDescriptor 객체

이름	필수	설명
id	예	조사 결과를 참조하는 데 사용되는 규칙의 고유 식별자입니다. 최대 길이: 1,024자
name	아니요	규칙의 디스플레이 이름입니다. 최대 길이: 1,024자
shortDescription.text	아니요	규칙에 대한 짧은 설명입니다. 최대 길이: 3,000자
fullDescription.text	아니요	규칙에 대한 전체 설명입니다. 최대 길이: 3,000자
helpUri	아니요	규칙에 대한 기본 설명서의 절대 URI를 포함하도록 현지화할 수 있는 문자열입니다. 최대 길이: 3,000자
properties.unscore	아니요	스캔 조사 결과의 점수가 매겨졌는지 여부를 나타내는 플래그입니다.
properties.score.severity	아니요	조사 결과의 심각도 수준을 지정하는 고정된 문자열 세트입니다. 최대 길이: 1,024자

이름	필수	설명
properties.cvssv3_baseSeverity	아니요	일반 취약성 점수 체계 v3.1 의 정성적 심각도 등급입니다.
properties.cvssv3_baseScore	아니요	CVSS v3 기본 점수 범위는 0.0~10.0 입니다.
properties.cvssv2_severity	아니요	CVSS v3 값을 사용할 수 없는 경우 CodeCatalyst는 CVSS v2 값을 검색합니다.
properties.cvssv2_score	아니요	CVSS v2 기본 점수 범위는 0.0~10.0 입니다.
properties.severity	아니요	조사 결과의 심각도 수준을 지정하는 고정된 문자열 세트입니다. 최대 길이: 1,024자
defaultConfiguration.level	아니요	규칙의 기본 심각도입니다.

result 객체

이름	필수	설명
ruleId	예	조사 결과를 참조하는 데 사용되는 규칙의 고유 식별자입니다. 최대 길이: 1,024자
ruleIndex	예	도구 구성 요소 rules[]에서 연결된 규칙의 인덱스입니다.

이름	필수	설명
message.text	예	결과를 설명하고 각 조사 결과에 대한 메시지를 표시하는 메시지입니다. 최대 길이: 3,000자
rank	아니요	결과의 우선 순위 또는 중요도를 나타내는 0.0~100.0 사이의 값입니다. 이 규모의 값은 0.0이 가장 낮은 우선 순위이고 100.0이 가장 높은 우선 순위입니다.
level	아니요	결과의 심각도입니다. 최대 길이: 1,024자
properties.unscore	아니요	스캔 조사 결과의 점수가 매겨졌는지 여부를 나타내는 플래그입니다.
properties.score.severity	아니요	조사 결과의 심각도 수준을 지정하는 고정된 문자열 세트입니다. 최대 길이: 1,024자
properties.cvssv3_baseSeverity	아니요	일반 취약성 점수 체계 v3.1 의 정성적 심각도 등급입니다.
properties.cvssv3_baseScore	아니요	CVSS v3 기본 점수 범위는 0.0~10.0 입니다.
properties.cvssv2_severity	아니요	CVSS v3 값을 사용할 수 없는 경우 CodeCatalyst는 CVSS v2 값을 검색합니다.

이름	필수	설명
<code>properties.cvssv2_score</code>	아니요	CVSS v2 기본 점수 범위는 0.0~10.0 입니다.
<code>properties.severity</code>	아니요	조사 결과의 심각도 수준을 지정하는 고정된 문자열 세트입니다. 최대 길이: 1,024자
<code>locations[]</code>	예	결과가 감지된 위치 집합입니다. 지정된 모든 위치에서 변경해야만 문제를 해결할 수 있는 경우가 아니라면 한 위치만 포함해야 합니다. CodeCatalyst는 위치 배열의 첫 번째 값을 사용하여 결과에 주석을 지정합니다. 최대 location 객체 개수: 10
<code>relatedLocations[]</code>	아니요	조사 결과에서 참조하는 추가 위치 목록입니다. 최대 location 객체 개수: 50
<code>fixes[]</code>	아니요	스캔 도구에서 제공하는 권장 사항을 나타내는 fix 객체 배열입니다. CodeCatalyst는 fixes 배열에서 첫 번째 권장 사항을 사용합니다.

location 객체

이름	필수	설명
<code>physicalLocation</code>	예	아티팩트와 리전을 식별합니다.
<code>logicalLocations[]</code>	아니요	아티팩트를 참조하지 않고 이름으로 설명하는 위치 집합입니다.

physicalLocation 객체

이름	필수	설명
<code>artifactLocation.uri</code>	예	아티팩트의 위치를 나타내는 URI로, 일반적으로 리포지토리에 있거나 빌드 중에 생성된 파일입니다.
<code>fileLocation.uri</code>	아니요	파일 위치를 나타내는 대체 URI입니다. <code>artifactLocation.uri</code> 가 빈 상태를 반환하는 경우 사용됩니다.
<code>region.startLine</code>	예	리전의 첫 번째 문자의 행 번호입니다.
<code>region.startColumn</code>	예	리전의 첫 번째 문자의 열 번호입니다.
<code>region.endLine</code>	예	리전의 마지막 문자의 행 번호입니다.
<code>region.endColumn</code>	예	리전의 마지막 문자의 열 번호입니다.

logicalLocation 객체

이름	필수	설명
fullyQualifiedName	아니요	결과물의 위치를 설명하는 추가 정보입니다. 최대 길이: 1,024자

fix 객체

이름	필수	설명
description.text	아니요	각 조사 결과에 대한 권장 사항을 표시하는 메시지입니다. 최대 길이: 3,000자
artifactChanges.[0].artifactLocation.uri	아니요	업데이트해야 하는 아티팩트의 위치를 나타내는 URI입니다.

워크플로를 사용하여 배포

[CodeCatalyst 워크플로](#)를 사용하면 Amazon ECS 등과 같은 다양한 대상에 애플리케이션 및 기타 리소스를 배포할 수 있습니다.

애플리케이션을 배포하려면 어떻게 해야 하나요?

CodeCatalyst를 통해 애플리케이션 또는 리소스를 배포하려면 먼저 워크플로를 생성한 다음 그 안에 배포 작업을 지정합니다. 배포 작업은 배포하려는 대상, 배포하려는 위치 및 배포 방법(예: 블루/그린 스키마 사용)을 정의하는 워크플로 구성 요소입니다. CodeCatalyst 콘솔의 시각적 편집기 또는 YAML 편집기를 사용하여 워크플로에 배포 작업을 추가합니다.

애플리케이션 또는 리소스를 배포하는 상위 단계는 다음과 같습니다.

애플리케이션을 배포하려면(고급 작업)

1. CodeCatalyst 프로젝트에서 배포하려는 애플리케이션의 소스 코드를 추가합니다. 자세한 내용은 [CodeCatalyst에서 프로젝트의 리포지토리에 소스 코드 저장](#) 단원을 참조하십시오.
2. CodeCatalyst 프로젝트에서 배포하려는 대상 AWS 계정 및 선택적 Amazon Virtual Private Cloud(VPC)를 정의하는 환경을 추가합니다. 자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 단원을 참조하십시오.
3. CodeCatalyst 프로젝트에서 워크플로를 생성합니다. 워크플로에서는 애플리케이션을 빌드, 테스트 및 배포하는 방법을 정의합니다. 자세한 내용은 [워크플로 시작하기](#) 섹션을 참조하세요.
4. 워크플로에서 트리거, 빌드 작업 및 선택적으로 테스트 작업을 추가합니다. 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#), [빌드 작업 추가](#), [테스트 작업 추가](#) 섹션을 참조하세요.
5. 워크플로에서 배포 작업을 추가합니다. 여러 CodeCatalyst 제공 배포 작업을 Amazon ECS와 같은 다양한 대상으로 애플리케이션에 선택할 수 있습니다. (구축 작업 또는 GitHub Actions를 사용하여 애플리케이션을 배포할 수도 있습니다. 빌드 작업 및 GitHub Actions에 대한 자세한 내용은 [작업을 배포하는 대안](#) 섹션을 참조하세요.)
6. 트리거를 통해 워크플로를 수동으로 또는 자동으로 시작합니다. 워크플로는 빌드, 테스트 및 배포 작업을 순서대로 실행하여 애플리케이션 및 리소스를 대상에 배포합니다. 자세한 내용은 [워크플로 수동 실행 시작](#) 섹션을 참조하세요.

배포 작업 목록

다음 배포 작업을 사용할 수 있습니다.

- CloudFormation 스택 배포 - 이 작업은 [AWS Serverless Application Model](#) 사용자가 제공하는 [CloudFormation 템플릿](#) 또는 템플릿을 AWS 기반으로 CloudFormation 스택을 생성합니다. 자세한 내용은 [CloudFormation 스택 배포](#) 단원을 참조하십시오.
- Amazon ECS에 배포 - 이 작업은 사용자가 제공하는 [작업 정의](#) 파일을 등록합니다. 자세한 내용은 [워크플로를 사용하여 Amazon ECS에 배포](#) 섹션을 참조하세요.
- Kubernetes 클러스터에 배포 - 이 작업은 Amazon Elastic Kubernetes Service 클러스터에 애플리케이션을 배포합니다. 자세한 내용은 [워크플로를 사용하여 Amazon EKS에 배포](#) 단원을 참조하십시오.
- AWS CDK 배포 - 이 작업은 [AWS CDK 앱](#)에 배포합니다 AWS. 자세한 내용은 [워크플로를 사용하여 AWS CDK 앱 배포](#) 단원을 참조하십시오.

Note

리소스를 배포할 수 있는 다른 CodeCatalyst 작업이 있지만 배포 정보가 환경 페이지에 표시되지 않으므로 배포 작업으로 간주되지 않습니다. 환경 페이지 및 배포 보기에 대한 자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 및 [배포 정보 보기](#) 섹션을 참조하세요.

작업 배포의 이점

워크플로 내에서 배포 작업을 사용하면 다음과 같은 이점이 있습니다.

- 배포 기록 - 배포 기록을 확인하여 배포된 소프트웨어의 변경 사항을 관리하고 전달할 수 있습니다.
- 추적성 - CodeCatalyst 콘솔을 통해 배포 상태를 추적하고 각 애플리케이션 개정이 배포된 시기와 위치를 확인합니다.
- 롤백 - 오류가 있는 경우 배포를 자동으로 롤백합니다. 배포 롤백을 활성화하도록 경보를 구성할 수도 있습니다.
- 모니터링 - 워크플로의 다양한 단계를 거치면서 배포를 관찰합니다.
- 다른 CodeCatalyst 기능과의 통합 - 소스 코드를 저장한 다음 빌드, 테스트 및 배포합니다.

작업을 배포하는 대안

배포 작업은 이전 섹션에 설명된 이점을 제공하므로 사용할 필요가 없습니다. 대신 다음 [CodeCatalyst 작업](#)을 사용할 수 있습니다.

- 빌드 작업입니다.

일반적으로 해당 배포 작업이 없는 대상에 배포하려는 경우 또는 배포 절차를 더 잘 제어하려는 경우 빌드 작업을 사용합니다. 빌드 작업을 사용하여 리소스를 배포하는 방법에 대한 자세한 내용은 [워크플로로 빌드하기](#)를 참조하세요.

- GitHub Action.

CodeCatalyst 워크플로 내에서 [GitHub 작업](#)을 사용하여 애플리케이션 및 리소스를 배포할 수 있습니다(CodeCatalyst 작업 대신). CodeCatalyst 워크플로 내에서 GitHub Actions를 사용하는 방법에 대한 자세한 내용은 [GitHub Actions와 통합](#) 섹션을 참조하세요.

CodeCatalyst 워크플로를 사용하지 않으려면 다음 AWS 서비스를 사용하여 애플리케이션을 배포할 수도 있습니다.

- AWS CodeDeploy - [CodeDeploy란 무엇인가요?](#)를 참조하세요.
- AWS CodeBuild 및 AWS CodePipeline - [란 무엇입니까 AWS CodeBuild?](#) 및 [란 무엇입니까 AWS CodePipeline?](#)를 참조하세요.
- CloudFormation - [란 무엇인가요 CloudFormation?](#)를 참조하세요.

복잡한 엔터프라이즈 배포에는 CodeDeploy, CodeBuild, CodePipeline 및 CloudFormation 서비스를 사용합니다.

주제

- [워크플로를 사용하여 Amazon ECS에 배포](#)
- [워크플로를 사용하여 Amazon EKS에 배포](#)
- [CloudFormation 스택 배포](#)
- [워크플로를 사용하여 AWS CDK 앱 배포](#)
- [워크플로를 사용하여 AWS CDK 앱 부트스트래핑](#)
- [워크플로를 사용하여 Amazon S3에 파일 게시](#)
- [AWS 계정 및 VPCs에 배포](#)
- [워크플로 다이어그램에 앱 URL 표시](#)
- [배포 대상 제거](#)
- [커밋별 배포 상태 추적](#)
- [배포 로그 보기](#)
- [배포 정보 보기](#)

워크플로를 사용하여 Amazon ECS에 배포

이 섹션에서는 CodeCatalyst 워크플로를 사용하여 컨테이너화된 애플리케이션을 Amazon Elastic Container Service 클러스터에 배포하는 방법을 설명합니다. 이렇게 하려면 워크플로에 Amazon ECS에 배포 작업을 추가해야 합니다. 이 작업은 사용자가 제공하는 [작업 정의](#) 파일을 등록합니다. 등록 시 작업 정의는 [Amazon ECS 클러스터](#)에서 실행되는 [Amazon ECS 서비스](#)에 의해 인스턴스화됩니다. "작업 정의 설명"은 Amazon ECS에 애플리케이션을 배포하는 것과 동일합니다.

이 작업을 사용하려면 Amazon ECS 클러스터, 서비스 및 작업 정의 파일이 준비되어 있어야 합니다.

Amazon ECS에 대해 자세히 알아보려면 Amazon Elastic Container Service 개발자 안내서를 참조하세요.

i Tip

Amazon ECS에 배포 작업을 사용하는 방법을 보여주는 자습서는 [자습서: Amazon ECS에 애플리케이션 배포](#) 섹션을 참조하세요.

i Tip

Amazon ECS에 배포 작업의 작업 예시를 보려면 Node.js API with AWS Fargate 또는 Java API with AWS Fargate 블루프린트로 프로젝트를 생성합니다. 자세한 내용은 [블루프린트를 사용하여 프로젝트 생성](#) 섹션을 참조하세요.

주제

- ['Amazon ECS에 배포' 작업에 사용되는 런타임 이미지](#)
- [자습서: Amazon ECS에 애플리케이션 배포](#)
- ['Amazon ECS에 배포' 작업 추가](#)
- ['Amazon ECS에 배포' 변수](#)
- ['Amazon ECS에 배포' 작업 YAML](#)

'Amazon ECS에 배포' 작업에 사용되는 런타임 이미지

Amazon ECS에 배포 작업은 [2022년 11월 이미지](#)에서 실행됩니다. 자세한 내용은 [활성 이미지](#) 섹션을 참조하세요.

자습서: Amazon ECS에 애플리케이션 배포

이 자습서에서는 워크플로, Amazon ECS 및 기타 몇 가지 AWS 서비스를 사용하여 Amazon Elastic Container Service(Amazon ECS)에 서버리스 애플리케이션을 배포하는 방법을 알아봅니다. 배포된 애플리케이션은 Apache 웹 서버 Docker 이미지를 기반으로 구축된 간단한 Hello World 웹 사이트입니다. 자습서에서는 클러스터 설정과 같은 필수 준비 작업을 안내한 다음 애플리케이션을 빌드하고 배포하기 위한 워크플로를 생성하는 방법을 설명합니다.

i Tip

이 자습서를 진행하는 대신 전체 Amazon ECS 설정을 수행하는 블루프린트를 사용할 수 있습니다. Node.js API with AWS Fargate 또는 Java API with AWS Fargate 블루프린트를 사용해야 합니다. 자세한 내용은 [블루프린트를 사용하여 프로젝트 생성](#) 섹션을 참조하세요.

주제

- [사전 조건](#)
- [1단계: AWS 사용자 설정 및 AWS CloudShell](#)
- [2단계: Amazon ECS에 자리 표시자 애플리케이션 배포](#)
- [3단계: Amazon ECR 이미지 리포지토리 생성](#)
- [4단계: AWS 역할 생성](#)
- [5단계: CodeCatalyst에 AWS 역할 추가](#)
- [6단계: 소스 리포지토리 생성](#)
- [7단계: 소스 파일 추가](#)
- [8단계: 워크플로 생성 및 실행](#)
- [9단계: 소스 파일 변경](#)
- [정리](#)

사전 조건

시작하기 전:

- 연결된 AWS 계정이 있는 CodeCatalyst 스페이스가 필요합니다. 자세한 내용은 [스페이스 생성](#) 단원을 참조하십시오.
- 스페이스에는 다음과 같은 빈 프로젝트가 필요합니다.

```
codecatalyst-ecs-project
```

처음부터 시작 옵션을 사용하여 이 프로젝트를 생성합니다.

자세한 내용은 [Amazon CodeCatalyst에서 빈 프로젝트 생성](#) 섹션을 참조하세요.

- 프로젝트에는 다음과 같은 CodeCatalyst 환경이 필요합니다.

```
codecatalyst-ecs-environment
```

다음과 같이 이 환경을 구성합니다.

- 비프로덕션과 같은 유형을 선택합니다.
- AWS 계정에 연결합니다.
- 기본 IAM 역할의 경우 아무 역할이나 선택합니다. 나중에 다른 역할을 지정합니다.

자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 단원을 참조하십시오.

1단계: AWS 사용자 설정 및 AWS CloudShell

이 자습서의 첫 번째 단계는 사용자 생성이고 이 사용자로 인스턴스를 AWS IAM Identity Center AWS CloudShell 시작하는 것입니다. 이 자습서 기간 동안 CloudShell은 개발 컴퓨터이며 AWS 리소스와 서비스를 구성하는 곳입니다. 자습서를 완료한 후 이 사용자를 삭제합니다.

Note

이 자습서에는 루트 사용자를 사용하지 마세요. 별도의 사용자를 생성해야 합니다. 그렇지 않으면 나중에 AWS Command Line Interface (CLI)에서 작업을 수행할 때 문제가 발생할 수 있습니다.

IAM Identity Center 사용자 및 CloudShell에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서 및 AWS CloudShell 사용 설명서를 참조하세요.

IAM Identity Center 사용자를 생성하려면

1. 에 로그인 AWS Management Console 하고 <https://console.aws.amazon.com/singlesignon/> AWS IAM Identity Center 콘솔을 엽니다.

Note

CodeCatalyst 스페이스에 연결된 AWS 계정 를 사용하여 로그인해야 합니다. 스페이스로 이동하여 AWS 계정 탭을 선택하고 연결된 계정을 확인할 수 있습니다. 자세한 내용은 [스페이스 생성](#) 섹션을 참조하세요.

2. 탐색 창에서 Users와 Add user를 차례대로 선택합니다.

3. 사용자 이름에 다음을 입력합니다.

CodeCatalystECSUser

4. 암호에서 이 사용자와 공유할 수 있는 일회용 암호 생성을 선택합니다.
5. 이메일 주소 및 이메일 주소 확인에 IAM Identity Center에 아직 없는 이메일 주소를 입력합니다.
6. 이름 및 성에 다음을 입력합니다.

CodeCatalystECSUser

7. 표시 이름에 자동으로 생성된 이름을 유지합니다.

CodeCatalystECSUser CodeCatalystECSUser

8. 다음을 선택합니다.
9. 그룹에 사용자 추가 페이지에서 다음을 선택합니다.
10. 검토 및 사용자 추가 페이지에서 정보를 검토하고 사용자 추가를 선택합니다.

일회용 암호 대화 상자가 나타납니다.

11. 복사를 선택한 다음 AWS 액세스 포털 URL 및 일회용 암호를 포함하여 로그인 정보를 붙여넣습니다.
12. 닫기를 선택하세요.

권한 집합을 생성하려면

이 권한 세트를 나중에 CodeCatalystECSUser에 할당합니다.

1. 탐색 창에서 권한 세트를 선택한 다음 권한 세트 생성을 선택합니다.
2. 사전 정의된 권한 세트를 선택하고 AdministratorAccess를 선택합니다. 이 정책은 모든 AWS 서비스에 대한 전체 권한을 제공합니다.
3. 다음을 선택합니다.
4. 권한 세트 이름에 다음을 입력합니다.

CodeCatalystECSPermissionSet

5. 다음을 선택합니다.

6. 검토 및 생성 페이지에서 정보를 검토하고 생성을 선택합니다.

이 권한 세트를 CodeCatalystECSUser에게 할당하려면

1. 탐색 창에서 AWS 계정을 선택한 다음 현재 로그인 AWS 계정 한 옆의 확인란을 선택합니다.
2. 사용자 또는 그룹 할당을 선택합니다.
3. 사용자 탭을 선택합니다.
4. CodeCatalystECSUser 옆의 확인란을 선택합니다.
5. 다음을 선택합니다.
6. CodeCatalystECSPermissionSet 옆의 확인란을 선택합니다.
7. 다음을 선택합니다.
8. 정보를 검토하고 제출을 선택합니다.

이제 CodeCatalystECSUser 및 CodeCatalystECSPermissionSet에 할당하여 함께 AWS 계정바인딩했습니다.

로그아웃했다가 CodeCatalystECSUser로 다시 로그인하려면

1. 로그아웃하기 전에에 대한 AWS 액세스 포털 URL과 사용자 이름 및 일회용 암호가 있는지 확인합니다. CodeCatalystECSUser. 이전에 이 정보를 텍스트 편집기에 복사했어야 합니다.

Note

이 정보가 없는 경우 IAM Identity Center의 CodeCatalystECSUser 세부 정보 페이지로 이동하여 암호 재설정, 일회용 암호 생성 [...]을 선택한 후 암호 재설정을 다시 선택하면 화면에 해당 정보가 표시됩니다.

2. 로그아웃합니다 AWS.
3. AWS 액세스 포털 URL을 브라우저의 주소 표시줄에 붙여 넣습니다.
4. CodeCatalystECSUser의 사용자 이름과 일회용 암호로 로그인합니다.
5. 새 암호에서 암호를 입력하고 새 암호 설정을 선택합니다.

화면에 AWS 계정 상자가 나타납니다.

6. AWS 계정을 선택한 다음 CodeCatalystECSUser 사용자 및 권한 세트를 할당 AWS 계정 한의 이름을 선택합니다.

7. CodeCatalystECSPermissionSet 옆에 있는 Management Console을 선택합니다.

가 AWS Management Console 나타납니다. 이제 적절한 권한으로 CodeCatalystECSUser에 로그인되었습니다.

AWS CloudShell 인스턴스를 시작하려면

1. 의 상단 탐색 모음CodeCatalystECSUser에서 AWS 아이콘



을 선택합니다.

의 기본 페이지가 AWS Management Console 나타납니다.

2. 상단 탐색 모음에서 AWS CloudShell 아이콘



을 선택합니다.

CloudShell이 열립니다. CloudShell 환경이 생성되는 동안 기다립니다.

Note

CloudShell 아이콘이 보이지 않는 경우 [CloudShell에서 지원하는 리전에 있는지](#) 확인하세요. 이 자습서에서는 사용자가 미국 서부(오리건) 리전에 있다고 가정합니다.

AWS CLI 가 설치되었는지 확인하려면

1. CloudShell 터미널에서 다음을 입력합니다.

```
aws --version
```

2. 버전이 나타나는지 확인합니다.

AWS CLI 는 현재 사용자에게 이미 구성되어 CodeCatalystECSUser있으므로 일반적으로처럼 AWS CLI 키와 자격 증명을 구성할 필요가 없습니다.

2단계: Amazon ECS에 자리 표시자 애플리케이션 배포

이 섹션에서는 자리 표시자 애플리케이션을 Amazon ECS에 수동으로 배포합니다. 이 자리 표시자 애플리케이션은 워크플로에 의해 배포된 Hello World 애플리케이션으로 대체됩니다. 자리 표시자 애플리케이션은 Apache Web Server입니다.

Amazon ECS에 대해 자세히 알아보려면 Amazon Elastic Container Service 개발자 안내서를 참조하세요.

자리 표시자 애플리케이션을 배포하려면 다음 일련의 절차를 완료합니다.

태스크 실행 역할을 생성하는 방법

이 역할은 Amazon ECS에 사용자를 대신하여 API를 호출할 수 있는 AWS Fargate 권한을 부여합니다.

1. 신뢰 정책 생성:

- a. 에 다음 명령을 AWS CloudShell 입력합니다.

```
cat > codecatalyst-ecs-trust-policy.json
```

CloudShell 터미널에 깜박이는 프롬프트가 나타납니다.

- b. 프롬프트에서 다음 코드를 입력합니다.

JSON

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

- c. 커서를 마지막 중괄호(}) 뒤에 놓습니다.

- d. **Enter**를 누른 다음 **Ctrl+d**를 눌러 파일을 저장하고 cat을 종료합니다.
2. 태스크 실행 역할을 생성합니다.

```
aws iam create-role \  
    --role-name codecatalyst-ecs-task-execution-role \  
    --assume-role-policy-document file:///codecatalyst-ecs-trust-policy.json
```

3. AWS 관리형 AmazonECSTaskExecutionRolePolicy 정책을 역할에 연결합니다.

```
aws iam attach-role-policy \  
    --role-name codecatalyst-ecs-task-execution-role \  
    --policy-arn arn:aws:iam::aws:policy/service-role/  
AmazonECSTaskExecutionRolePolicy
```

4. 역할의 세부 정보를 표시합니다.

```
aws iam get-role \  
    --role-name codecatalyst-ecs-task-execution-role
```

5. 역할의 "Arn": 값을 기록해 둡니다. 예시: arn:aws:iam::111122223333:role/codecatalyst-ecs-task-execution-role. 나중에 이 Amazon 리소스 이름(ARN)이 필요합니다.

Amazon ECS 클러스터를 생성하려면

이 클러스터에는 Apache 자리 표시자 애플리케이션과 이후 Hello World 애플리케이션이 포함됩니다.

1. 에서 빈 클러스터를 CodeCatalystECSUser AWS CloudShell 생성합니다.

```
aws ecs create-cluster --cluster-name codecatalyst-ecs-cluster
```

2. (선택 사항) 클러스터가 성공적으로 생성되었는지 확인합니다.

```
aws ecs list-clusters
```

codecatalyst-ecs-cluster 클러스터의 ARN이 목록에 나타나 성공적으로 생성되었음을 나타냅니다.

태스크 정의 파일을 생성하는 방법

태스크 정의 파일은 DockerHub에서 가져온 [Apache 2.4 웹 서버](#) Docker 이미지(httpd:2.4)를 실행함을 나타냅니다.

1. 에서 작업 정의 파일을 CodeCatalystECSUser AWS CloudShell생성합니다.

```
cat > taskdef.json
```

2. 프롬프트에서 다음 코드를 붙여넣습니다.

```
{
  "executionRoleArn": "arn:aws:iam::111122223333:role/codecatalyst-ecs-task-  
execution-role",
  "containerDefinitions": [
    {
      "name": "codecatalyst-ecs-container",
      "image": "httpd:2.4",
      "essential": true,
      "portMappings": [
        {
          "hostPort": 80,
          "protocol": "tcp",
          "containerPort": 80
        }
      ]
    }
  ],
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "cpu": "256",
  "family": "codecatalyst-ecs-task-def",
  "memory": "512",
  "networkMode": "awsvpc"
}
```

앞의 코드에서 *arn:aws:iam::111122223333:role/codecatalyst-ecs-task-execution-role*을

[태스크 실행 역할을 생성하는 방법](#)에서 기록한 태스크 실행 역할의 ARN으로 바꿉니다.

3. 커서를 마지막 중괄호(}) 뒤에 놓습니다.

4. **Enter**를 누른 다음 **Ctrl+d**를 눌러 파일을 저장하고 cat을 종료합니다.

Amazon ECS로 태스크 정의 파일을 등록하려면

1. 의 CodeCatalystECSUser에서 작업 정의를 AWS CloudShell등록합니다.

```
aws ecs register-task-definition \
  --cli-input-json file://taskdef.json
```

2. (선택 사항) 태스크 정의가 등록되었는지 확인합니다.

```
aws ecs list-task-definitions
```

codecatalyst-ecs-task-def 태스크 정의가 목록에 나타나야 합니다.

Amazon ECS 서비스를 생성하는 방법

Amazon ECS 서비스는 Apache 자리 표시자 애플리케이션과 이후 Hello World 애플리케이션의 태스크(및 관련 Docker 컨테이너)를 실행합니다.

1. CodeCatalystECSUser로서 아직 하지 않았다면 Amazon Elastic Container Service 콘솔로 전환합니다.
2. 앞서 생성한 codecatalyst-ecs-cluster 클러스터를 선택합니다.
3. 서비스 탭에서 생성을 선택합니다.
4. 생성 페이지에서 다음을 수행합니다.
 - a. 다음에 나열된 설정을 제외한 모든 기본 설정을 유지합니다.
 - b. 시작 유형에서 FARGATE를 선택합니다.
 - c. 태스크 정의의 패밀리 드롭다운 목록에서 다음을 선택합니다.

codecatalyst-ecs-task-def

- d. 서비스 이름에 다음을 입력합니다.

```
codecatalyst-ecs-service
```

- e. 원하는 태스크에 다음을 입력합니다.

3

이 자습서에서는 각 작업이 단일 Docker 컨테이너를 시작합니다.

- f. 네트워킹 섹션을 확장합니다.
- g. VPC에서 VPC를 선택합니다.
- h. 서브넷에서 서브넷을 선택합니다.

Note

서브넷을 하나만 지정합니다. 이 자습서에 필요한 것은 이것뿐입니다.

Note

VPC와 서브넷이 없는 경우 생성합니다. Amazon VPC 사용 설명서의 [VPC에 서브넷 생성](#)을 참조하여 [VPC를 생성](#)합니다.

- i. 보안 그룹에서 새 보안 그룹 생성을 선택하고 다음을 수행합니다.
 - i. 보안 그룹 이름에 다음을 입력합니다.

codecatalyst-ecs-security-group

- ii. 보안 그룹 설명에 다음을 입력합니다.

CodeCatalyst ECS security group

- iii. 규칙 추가를 선택합니다. 유형에서 HTTP를 선택하고 소스에서 Anywhere를 선택합니다.
 - j. 하단에서 생성을 선택합니다.
 - k. 서비스가 생성되는 동안 기다립니다. 몇 분 정도 소요될 수 있습니다.
5. 태스크 탭을 선택한 후 새로 고침 버튼을 선택합니다. 세 태스크 모두에서 마지막 상태 열이 실행 중으로 설정되어 있는지 확인합니다.

(선택 사항) Apache 자리 표시자 애플리케이션이 실행 중인지 확인하려면

- 1. 태스크 탭에서 세 가지 작업 중 하나를 선택합니다.

2. 퍼블릭 IP 필드에서 개방형 주소를 선택합니다.

It Works! 페이지가 나타납니다. 이는 Amazon ECS 서비스가 Apache 이미지로 Docker 컨테이너를 시작한 태스크를 성공적으로 시작했음을 나타냅니다.

자습서의 이 시점에서 Amazon ECS 클러스터, 서비스 및 태스크 정의와 Apache 자리 표시자 애플리케이션을 수동으로 배포했습니다. 이러한 모든 항목이 준비되면 이제 Apache 자리 표시자 애플리케이션을 자습서의 Hello World 애플리케이션으로 대체하는 워크플로를 생성할 준비가 된 것입니다.

3단계: Amazon ECR 이미지 리포지토리 생성

이 섹션에서는 Amazon Elastic Container Registry(Amazon ECR)에서 프라이빗 이미지 리포지토리를 생성합니다. 이 리포지토리에는 이전에 배포한 Apache 자리 표시자 이미지를 대체하는 자습서의 Docker 이미지가 저장됩니다.

Amazon ECR에 대한 자세한 내용은 Amazon Elastic Container Registry 사용 설명서를 참조합니다.

Amazon ECR에서 이미지 리포지토리를 생성하려면

1. 에서 Amazon ECR에 빈 리포지토리를 CodeCatalystECSUser AWS CloudShell 생성합니다.

```
aws ecr create-repository --repository-name codecatalyst-ecs-image-repo
```

2. Amazon ECR 리포지토리의 세부 정보를 표시합니다.

```
aws ecr describe-repositories \
    --repository-names codecatalyst-ecs-image-repo
```

3. “repositoryUri”: 값을 기록해 둡니다. 예시: 111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-ecs-image-repo.

나중에 워크플로에 리포지토리를 추가할 때 필요합니다.

4단계: AWS 역할 생성

이 섹션에서는 CodeCatalyst 워크플로가 작동하는 데 필요한 AWS IAM 역할을 생성합니다. 이러한 역할은 다음과 같습니다.

- 빌드 역할 - CodeCatalyst 빌드 작업(워크플로 내)에 AWS 계정에 액세스하고 Amazon ECR 및 Amazon EC2에 쓸 수 있는 권한을 부여합니다.
- 역할 배포 - CodeCatalyst ECS에 배포 작업(워크플로 내)에 AWS 계정, Amazon ECS 및 기타 몇 가지 AWS 서비스에 액세스할 수 있는 권한을 부여합니다.

IAM 역할에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 [IAM 역할](#) 섹션을 참조하세요.

Note

시간을 절약하기 위해 이전에 나열한 두 역할 대신

CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할이라

는 단일 역할을 생성할 수 있습니다. 자세한 내용은 [계정 및 스페이스의](#)

[CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 생성](#) 섹션을 참조하세요.

CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할에는 보안 위험을 초래할 수 있는 매우 광범위한 권한이 있음을 이해합니다. 보안에 대한 우려가 적은 자습서 및 시나리오에서만 이 역할을 사용하는 것이 좋습니다. 이 자습서에서는 이전에 나열된 두 역할을 생성하고 있다고 가정합니다.

빌드 및 배포 역할을 생성하려면 AWS Management Console 또는를 사용할 수 있습니다 AWS CLI.

AWS Management Console

빌드 및 배포 역할을 생성하려면 다음 절차 시리즈를 완료합니다.

빌드 역할을 생성하려면

1. 역할에 대한 정책을 다음과 같이 생성합니다.
 - a. 에 로그인합니다 AWS.
 - b. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
 - c. 탐색 창에서 Policies를 선택합니다.
 - d. 정책 생성을 선택합니다.
 - e. JSON 탭을 선택합니다.
 - f. 기존 코드를 삭제합니다.
 - g. 다음 코드를 붙여넣습니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:*",
        "ec2:*"
      ],
      "Resource": "*"
    }
  ]
}
```

 Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

- h. 다음: 태그를 선택합니다.
- i. 다음: 검토를 선택합니다.
- j. 이름에 다음과 같이 입력합니다.

```
codecatalyst-ecs-build-policy

```

- k. 정책 생성을 선택합니다.

이제 권한 정책을 생성했습니다.

2. 다음과 같이 빌드 역할을 생성합니다.
 - a. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
 - b. 사용자 지정 신뢰 정책을 선택합니다.

- c. 기존 사용자 지정 신뢰 정책을 삭제합니다.
- d. 다음 사용자 지정 신뢰 정책을 추가합니다.
- e. 다음을 선택합니다.
- f. 권한 정책에서 `codecatalyst-ecs-build-policy`를 검색하고 해당 확인란을 선택합니다.
- g. 다음을 선택합니다.
- h. 역할 이름에 다음과 같이 입력합니다.

codecatalyst-ecs-build-role

- i. 역할 설명에 다음과 같이 입력합니다.

CodeCatalyst ECS build role

- j. 역할 생성을 선택합니다.

이제 신뢰 정책 및 권한 정책으로 빌드 역할을 생성했습니다.

3. 다음과 같이 빌드 역할 ARN을 가져옵니다.
 - a. 탐색 창에서 Roles를 선택합니다.
 - b. 검색 상자에 방금 생성한 역할의 이름을 입력합니다(`codecatalyst-ecs-build-role`).
 - c. 목록에서 역할을 선택합니다.

역할의 요약 페이지가 나타납니다.

 - d. 상단에서 ARN 값을 복사합니다. 나중에 필요합니다.

배포 역할을 생성하려면

1. 역할에 대한 정책을 다음과 같이 생성합니다.
 - a. 에 로그인합니다 AWS.
 - b. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
 - c. 탐색 창에서 Policies를 선택합니다.
 - d. 정책 생성을 선택하세요.

- e. JSON 탭을 선택합니다.
- f. 기존 코드를 삭제합니다.
- g. 다음 코드를 붙여넣습니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Action": [
      "ecs:DescribeServices",
      "ecs:CreateTaskSet",
      "ecs>DeleteTaskSet",
      "ecs:ListClusters",
      "ecs:RegisterTaskDefinition",
      "ecs:UpdateServicePrimaryTaskSet",
      "ecs:UpdateService",
      "elasticloadbalancing:DescribeTargetGroups",
      "elasticloadbalancing:DescribeListeners",
      "elasticloadbalancing:ModifyListener",
      "elasticloadbalancing:DescribeRules",
      "elasticloadbalancing:ModifyRule",
      "lambda:InvokeFunction",
      "lambda:ListFunctions",
      "cloudwatch:DescribeAlarms",
      "sns:Publish",
      "sns:ListTopics",
      "s3:GetObject",
      "s3:GetObjectVersion",
      "codedeploy:CreateApplication",
      "codedeploy:CreateDeployment",
      "codedeploy:CreateDeploymentGroup",
      "codedeploy:GetApplication",
      "codedeploy:GetDeployment",
      "codedeploy:GetDeploymentGroup",
      "codedeploy:ListApplications",
      "codedeploy:ListDeploymentGroups",
      "codedeploy:ListDeployments",
      "codedeploy:StopDeployment",
      "codedeploy:GetDeploymentTarget",
      "codedeploy:ListDeploymentTargets",
      "codedeploy:GetDeploymentConfig",
```

```

    "codedeploy:GetApplicationRevision",
    "codedeploy:RegisterApplicationRevision",
    "codedeploy:BatchGetApplicationRevisions",
    "codedeploy:BatchGetDeploymentGroups",
    "codedeploy:BatchGetDeployments",
    "codedeploy:BatchGetApplications",
    "codedeploy:ListApplicationRevisions",
    "codedeploy:ListDeploymentConfigs",
    "codedeploy:ContinueDeployment"
  ],
  "Resource": "*",
  "Effect": "Allow"
}, {"Action": [
    "iam:PassRole"
  ],
  "Effect": "Allow",
  "Resource": "*",
  "Condition": {"StringLike": {"iam:PassedToService": [
    "ecs-tasks.amazonaws.com",
    "codedeploy.amazonaws.com"
  ]}
}
}]
}
```

 Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문의 와일드카드를 사용합니다. 그런 다음 사용 가능한 리소스 이름을 사용하여 정책의 범위를 좁힐 수 있습니다.

```
"Resource": "*"

```

- h. 다음: 태그를 선택합니다.
- i. 다음: 검토를 선택합니다.
- j. 이름에 다음과 같이 입력합니다.

codecatalyst-ecs-deploy-policy

- k. 정책 생성을 선택합니다.

이제 권한 정책을 생성했습니다.

- 2. 다음과 같이 배포 역할을 생성합니다.

- a. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
- b. 사용자 지정 신뢰 정책을 선택합니다.
- c. 기존 사용자 지정 신뢰 정책을 삭제합니다.
- d. 다음 사용자 지정 신뢰 정책을 추가합니다.
- e. 다음을 선택합니다.
- f. 권한 정책에서 `codecatalyst-ecs-deploy-policy`를 검색하고 해당 확인란을 선택합니다.
- g. 다음을 선택합니다.
- h. 역할 이름에 다음과 같이 입력합니다.

codecatalyst-ecs-deploy-role

- i. 역할 설명에 다음과 같이 입력합니다.

CodeCatalyst ECS deploy role

- j. 역할 생성을 선택합니다.

이제 신뢰 정책으로 배포 역할을 생성했습니다.

- 3. 다음과 같이 배포 역할 ARN을 가져옵니다.

- a. 탐색 창에서 Roles를 선택합니다.
- b. 검색 상자에 방금 생성한 역할의 이름을 입력합니다(`codecatalyst-ecs-deploy-role`).
- c. 목록에서 역할을 선택합니다.

역할의 요약 페이지가 나타납니다.

- d. 상단에서 ARN 값을 복사합니다. 나중에 필요합니다.

AWS CLI

빌드 및 배포 역할을 생성하려면 다음 절차 시리즈를 완료합니다.

두 역할에 대한 신뢰 정책을 생성하려면

CodeCatalystECSUser에서 로 신뢰 정책 파일을 AWS CloudShell생성합니다.

1. 다음 파일을 생성합니다.

```
cat > codecatalyst-ecs-trust-policy.json
```

2. 터미널 프롬프트에서 다음 코드를 붙여넣습니다.
3. 커서를 마지막 중괄호(}) 뒤에 놓습니다.
4. **Enter**를 누른 다음 **Ctrl+d**를 눌러 파일을 저장하고 cat을 종료합니다.

빌드 정책 및 빌드 역할을 생성하려면

1. 다음 빌드 정책을 생성합니다.
 - a. 의 CodeCatalystECSUser에서 빌드 정책 파일을 AWS CloudShell생성합니다.

```
cat > codecatalyst-ecs-build-policy.json
```

- b. 프롬프트에서 다음 코드를 입력합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:*",
        "ec2:*"
      ],
      "Resource": "*"
    }
  ]
}
```


}

- c. 커서를 마지막 중괄호(}) 뒤에 놓습니다.
- d. **Enter**를 누른 다음 **Ctrl+d**를 눌러 파일을 저장하고 cat을 종료합니다.

2. 빌드 정책을 AWS다음에 추가합니다.

```
aws iam create-policy \
  --policy-name codecatalyst-ecs-build-policy \
  --policy-document file:///codecatalyst-ecs-build-policy.json
```

- 3. 명령 출력에서 와 같은 "arn": 값을 기록해 둡니다(예: `arn:aws:iam::111122223333:policy/codecatalyst-ecs-build-policy`). 나중에 이 ARN이 필요합니다.
- 4. 빌드 역할을 생성하여 신뢰 정책에 연결합니다.

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

5. 빌드 정책을 빌드 역할에 연결합니다.

```
aws iam attach-role-policy \
  --role-name codecatalyst-ecs-build-role \
  --policy-arn arn:aws:iam::111122223333:policy/codecatalyst-ecs-build-policy
```

*arn:aws:iam::111122223333:policy/codecatalyst-ecs-build-policy*가 앞서 언급한 빌드 정책의 ARN으로 대체되는 경우.

6. 빌드 역할의 세부 정보를 표시합니다.

```
aws iam get-role \
  --role-name codecatalyst-ecs-build-role
```

- 7. 역할의 "Arn": 값을 기록해 둡니다. 예시: `arn:aws:iam::111122223333:role/codecatalyst-ecs-build-role`. 나중에 이 ARN이 필요합니다.

배포 정책을 생성하고 역할을 배포하려면

1. 배포 정책 생성:

- a. 에서 배포 정책 파일을 AWS CloudShell 생성합니다.

```
cat > codecatalyst-ecs-deploy-policy.json
```

- b. 프롬프트에서 다음 코드를 입력합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Action": [
      "ecs:DescribeServices",
      "ecs:CreateTaskSet",
      "ecs>DeleteTaskSet",
      "ecs:ListClusters",
      "ecs:RegisterTaskDefinition",
      "ecs:UpdateServicePrimaryTaskSet",
      "ecs:UpdateService",
      "elasticloadbalancing:DescribeTargetGroups",
      "elasticloadbalancing:DescribeListeners",
      "elasticloadbalancing:ModifyListener",
      "elasticloadbalancing:DescribeRules",
      "elasticloadbalancing:ModifyRule",
      "lambda:InvokeFunction",
      "lambda:ListFunctions",
      "cloudwatch:DescribeAlarms",
      "sns:Publish",
      "sns:ListTopics",
      "s3:GetObject",
      "s3:GetObjectVersion",
      "codedeploy:CreateApplication",
      "codedeploy:CreateDeployment",
      "codedeploy:CreateDeploymentGroup",
      "codedeploy:GetApplication",
      "codedeploy:GetDeployment",
      "codedeploy:GetDeploymentGroup",
      "codedeploy:ListApplications",
      "codedeploy:ListDeploymentGroups",
      "codedeploy:ListDeployments",
      "codedeploy:StopDeployment",
      "codedeploy:GetDeploymentTarget",
    ]
  }]
```

```

    "codedeploy:ListDeploymentTargets",
    "codedeploy:GetDeploymentConfig",
    "codedeploy:GetApplicationRevision",
    "codedeploy:RegisterApplicationRevision",
    "codedeploy:BatchGetApplicationRevisions",
    "codedeploy:BatchGetDeploymentGroups",
    "codedeploy:BatchGetDeployments",
    "codedeploy:BatchGetApplications",
    "codedeploy:ListApplicationRevisions",
    "codedeploy:ListDeploymentConfigs",
    "codedeploy:ContinueDeployment"
  ],
  "Resource": "*",
  "Effect": "Allow"
}, {"Action": [
    "iam:PassRole"
  ],
  "Effect": "Allow",
  "Resource": "*",
  "Condition": {"StringLike": {"iam:PassedToService": [
    "ecs-tasks.amazonaws.com",
    "codedeploy.amazonaws.com"
  ]
  }
}
}]
}
```

 Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

- c. 커서를 마지막 중괄호(`}`) 뒤에 놓습니다.
 - d. **Enter**를 누른 다음 **Ctrl+d**를 눌러 파일을 저장하고 cat을 종료합니다.
2. 배포 정책을 AWS다음에 추가합니다.

```
aws iam create-policy \
```

```
--policy-name codecatalyst-ecs-deploy-policy \
--policy-document file://codecatalyst-ecs-deploy-policy.json
```

- 명령 출력에서 와 같은 "arn": 값을 기록해 둡니다(예: `arn:aws:iam::111122223333:policy/codecatalyst-ecs-deploy-policy`). 나중에 이 ARN이 필요합니다.
- 빌드 역할을 생성하여 신뢰 정책에 연결합니다.

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

- 배포 정책을 배포 역할에 연결합니다. 여기서 `arn:aws:iam::111122223333:policy/codecatalyst-ecs-deploy-policy`는 앞서 언급한 배포 정책의 ARN으로 대체됩니다.

```
aws iam attach-role-policy \
  --role-name codecatalyst-ecs-deploy-role \
  --policy-arn arn:aws:iam::111122223333:policy/codecatalyst-ecs-deploy-policy
```

- 배포 역할의 세부 정보를 표시합니다.

```
aws iam get-role \
  --role-name codecatalyst-ecs-deploy-role
```

- 역할의 "Arn": 값을 기록해 둡니다. 예시: `arn:aws:iam::111122223333:role/codecatalyst-ecs-deploy-role`. 나중에 이 ARN이 필요합니다.

5단계: CodeCatalyst에 AWS 역할 추가

이 단계에서는 빌드 역할(`codecatalyst-ecs-build-role`)을 추가하고 스페이스의 CodeCatalyst 계정 연결에 역할(`codecatalyst-ecs-deploy-role`)을 배포합니다.

계정 연결에 빌드 및 배포 역할을 추가하려면

- CodeCatalyst에서 스페이스로 이동합니다.
- AWS 계정을 선택합니다. 계정 연결 목록이 나타납니다.
- 빌드 및 배포 역할을 생성한 계정을 나타내는 AWS 계정 연결을 선택합니다.
- 관리 콘솔에서 역할 AWS 관리를 선택합니다.

Amazon CodeCatalyst 스페이스에 IAM 역할 추가 페이지가 나타납니다. 페이지에 액세스하려면 로그인해야 할 수 있습니다.

5. IAM에서 생성한 기존 역할을 추가를 선택합니다.

드롭다운 목록이 나타납니다. 목록에는 `codecatalyst-runner.amazonaws.com` 및 `codecatalyst.amazonaws.com` 서비스 위탁자가 포함된 신뢰 정책이 있는 모든 IAM 역할이 표시됩니다.

6. 드롭다운 목록에서 `codecatalyst-ecs-build-role`을 선택하고 역할을 추가를 선택합니다.

Note

The security token included in the request is invalid가 표시되면 적절한 권한이 없기 때문일 수 있습니다. 이 문제를 해결하려면 CodeCatalyst 스페이스를 생성할 때 사용한 AWS 계정으로 다시 로그인 AWS 하면에서 로그아웃합니다.

7. IAM 역할을 추가를 선택하고 IAM에서 생성한 기존 역할을 추가를 선택한 다음 드롭다운 목록에서 `codecatalyst-ecs-deploy-role`을 선택합니다. [Add role]을 선택합니다.

이제 스페이스에 빌드 및 배포 역할을 추가했습니다.

8. Amazon CodeCatalyst 표시 이름의 값을 복사합니다. 워크플로를 생성할 때 나중에 이 값이 필요합니다.

6단계: 소스 리포지토리 생성

이 단계에서는 CodeCatalyst에 소스 리포지토리를 생성합니다. 이 리포지토리는 작업 정의 파일과 같은 자습서의 소스 파일을 저장합니다.

소스 리포지토리에 대한 자세한 정보는 [소스 리포지토리 생성](#) 섹션을 참조하세요.

소스 리포지토리를 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. `codecatalyst-ecs-project` 프로젝트로 이동합니다.
3. 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
4. 리포지토리 추가를 선택하고 리포지토리 생성을 선택합니다.
5. 리포지토리 이름에 다음과 같이 입력합니다.

codecatalyst-ecs-source-repository

6. 생성(Create)을 선택합니다.

7단계: 소스 파일 추가

이 섹션에서는 Hello World 소스 파일을 CodeCatalyst 리포지토리 `codecatalyst-ecs-source-repository`에 추가합니다. 다음과 같이 구성됩니다.

- `index.html` 파일 - 브라우저에 Hello World 메시지를 표시합니다.
- `Dockerfile` - Docker 이미지에 사용할 기본 이미지와 Docker 명령으로 적용할 기본 이미지를 설명합니다.
- `taskdef.json` 파일 - 클러스터에서 작업을 시작할 때 사용할 Docker 이미지를 정의합니다.

폴더의 구조는 다음과 같습니다.

```
.
├─ public-html
│   └─ index.html
├─ Dockerfile
└─ taskdef.json
```

Note

다음 지침은 CodeCatalyst 콘솔을 사용하여 파일을 추가하는 방법을 보여주지만 원하는 경우 Git을 사용할 수 있습니다. 자세한 내용은 [소스 리포지토리 복제](#)를 참조하세요.

주제

- [index.html](#)
- [Dockerfile](#)
- [taskdef.json](#)

index.html

`index.html` 파일에 브라우저에 Hello World 메시지가 표시됩니다.

index.html 파일을 추가하려면

1. CodeCatalyst 콘솔에서 `codecatalyst-ecs-source-repository` 소스 리포지토리로 이동합니다.
2. 파일에서 파일 생성을 선택합니다.
3. 파일 이름에 다음과 같이 입력합니다.

```
public-html/index.html
```

Important

동일한 이름의 폴더를 만들려면 `public-html/` 접두사를 포함해야 합니다.
`index.html`이 폴더에 있을 것으로 예상됩니다.

4. 다음 코드를 텍스트 상자에 입력합니다.

```
<html>
  <head>
    <title>Hello World</title>
    <style>
      body {
        background-color: black;
        text-align: center;
        color: white;
        font-family: Arial, Helvetica, sans-serif;
      }
    </style>
  </head>
  <body>
    <h1>Hello World</h1>
  </body>
</html>
```

5. 커밋을 선택한 다음 커밋을 다시 선택합니다.

`index.html`은 `public-html` 폴더의 리포지토리에 추가됩니다.

Dockerfile

Dockerfile은 사용할 기본 Docker 이미지와 적용할 Docker 명령을 설명합니다. Dockerfile에 대한 자세한 내용은 [Dockerfile 참조](#)를 참조하세요.

여기에 지정된 Dockerfile은 Apache 2.4 기본 이미지(httpd)를 사용함을 나타냅니다. 또한 index.html 소스 파일을 웹 페이지를 제공하는 Apache 서버의 폴더에 복사하는 방법도 포함되어 있습니다. Dockerfile의 EXPOSE 지침은 컨테이너가 포트 80에서 수신 중임을 Docker에게 알립니다.

Dockerfile을 추가하려면

1. 소스 리포지토리에서 파일 생성을 선택합니다.
2. 파일 이름에 다음과 같이 입력합니다.

Dockerfile

파일 확장자는 제외합니다.

Important

Dockerfile은 리포지토리의 루트 폴더에 있어야 합니다. 워크플로의 Docker build 명령은 해당 파일이 있을 것으로 예상합니다.

3. 다음 코드를 텍스트 상자에 입력합니다.

```
FROM httpd:2.4
COPY ./public-html/index.html /usr/local/apache2/htdocs/index.html
EXPOSE 80
```

4. 커밋을 선택한 다음 커밋을 다시 선택합니다.

Dockerfile이 리포지토리에 추가됩니다.

taskdef.json

이 단계에서 추가하는 taskdef.json 파일은 [2단계: Amazon ECS에 자리 표시자 애플리케이션 배포](#)에서 이미 지정한 파일과 동일하며 다음과 같은 차이점이 있습니다.

image: 필드(httpd:2.4)에 하드코딩된 Docker 이미지 이름을 지정하는 대신, 여기서 태스크 정의는 몇 가지 변수를 사용하여 이미지(\$REPOSITORY_URI 및 \$IMAGE_TAG)를 나타냅니다. 이러한 변수는 이후 단계에서 워크플로를 실행할 때 워크플로의 빌드 작업에서 생성된 실제 값으로 대체됩니다.

태스크 정의 파라미터에 대한 자세한 정보는 Amazon Elastic Container Service 개발자 안내서의 [태스크 정의 파라미터](#)를 참조하세요.

taskdef.json 파일을 추가하려면

1. 소스 리포지토리에서 파일 생성을 선택합니다.
2. 파일 이름에 다음과 같이 입력합니다.

taskdef.json

3. 다음 코드를 텍스트 상자에 입력합니다.

```
{
  "executionRoleArn": "arn:aws:iam::account_ID:role/codecatalyst-ecs-task-  
execution-role",
  "containerDefinitions": [
    {
      "name": "codecatalyst-ecs-container",
      # The $REPOSITORY_URI and $IMAGE_TAG variables will be replaced
      # by the workflow at build time (see the build action in the
      # workflow)
      "image": $REPOSITORY_URI:$IMAGE_TAG,
      "essential": true,
      "portMappings": [
        {
          "hostPort": 80,
          "protocol": "tcp",
          "containerPort": 80
        }
      ]
    }
  ],
  "requiresCompatibilities": [
    "FARGATE"
  ],
  "networkMode": "awsvpc",
  "cpu": "256",
  "memory": "512",
}
```

```
"family": "codecatalyst-ecs-task-def"
}
```

앞의 코드에서

arn:aws:iam::account_ID:role/codecatalyst-ecs-task-execution-role

[태스크 실행 역할을 생성하는 방법](#)에서 기록한 태스크 실행 역할의 ARN으로 바꿉니다.

4. 커밋을 선택한 다음 커밋을 다시 선택합니다.

taskdef.json 파일이 리포지토리에 추가됩니다.

8단계: 워크플로 생성 및 실행

이 단계에서는 소스 파일을 가져와 Docker 이미지로 빌드한 다음 Amazon ECS 클러스터에 이미지를 배포하는 워크플로를 생성합니다. 이 배포는 기존 Apache 자리 표시자 애플리케이션을 대체합니다.

워크플로는 순차적으로 실행되는 다음 구성 요소로 구성됩니다.

- 트리거 - 이 트리거는 소스 리포지토리에 변경 사항을 푸시할 때 워크플로 실행을 자동으로 시작합니다. 트리거에 대한 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 주제를 참조하세요.
- 빌드 작업(BuildBackend) - 트리거 시 작업은 Dockerfile을 사용하여 Docker 이미지를 빌드하고 Amazon ECR에 이미지를 푸시합니다. 빌드 작업은 또한 taskdef.json를 올바른 image 필드 값으로 업데이트한 다음 이 파일의 출력 아티팩트를 생성합니다. 이 아티팩트는 다음 배포 작업의 입력으로 사용됩니다.

빌드 작업에 대한 자세한 내용은 [워크플로로 빌드하기](#) 섹션을 참조하세요.

- 배포 작업(DeployToECS) - 빌드 작업이 완료되면 배포 작업은 빌드 작업(TaskDefArtifact)에서 생성된 출력 아티팩트를 찾고, 그 안에서 taskdef.json을 찾아 Amazon ECS 서비스에 등록합니다. 그런 다음 Amazon ECS 서비스는 taskdef.json 파일의 지침에 따라 Amazon ECS 클러스터 내에서 Amazon ECS 작업 및 연결된 Docker 컨테이너를 실행합니다.

워크플로 생성

1. CodeCatalyst 콘솔의 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
2. 워크플로 생성을 선택합니다.
3. 소스 리포지토리에서 codecatalyst-ecs-source-repository를 선택합니다.
4. 브랜치에서 main을 선택합니다.

5. 생성(Create)을 선택합니다.
6. YAML 샘플 코드를 삭제합니다.
7. 다음 YAML 코드를 추가합니다.

Note

다음 YAML 코드에서 원하는 경우 Connections: 섹션을 생략할 수 있습니다. 이 섹션을 생략하는 경우 환경의 기본 IAM 역할 필드에 지정된 역할에 [5단계: CodeCatalyst에 AWS 역할 추가](#)에 설명된 두 역할의 권한 및 신뢰 정책이 포함되어 있는지 확인해야 합니다. 기본 IAM 역할이 있는 환경 설정에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.

```
Name: codecatalyst-ecs-workflow
SchemaVersion: 1.0

Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  BuildBackend:
    Identifier: aws/build@v1
    Environment:
      Name: codecatalyst-ecs-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-ecs-build-role
    Inputs:
      Sources:
        - WorkflowSource
      Variables:
        - Name: REPOSITORY_URI
          Value: 111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-ecs-image-repo
        - Name: IMAGE_TAG
          Value: ${WorkflowSource.CommitId}
    Configuration:
      Steps:
        #pre_build:
          - Run: echo Logging in to Amazon ECR...
          - Run: aws --version
```

```

- Run: aws ecr get-login-password --region us-west-2 | docker login --
username AWS --password-stdin 111122223333.dkr.ecr.us-west-2.amazonaws.com
#build:
- Run: echo Build started on `date`
- Run: echo Building the Docker image...
- Run: docker build -t $REPOSITORY_URI:latest .
- Run: docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
#post_build:
- Run: echo Build completed on `date`
- Run: echo Pushing the Docker images...
- Run: docker push $REPOSITORY_URI:latest
- Run: docker push $REPOSITORY_URI:$IMAGE_TAG
# Replace the variables in taskdef.json
- Run: find taskdef.json -type f | xargs sed -i "s|\\$REPOSITORY_URI|
$REPOSITORY_URI|g"
- Run: find taskdef.json -type f | xargs sed -i "s|\\$IMAGE_TAG|$IMAGE_TAG|
g"
- Run: cat taskdef.json
# The output artifact will be a zip file that contains a task definition
file.
  Outputs:
    Artifacts:
      - Name: TaskDefArtifact
      Files:
        - taskdef.json
  DeployToECS:
    DependsOn:
      - BuildBackend
  Identifier: aws/ecs-deploy@v1
  Environment:
    Name: codecatalyst-ecs-environment
  Connections:
    - Name: codecatalyst-account-connection
      Role: codecatalyst-ecs-deploy-role
  Inputs:
    Sources: []
    Artifacts:
      - TaskDefArtifact
  Configuration:
    region: us-west-2
    cluster: codecatalyst-ecs-cluster
    service: codecatalyst-ecs-service
    task-definition: taskdef.json

```

앞의 코드에서

- `codecatalyst-ecs-environment` 인스턴스 두 개를 모두 [사전 조건](#)에서 생성한 환경 이름으로 바꿉니다.
- `codecatalyst-account-connection` 인스턴스 두 개를 모두 계정 연결의 표시 이름으로 바꿉니다. 표시 이름은 숫자일 수 있습니다. 자세한 내용은 [5단계: CodeCatalyst에 AWS 역할 추가](#) 섹션을 참조하세요.
- `codecatalyst-ecs-build-role`을 [4단계: AWS 역할 생성](#)에서 생성한 빌드 역할 이름으로 바꿉니다.
- `111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-ecs-image-repo`(Value: 속성 내)를 [3단계: Amazon ECR 이미지 리포지토리 생성](#)에서 생성한 Amazon ECR 리포지토리의 URI로 바꿉니다.
- `111122223333.dkr.ecr.us-west-2.amazonaws.com`(Run: aws ecr 명령 내)을 이미지 접미사(/codecatalyst-ecs-image-repo) 제외 Amazon ECR 리포지토리의 URI로 바꿉니다.
- `codecatalyst-ecs-deploy-role`을 [4단계: AWS 역할 생성](#)에서 생성한 배포 역할의 이름으로 바꿉니다.
- AWS 리전 코드가 있는 `us-west-2`의 두 인스턴스. 리전 코드 목록은 AWS 일반 참조의 [리전 엔드포인트](#)를 참조하세요.

Note

빌드 및 배포 역할을 생성하지 않기로 결정한 경우 `codecatalyst-ecs-build-role` 및 `codecatalyst-ecs-deploy-role`을 CodeCatalystWorkflowDevelopmentRole-`spaceName` 역할 이름으로 바꿉니다. 이에 대한 자세한 내용은 [4단계: AWS 역할 생성](#) 섹션을 참조하세요.

Tip

이전 워크플로 코드에 표시된 find 및 sed 명령을 사용하여 리포지토리 및 이미지 이름을 업데이트하는 대신 이 용도로 Amazon ECS 작업 정의 렌더링 작업을 사용할 수 있습니다. 자세한 내용은 [Amazon ECS 작업 정의 수정](#) 섹션을 참조하세요.

8. (선택 사항) 커밋하기 전에 YAML 코드가 유효한지 확인하려면 검증을 선택합니다.

9. 커밋을 선택합니다.
10. 워크플로 커밋 대화 상자에서 다음을 입력합니다.
 - a. 커밋 메시지에서 텍스트를 제거하고 다음을 입력합니다.

Add first workflow

- b. 리포지토리에서 `codecatalyst-ecs-source-repository`를 선택합니다.
- c. 브랜치 이름에서 기본을 선택합니다.
- d. 커밋을 선택합니다.

이제 워크플로를 생성했습니다. 워크플로 상단에 정의된 트리거로 인해 워크플로 실행이 자동으로 시작됩니다. 특히 `workflow.yaml` 파일을 소스 리포지토리에 커밋(및 푸시)할 때 트리거가 워크플로 실행을 시작했습니다.

진행 중인 워크플로 실행을 보려면

1. CodeCatalyst 콘솔의 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
2. 방금 생성한 `codecatalyst-ecs-workflow` 워크플로를 선택합니다.
3. BuildBackend를 선택하여 빌드 진행 상황을 확인합니다.
4. DeployToECS를 선택하여 배포 진행 상황을 확인합니다.

실행 세부 정보 보기에 대한 자세한 내용은 [워크플로 실행 상태 및 세부 정보 보기](#)를 참조하세요.

배포를 확인하려면

1. <https://console.aws.amazon.com/ecs/>에서 Amazon ECS 클래식 콘솔을 엽니다.
2. `codecatalyst-ecs-cluster` 클러스터를 선택합니다.
3. 작업 탭을 선택합니다.
4. 세 가지 태스크 중 하나를 선택합니다.
5. 퍼블릭 IP 필드에서 개방형 주소를 선택합니다.

브라우저에 "Hello World" 페이지가 나타나 Amazon ECS 서비스가 애플리케이션을 성공적으로 배포했음을 나타냅니다.

9단계: 소스 파일 변경

이 섹션에서는 소스 리포지토리의 `index.html` 파일을 변경합니다. 이 변경으로 인해 워크플로는 새 Docker 이미지를 빌드하고 커밋 ID로 태그를 지정하고 Amazon ECR에 푸시한 다음 Amazon ECS에 배포합니다.

`index.html`을 변경하려면

1. CodeCatalyst 콘솔의 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택하고 `codecatalyst-ecs-source-repository` 리포지토리를 선택합니다.
2. `[public-html]`를 선택한 다음 `[index.html]`를 선택합니다.

`index.html`의 내용이 나타납니다.

3. 편집을 선택합니다.
4. 14행에서 Hello World 텍스트를 Tutorial complete!로 변경합니다.
5. 커밋을 선택한 다음 커밋을 다시 선택합니다.

커밋으로 인해 새 워크플로 실행이 시작됩니다.

6. (선택 사항) 소스 리포지토리의 메인 페이지로 이동하여 커밋 보기를 선택한 다음 `index.html` 변경 사항에 대한 커밋 ID를 기록해 둡니다.
7. 배포 진행 상황 보기:
 - a. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
 - b. 최신 실행을 보려면 `codecatalyst-ecs-workflow`를 선택합니다.
 - c. BuildBackend 및 DeployToECS를 선택하여 워크플로 실행 진행 상황을 확인합니다.
8. 다음과 같이 애플리케이션이 업데이트되었는지 확인합니다.
 - a. <https://console.aws.amazon.com/ecs/>에서 Amazon ECS 클래식 콘솔을 엽니다.
 - b. `codecatalyst-ecs-cluster` 클러스터를 선택합니다.
 - c. 작업 탭을 선택합니다.
 - d. 세 가지 태스크 중 하나를 선택합니다.
 - e. 퍼블릭 IP 필드에서 개방형 주소를 선택합니다.

Tutorial complete! 페이지가 나타납니다.

9. (선택 사항)에서 Amazon ECR 콘솔로 AWS 전환하고 새 Docker 이미지에 6단계의 커밋 ID로 태그가 지정되었는지 확인합니다.

정리

요금이 부과되지 않도록 이 자습서에서 사용하는 파일과 서비스를 정리합니다.

에서 다음 순서로 AWS Management Console정리합니다.

1. Amazon ECS에서 다음을 수행합니다.
 - a. `codecatalyst-ecs-service`을 삭제합니다.
 - b. `codecatalyst-ecs-cluster`을 삭제합니다.
 - c. `codecatalyst-ecs-task-definition` 등록을 취소합니다.
2. Amazon ECR에서 `codecatalyst-ecs-image-repo`를 삭제합니다.
3. Amazon EC2에서 `codecatalyst-ecs-security-group`를 삭제합니다.
4. IAM Identity Center에서 다음을 삭제합니다.
 - a. `CodeCatalystECSUser`
 - b. `CodeCatalystECSPermissionSet`

CodeCatalyst 콘솔에서 다음과 같이 정리합니다.

1. `codecatalyst-ecs-workflow`를 삭제합니다.
2. `codecatalyst-ecs-environment`를 삭제합니다.
3. `codecatalyst-ecs-source-repository`를 삭제합니다.
4. `codecatalyst-ecs-project`를 삭제합니다.

이 자습서에서는 CodeCatalyst 워크플로와 Amazon ECS에 배포 작업을 사용하여 Amazon ECS 서비스에 애플리케이션을 배포하는 방법을 배웠습니다.

'Amazon ECS에 배포' 작업 추가

다음 지침을 사용하여 워크플로에 Amazon ECS에 배포 작업을 추가합니다.

Visual

시각적 편집기를 사용하여 'Amazon ECS에 배포' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.

3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.
8. 드롭다운 목록에서 Amazon CodeCatalyst를 선택합니다.
9. Amazon ECS에 배포 작업을 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
- 또는
 - Amazon ECS에 배포를 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 다운로드를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
10. 입력 및 구성 탭에서 필요에 따라 필드를 작성합니다. 각 필드의 설명은 '[Amazon ECS에 배포](#)' [작업 YAML](#) 섹션을 참조하세요. 이 참조는 YAML 및 시각적 편집기 모두에 나타나는 각 필드(및 해당 YAML 속성 값)에 대한 자세한 정보를 제공합니다.
11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 'Amazon ECS에 배포' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.

8. 드롭다운 목록에서 Amazon CodeCatalyst를 선택합니다.
9. Amazon ECS에 배포 작업을 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
 또는
 - Amazon ECS에 배포를 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 다운로드를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
10. 필요에 따라 YAML 코드의 속성을 수정합니다. 사용 가능한 각 속성에 대한 설명은 '[Amazon ECS에 배포' 작업 YAML](#)에서 볼 수 있습니다.
11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

'Amazon ECS에 배포' 변수

Amazon ECS에 배포 작업은 런타임에 다음 변수를 생성하고 설정합니다. 이를 사전 정의된 변수라고 합니다.

워크플로에서 이러한 변수를 참조하는 방법에 대한 자세한 내용은 [사전 정의된 변수 사용](#) 섹션을 참조하세요.

Key(키)	값
클러스터	워크플로 실행 중에 배포된 Amazon ECS 클러스터의 이름입니다. 예시: codecatalyst-ecs-cluster
deployment-platform	배포 플랫폼의 이름입니다. AWS: ECS로 하드코딩되었습니다.
서비스	워크플로 실행 중에 배포된 Amazon ECS 서비스의 이름입니다. 예시: codecatalyst-ecs-service

Key(키)	값
task-definition-arn	워크플로 실행 중에 등록된 작업 정의의 Amazon 리소스 이름(ARN)입니다. 예시: <code>arn:aws:ecs:us-west-2:11112223333:task-definition/cod ecatalyst-task-def:8</code> 위 예시의 :8은 등록된 리비전을 나타냅니다.
deployment-url	워크플로 실행과 연결된 Amazon ECS 배포의 세부 정보를 볼 수 있는 Amazon ECS 콘솔의 이벤트 탭에 대한 링크입니다. 예시: <code>https://console.aws.amazon.com/ecs/home?region=us-west-2#/clusters/codecatalyst-ecs-cluster/services/codecatalyst-ecs-service/events</code>
리전	워크플로 실행 중에 배포 AWS 리전 정의의 리전 코드입니다. 예시: <code>us-west-2</code>

'Amazon ECS에 배포' 작업 YAML

다음은 Amazon ECS에 배포 작업의 YAML 정의입니다. 이러한 작업 사용 방법을 배우려면 [워크플로를 사용하여 Amazon ECS에 배포](#) 섹션을 참조하세요.

이 작업 정의는 더 광범위한 워크플로 정의 파일 내의 섹션으로 존재합니다. 이 파일에 대한 자세한 내용은 [워크플로 YAML 정의](#)를 참조합니다.

Note

이어지는 대부분의 YAML 속성에는 시각적 편집기에 해당 UI 요소가 있습니다. UI 요소를 찾으려면 Ctrl+F를 사용합니다. 요소가 연결된 YAML 속성과 함께 나열됩니다.

```

# The workflow definition starts here.
# See ### ## for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
ECSDeployAction\_nn:
  Identifier: aws/ecs-deploy@v1
  DependsOn:
    - build-action
  Compute:
    Type: EC2 | Lambda
    Fleet: fleet-name
    Timeout: timeout-minutes
  Environment:
    Name: environment-name
  Connections:
    - Name: account-connection-name
      Role: iam-role-name
  Inputs:
    # Specify a source or an artifact, but not both.
    Sources:
      - source-name-1
    Artifacts:
      - task-definition-artifact
  Configuration:
    region: us-east-1
    cluster: ecs-cluster
    service: ecs-service
    task-definition: task-definition-path
    force-new-deployment: false|true
    codedeploy-appspec: app-spec-file-path
    codedeploy-application: application-name
    codedeploy-deployment-group: deployment-group-name
    codedeploy-deployment-description: deployment-description

```

ECSDeployAction

(필수)

작업 이름을 지정합니다. 워크플로 내의 모든 작업 이름은 고유해야 합니다. 작업 이름은 영숫자 문자 (a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 작업 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

기본값: ECSDeployAction_nn.

해당 UI: 구성 탭/작업 표시 이름

Identifier

(*ECSDeployAction*/Identifier)

(필수)

작업을 식별합니다. 버전을 변경하려는 경우가 아니면 이 속성을 변경하지 마세요. 자세한 내용은 [사용할 작업 버전 지정](#) 섹션을 참조하세요.

기본값: aws/ecs-deploy@v1.

해당 UI: 워크플로 다이어그램/ECSDeployAction_nn/aws/ecs-deploy@v1 레이블

DependsOn

(*ECSDeployAction*/DependsOn)

(선택 사항)

이 작업을 실행하기 위해 성공적으로 실행해야 하는 작업, 작업 그룹 또는 게이트를 지정합니다.

'depends on' 함수에 대한 자세한 내용은 [작업 순서 지정](#) 섹션을 참조하세요.

해당 UI: 입력 탭/의존 - 선택 사항

Compute

(*ECSDeployAction*/Compute)

(선택 사항)

워크플로 작업을 실행하는 데 사용되는 컴퓨팅 엔진입니다. 워크플로 수준 또는 작업 수준에서 컴퓨팅을 지정할 수 있지만 둘 다 지정할 수는 없습니다. 워크플로 수준에서 지정하면 컴퓨팅 구성이 워크플로에 정의된 모든 작업에 적용됩니다. 워크플로 수준에서는 동일한 인스턴스에서 여러 작업을 실행할 수도 있습니다. 자세한 내용은 [작업 간에 컴퓨팅 공유](#) 섹션을 참조하세요.

해당 UI: 없음

Type

(*ECSDeployAction*/Compute/Type)

([Compute](#) 포함 시 필수)

컴퓨팅 엔진의 유형입니다. 다음 값 중 하나를 사용할 수 있습니다.

- EC2(시각 편집기) 또는 EC2(YAML 편집기)
작업 실행 중 유연성을 위해 최적화되었습니다.
- Lambda(시각 편집기) 또는 Lambda(YAML 편집기)
작업 시작 속도를 최적화했습니다.

컴퓨팅 유형에 대한 자세한 정보는 [컴퓨팅 유형](#)을 참고하세요.

해당 UI: 구성 탭/고급 - 선택 사항/컴퓨팅 유형

Fleet

(*ECSDeployAction*/Compute/Fleet)

(선택 사항)

워크플로 또는 워크플로 작업을 실행할 시스템 또는 플릿을 지정합니다. 온디맨드 플릿의 경우 작업이 시작되면 워크플로가 필요한 리소스를 프로비저닝하고 작업이 완료되면 시스템이 파괴됩니다. 온디맨드 플릿의 예시: Linux.x86-64.Large, Linux.x86-64.XLarge. 온디맨드 플릿에 대한 자세한 내용은 [온디맨드 플릿 속성](#) 섹션을 참조하세요.

프로비저닝된 플릿을 사용하면 워크플로 작업을 실행하도록 전용 시스템 세트를 구성할 수 있습니다. 이러한 시스템은 유휴 상태로 유지되므로 작업을 즉시 처리할 수 있습니다. 프로비저닝된 플릿에 대한 자세한 내용은 [프로비저닝된 플릿 속성](#) 섹션을 참조하세요.

Fleet 생략 시 기본값은 Linux.x86-64.Large입니다.

해당 UI: 구성 탭/고급 - 선택적/컴퓨팅 플릿

Timeout

(*ECSDeployAction*/Timeout)

(선택 사항)

CodeCatalyst가 작업을 종료하기 전에 작업을 실행할 수 있는 시간을 분(YAML 편집기) 또는 시간 및 분(시각적 편집기) 단위로 지정합니다. 최소값은 5분이고 최대값은 [CodeCatalyst의 워크플로 할당량](#)에 설명되어 있습니다. 기본 제한 시간은 최대 제한 시간과 동일합니다.

해당 UI: 구성 탭/제한 시간 - 선택 사항

Environment

(*ECSDeployAction*/Environment)

(필수)

작업에 사용할 CodeCatalyst 환경을 지정합니다. 작업은 선택한 환경에 지정된 AWS 계정 및 선택적 Amazon VPC에 연결됩니다. 작업은 환경에 지정된 기본 IAM 역할을 사용하여 연결하고 [Amazon VPC 연결](#)에 지정된 IAM 역할을 AWS 계정으로 사용하여 Amazon VPC에 연결합니다.

Note

기본 IAM 역할에 작업에 필요한 권한이 없는 경우 다른 역할을 사용하도록 작업을 구성할 수 있습니다. 자세한 내용은 [작업의 IAM 역할 변경](#) 섹션을 참조하세요.

환경에 대한 자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 및 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 구성 탭/환경

Name

(*ECSDeployAction*/Environment/Name)

([Environment](#) 포함 시 필수)

작업과 연결하려는 기존 환경의 이름을 지정합니다.

해당 UI: 구성 탭/환경

Connections

(*ECSDeployAction*/Environment/Connections)

(최신 버전의 작업에서는 선택 사항, 이전 버전에서는 필수)

작업과 연결할 계정 연결을 지정합니다. Environment에서 계정 연결을 최대 1개까지 지정할 수 있습니다.

계정 연결을 지정하지 않는 경우:

- 작업은 CodeCatalyst 콘솔의 환경에 지정된 AWS 계정 연결 및 기본 IAM 역할을 사용합니다. 환경에 계정 연결 및 기본 IAM 역할을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.
- 기본 IAM 역할에는 작업에 필요한 정책 및 권한이 포함되어야 합니다. 이러한 정책 및 권한이 무엇인지 확인하려면 작업의 YAML 정의 설명서에서 역할 속성에 대한 설명을 참조하세요.

계정 연결에 대한 자세한 정보는 [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 섹션을 참조하세요. 환경에 계정 연결을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/AWS 계정 연결

Name

(*ECSDeployAction*/Environment/Connections/Name)

([Connections](#) 포함 시 필수)

계정 연결의 이름을 지정합니다.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/AWS 계정 연결

Role

(*ECSDeployAction*/Environment/Connections/Role)

([Connections](#) 포함 시 필수)

Amazon ECS에 배포 작업이 AWS에 액세스하는 데 사용하는 IAM 역할의 이름을 지정합니다. [CodeCatalyst 스페이스에 역할을 추가](#)했고 역할에 다음 정책이 포함되어 있는지 확인합니다.

IAM 역할을 지정하지 않으면 작업은 CodeCatalyst 콘솔의 [환경](#)에 나열된 기본 IAM 역할을 사용합니다. 환경에서 기본 역할을 사용하는 경우 다음 정책이 있는지 확인합니다.

- 다음 권한 정책:

 Warning

다음 정책에 표시된 대로 권한을 제한합니다. 더 광범위한 권한을 가진 역할을 사용하면 보안 위험이 발생할 수 있습니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Action": [
      "ecs:DescribeServices",
      "ecs:CreateTaskSet",
      "ecs>DeleteTaskSet",
      "ecs:ListClusters",
      "ecs:RegisterTaskDefinition",
      "ecs:UpdateServicePrimaryTaskSet",
      "ecs:UpdateService",
      "elasticloadbalancing:DescribeTargetGroups",
      "elasticloadbalancing:DescribeListeners",
      "elasticloadbalancing:ModifyListener",
      "elasticloadbalancing:DescribeRules",
      "elasticloadbalancing:ModifyRule",
      "lambda:InvokeFunction",
      "lambda:ListFunctions",
      "cloudwatch:DescribeAlarms",
      "sns:Publish",
      "sns:ListTopics",
      "s3:GetObject",
      "s3:GetObjectVersion",
      "codedeploy:CreateApplication",
      "codedeploy:CreateDeployment",
      "codedeploy:CreateDeploymentGroup",
```

```

        "codedeploy:GetApplication",
        "codedeploy:GetDeployment",
        "codedeploy:GetDeploymentGroup",
        "codedeploy:ListApplications",
        "codedeploy:ListDeploymentGroups",
        "codedeploy:ListDeployments",
        "codedeploy:StopDeployment",
        "codedeploy:GetDeploymentTarget",
        "codedeploy:ListDeploymentTargets",
        "codedeploy:GetDeploymentConfig",
        "codedeploy:GetApplicationRevision",
        "codedeploy:RegisterApplicationRevision",
        "codedeploy:BatchGetApplicationRevisions",
        "codedeploy:BatchGetDeploymentGroups",
        "codedeploy:BatchGetDeployments",
        "codedeploy:BatchGetApplications",
        "codedeploy:ListApplicationRevisions",
        "codedeploy:ListDeploymentConfigs",
        "codedeploy:ContinueDeployment"
    ],
    "Resource": "*",
    "Effect": "Allow"
  }, {
    "Action": [
      "iam:PassRole"
    ],
    "Effect": "Allow",
    "Resource": "*",
    "Condition": {
      "StringLike": {
        "iam:PassedToService": [
          "ecs-tasks.amazonaws.com",
          "codedeploy.amazonaws.com"
        ]
      }
    }
  }
}]
}

```

Note

역할을 처음 사용할 때는 리소스 정책 설명에서 다음 와일드카드를 사용한 다음 사용 가능한 리소스 이름으로 정책의 범위를 좁힙니다.

```
"Resource": "*"

```

- 다음 사용자 지정 신뢰 정책:

Note

원하는 경우 이 작업에서 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 사용할 수 있습니다. 이에 대한 자세한 내용은 [계정 및 스페이스의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 생성](#) 섹션을 참조하세요.

CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할에 보안 위험을 초래할 수 있는 전체 액세스 권한이 있음을 이해합니다. 보안에 대한 우려가 적은 자습서 및 시나리오에서만 이 역할을 사용하는 것이 좋습니다.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/# ##의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/역할

Inputs

(*ECSDeployAction*/Inputs)

(선택 사항)

이 Inputs 섹션에서는 워크플로 실행 중 ECSDeployAction에 필요한 데이터를 정의합니다.

Note

Amazon ECS에 배포 작업당 하나의 입력(소스 또는 아티팩트)만 허용됩니다.

해당 UI: 입력 탭

Sources

(*ECSDeployAction*/Inputs/Sources)

(작업 정의 파일이 소스 리포지토리에 저장된 경우 필수)

태스크 정의 파일이 소스 리포지토리에 저장되어 있는 경우 해당 소스 리포지토리의 레이블을 지정합니다. 현재, WorkflowSource 레이블만 지원됩니다.

태스크 정의 파일이 소스 리포지토리에 포함되어 있지 않은 경우, 다른 작업에서 생성된 아티팩트에 있어야 합니다.

소스에 대한 자세한 내용은 [워크플로에 소스 리포지토리 연결](#) 섹션을 참조하세요.

해당 UI: 입력 탭/소스 - 선택 사항

Artifacts - input

(*ECSDeployAction*/Inputs/Artifacts)

(작업 정의 파일이 이전 작업의 [출력 아티팩트](#)에 저장된 경우 필수)

배포하려는 작업 정의 파일이 이전 작업에서 생성된 아티팩트에 포함된 경우 여기에 해당 아티팩트를 지정합니다. 태스크 정의 파일이 아티팩트 내에 포함되어 있지 않은 경우 소스 리포지토리에 있어야 합니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

해당 UI: 구성 탭/아티팩트 - 선택 사항

Configuration

(*ECSDeployAction*/Configuration)

(필수)

작업의 구성 속성을 정의할 수 있는 섹션입니다.

해당 UI: 구성 탭

region

(Configuration/region)

(필수)

Amazon ECS 클러스터 및 서비스가 상주하는 AWS 리전을 지정합니다. 리전 코드 목록은 AWS 일반 참조의 [리전 엔드포인트](#)를 참조하세요.

해당 UI: 구성 탭/리전

cluster

(*ECSDeployAction*/Configuration/cluster)

(필수)

기존 Amazon ECS 클러스터의 이름을 지정합니다. Amazon ECS에 배포 작업을 수행하면 컨테이너화된 애플리케이션이 이 클러스터에 작업으로 배포됩니다. 자세한 내용을 알아보려면 Amazon Elastic Container Service 개발자 안내서에서 [클러스터](#)를 참조하세요.

해당 UI: 구성 탭/클러스터

service

(*ECSDeployAction*/Configuration/service)

(필수)

작업 정의 파일을 인스턴스화할 기존 Amazon ECS 서비스의 이름을 지정합니다. 이 서비스는 `cluster` 필드에 지정된 클러스터 아래에 있어야 합니다. Amazon ECS에 대한 자세한 내용은 Amazon Elastic Container Service 개발자 안내서의 [Amazon ECS 서비스](#)를 참조하세요.

해당 UI: 구성 탭/서비스

task-definition

(*ECSDeployAction*/Configuration/task-definition)

(필수)

기존 작업 정의 파일의 경로를 지정합니다. 파일이 소스 리포지토리에 있는 경우 경로는 소스 리포지토리 루트 폴더를 기준으로 합니다. 파일이 이전 워크플로 작업의 아티팩트에 있는 경우 경로는 아티팩트 루트 폴더를 기준으로 합니다. 자세한 내용을 알아보려면 Amazon Elastic Container Service 개발자 안내서의 [작업 정의](#)를 참조하세요.

해당 UI: 구성 탭/작업 정의

force-new-deployment

(*ECSDeployAction*/Configuration/force-new-deployment)

(필수)

활성화된 경우 Amazon ECS 서비스는 서비스 정의 변경 없이 새 배포를 시작할 수 있습니다. 배포를 강제하면 서비스가 현재 실행 중인 모든 작업을 중지하고 새 작업을 시작합니다. 새 배포 강제 적용에 대한 자세한 내용은 Amazon Elastic Container Service 개발자 안내서의 [서비스 업데이트](#)를 참조하세요.

기본값: false

해당 UI: 구성 탭/서비스의 새 배포 강제 적용

codedeploy-appspec

(*ECSDeployAction*/Configuration/codedeploy-appspec)

(블루/그린 배포를 사용하도록 Amazon ECS 서비스를 구성한 경우 필수, 그렇지 않으면 생략)

기존 CodeDeploy 애플리케이션 사양(AppSpec) 파일의 이름과 경로를 지정합니다. 이 파일은 CodeCatalyst 소스 리포지토리의 루트에 있어야 합니다. AppSpec 파일에 대한 자세한 내용은 AWS CodeDeploy 사용 설명서의 [CodeDeploy 애플리케이션 사양\(AppSpec\) 파일](#)을 참조하세요.

Note

블루/그린 배포를 수행하도록 Amazon ECS 서비스를 구성한 경우에만 CodeDeploy 정보를 제공합니다. 롤링 업데이트 배포(기본값)의 경우 CodeDeploy 정보를 생략합니다. Amazon ECS 배포에 대해 자세한 내용을 알아보려면 Amazon Elastic Container Service 개발자 안내서의 [Amazon ECS 배포 유형](#)을 참조하세요.

Note

CodeDeploy 필드는 시각적 편집기에 숨겨질 수 있습니다. 해당 항목이 표시되도록 하려면 [시각적 편집기에서 CodeDeploy 필드가 누락된 이유는 무엇인가요?](#) 섹션을 참조하세요.

해당 UI: 구성 탭/CodeDeploy AppSpec

codedeploy-application

(*ECSDeployAction*/Configuration/codedeploy-application)

(codedeploy-appspec 포함 시 필수)

기존 CodeDeploy 애플리케이션의 이름을 지정합니다. CodeDeploy 애플리케이션에 대한 자세한 내용은 AWS CodeDeploy 사용 설명서의 [CodeDeploy에서 애플리케이션 작업](#)을 참조하세요.

해당 UI: 구성 탭/CodeDeploy 애플리케이션

codedeploy-deployment-group

(*ECSDeployAction*/Configuration/codedeploy-deployment-group)

(codedeploy-appspec 포함 시 필수)

기존 CodeDeploy 배포 그룹의 이름을 지정합니다. CodeDeploy 배포 그룹에 대한 자세한 내용은 AWS CodeDeploy 사용 설명서의 [CodeDeploy에서 배포 그룹 작업](#)을 참조하세요.

해당 UI: 구성 탭/CodeDeploy 배포 그룹

codedeploy-deployment-description

(*ECSDeployAction*/Configuration/codedeploy-deployment-description)

(선택 사항)

이 작업에서 생성할 배포에 대한 설명을 지정합니다. 자세한 정보는 AWS CodeDeploy 사용자 가이드의 [CodeDeploy에서 배포로 작업](#)을 참조하세요.

해당 UI: 구성 탭/CodeDeploy 배포 설명

워크플로를 사용하여 Amazon EKS에 배포

Tip

Kubernetes 클러스터에 배포 작업을 사용하는 방법을 보여주는 자습서는 [자습서: Amazon EKS에 애플리케이션 배포](#) 섹션을 참조하세요.

이 섹션에서는 CodeCatalyst 워크플로를 사용하여 컨테이너화된 애플리케이션을 Kubernetes 클러스터에 배포하는 방법을 설명합니다. 이렇게 하려면 워크플로에 Kubernetes 클러스터에 배포 작업을 추가해야 합니다. 이 작업은 하나 이상의 Kubernetes 매니페스트 파일을 사용하여 Amazon Elastic Kubernetes Service(EKS)에서 설정한 Kubernetes 클러스터에 애플리케이션을 배포합니다. 샘플 매니페스트는 [자습서: Amazon EKS에 애플리케이션 배포](#)의 [deployment.yaml](#) 섹션을 참조하세요.

Kubernetes에 대한 자세한 내용은 [Kubernetes 문서](#)를 참조하세요.

Amazon EKS에 관한 자세한 내용은 Amazon EKS 사용 설명서의 [Amazon EKS란?](#)을 참조하세요.

주제

- ['Kubernetes 클러스터에 배포' 작업 작동 방식](#)
- ['Amazon EKS에 배포' 작업에서 사용되는 런타임 이미지](#)
- [자습서: Amazon EKS에 애플리케이션 배포](#)
- ['Kubernetes 클러스터에 배포' 작업 추가](#)
- ['Kubernetes 클러스터에 배포' 변수](#)
- ['Kubernetes 클러스터에 배포' 작업 YAML](#)

'Kubernetes 클러스터에 배포' 작업 작동 방식

Kubernetes 클러스터에 배포는 다음과 같이 작동합니다.

1. 런타임 시 작업은 작업이 실행 중인 CodeCatalyst 컴퓨팅 시스템에 Kubernetes kubectl 유틸리티를 설치합니다. 작업을 구성할 때 제공한 Amazon EKS 클러스터를 가리키도록 작업이 kubectl을 구성합니다. 다음에는 kubectl apply 명령을 실행하는 데 kubectl 유틸리티가 필요합니다.
2. 이 작업은 *my-manifest.yaml*의 kubectl apply -f *my-manifest.yaml* 명령을 실행하여 애플리케이션을 컨테이너 및 포드 세트로 구성된 클러스터에 배포합니다. 이 명령에 대한 자세한 내용은 Kubernetes 참조 설명서의 [kubectl 적용](#) 주제를 참조하세요.

'Amazon EKS에 배포' 작업에서 사용되는 런타임 이미지

Amazon EKS에 배포 작업은 [2022년 11월 이미지](#)에서 실행됩니다. 자세한 내용은 [활성 이미지](#) 섹션을 참조하세요.

자습서: Amazon EKS에 애플리케이션 배포

이 자습서에서는 Amazon CodeCatalyst 워크플로, Amazon EKS 및 기타 몇 가지 서비스를 사용하여 컨테이너화된 애플리케이션을 Amazon Elastic Kubernetes AWS Service에 배포하는 방법을 알아봅니다. 배포된 애플리케이션은 간단한 'Hello, World!'입니다. Apache 웹 서버 Docker 이미지를 기반으로 구축된 웹 사이트입니다. 이 자습서에서는 개발 머신 및 Amazon EKS 클러스터 설정과 같은 필수 준비 작업을 안내한 다음 애플리케이션을 빌드하고 클러스터에 배포하기 위한 워크플로를 생성하는 방법을 설명합니다.

초기 배포가 완료되면 자습서에서 애플리케이션 소스를 변경하도록 지시합니다. 이 변경으로 인해 새 Docker 이미지가 빌드되고 새 개정 정보와 함께 Docker 이미지 리포지토리로 푸시됩니다. 그런 다음 Docker 이미지의 새 개정이 Amazon EKS에 배포됩니다.

Tip

이 자습서를 진행하는 대신 전체 Amazon EKS 설정을 수행하는 블루프린트를 사용할 수 있습니다. EKS 앱 배포 블루프린트를 사용해야 합니다. 자세한 내용은 [블루프린트를 사용하여 프로젝트 생성](#) 섹션을 참조하세요.

주제

- [사전 조건](#)
- [1단계: 개발 머신 설정](#)
- [2단계: Amazon EKS 클러스터 생성](#)
- [3단계: Amazon ECR 이미지 리포지토리 생성](#)
- [4단계: 소스 파일 추가](#)
- [5단계: AWS 역할 생성](#)
- [6단계: CodeCatalyst에 AWS 역할 추가](#)
- [7단계: ConfigMap 업데이트](#)
- [8단계: 워크플로 생성 및 실행](#)
- [9단계: 소스 파일 변경](#)
- [정리](#)

사전 조건

이 자습서를 시작하기 전에 다음을 수행합니다.

- 연결된 AWS 계정이 있는 Amazon CodeCatalyst 스페이스가 필요합니다. 자세한 내용은 [스페이스 생성](#) 단원을 참조하십시오.
- 스페이스에는 다음과 같은 빈 프로젝트가 필요합니다.

```
codecatalyst-eks-project
```

처음부터 시작 옵션을 사용하여 이 프로젝트를 생성합니다.

자세한 내용은 [Amazon CodeCatalyst에서 빈 프로젝트 생성](#) 섹션을 참조하세요.

- 프로젝트에는 다음과 같은 빈 CodeCatalyst 소스 리포지토리가 필요합니다.

```
codecatalyst-eks-source-repository
```

자세한 내용은 [CodeCatalyst의 소스 리포지토리로 코드 저장 및 협업](#) 섹션을 참조하세요.

- 프로젝트에는 다음과 같은 CodeCatalyst CI/CD 환경(개발 환경 아님)이 필요합니다.

```
codecatalyst-eks-environment
```

다음과 같이 이 환경을 구성합니다.

- 비프로덕션과 같은 유형을 선택합니다.
- AWS 계정에 연결합니다.
- 기본 IAM 역할의 경우 아무 역할이나 선택합니다. 나중에 다른 역할을 지정합니다.

자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 섹션을 참조하세요.

1단계: 개발 머신 설정

이 자습서의 첫 번째 단계는 이 자습서 전체에서 사용할 몇 가지 도구를 사용하여 개발 머신을 구성하는 것입니다. 이러한 도구는 다음과 같습니다.

- eksctl 유틸리티 - 클러스터 생성용
- kubectl 유틸리티 - eksctl의 사전 조건
- AWS CLI -에 대한 사전 조건이기도 합니다. eksctl

이러한 도구가 있는 경우 기존 개발 시스템에 설치하거나 클라우드 기반 CodeCatalyst 개발 환경을 사용할 수 있습니다. CodeCatalyst 개발 환경의 이점은 스핀 업 및 테이크 다운이 쉽고 다른 CodeCatalyst 서비스와 통합되어 있어 이 자습서를 더 적은 단계로 진행할 수 있다는 것입니다.

이 자습서에서는 CodeCatalyst 개발 환경을 사용한다고 가정합니다.

다음 지침은 CodeCatalyst 개발 환경을 시작하고 필요한 도구로 구성하는 빠른 방법을 설명하지만 자세한 지침은 다음을 참조하세요.

- [개발 환경 생성](#) 섹션을 참조하세요.
- Amazon EKS 사용 설명서의 [kubectl 설치](#).
- Amazon EKS 사용 설명서의 [eksctl 설치 또는 업그레이드](#).
- AWS Command Line Interface 사용자 안내서의 [AWS CLI 최신 버전 설치 또는 업데이트](#).

새 개발 환경을 시작하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. codecatalyst-eks-project 프로젝트로 이동합니다.
3. 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
4. 소스 리포지토리 codecatalyst-eks-source-repository의 이름을 선택합니다.
5. 상단 근처에서 개발 환경 생성을 선택한 다음 AWS Cloud9 (브라우저에서)를 선택합니다.
6. 기존 브랜치에서 작업 및 메인이 선택되어 있는지 확인한 다음 생성을 선택합니다.

개발 환경이 새 브라우저 탭에서 시작되고 리포지토리(codecatalyst-eks-source-repository)가 복제됩니다.

kubectl을 설치 및 구성하려면

1. 개발 환경 터미널에 다음을 입력합니다.

```
curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.18.9/2020-11-02/bin/linux/amd64/kubectl
```

2. 입력:

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

3. 입력:

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

4. 입력:

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

5. 입력:

```
kubectl version --short --client
```

6. 버전이 나타나는지 확인합니다.

이제 kubectl이 설치됐습니다.

eksctl을 설치 및 구성하려면

Note

대신 kubectl을 사용할 수 있으므로 eksctl이 반드시 필요한 것은 아닙니다. 그러나 eksctl은 클러스터 구성의 대부분을 자동화하는 이점을 제공하므로 이 자습서에 권장되는 도구입니다.

1. 개발 환경 터미널에 다음을 입력합니다.

```
curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
```

2. 입력:

```
sudo cp /tmp/eksctl /usr/bin
```

3. 입력:

```
eksctl version
```

4. 버전이 나타나는지 확인합니다.

이제 eksctl이 설치됐습니다.

AWS CLI 가 설치되었는지 확인하려면

1. 개발 환경 터미널에 다음을 입력합니다.

```
aws --version
```

2. 버전이 표시되어 AWS CLI 가 설치되었는지 확인합니다.

나머지 절차를 완료하여 액세스하는 데 필요한 권한 AWS CLI 으로를 구성합니다 AWS.

를 구성하려면 AWS CLI

AWS 서비스에 대한 AWS CLI 액세스 권한을 부여하려면 액세스 키와 세션 토큰으로 구성해야 합니다. 다음 지침은 키와 토큰을 구성하는 빠른 방법을 제공하지만 자세한 지침은 AWS Command Line Interface 사용자 설명서의 [AWS CLI 구성](#)을 참조하세요.

1. 다음과 같이 IAM Identity Center 사용자를 생성합니다.

- a. 에 로그인 AWS Management Console 하고 <https://console.aws.amazon.com/singlesignon/> AWS IAM Identity Center 콘솔을 엽니다.

(IAM Identity Center에 로그인한 적이 없는 경우 활성화를 선택해야 할 수 있습니다.)

Note

CodeCatalyst 스페이스에 AWS 계정 연결된를 사용하여 로그인해야 합니다. 스페이스로 이동하여 AWS 계정 탭을 선택하고 연결된 계정을 확인할 수 있습니다. 자세한 내용은 [스페이스 생성](#) 섹션을 참조하세요.

- b. 탐색 창에서 Users와 Add user를 차례대로 선택합니다.
- c. 사용자 이름에 다음을 입력합니다.

codecatalyst-eks-user

- d. 암호에서 이 사용자와 공유할 수 있는 일회용 암호 생성을 선택합니다.
- e. 이메일 주소 및 이메일 주소 확인에 IAM Identity Center에 아직 없는 이메일 주소를 입력합니다.
- f. 이름에 다음을 입력합니다.

codecatalyst-eks-user

- g. 성에 다음을 입력합니다.

codecatalyst-eks-user

- h. 표시 이름에서 다음을 유지합니다.

codecatalyst-eks-user codecatalyst-eks-user

- i. 다음을 선택합니다.
- j. 그룹에 사용자 추가 페이지에서 다음을 선택합니다.
- k. 검토 및 사용자 추가 페이지에서 정보를 검토하고 사용자 추가를 선택합니다.

일회용 암호 대화 상자가 나타납니다.

- l. 복사를 선택한 다음 로그인 정보를 텍스트 파일에 붙여넣습니다. 로그인 정보는 AWS 액세스 포털 URL, 사용자 이름 및 일회용 암호로 구성됩니다.
 - m. 닫기를 선택하세요.
2. 다음과 같이 권한 세트를 생성합니다.
- a. 탐색 창에서 권한 세트를 선택한 다음 권한 세트 생성을 선택합니다.
 - b. 사전 정의된 권한 세트를 선택하고 AdministratorAccess를 선택합니다. 이 정책은 모든 AWS 서비스에 대한 전체 권한을 제공합니다.
 - c. 다음을 선택합니다.
 - d. 권한 세트 이름에서 AdministratorAccess를 제거하고 다음을 입력합니다.

codecatalyst-eks-permission-set

- e. 다음을 선택합니다.
 - f. 검토 및 생성 페이지에서 정보를 검토하고 생성을 선택합니다.
3. 다음과 같이 권한 세트를 codecatalyst-eks-user에 할당합니다.
- a. 탐색 창에서 AWS 계정을 선택한 다음 현재 로그인 AWS 계정 한 옆의 확인란을 선택합니다.
 - b. 사용자 또는 그룹 할당을 선택합니다.
 - c. 사용자 탭을 선택합니다.
 - d. codecatalyst-eks-user 옆의 확인란을 선택합니다.
 - e. 다음을 선택합니다.
 - f. codecatalyst-eks-permission-set 옆의 확인란을 선택합니다.
 - g. 다음을 선택합니다.
 - h. 정보를 검토하고 제출을 선택합니다.

이제 `codecatalyst-eks-user` 및 `codecatalyst-eks-permission-set`에 할당하여 함께 AWS 계정바인딩했습니다.

4. 다음과 같이 `codecatalyst-eks-user`의 액세스 키 및 세션 토큰을 가져옵니다.

- a. 에 대한 AWS 액세스 포털 URL과 사용자 이름 및 일회용 암호가 있는지 확인합니다 `codecatalyst-eks-user`. 이전에 이 정보를 텍스트 편집기에 복사했어야 합니다.

 Note

이 정보가 없는 경우 IAM Identity Center의 `codecatalyst-eks-user` 세부 정보 페이지로 이동하여 암호 재설정, 일회용 암호 생성 [...]을 선택한 후 암호 재설정을 다시 선택하면 화면에 해당 정보가 표시됩니다.

- b. 로그아웃합니다 AWS.
- c. AWS 액세스 포털 URL을 브라우저의 주소 표시줄에 붙여 넣습니다.
- d. 다음을 사용하여 로그인:

- 사용자 이름:

`codecatalyst-eks-user`

- 암호:

##

- e. 새 암호 설정에서 새 암호를 입력하고 새 암호 설정을 선택합니다.

화면에 AWS 계정 상자가 나타납니다.

- f. AWS 계정을 선택한 다음 `codecatalyst-eks-user` 사용자 및 권한 세트를 할당한 AWS 계정 의 이름을 선택합니다.
- g. `codecatalyst-eks-permission-set` 옆의 명령줄 또는 프로그래밍 액세스를 선택합니다.
- h. 페이지 중간에 있는 명령을 복사합니다. 다음처럼 보일 것입니다.

```
export AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
export AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
export AWS_SESSION_TOKEN="session-token"
```

...여기서 `## ##`은 긴 무작위 문자열입니다.

5. 다음과 AWS CLI같이 액세스 키와 세션 토큰에 추가합니다.

- a. CodeCatalyst 개발 환경으로 돌아갑니다.
- b. 터미널 프롬프트에서 복사한 명령을 붙여넣습니다. 입력을 누릅니다.

이제 액세스 키와 세션 토큰 AWS CLI 으로를 구성했습니다. 이제 AWS CLI 를 사용하여이 자습서에 필요한 작업을 완료할 수 있습니다.

Important

이 자습서 중에 언제든지 다음과 유사한 메시지가 표시되는 경우:

Unable to locate credentials. You can configure credentials by running "aws configure".

또는 다음과 같습니다.

ExpiredToken: The security token included in the request is expired

... AWS CLI 세션이 만료되었기 때문입니다. 이 경우 `aws configure` 명령을 실행하지 않습니다. 대신 Obtain codecatalyst-eks-user's access key and session token로 시작하는 이 절차의 4단계에 있는 지침을 사용하여 세션을 새로 고칩니다.

2단계: Amazon EKS 클러스터 생성

이 섹션에서는 Amazon EKS에서 클러스터를 생성합니다. 아래 지침은 `eksctl`를 사용하여 클러스터를 생성하는 빠른 방법을 설명하지만 자세한 지침은 다음을 참조하세요.

- Amazon EKS 사용 설명서의 [eksctl 시작하기](#)

또는

- Amazon EKS 사용 설명서의 [콘솔 및 AWS CLI 시작하기](#)(이 주제에서는 클러스터 생성 `kubectl` 지침을 제공합니다)

 Note

프라이빗 클러스터는 Amazon EKS와의 CodeCatalyst 통합에서 지원되지 않습니다.

시작하기 전에

개발 시스템에서 다음 작업을 완료했는지 확인합니다.

- eksctl 유틸리티를 설치했습니다.
- kubectl 유틸리티를 설치했습니다.
- 를 설치하고 액세스 키와 세션 토큰으로 AWS CLI 구성했습니다.

이러한 작업을 완료하는 방법에 대한 자세한 정보는 [1단계: 개발 머신 설정](#) 섹션을 참조하세요.

클러스터 생성

 Important

클러스터가 올바르게 구성되지 않으므로 Amazon EKS 서비스의 사용자 인터페이스를 사용하여 클러스터를 생성하지 마세요. 다음 단계에 설명된 대로 eksctl 유틸리티를 사용합니다.

1. 개발 환경으로 이동합니다.
2. 클러스터 및 노드를 생성합니다.

```
eksctl create cluster --name codecatalyst-eks-cluster --region us-west-2
```

위치:

- *codecatalyst-eks-cluster*는 클러스터에 부여하려는 이름으로 대체됩니다.
- *us-west-2*를 해당 리전으로 바꿉니다.

10~20분 후 다음과 비슷한 메시지가 나타납니다.

EKS cluster "codecatalyst-eks-cluster" in "us-west-2" region is ready

Note

AWS 에서 클러스터를 생성하는 동안 여러 `waiting for CloudFormation stack` 메시지가 표시됩니다. 이는 예상된 동작입니다.

- 클러스터가 성공적으로 생성되었는지 확인합니다.

```
kubectl cluster-info
```

성공적인 클러스터 생성을 나타내는 다음과 유사한 메시지가 표시됩니다.

```
Kubernetes master is running at https://long-string.gr7.us-west-2.eks.amazonaws.com
CoreDNS is running at https://long-string.gr7.us-west-2.eks.amazonaws.com/api/v1/
namespaces/kube-system/services/kube-dns:dns/proxy
```

3단계: Amazon ECR 이미지 리포지토리 생성

이 섹션에서는 Amazon Elastic Container Registry(Amazon ECR)에서 프라이빗 이미지 리포지토리를 생성합니다. 이 리포지토리는 자습서의 Docker 이미지를 저장합니다.

Amazon ECR에 대한 자세한 내용은 Amazon Elastic Container Registry 사용 설명서를 참조합니다.

Amazon ECR에서 이미지 리포지토리를 생성하려면

- 개발 환경으로 이동합니다.
- Amazon ECR에 빈 리포지토리를 생성합니다.

```
aws ecr create-repository --repository-name codecatalyst-eks-image-repo
```

*codecatalyst-eks-image-repo*를 Amazon ECR 리포지토리에 부여하려는 이름으로 바꿉니다.

이 자습서에서는 리포지토리 이름을 *codecatalyst-eks-image-repo*로 지정했다고 가정합니다.

- Amazon ECR 리포지토리의 세부 정보를 표시합니다.

```
aws ecr describe-repositories \
```

```
--repository-names codecatalyst-eks-image-repo
```

4. “repositoryUri”: 값을 기록해 둡니다. 예시: 111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-eks-image-repo.

나중에 워크플로에 리포지토리를 추가할 때 필요합니다.

4단계: 소스 파일 추가

이 섹션에서는 소스 리포지토리(codecatalyst-eks-source-repository)에 애플리케이션 소스 파일을 추가합니다. 다음과 같이 구성됩니다.

- index.html 파일 - 'Hello, World!' 메시지를 브라우저에 표시합니다.
- Dockerfile - Docker 이미지에 사용할 기본 이미지와 Docker 명령으로 적용할 기본 이미지를 설명합니다.
- deployment.yaml 파일 - Kubernetes 서비스 및 배포를 정의하는 Kubernetes 매니페스트입니다.

폴더의 구조는 다음과 같습니다.

```
|— codecatalyst-eks-source-repository
  |— Kubernetes
    |— deployment.yaml
  |— public-html
    | |— index.html
  |— Dockerfile
```

주제

- [index.html](#)
- [Dockerfile](#)
- [deployment.yaml](#)

index.html

index.html 파일 - 'Hello, World!' 메시지를 브라우저에 표시합니다.

index.html 파일을 추가하려면

1. 개발 환경으로 이동합니다.

2. codecatalyst-eks-source-repository에서 이름이 public-html인 폴더를 생성합니다.
3. /public-html에서 다음 콘텐츠로 index.html 파일을 생성합니다.

```
<html>
  <head>
    <title>Hello World</title>
    <style>
      body {
        background-color: black;
        text-align: center;
        color: white;
        font-family: Arial, Helvetica, sans-serif;
      }
    </style>
  </head>
  <body>
    <h1>Hello, World!</h1>
  </body>
</html>
```

4. 터미널 프롬프트에서 다음을 입력합니다.

```
cd /projects/codecatalyst-eks-source-repository
```

5. 추가, 커밋 및 푸시:

```
git add .
git commit -m "add public-html/index.html"
git push
```

index.html은 public-html 폴더의 리포지토리에 추가됩니다.

Dockerfile

Dockerfile은 사용할 기본 Docker 이미지와 적용할 Docker 명령을 설명합니다. Dockerfile에 대한 자세한 내용은 [Dockerfile 참조](#)를 참조하세요.

여기에 지정된 Dockerfile은 Apache 2.4 기본 이미지(httpd)를 사용함을 나타냅니다. 또한 index.html 소스 파일을 웹 페이지를 제공하는 Apache 서버의 폴더에 복사하는 방법도 포함되어 있습니다. Dockerfile의 EXPOSE 지침은 컨테이너가 포트 80에서 수신 중임을 Docker에게 알립니다.

Dockerfile을 추가하려면

1. codecatalyst-eks-source-repository에서 다음 콘텐츠로 Dockerfile 파일을 생성합니다.

```
FROM httpd:2.4
COPY ./public-html/index.html /usr/local/apache2/htdocs/index.html
EXPOSE 80
```

파일 확장자는 제외합니다.

Important

Dockerfile은 리포지토리의 루트 폴더에 있어야 합니다. 워크플로의 Docker build 명령은 해당 파일이 있을 것으로 예상합니다.

2. 추가, 커밋 및 푸시:

```
git add .
git commit -m "add Dockerfile"
git push
```

Dockerfile이 리포지토리에 추가됩니다.

deployment.yaml

이 섹션에서는 리포지토리에 deployment.yaml 파일을 추가합니다. deployment.yaml 파일은 '서비스'와 '배포'라는 두 가지 Kubernetes 리소스 유형 또는 실행 유형을 정의하는 Kubernetes 매니페스트입니다.

- '서비스'는 Amazon EC2에 로드 밸런서를 배포합니다. 로드 밸런서는 'Hello, World!'를 탐색하는 데 사용할 수 있는 인터넷 연결 퍼블릭 URL 및 표준 포트(포트 80)를 제공합니다. 애플리케이션을 배포합니다.
- '배포'는 포트 3개를 배포하며 각 포트에는 'Hello, World!'가 포함된 Docker 컨테이너가 포함됩니다. 애플리케이션을 배포합니다. 세 포트는 클러스터를 생성할 때 생성된 노드에 배포됩니다.

이 자습서의 매니페스트는 짧지만 매니페스트에는 포드, 작업, 수신 및 네트워크 정책과 같은 다양한 Kubernetes 리소스 유형이 포함될 수 있습니다. 또한 배포가 복잡한 경우 여러 매니페스트 파일을 사용할 수 있습니다.

deployment.yaml 파일을 추가하려면

1. codecatalyst-eks-source-repository에서 이름이 Kubernetes인 폴더를 생성합니다.
2. /Kubernetes에서 다음 콘텐츠로 deployment.yaml 파일을 생성합니다.

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  labels:
    app: my-app
spec:
  type: LoadBalancer
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
  labels:
    app: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: codecatalyst-eks-container
```

```
# The $REPOSITORY_URI and $IMAGE_TAG placeholders will be replaced by
actual values supplied by the build action in your workflow
image: $REPOSITORY_URI:$IMAGE_TAG
ports:
  - containerPort: 80
```

3. 추가, 커밋 및 푸시:

```
git add .
git commit -m "add Kubernetes/deployment.yaml"
git push
```

deployment.yaml은 Kubernetes 폴더의 리포지토리에 추가됩니다.

이제 모든 소스 파일을 추가했습니다.

잠시 시간을 내어 작업을 다시 확인하고 모든 파일을 올바른 폴더에 넣었는지 확인하세요. 폴더의 구조는 다음과 같습니다.

```
|— codecatalyst-eks-source-repository
   |— Kubernetes
       |— deployment.yaml
   |— public-html
       | |— index.html
   |— Dockerfile
```

5단계: AWS 역할 생성

이 섹션에서는 CodeCatalyst 워크플로가 작동하는 데 필요한 AWS IAM 역할을 생성합니다. 이러한 역할은 다음과 같습니다.

- 빌드 역할 - CodeCatalyst 빌드 작업(워크플로 내)에 AWS 계정에 액세스하고 Amazon ECR 및 Amazon EC2에 쓸 수 있는 권한을 부여합니다.
- 역할 배포 - CodeCatalyst Kubernetes 클러스터에 배포 작업(워크플로 내)에 AWS 계정 및 Amazon EKS에 액세스할 수 있는 권한을 부여합니다.

IAM 역할에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 [IAM 역할](#) 섹션을 참조하세요.

Note

시간을 절약하기 위해 이전에 나열한 두 역할 대신

CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할이라

는 단일 역할을 생성할 수 있습니다. 자세한 내용은 [계정 및 스페이스의](#)

[CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 생성](#) 섹션을 참조하세요.

CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할에는 보안 위험을 초래할 수 있는 매우 광범위한 권한이 있음을 이해합니다. 보안에 대한 우려가 적은 자습서 및 시나리오에서만 이 역할을 사용하는 것이 좋습니다. 이 자습서에서는 이전에 나열된 두 역할을 생성하고 있다고 가정합니다.

빌드 및 배포 역할을 생성하려면 다음 절차 시리즈를 완료합니다.

1. 두 역할에 대한 신뢰 정책을 생성하려면

1. 개발 환경으로 이동합니다.
2. Cloud9-*long-string* 디렉터리에 다음 콘텐츠로 codecatalyst-eks-trust-policy.json 파일을 생성합니다.

2. 빌드 역할에 대한 빌드 정책을 생성하려면

- Cloud9-*long-string* 디렉터리에 다음 콘텐츠로 codecatalyst-eks-build-policy.json 파일을 생성합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:*",
        "ec2:*"
      ],
      "Resource": "*"
    }
  ]
}
```


}

Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

3. 배포 정책을 생성하고 역할을 배포하려면

- Cloud9-*long-string* 디렉터리에 다음 콘텐츠로 codecatalyst-eks-deploy-policy.json 파일을 생성합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks:DescribeCluster",
        "eks:ListClusters"
      ],
      "Resource": "*"
    }
  ]
}
```

Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

이제 개발 환경에 세 개의 정책 문서를 추가했습니다. 디렉터리 구조는 다음과 같습니다.

```
|– Cloud9-long-string
  |– .c9
  |– codecatalyst-eks-source-repository
    |– Kubernetes
    |– public-html
    |– Dockerfile
  codecatalyst-eks-build-policy.json
  codecatalyst-eks-deploy-policy.json
  codecatalyst-eks-trust-policy.json

```

4. 에 빌드 정책을 추가하려면 AWS

1. 개발 환경 터미널에 다음을 입력합니다.

```
cd /projects

```

2. 입력:

```
aws iam create-policy \
  --policy-name codecatalyst-eks-build-policy \
  --policy-document file:///codecatalyst-eks-build-policy.json

```

3. Enter를 누릅니다.

4. 명령 출력에서와 같은 "arn": 값을 기록해 둡니다(예: `arn:aws:iam::111122223333:policy/codecatalyst-eks-build-policy`). 나중에 이 ARN이 필요합니다.

5. 에 배포 정책을 추가하려면 AWS

1. 입력:

```
aws iam create-policy \
  --policy-name codecatalyst-eks-deploy-policy \

```

```
--policy-document file://codecatalyst-eks-deploy-policy.json
```

2. Enter를 누릅니다.
3. 명령 출력에서 와 같은 "arn": 값을 기록해 둡니다(예: `arn:aws:iam::111122223333:policy/codecatalyst-eks-deploy-policy`). 나중에 이 ARN이 필요합니다.

6. 빌드 역할을 생성하려면

1. 입력:

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

2. Enter를 누릅니다.

3. 입력:

```
aws iam attach-role-policy \  
    --role-name codecatalyst-eks-build-role \  
    --policy-arn arn:aws:iam::111122223333:policy/codecatalyst-eks-build-policy
```

*arn:aws:iam::111122223333:policy/codecatalyst-eks-build-policy*가 앞서 언급한 빌드 정책의 ARN으로 대체되는 경우.

4. Enter를 누릅니다.
5. 터미널 프롬프트에서 다음을 입력합니다.

```
aws iam get-role \  
    --role-name codecatalyst-eks-build-role
```

6. Enter를 누릅니다.
7. 역할의 "Arn": 값을 기록해 둡니다. 예시: `arn:aws:iam::111122223333:role/codecatalyst-eks-build-role`. 나중에 이 ARN이 필요합니다.

7. 배포 역할을 생성하려면

1. 입력:

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

2. Enter를 누릅니다.

3. 입력:

```
aws iam attach-role-policy \  
    --role-name codecatalyst-eks-deploy-role \  
    --policy-arn arn:aws:iam::111122223333:policy/codecatalyst-eks-deploy-policy
```

*arn:aws:iam::111122223333:policy/codecatalyst-eks-deploy-policy*가 앞서 언급한 배포 정책의 ARN으로 대체되는 경우.

4. Enter를 누릅니다.

5. 입력:

```
aws iam get-role \  
    --role-name codecatalyst-eks-deploy-role
```

6. Enter를 누릅니다.

7. 역할의 "Arn": 값을 기록해 둡니다. 예시: `arn:aws:iam::111122223333:role/codecatalyst-eks-deploy-role`. 나중에 이 ARN이 필요합니다.

이제 빌드 및 배포 역할을 생성하고 해당 ARN을 기록했습니다.

6단계: CodeCatalyst에 AWS 역할 추가

이 단계에서는 스페이스에 연결 AWS 계정 한에 빌드 역할(`codecatalyst-eks-build-role`) 및 배포 역할(`codecatalyst-eks-deploy-role`)을 추가합니다. 이렇게 하면 워크플로에서 역할을 사용할 수 있습니다.

에 빌드 및 배포 역할을 추가하려면 AWS 계정

1. CodeCatalyst 콘솔에서 스페이스로 이동합니다.
2. 상단에서 설정을 선택합니다.
3. 탐색 창에서 AWS 계정을 선택합니다. 계정 목록이 나타납니다.

4. Amazon CodeCatalyst 표시 이름 열에서 빌드 및 배포 역할을 생성한 AWS 계정 의 표시 이름을 복사합니다. (숫자일 수 있습니다.) 워크플로를 생성할 때 나중에 이 값이 필요합니다.
5. 표시 이름을 선택합니다.
6. 관리 콘솔에서 역할 AWS 관리를 선택합니다.

Amazon CodeCatalyst 스페이스에 IAM 역할 추가 페이지가 나타납니다. 페이지에 액세스하려면 로그인해야 할 수 있습니다.

7. IAM에서 생성한 기존 역할 추가를 선택합니다.

드롭다운 목록이 나타납니다. 목록에는 빌드 및 배포 역할, `codecatalyst-runner.amazonaws.com` 및 `codecatalyst.amazonaws.com` 서비스 위탁자가 포함된 신뢰 정책이 있는 기타 IAM 역할이 표시됩니다.

8. 드롭다운 목록에서 다음을 추가합니다.

- `codecatalyst-eks-build-role`
- `codecatalyst-eks-deploy-role`

Note

The security token included in the request is invalid가 표시되면 적절한 권한이 없기 때문일 수 있습니다. 이 문제를 해결하려면 CodeCatalyst 스페이스를 생성할 때 사용한 AWS 계정으로 다시 로그인 AWS 하면에서 로그아웃합니다.

9. CodeCatalyst 콘솔로 돌아가 페이지를 새로 고칩니다.

이제 빌드 및 배포 역할이 IAM 역할 아래에 표시됩니다.

이제 CodeCatalyst 워크플로에서 이러한 역할을 사용할 수 있습니다.

7단계: ConfigMap 업데이트

5단계: AWS 역할 생성에서 생성한 배포 역할을 Kubernetes ConfigMap 파일에 추가하여 (워크플로의) Kubernetes 클러스터에 배포 작업에 클러스터에 액세스하고 상호 작용할 수 있는 기능을 부여해야 합니다. `eksctl` 또는 `kubectl`를 사용하여 이러한 작업을 수행할 수 있습니다.

`eksctl`을 사용하여 Kubernetes ConfigMap 파일을 구성하려면

- 개발 환경 터미널에 다음을 입력합니다.

```
eksctl create iamidentitymapping --cluster codecatalyst-eks-cluster --
arn arn:aws:iam::111122223333:role/codecatalyst-eks-deploy-role --group
system:masters --username codecatalyst-eks-deploy-role --region us-west-2
```

위치:

- *codecatalyst-eks-cluster*는 Amazon EKS 클러스터의 클러스터 이름으로 대체됩니다.
- *arn:aws:iam::111122223333:role/codecatalyst-eks-deploy-role*은 [5단계: AWS 역할 생성](#)에서 생성한 배포 역할의 ARN으로 대체됩니다.
- *codecatalyst-eks-deploy-role*(--username 옆)은 [5단계: AWS 역할 생성](#)에서 생성한 배포 역할의 이름으로 대체됩니다.

Note

배포 역할을 생성하지 않기로 결정한 경우 *codecatalyst-eks-deploy-role*을 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 이름으로 바꿉니다. 이에 대한 자세한 내용은 [5단계: AWS 역할 생성](#) 섹션을 참조하세요.

- *us-west-2*를 해당 리전으로 바꿉니다.

이 명령에 대한 자세한 내용은 [IAM 사용자 및 역할 관리](#)를 참조하세요.

다음과 비슷한 메시지가 나타납니다.

```
2023-06-09 00:58:29 [#] checking arn arn:aws:iam::111122223333:role/codecatalyst-
eks-deploy-role against entries in the auth ConfigMap
2023-06-09 00:58:29 [#] adding identity "arn:aws:iam::111122223333:role/
codecatalyst-eks-deploy-role" to auth ConfigMap
```

kubectl을 사용하여 Kubernetes ConfigMap 파일을 구성하려면

1. 개발 환경 터미널에 다음을 입력합니다.

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

ConfigMap 파일이 화면에 나타납니다.

2. 빨간색 기울임꼴로 텍스트를 추가합니다.

```
# Please edit the object below. Lines beginning with a '#' will be ignored,
# and an empty file will abort the edit. If an error occurs while saving this file
# will be
# reopened with the relevant failures.
#
apiVersion: v1
data:
  mapRoles: |
    - groups:
      - system:bootstrappers
      - system:nodes
      rolearn: arn:aws:iam::111122223333:role/eksctl-codecatalyst-eks-cluster-n-
NodeInstanceRole-16BC456ME6YR5
      username: system:node:{{EC2PrivateDNSName}}
    - groups:
      - system:masters
      rolearn: arn:aws:iam::111122223333:role/codecatalyst-eks-deploy-role
      username: codecatalyst-eks-deploy-role
  mapUsers: |
    []
kind: ConfigMap
metadata:
  creationTimestamp: "2023-06-08T19:04:39Z"
  managedFields:
    ...
```

위치:

- *arn:aws:iam::111122223333:role/codecatalyst-eks-deploy-role*은 [5단계: AWS 역할 생성](#)에서 생성한 배포 역할의 ARN으로 대체됩니다.
- *codecatalyst-eks-deploy-role*(username: 옆)은 [5단계: AWS 역할 생성](#)에서 생성한 배포 역할의 이름으로 대체됩니다.

Note

배포 역할을 생성하지 않기로 결정한 경우 *codecatalyst-eks-deploy-role*을 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 이름으로 바꿉니다. 이에 대한 자세한 내용은 [5단계: AWS 역할 생성](#) 섹션을 참조하세요.

자세한 내용은 Amazon EKS 사용 설명서의 [클러스터에 대한 IAM 위탁자 액세스 활성화](#)를 참조하세요.

이제 배포 역할을 부여하고 Amazon EKS에 배포 작업을 확장하여 Kubernetes 클러스터에 대한 `system:masters` 권한을 부여했습니다.

8단계: 워크플로 생성 및 실행

이 단계에서는 소스 파일을 가져와 Docker 이미지로 빌드한 다음 Amazon EKS 클러스터의 트리 포드에 이미지를 배포하는 워크플로를 생성합니다.

워크플로는 순차적으로 실행되는 다음 구성 요소로 구성됩니다.

- 트리거 - 이 트리거는 소스 리포지토리에 변경 사항을 푸시할 때 워크플로 실행을 자동으로 시작합니다. 트리거에 대한 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 주제를 참조하세요.
- 빌드 작업(BuildBackend) - 트리거 시 작업은 Dockerfile을 사용하여 Docker 이미지를 빌드하고 Amazon ECR에 이미지를 푸시합니다. 빌드 작업은 `deployment.yaml` 파일의 `$REPOSITORY_URI` 및 `$IMAGE_TAG` 변수를 올바른 값으로 업데이트한 다음 이 파일과 Kubernetes 폴더의 다른 모든 출력 아티팩트를 생성합니다. 이 자습서에서 Kubernetes 폴더의 유일한 파일은 `deployment.yaml`이지만 더 많은 파일을 포함할 수 있습니다. 아티팩트는 배포 작업의 입력으로 사용되며, 그 다음입니다.

빌드 작업에 대한 자세한 내용은 [워크플로로 빌드하기](#) 섹션을 참조하세요.

- 배포 작업(DeployToEKS) - 빌드 작업이 완료되면 배포 작업은 빌드 작업(Manifests)에서 생성된 출력 아티팩트를 찾아 그 안의 `deployment.yaml` 파일을 찾습니다. 그런 다음 이 작업은 `deployment.yaml` 파일의 지침에 따라 각각 단일 'Hello, World!'가 포함된 세 개의 포드를 실행합니다. Docker 컨테이너 - Amazon EKS 클러스터 내부.

워크플로 생성

1. CodeCatalyst 콘솔로 이동합니다.
2. 프로젝트(`codecatalyst-eks-project`)로 이동합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로 생성을 선택합니다.
5. 소스 리포지토리에서 `codecatalyst-eks-source-repository`를 선택합니다.

6. 브랜치에서 main을 선택합니다.
7. 생성(Create)을 선택합니다.
8. YAML 샘플 코드를 삭제합니다.
9. 다음 YAML 코드를 추가하여 새 워크플로 정의 파일을 생성합니다.

 Note

워크플로 정의 파일에 대한 자세한 내용은 [워크플로 YAML 정의](#) 섹션을 참조하세요.

 Note

다음 YAML 코드에서 원하는 경우 Connections: 섹션을 생략할 수 있습니다. 이 섹션을 생략하는 경우 환경의 기본 IAM 역할 필드에 지정된 역할에 [6단계: CodeCatalyst에 AWS 역할 추가](#)에 설명된 두 역할의 권한 및 신뢰 정책이 포함되어 있는지 확인해야 합니다. 기본 IAM 역할이 있는 환경 설정에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.

```
Name: codecatalyst-eks-workflow
SchemaVersion: 1.0

Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  BuildBackend:
    Identifier: aws/build@v1
    Environment:
      Name: codecatalyst-eks-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-eks-build-role
    Inputs:
      Sources:
        - WorkflowSource
      Variables:
        - Name: REPOSITORY_URI
```

```

    Value: 111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-eks-
image-repo
    - Name: IMAGE_TAG
      Value: ${WorkflowSource.CommitId}
  Configuration:
    Steps:
      #pre_build:
      - Run: echo Logging in to Amazon ECR...
      - Run: aws --version
      - Run: aws ecr get-login-password --region us-west-2 | docker login --
username AWS --password-stdin 111122223333.dkr.ecr.us-west-2.amazonaws.com
      #build:
      - Run: echo Build started on `date`
      - Run: echo Building the Docker image...
      - Run: docker build -t $REPOSITORY_URI:latest .
      - Run: docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
      #post_build:
      - Run: echo Build completed on `date`
      - Run: echo Pushing the Docker images...
      - Run: docker push $REPOSITORY_URI:latest
      - Run: docker push $REPOSITORY_URI:$IMAGE_TAG
      # Replace the variables in deployment.yaml
      - Run: find Kubernetes/ -type f | xargs sed -i "s|\\$REPOSITORY_URI|
$REPOSITORY_URI|g"
      - Run: find Kubernetes/ -type f | xargs sed -i "s|\\$IMAGE_TAG|$IMAGE_TAG|g"
      - Run: cat Kubernetes/*
      # The output artifact will be a zip file that contains Kubernetes manifest
files.
    Outputs:
      Artifacts:
        - Name: Manifests
          Files:
            - "Kubernetes/*"
  DeployToEKS:
    DependsOn:
      - BuildBackend
  Identifier: aws/kubernetes-deploy@v1
  Environment:
    Name: codecatalyst-eks-environment
  Connections:
    - Name: codecatalyst-account-connection
      Role: codecatalyst-eks-deploy-role
  Inputs:
    Artifacts:

```

```

- Manifests
Configuration:
  Namespace: default
  Region: us-west-2
  Cluster: codecatalyst-eks-cluster
  Manifests: Kubernetes/

```

앞의 코드에서

- *codecatalyst-eks-environment* 인스턴스 두 개를 모두 [사전 조건](#)에서 생성한 환경 이름으로 바꿉니다.
- *codecatalyst-account-connection* 인스턴스 두 개를 모두 계정 연결의 표시 이름으로 바꿉니다. 표시 이름은 숫자일 수 있습니다. 자세한 내용은 [6단계: CodeCatalyst에 AWS 역할 추가](#) 섹션을 참조하세요.
- *codecatalyst-eks-build-role*을 [5단계: AWS 역할 생성](#)에서 생성한 빌드 역할 이름으로 바꿉니다.
- *111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-eks-image-repo*(Value: 속성 내)를 [3단계: Amazon ECR 이미지 리포지토리 생성](#)에서 생성한 Amazon ECR 리포지토리의 URI로 바꿉니다.
- *111122223333.dkr.ecr.us-west-2.amazonaws.com*(Run: aws ecr 명령 내)을 이미지 접미사(/codecatalyst-eks-image-repo) 제외 Amazon ECR 리포지토리의 URI로 바꿉니다.
- *codecatalyst-eks-deploy-role*을 [5단계: AWS 역할 생성](#)에서 생성한 배포 역할의 이름으로 바꿉니다.
- AWS 리전 코드가 있는 *us-west-2*의 두 인스턴스. 리전 코드 목록은 AWS 일반 참조의 [리전 엔드포인트](#)를 참조하세요.

Note

빌드 및 배포 역할을 생성하지 않기로 결정한 경우 *codecatalyst-eks-build-role* 및 *codecatalyst-eks-deploy-role*을 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 이름으로 바꿉니다. 이에 대한 자세한 내용은 [5단계: AWS 역할 생성](#) 섹션을 참조하세요.

10. (선택 사항) 커밋하기 전에 YAML 코드가 유효한지 확인하려면 검증을 선택합니다.

11. 커밋을 선택합니다.

12. 워크플로 커밋 대화 상자에서 다음을 입력합니다.

- a. 커밋 메시지에서 텍스트를 제거하고 다음을 입력합니다.

Add first workflow

- b. 리포지토리에서 `codecatalyst-eks-source-repository`를 선택합니다.
 c. 브랜치 이름에서 기본을 선택합니다.
 d. 커밋을 선택합니다.

이제 워크플로를 생성했습니다. 워크플로 상단에 정의된 트리거로 인해 워크플로 실행이 자동으로 시작됩니다. 특히 `workflow.yaml` 파일을 소스 리포지토리에 커밋(및 푸시)할 때 트리거가 워크플로 실행을 시작했습니다.

진행 중인 워크플로 실행을 보려면

1. CodeCatalyst 콘솔의 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
2. 방금 생성한 `codecatalyst-eks-workflow` 워크플로를 선택합니다.
3. BuildBackend를 선택하여 빌드 진행 상황을 확인합니다.
4. DeployToEKS를 선택하여 배포 진행 상황을 확인합니다.

실행 세부 정보 보기에 대한 자세한 내용은 [워크플로 실행 상태 및 세부 정보 보기](#)를 참조하세요.

배포를 확인하려면

1. <https://console.aws.amazon.com/ec2/>에서 Amazon EC2 콘솔을 엽니다.
2. 왼쪽 하단에서 로드 밸런서를 선택합니다.
3. Kubernetes 배포의 일부로 생성된 로드 밸런서를 선택합니다. 선택할 로드 밸런서가 확실하지 않은 경우 태그 탭에서 다음 태그를 찾습니다.
 - `kubernetes.io/service-name`
 - `kubernetes.io/cluster/ekstutorialcluster`
4. 올바른 로드 밸런서를 선택한 상태에서 설명 탭을 선택합니다.
5. DNS 이름 값을 복사하여 브라우저의 주소 표시줄에 붙여 넣습니다.

“Hello World!” 웹페이지가 브라우저에 표시되며 애플리케이션을 성공적으로 배포했음을 나타냅니다.

9단계: 소스 파일 변경

이 섹션에서는 소스 리포지토리의 `index.html` 파일을 변경합니다. 이 변경으로 인해 워크플로는 새 Docker 이미지를 빌드하고 커밋 ID로 태그를 지정하고 Amazon ECR에 푸시한 다음 Amazon ECS에 배포합니다.

`index.html`을 변경하려면

1. 개발 환경으로 이동합니다.
2. 터미널 프롬프트에서 소스 리포지토리로 변경합니다.

```
cd /projects/codecatalyst-eks-source-repository
```

3. 최신 워크플로 변경 사항 가져오기:

```
git pull
```

4. `codecatalyst-eks-source-repository/public-html/index.html`를 엽니다.
5. 14행에서 `Hello, World!` 텍스트를 `Tutorial complete!`로 변경합니다.
6. 추가, 커밋 및 푸시:

```
git add .  
git commit -m "update index.html title"  
git push
```

워크플로 실행이 자동으로 시작됩니다.

7. (선택 사항) 다음을 입력합니다.

```
git show HEAD
```

`index.html` 변경 사항에 대한 커밋 ID를 기록해 둡니다. 이 커밋 ID는 방금 시작한 워크플로 실행에 의해 배포될 Docker 이미지에 태그가 지정됩니다.

8. 배포 진행 상황 보기:

- a. CodeCatalyst 콘솔의 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
 - b. 최신 실행을 보려면 `codecatalyst-eks-workflow`를 선택합니다.
 - c. BuildBackend 및 DeployToEKS를 선택하여 워크플로 실행 진행 상황을 확인합니다.
9. 다음과 같이 애플리케이션이 업데이트되었는지 확인합니다.

- a. <https://console.aws.amazon.com/ec2/>에서 Amazon EC2 콘솔을 엽니다.
- b. 왼쪽 하단에서 로드 밸런서를 선택합니다.
- c. Kubernetes 배포의 일부로 생성된 로드 밸런서를 선택합니다.
- d. DNS 이름 값을 복사하여 브라우저의 주소 표시줄에 붙여 넣습니다.

‘자습서가 완료되었습니다!’ 웹 페이지가 브라우저에 나타나 애플리케이션의 새 버전을 성공적으로 배포했음을 나타냅니다.

10. (선택 사항)에서 Amazon ECR 콘솔로 AWS전환하고 새 Docker 이미지에이 절차의 7단계에서 커 및 ID로 태그가 지정되었는지 확인합니다.

정리

이 자습서에서 사용하는 스토리지 및 컴퓨팅 리소스에 대해 불필요한 요금이 부과되지 않도록 환경을 정리해야 합니다.

정리하려면

1. 클러스터 삭제:

- 개발 환경 터미널에 다음을 입력합니다.

```
eksctl delete cluster --region=us-west-2 --name=codecatalyst-eks-cluster
```

위치:

- *us-west-2*를 해당 리전으로 바꿉니다.
- *codecatalyst-eks-cluster*는 생성한 클러스터의 이름으로 대체됩니다.

5~10분이 지나면 CloudFormation 스택, 노드 그룹(Amazon EC2) 및 로드 밸런서를 포함하되 이에 국한되지 않는 클러스터 및 관련 리소스가 삭제됩니다.

⚠ Important

eksctl delete cluster 명령이 작동하지 않으면 자격 AWS 증명 또는 자격 kubectl 증명을 새로 고쳐야 할 수 있습니다. 어떤 자격 증명을 새로 고칠지 확실하지 않은 경우 먼저 AWS 자격 증명을 새로 고칩니다. AWS 자격 증명을 새로 고치려면 ['Unable to locate credentials' 및 'ExpiredToken' 오류를 해결하려면 어떻게 해야 하나요?](#) 섹션을 참조하세요. kubectl 자격 증명을 새로 고치려면 ['Unable to connect to the server' 오류를 해결하려면 어떻게 해야 하나요?](#) 섹션을 참조하세요.

2. AWS 콘솔에서 다음과 같이 정리합니다.
 1. Amazon ECR에서 codecatalyst-eks-image-repo를 삭제합니다.
 2. IAM Identity Center에서 다음을 삭제합니다.
 - a. codecatalyst-eks-user
 - b. codecatalyst-eks-permission-set
 3. IAM에서 다음을 삭제합니다.
 - codecatalyst-eks-build-role
 - codecatalyst-eks-deploy-role
 - codecatalyst-eks-build-policy
 - codecatalyst-eks-deploy-policy
3. CodeCatalyst 콘솔에서 다음과 같이 정리합니다.
 1. codecatalyst-eks-workflow를 삭제합니다.
 2. codecatalyst-eks-environment를 삭제합니다.
 3. codecatalyst-eks-source-repository를 삭제합니다.
 4. 개발 환경을 삭제합니다.
 5. codecatalyst-eks-project를 삭제합니다.

이 자습서에서는 CodeCatalyst 워크플로와 Kubernetes 클러스터에 배포 작업을 사용하여 Amazon EKS 서비스에 애플리케이션을 배포하는 방법을 배웠습니다.

'Kubernetes 클러스터에 배포' 작업 추가

다음 지침을 사용하여 워크플로에 Kubernetes 클러스터에 배포 작업을 추가합니다.

시작하기 전에

워크플로에 Kubernetes 클러스터에 배포 작업을 추가하기 전에 다음을 준비해야 합니다.

Tip

이러한 사전 조건을 빠르게 설정하려면 [자습서: Amazon EKS에 애플리케이션 배포](#)의 지침을 따르세요.

- Amazon EKS의 Kubernetes 클러스터. 클러스터에 대한 자세한 내용은 Amazon EKS 사용 설명서의 [Amazon EKS 클러스터](#)를 참조하세요.
- 애플리케이션을 Docker 이미지로 조립하는 방법을 설명하는 Dockerfile이 하나 이상 있습니다. Dockerfile에 대한 자세한 내용은 [Dockerfile 참조](#)를 참조하세요.
- Kubernetes 설명서에서 구성 파일 또는 구성이라는 하나 이상의 Kubernetes 매니페스트 파일. 자세한 내용은 Kubernetes 문서의 [리소스 관리](#)를 참조하세요.
- Kubernetes 클러스터에 배포 작업에 Amazon EKS 클러스터에 액세스하고 상호 작용할 수 있는 기능을 제공하는 IAM 역할입니다. 자세한 내용은 ['Kubernetes 클러스터에 배포' 작업 YAML에서 Role](#) 주제를 참조하세요.

이 역할을 생성한 후 다음 위치에 추가해야 합니다.

- Kubernetes ConfigMap 파일. ConfigMap 파일에 역할을 추가하는 방법을 알아보려면 Amazon EKS 사용 설명서의 [클러스터에 대한 IAM 위탁자 액세스 활성화](#)를 참조하세요.
- CodeCatalyst CodeCatalyst에 IAM 역할을 추가하는 방법은 [계정 연결에 IAM 역할 추가](#) 섹션을 참조하세요.
- CodeCatalyst 스페이스, 프로젝트 및 환경. 스페이스와 환경은 모두 애플리케이션을 배포할 AWS 계정에 연결되어야 합니다. 자세한 내용은 [스페이스 생성](#), [Amazon CodeCatalyst에서 빈 프로젝트 생성](#), [AWS 계정 및 VPCs에 배포](#) 섹션을 참조하세요.
- CodeCatalyst에서 지원하는 소스 리포지토리입니다. 리포지토리는 애플리케이션 소스 파일, Dockerfiles 및 Kubernetes 매니페스트를 저장합니다. 자세한 내용은 [CodeCatalyst의 소스 리포지토리 코드 저장 및 협업](#) 섹션을 참조하세요.

Visual

시각적 편집기를 사용하여 'Kubernetes 클러스터에 배포' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.

2. 프로젝트를 선택합니다.
 3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
 4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
 5. 편집을 선택합니다.
 6. 비주얼을 선택합니다.
 7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.
 8. 드롭다운 목록에서 Amazon CodeCatalyst를 선택합니다.
 9. Kubernetes에 배포 클러스터 작업을 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
- 또는
- Kubernetes 클러스터에 배포를 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 다운로드를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
10. 입력 및 구성 탭에서 필요에 따라 필드를 작성합니다. 각 필드의 설명은 '[Kubernetes 클러스터에 배포' 작업 YAML](#) 섹션을 참조하세요. 이 참조는 YAML 및 시각적 편집기 모두에 나타나는 각 필드(및 해당 YAML 속성 값)에 대한 자세한 정보를 제공합니다.
 11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
 12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 'Kubernetes 클러스터에 배포' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.

7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.
8. 드롭다운 목록에서 Amazon CodeCatalyst를 선택합니다.
9. Kubernetes에 배포 클러스터 작업을 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.

또는

 - Kubernetes 클러스터에 배포를 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 다운로드를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
10. 필요에 따라 YAML 코드의 속성을 수정합니다. 사용 가능한 각 속성에 대한 설명은 ['Kubernetes 클러스터에 배포' 작업 YAML](#)에서 볼 수 있습니다.
11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

'Kubernetes 클러스터에 배포' 변수

Kubernetes 클러스터에 배포 작업은 런타임에 다음 변수를 생성하고 설정합니다. 이를 사전 정의된 변수라고 합니다.

워크플로에서 이러한 변수를 참조하는 방법에 대한 자세한 내용은 [사전 정의된 변수 사용](#) 섹션을 참조하세요.

Key(키)	값
클러스터	워크플로 실행 중에 배포된 Amazon EKS 클러스터의 Amazon.com 리소스 이름(ARN)입니다. 예시: <code>arn:aws:eks:us-west-2:111122223333:cluster/codecatalyst-eks-cluster</code>
deployment-platform	배포 플랫폼의 이름입니다. AWS:EKS로 하드코딩되었습니다.

Key(키)	값
metadata	예약. 워크플로 실행 중에 배포된 클러스터와 관련된 JSON 형식 메타데이터입니다.
네임스페이스	클러스터가 배포된 Kubernetes 네임스페이스입니다. 예시: default
리소스	예약. 워크플로 실행 중에 배포된 리소스와 관련된 JSON 형식 메타데이터입니다.
서버	kubectl 같은 관리 도구를 사용하여 클러스터와 통신하는 데 사용할 수 있는 API 서버 엔드포인트의 이름입니다. 자세한 내용은 Amazon EKS 사용 설명서의 Amazon EKS 클러스터 엔드포인트 액세스 제어 를 참조하세요. 예시: https:// <i>random-string</i> .gr7.us-west-2.eks.amazonaws.com

'Kubernetes 클러스터에 배포' 작업 YAML

다음은 Kubernetes에 배포 클러스터 작업의 YAML 정의입니다. 이러한 작업 사용 방법을 배우려면 [워크플로를 사용하여 Amazon EKS에 배포](#) 섹션을 참조하세요.

이 작업 정의는 더 광범위한 워크플로 정의 파일 내의 섹션으로 존재합니다. 이 파일에 대한 자세한 내용은 [워크플로 YAML 정의](#)를 참조합니다.

Note

이어지는 대부분의 YAML 속성에는 시각적 편집기에 해당 UI 요소가 있습니다. UI 요소를 찾으려면 Ctrl+F를 사용합니다. 요소가 연결된 YAML 속성과 함께 나열됩니다.

```
# The workflow definition starts here.
```

```
# See ### ## for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
DeployToKubernetesCluster\_nn:
  Identifier: aws/kubernetes-deploy@v1
  DependsOn:
    - build-action
  Compute:
    - Type: EC2 | Lambda
    - Fleet: fleet-name
  Timeout: timeout-minutes
  Environment:
    Name: environment-name
  Connections:
    - Name: account-connection-name
    Role: DeployToEKS
  Inputs:
    # Specify a source or an artifact, but not both.
    Sources:
      - source-name-1
    Artifacts:
      - manifest-artifact
  Configuration:
    Namespace: namespace
    Region: us-east-1
    Cluster: eks-cluster
    Manifests: manifest-path
```

DeployToKubernetesCluster

(필수)

작업 이름을 지정합니다. 워크플로 내의 모든 작업 이름은 고유해야 합니다. 작업 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 작업 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

기본값: DeployToKubernetesCluster_nn.

해당 UI: 구성 탭/작업 표시 이름

Identifier

(*DeployToKubernetesCluster*/Identifier)

(필수)

작업을 식별합니다. 버전을 변경하려는 경우가 아니면 이 속성을 변경하지 마세요. 자세한 내용은 [사용할 작업 버전 지정](#) 섹션을 참조하세요.

기본값: aws/kubernetes-deploy@v1.

해당 UI: 워크플로 다이어그램/DeployToKubernetesCluster_nn/aws/kubernetes-deploy@v1 레이블

DependsOn

(*DeployToKubernetesCluster*/DependsOn)

(선택 사항)

이 작업을 실행하기 위해 성공적으로 실행해야 하는 작업, 작업 그룹 또는 게이트를 지정합니다.

'depends on' 함수에 대한 자세한 내용은 [작업 순서 지정](#) 섹션을 참조하세요.

해당 UI: 입력 탭/의존 - 선택 사항

Compute

(*DeployToKubernetesCluster*/Compute)

(선택 사항)

워크플로 작업을 실행하는 데 사용되는 컴퓨팅 엔진입니다. 워크플로 수준 또는 작업 수준에서 컴퓨팅을 지정할 수 있지만 둘 다 지정할 수는 없습니다. 워크플로 수준에서 지정하면 컴퓨팅 구성이 워크플로에 정의된 모든 작업에 적용됩니다. 워크플로 수준에서는 동일한 인스턴스에서 여러 작업을 실행할 수도 있습니다. 자세한 내용은 [작업 간에 컴퓨팅 공유](#) 섹션을 참조하세요.

해당 UI: 없음

Type

(*DeployToKubernetesCluster*/Compute/Type)

([Compute](#) 포함 시 필수)

컴퓨팅 엔진의 유형입니다. 다음 값 중 하나를 사용할 수 있습니다.

- EC2(시각 편집기) 또는 EC2(YAML 편집기)

작업 실행 중 유연성을 위해 최적화되었습니다.

- Lambda(시각 편집기) 또는 Lambda(YAML 편집기)

작업 시작 속도를 최적화했습니다.

컴퓨팅 유형에 대한 자세한 정보는 [컴퓨팅 유형](#)을 참고하세요.

해당 UI: 구성 탭/고급 - 선택 사항/컴퓨팅 유형

Fleet

(*DeployToKubernetesCluster*/Compute/Fleet)

(선택 사항)

워크플로 또는 워크플로 작업을 실행할 시스템 또는 플릿을 지정합니다. 온디맨드 플릿의 경우 작업이 시작되면 워크플로가 필요한 리소스를 프로비저닝하고 작업이 완료되면 시스템이 파괴됩니다. 온디맨드 플릿의 예시: Linux.x86-64.Large, Linux.x86-64.XLarge. 온디맨드 플릿에 대한 자세한 내용은 [온디맨드 플릿 속성](#) 섹션을 참조하세요.

프로비저닝된 플릿을 사용하면 워크플로 작업을 실행하도록 전용 시스템 세트를 구성할 수 있습니다. 이러한 시스템은 유휴 상태로 유지되므로 작업을 즉시 처리할 수 있습니다. 프로비저닝된 플릿에 대한 자세한 내용은 [프로비저닝된 플릿 속성](#) 섹션을 참조하세요.

Fleet 생략 시 기본값은 Linux.x86-64.Large입니다.

해당 UI: 구성 탭/고급 - 선택적/컴퓨팅 플릿

Timeout

(*DeployToKubernetesCluster*/Timeout)

(선택 사항)

CodeCatalyst가 작업을 종료하기 전에 작업을 실행할 수 있는 시간을 분(YAML 편집기) 또는 시간 및 분(시각적 편집기) 단위로 지정합니다. 최소값은 5분이고 최대값은 [CodeCatalyst의 워크플로 할당량](#)에 설명되어 있습니다. 기본 제한 시간은 최대 제한 시간과 동일합니다.

해당 UI: 구성 탭/제한 시간 - 선택 사항

Environment

(*DeployToKubernetesCluster*/Environment)

(필수)

작업에 사용할 CodeCatalyst 환경을 지정합니다. 작업은 선택한 환경에 지정된 AWS 계정 및 선택적 Amazon VPC에 연결됩니다. 작업은 환경에 지정된 기본 IAM 역할을 사용하여 연결하고 [Amazon VPC 연결](#)에 지정된 IAM 역할을 AWS 계정으로 사용하여 Amazon VPC에 연결합니다.

Note

기본 IAM 역할에 작업에 필요한 권한이 없는 경우 다른 역할을 사용하도록 작업을 구성할 수 있습니다. 자세한 내용은 [작업의 IAM 역할 변경](#) 섹션을 참조하세요.

환경에 대한 자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 및 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 구성 탭/환경

Name

(*DeployToKubernetesCluster*/Environment/Name)

([Environment](#) 포함 시 필수)

작업과 연결하려는 기존 환경의 이름을 지정합니다.

해당 UI: 구성 탭/환경

Connections

(*DeployToKubernetesCluster*/Environment/Connections)

(최신 버전의 작업에서는 선택 사항, 이전 버전에서는 필수)

작업과 연결할 계정 연결을 지정합니다. Environment에서 계정 연결을 최대 1개까지 지정할 수 있습니다.

계정 연결을 지정하지 않는 경우:

- 작업은 CodeCatalyst 콘솔의 환경에 지정된 AWS 계정 연결 및 기본 IAM 역할을 사용합니다. 환경에 계정 연결 및 기본 IAM 역할을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.

- 기본 IAM 역할에는 작업에 필요한 정책 및 권한이 포함되어야 합니다. 이러한 정책 및 권한이 무엇인지 확인하려면 작업의 YAML 정의 설명서에서 역할 속성에 대한 설명을 참조하세요.

계정 연결에 대한 자세한 정보는 [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 섹션을 참조하세요. 환경에 계정 연결을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/AWS 계정 연결

Name

(*DeployToKubernetesCluster*/Environment/Connections/Name)

(선택 사항)

계정 연결의 이름을 지정합니다.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/AWS 계정 연결

Role

(*DeployToKubernetesCluster*/Environment/Connections/Role)

([Connections](#) 포함 시 필수)

Kubernetes 클러스터에 배포 작업이 AWS에 액세스하는 데 사용하는 IAM 역할의 이름을 지정합니다. [CodeCatalyst 스페이스에 역할을 추가](#)했고 역할에 다음 정책이 포함되어 있는지 확인합니다.

IAM 역할을 지정하지 않으면 작업은 CodeCatalyst 콘솔의 [환경](#)에 나열된 기본 IAM 역할을 사용합니다. 환경에서 기본 역할을 사용하는 경우 다음 정책이 있는지 확인합니다.

- 다음 권한 정책:

⚠ Warning

다음 정책에 표시된 대로 권한을 제한합니다. 더 광범위한 권한을 가진 역할을 사용하면 보안 위험이 발생할 수 있습니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "eks:DescribeCluster",
        "eks:ListClusters"
      ],
      "Resource": "*"
    }
  ]
}
```

ℹ Note

역할을 처음 사용할 때는 리소스 정책 설명에서 다음 와일드카드를 사용한 다음 사용 가능한 리소스 이름으로 정책의 범위를 좁힙니다.

```
"Resource": "*"
```

- 다음 사용자 지정 신뢰 정책:

이 역할이 다음에 추가되었는지 확인합니다.

- 계정 연결. IAM 역할을 계정 연결에 추가하는 방법에 대한 자세한 내용은 [계정 연결에 IAM 역할 추가](#) 섹션을 참조하세요.
- Kubernetes ConfigMap. ConfigMap에 IAM 역할을 추가하는 방법에 대한 자세한 내용은 eksctl 설명서의 [IAM 사용자 및 역할 관리](#)를 참조하세요.

Tip

계정 연결 및 ConfigMap에 am IAM 역할을 추가하는 방법에 대한 지침은 [자습서: Amazon EKS에 애플리케이션 배포](#) 섹션을 참조하세요.

Note

원하는 경우 이 작업에서 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 사용할 수 있습니다. 이에 대한 자세한 내용은 [계정 및 스페이스의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 생성](#) 섹션을 참조하세요. CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할에 보안 위험을 초래할 수 있는 전체 액세스 권한이 있음을 이해합니다. 보안에 대한 우려가 적은 자습서 및 시나리오에서만 이 역할을 사용하는 것이 좋습니다.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/# ##의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/역할

Inputs

(*DeployToKubernetesCluster*/Inputs)

([Connections](#) 포함 시 필수)

이 Inputs 섹션에서는 워크플로 실행 중 DeployToKubernetesCluster에 필요한 데이터를 정의합니다.

Note

Amazon EKS에 배포 작업당 하나의 입력(소스 또는 아티팩트)만 허용됩니다.

해당 UI: 입력 탭

Sources

(*DeployToKubernetesCluster*/Inputs/Sources)

(매니페스트 파일이 소스 리포지토리에 저장된 경우 필수)

Kubernetes 매니페스트 파일 또는 파일이 소스 리포지토리에 저장되는 경우 해당 소스 리포지토리의 레이블을 지정합니다. 현재, WorkflowSource 레이블만 지원됩니다.

매니페스트 파일이 소스 리포지토리에 포함되지 않은 경우 다른 작업에서 생성된 아티팩트에 있어야 합니다.

소스에 대한 자세한 내용은 [워크플로에 소스 리포지토리 연결](#) 섹션을 참조하세요.

해당 UI: 입력 탭/소스 - 선택 사항

Artifacts - input

(*DeployToKubernetesCluster*/Inputs/Artifacts)

(매니페스트 파일이 이전 작업의 [출력 아티팩트](#)에 저장된 경우 필수)

Kubernetes 매니페스트 파일 또는 파일이 이전 작업에서 생성된 아티팩트에 포함된 경우 여기에 해당 아티팩트를 지정합니다. 매니페스트 파일이 아티팩트 내에 포함되어 있지 않은 경우 해당 파일은 소스 리포지토리에 있어야 합니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

해당 UI: 구성 탭/아티팩트 - 선택 사항

Configuration

(*DeployToKubernetesCluster*/Configuration)

(필수)

작업의 구성 속성을 정의할 수 있는 섹션입니다.

해당 UI: 구성 탭

Namespace

(*DeployToKubernetesCluster*/Configuration/Namespace)

(선택 사항)

Kubernetes 애플리케이션을 배포할 Kubernetes 네임스페이스를 지정합니다. 클러스터에서 네임스페이스를 사용하지 않는 경우 default를 사용합니다. 네임스페이스에 대한 자세한 내용은 Kubernetes 설명서의 [Kubernetes 네임스페이스를 사용하여 클러스터 세분화](#)를 참조하세요.

네임스페이스를 생략하면 default 값이 사용됩니다.

해당 UI: 구성 탭/네임스페이스

Region

(*DeployToKubernetesCluster*/Configuration/Region)

(필수)

Amazon EKS 클러스터 및 서비스가 상주하는 AWS 리전을 지정합니다. 리전 코드 목록은 AWS 일반 참조의 [리전 엔드포인트](#)를 참조하세요.

해당 UI: 구성 탭/리전

Cluster

(*DeployToKubernetesCluster*/Configuration/Cluster)

(필수)

기존 Amazon EKS 클러스터의 이름을 지정합니다. Kubernetes 클러스터에 배포 작업을 수행하면 컨테이너화된 애플리케이션이 이 클러스터에 배포됩니다. Amazon EKS 클러스터에 대한 자세한 내용은 Amazon EKS 사용 설명서의 [Amazon EKS 클러스터](#)를 참조하세요.

해당 UI: 구성 탭/클러스터

Manifests

(*DeployToKubernetesCluster*/Configuration/Manifests)

(필수)

Kubernetes 설명서에서 구성 파일, 구성 파일, config 파일 또는 단순히 구성이라고 하는 YAML 형식의 Kubernetes 매니페스트 파일(들)의 경로를 지정합니다.

여러 매니페스트 파일을 사용하는 경우 단일 폴더에 배치하고 해당 폴더를 참조합니다. 매니페스트 파일은 Kubernetes에서 영숫자로 처리되므로 파일 이름 앞에 숫자 또는 문자가 늘어나도록 하여 처리 순서를 제어해야 합니다. 예:

00-namespace.yaml

01-deployment.yaml

매니페스트 파일이 소스 리포지토리에 있는 경우 경로는 소스 리포지토리 루트 폴더를 기준으로 합니다. 파일이 이전 워크플로 작업의 아티팩트에 있는 경우 경로는 아티팩트 루트 폴더를 기준으로 합니다.

예시:

Manifests/

deployment.yaml

my-deployment.yaml

와일드카드(*)를 사용하지 않습니다.

Note

[헬름 차트](#) 및 [kustomization 파일](#)은 지원되지 않습니다.

매니페스트 파일에 대한 자세한 내용은 Kubernetes 설명서의 [리소스 구성 조직하기](#) 섹션을 참조하세요.

해당 UI: 구성 탭/매니페스트

CloudFormation 스택 배포

이 섹션에서는 CodeCatalyst 워크플로를 사용하여 AWS CloudFormation 스택을 배포하는 방법을 설명합니다. 이렇게 하려면 CloudFormation 스택 배포 작업을 워크플로에 추가해야 합니다. 작업은 사용자가 제공한 템플릿을 AWS 기반으로 리소스의 CloudFormation 스택에 배포합니다. 템플릿은 다음과 같을 수 있습니다.

- CloudFormation 템플릿 - 자세한 내용은 [템플릿 작업을 CloudFormation](#) 참조하세요.

- AWS SAM 템플릿 - 자세한 내용은 [AWS Serverless Application Model \(AWS SAM\) 사양](#)을 참조하세요.

Note

AWS SAM 템플릿을 사용하려면 먼저 [sam package](#) 작업을 사용하여 AWS SAM 애플리케이션을 패키징해야 합니다. Amazon CodeCatalyst 워크플로의 일부로 이 패키징을 자동으로 수행하는 방법을 보여주는 자습서는 [자습서: 서버리스 애플리케이션 배포](#) 섹션을 참조하세요.

스택이 이미 있는 경우 작업은 CloudFormation [CreateChangeSet](#) 작업을 실행한 다음 [ExecuteChangeSet](#) 작업을 실행합니다. 그러면 작업이 변경 사항이 배포될 때까지 대기한 뒤 결과에 따라 성공 또는 실패로 표시됩니다.

배포하려는 리소스가 포함된 CloudFormation 또는 AWS SAM 템플릿이 이미 있거나 AWS SAM 및와 같은 도구를 사용하여 워크플로 빌드 작업의 일부로 스택을 자동으로 생성할 계획인 경우 CloudFormation 스택 배포 작업을 사용합니다. [AWS Cloud Development Kit \(AWS CDK\)](#). ???

CloudFormation에서 작성하거나 CloudFormation 스택 배포 작업과 함께 AWS SAM 사용할 수 있는 템플릿에는 제한이 없습니다.

Tip

CloudFormation 스택 배포 작업을 사용하여 서버리스 애플리케이션을 배포하는 방법을 보여주는 자습서는 [자습서: 서버리스 애플리케이션 배포](#) 섹션을 참조하세요.

주제

- ['스 CloudFormation 텍 배포' 작업에서 사용하는 런타임 이미지](#)
- [자습서: 서버리스 애플리케이션 배포](#)
- ['스 CloudFormation 텍 배포' 작업 추가](#)
- [롤백 구성](#)
- ['스 CloudFormation 텍 배포' 변수](#)
- ['스 CloudFormation 텍 배포' 작업 YAML](#)

'스 CloudFormation 템플릿 배포' 작업에서 사용하는 런타임 이미지

CloudFormation 스택 배포 작업은 [2022년 11월 이미지](#)에서 실행됩니다. 자세한 내용은 [활성 이미지](#) 단원을 참조하십시오.

자습서: 서버리스 애플리케이션 배포

이 자습서에서는 워크플로를 사용하여 CloudFormation 스택으로 서버리스 애플리케이션을 빌드, 테스트 및 배포하는 방법을 알아봅니다.

이 자습서의 애플리케이션은 'Hello World' 메시지를 출력하는 간단한 웹 애플리케이션입니다. AWS Lambda 함수와 Amazon API Gateway로 구성되며의 확장인 [AWS Serverless Application Model \(AWS SAM\)](#)을 사용하여 빌드합니다 [CloudFormation](#).

주제

- [사전 조건](#)
- [1단계: 소스 리포지토리 생성](#)
- [2단계: AWS 역할 생성](#)
- [3단계: CodeCatalyst에 AWS 역할 추가](#)
- [4단계: Amazon S3 버킷 생성](#)
- [5단계: 소스 파일 추가](#)
- [6단계: 워크플로 생성 및 실행](#)
- [7단계: 변경](#)
- [정리](#)

사전 조건

시작하기 전:

- 연결된 AWS 계정이 있는 CodeCatalyst 스페이스가 필요합니다. 자세한 내용은 [스페이스 생성](#) 단원을 참조하십시오.
- 스페이스에는 다음과 같은 빈 프로젝트가 필요합니다.

```
codecatalyst-cfn-project
```

처음부터 시작 옵션을 사용하여 이 프로젝트를 생성합니다.

자세한 내용은 [Amazon CodeCatalyst에서 빈 프로젝트 생성](#) 섹션을 참조하세요.

- 프로젝트에는 다음과 같은 CodeCatalyst 환경이 필요합니다.

```
codecatalyst-cfn-environment
```

다음과 같이 이 환경을 구성합니다.

- 비프로덕션과 같은 유형을 선택합니다.
- AWS 계정에 연결합니다.
- 기본 IAM 역할의 경우 아무 역할이나 선택합니다. 나중에 다른 역할을 지정합니다.

자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 섹션을 참조하세요.

1단계: 소스 리포지토리 생성

이 단계에서는 CodeCatalyst에 소스 리포지토리를 생성합니다. 이 리포지토리는 Lambda 함수 파일과 같은 자습서의 소스 파일을 저장하는 데 사용됩니다.

소스 리포지토리에 대한 자세한 정보는 [소스 리포지토리 생성](#) 섹션을 참조하세요.

소스 리포지토리를 생성하려면

- CodeCatalyst의 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
- 리포지토리 추가를 선택하고 리포지토리 생성을 선택합니다.
- 리포지토리 이름에 다음과 같이 입력합니다.

```
codecatalyst-cfn-source-repository
```

- 생성(Create)을 선택합니다.

이제 codecatalyst-cfn-source-repository 리포지토리를 생성했습니다.

2단계: AWS 역할 생성

이 단계에서는 다음 AWS IAM 역할을 생성합니다.

- 역할 배포 - CodeCatalyst CloudFormation 스택 배포 작업에 서버리스 애플리케이션을 배포할 AWS 계정 및 CloudFormation 서비스에 액세스할 수 있는 권한을 부여합니다. CloudFormation 스택 배포 작업은 워크플로의 일부입니다.

- 빌드 역할 - CodeCatalyst 빌드 작업에 AWS 계정에 액세스하고 서버리스 애플리케이션 패키지가 저장될 Amazon S3에 쓸 수 있는 권한을 부여합니다. 빌드 작업은 워크플로의 일부입니다.
- 스택 역할 - 나중에 제공할 AWS SAM 템플릿에 지정된 리소스를 읽고 수정할 수 있는 CloudFormation 권한을 부여합니다. 또한 CloudWatch에 권한을 부여합니다.

IAM 역할에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 [IAM 역할](#) 섹션을 참조하세요.

Note

시간을 절약하기 위해 이전에 나열한 세 가지 역할 대신

CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할이라

는 단일 역할을 생성할 수 있습니다. 자세한 내용은 [계정 및 스페이스의](#)

[CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 생성](#) 섹션을 참조하세요.

CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할에는 보안 위험을 초래할 수 있는 매우 광범위한 권한이 있음을 이해합니다. 보안에 대한 우려가 적은 자습서 및 시나리오에서만 이 역할을 사용하는 것이 좋습니다. 이 자습서에서는 이전에 나열된 세 가지 역할을 생성하고 있다고 가정합니다.

Note

[Lambda 실행 역할](#)도 필요하지만 5단계에서 워크플로를 실행할 때 sam-template.yml 파일이 생성하므로 지금 생성할 필요가 없습니다.

배포 역할을 생성하려면

1. 역할에 대한 정책을 다음과 같이 생성합니다.
 - a. 에 로그인합니다 AWS.
 - b. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
 - c. 탐색 창에서 Policies를 선택합니다.
 - d. 정책 생성을 선택합니다.
 - e. JSON 탭을 선택합니다.
 - f. 기존 코드를 삭제합니다.

- g. 다음 코드를 붙여넣습니다.

 Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

- h. 다음: 태그를 선택합니다.
i. 다음: 검토를 선택합니다.
j. 이름에 다음과 같이 입력합니다.

```
codecatalyst-deploy-policy

```

- k. 정책 생성을 선택합니다.

이제 권한 정책을 생성했습니다.

2. 다음과 같이 배포 역할을 생성합니다.

- a. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
b. 사용자 지정 신뢰 정책을 선택합니다.
c. 기존 사용자 지정 신뢰 정책을 삭제합니다.
d. 다음 사용자 지정 신뢰 정책을 추가합니다.
e. 다음을 선택합니다.
f. 권한 정책에서 codecatalyst-deploy-policy를 검색하고 해당 확인란을 선택합니다.
g. 다음을 선택합니다.
h. 역할 이름에 다음과 같이 입력합니다.

```
codecatalyst-deploy-role

```

- i. 역할 설명에 다음과 같이 입력합니다.

```
CodeCatalyst deploy role

```

- j. 역할 생성을 선택합니다.

이제 신뢰 정책 및 권한 정책으로 배포 역할을 생성했습니다.

3. 다음과 같이 배포 역할 ARN을 가져옵니다.

- a. 탐색 창에서 Roles를 선택합니다.
- b. 검색 상자에 방금 생성한 역할의 이름을 입력합니다(codecatalyst-deploy-role).
- c. 목록에서 역할을 선택합니다.

역할의 요약 페이지가 나타납니다.

- d. 상단에서 ARN 값을 복사합니다.

이제 적절한 권한으로 배포 역할을 생성하고 ARN을 획득했습니다.

빌드 역할을 생성하려면

1. 역할에 대한 정책을 다음과 같이 생성합니다.

- a. 에 로그인합니다 AWS.
- b. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
- c. 탐색 창에서 Policies를 선택합니다.
- d. 정책 생성을 선택합니다.
- e. JSON 탭을 선택합니다.
- f. 기존 코드를 삭제합니다.
- g. 다음 코드를 붙여넣습니다.

 Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

- h. 다음: 태그를 선택합니다.
- i. 다음: 검토를 선택합니다.
- j. 이름에 다음과 같이 입력합니다.

codecatalyst-build-policy

- k. 정책 생성을 선택합니다.

이제 권한 정책을 생성했습니다.

- 2. 다음과 같이 빌드 역할을 생성합니다.

- a. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
- b. 사용자 지정 신뢰 정책을 선택합니다.
- c. 기존 사용자 지정 신뢰 정책을 삭제합니다.
- d. 다음 사용자 지정 신뢰 정책을 추가합니다.
- e. 다음을 선택합니다.
- f. 권한 정책에서 **codecatalyst-build-policy**를 검색하고 해당 확인란을 선택합니다.
- g. 다음을 선택합니다.
- h. 역할 이름에 다음과 같이 입력합니다.

codecatalyst-build-role

- i. 역할 설명에 다음과 같이 입력합니다.

CodeCatalyst build role

- j. 역할 생성을 선택합니다.

이제 신뢰 정책 및 권한 정책으로 빌드 역할을 생성했습니다.

- 3. 다음과 같이 빌드 역할 ARN을 가져옵니다.

- a. 탐색 창에서 Roles를 선택합니다.
- b. 검색 상자에 방금 생성한 역할의 이름을 입력합니다(**codecatalyst-build-role**).
- c. 목록에서 역할을 선택합니다.

역할의 요약 페이지가 나타납니다.

- d. 상단에서 ARN 값을 복사합니다.

이제 적절한 권한으로 빌드 역할을 생성하고 ARN을 획득했습니다.

스택 역할을 생성하려면

1. 스택을 배포하려는 계정을 AWS 사용하여 로그인합니다.
2. IAM 콘솔(<https://console.aws.amazon.com/iam/>)을 엽니다.
3. 다음과 같이 스택 역할을 생성합니다.
 - a. 탐색 창에서 Roles를 선택합니다.
 - b. 역할 생성(Create role)을 선택합니다.
 - c. AWS 서비스를 선택합니다.
 - d. 사용 사례 섹션의 드롭다운 목록에서 CloudFormation을 선택합니다.
 - e. CloudFormation 라디오 버튼을 선택합니다.
 - f. 하단에서 다음을 선택합니다.
 - g. 검색 상자를 사용하여 다음 권한 정책을 찾은 다음 해당 확인란을 선택합니다.

Note

정책을 검색했지만 표시되지 않는 경우 필터 지우기를 선택하고 다시 시도해야 합니다.

- CloudWatchFullAccess
- AWS CloudFormationFullAccess
- IAMFullAccess
- AWS Lambda_FullAccess
- AmazonAPIGatewayAdministrator
- AmazonS3FullAccess
- AmazonEC2ContainerRegistryFullAccess

첫 번째 정책은 경보가 발생할 때 스택 롤백을 활성화하기 위해 CloudWatch에 대한 액세스를 허용합니다.

나머지 정책은가이 자습서에서 배포할 스택의 서비스 및 리소스에 AWS SAM 액세스할 수 있도록 허용합니다. 자세한 내용은 AWS Serverless Application Model 개발자 안내서의 [권한](#) 섹션을 참조하세요.

- h. 다음을 선택합니다.
- i. 역할 이름에 다음과 같이 입력합니다.

`codecatalyst-stack-role`

- j. 역할 생성을 선택합니다.
4. 다음과 같이 스택 역할 ARN을 가져옵니다.
- a. 탐색 창에서 Roles를 선택합니다.
 - b. 검색 상자에 방금 생성한 역할의 이름을 입력합니다(`codecatalyst-stack-role`).
 - c. 목록에서 역할을 선택합니다.
 - d. 요약 섹션에 표시된 ARN 값을 복사합니다. 나중에 필요합니다.

이제 적절한 권한으로 스택 역할을 생성했으며 ARN을 획득했습니다.

3단계: CodeCatalyst에 AWS 역할 추가

이 단계에서는 빌드 역할(`codecatalyst-build-role`)을 추가하고 스페이스의 CodeCatalyst 계정 연결에 역할(`codecatalyst-deploy-role`)을 배포합니다.

Note

스택 역할(`codecatalyst-stack-role`)을 연결에 추가할 필요가 없습니다. 이는 배포 역할을 사용하여 CodeCatalyst와 AWS 간에 연결이 이미 설정된 후 CloudFormation(CodeCatalyst 아님)에서 스택 역할을 사용하기 때문입니다. 스택 역할은 CodeCatalyst에서 AWS에 대한 액세스 권한을 얻는 데 사용되지 않으므로 계정 연결과 연결할 필요가 없습니다.

계정 연결에 빌드 및 배포 역할을 추가하려면

1. CodeCatalyst에서 스페이스로 이동합니다.
2. AWS 계정을 선택합니다. 계정 연결 목록이 나타납니다.
3. 빌드 및 배포 역할을 생성한 계정을 나타내는 AWS 계정 연결을 선택합니다.
4. 관리 콘솔에서 역할 AWS 관리를 선택합니다.

Amazon CodeCatalyst 스페이스에 IAM 역할 추가 페이지가 나타납니다. 페이지에 액세스하려면 로그인해야 할 수 있습니다.

5. IAM에서 생성한 기존 역할 추가를 선택합니다.

드롭다운 목록이 나타납니다. 목록에는 `codecatalyst-runner.amazonaws.com` 및 `codecatalyst.amazonaws.com` 서비스 위탁자가 포함된 신뢰 정책이 있는 모든 IAM 역할이 표시됩니다.

6. 드롭다운 목록에서 `codecatalyst-build-role`을 선택하고 역할 추가를 선택합니다.

7. IAM 역할 추가를 선택하고 IAM에서 생성한 기존 역할 추가를 선택한 다음 드롭다운 목록에서 `codecatalyst-deploy-role`을 선택합니다. [Add role]을 선택합니다.

이제 스페이스에 빌드 및 배포 역할을 추가했습니다.

8. Amazon CodeCatalyst 표시 이름의 값을 복사합니다. 워크플로를 생성할 때 나중에 이 값이 필요합니다.

4단계: Amazon S3 버킷 생성

이 단계에서는 서버리스 애플리케이션의 배포 패키지 .zip 파일을 저장하는 Amazon S3 버킷을 생성합니다.

Amazon S3 버킷을 생성하려면

1. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.
2. 기본 창에서 버킷 생성을 선택합니다.
3. 버킷 이름에 다음과 같이 입력합니다.

codecatalyst-cfn-s3-bucket

4. AWS 리전에서 리전을 선택합니다. 이 자습서에서는 미국 서부(오리건) `us-west-2`를 선택했다고 가정합니다. Amazon S3에서 지원되는 리전에 대한 자세한 내용은 AWS 일반 참조의 [Amazon Simple Storage Service 엔드포인트 및 할당량](#)을 참조하세요.
5. 페이지 맨 아래에 있는 버킷 생성 버튼을 선택합니다.

이제 미국 서부(오리건) `us-west-2` 리전에 **codecatalyst-cfn-s3-bucket** 버킷을 생성했습니다.

5단계: 소스 파일 추가

이 단계에서는 CodeCatalyst 소스 리포지토리에 여러 애플리케이션 소스 파일을 추가합니다. `hello-world` 폴더에는 배포할 애플리케이션 파일이 포함되어 있습니다. `tests` 폴더에는 단위 테스트가 포함되어 있습니다. 폴더의 구조는 다음과 같습니다.

```
.
|- hello-world
|  |- tests
|    |- unit
|      |- test-handler.js
|  |- app.js
|- .npmignore
|- package.json
|- sam-template.yml
|- setup-sam.sh
```

.npmignore 파일

.npmignore 파일은 npm이 애플리케이션 패키지에서 제외해야 하는 파일 및 폴더를 나타냅니다. 이 자습서에서는 npm이 애플리케이션의 일부가 아니기 때문에 tests 폴더를 제외합니다.

.npmignore 파일을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. codecatalyst-cfn-project 프로젝트를 선택합니다.
3. 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
4. 소스 리포지토리 목록에서 codecatalyst-cfn-source-repository 리포지토리를 선택합니다.
5. 파일에서 파일 생성을 선택합니다.
6. 파일 이름에 다음과 같이 입력합니다.

.npmignore

7. 다음 코드를 텍스트 상자에 입력합니다.

tests/*

8. 커밋을 선택한 다음 커밋을 다시 선택합니다.

이제 리포지토리 루트에서 .npmignore 파일을 생성했습니다.

package.json 파일

package.json 파일에는 프로젝트 이름, 버전 번호, 설명, 종속성 및 애플리케이션과 상호 작용하고 실행하는 방법을 설명하는 기타 세부 정보와 같은 노드 프로젝트에 대한 중요한 메타데이터가 포함되어 있습니다.

package.json 이 자습서의 test 에는 종속성 목록과 스크립트가 포함되어 있습니다. 테스트 스크립트는 다음 작업을 수행합니다.

- [mocha](#)를 사용하여 테스트 스크립트는 hello-world/tests/unit/에 지정된 단위 테스트를 실행하고 [xunit](#) 리포터를 사용하여 junit.xml 파일에 결과를 기록합니다.
- [이stanbul\(nyc\)](#)을 사용하면 테스트 스크립트가 [클로버](#) 리포터를 사용하여 코드 적용 범위 보고서(clover.xml)를 생성합니다. 자세한 내용은 이stanbul 설명서의 [대체 리포터 사용](#)을 참조하세요.

package.json 파일을 추가하려면

1. 리포지토리의 파일에서 파일 생성을 선택합니다.
2. 파일 이름에 다음과 같이 입력합니다.

package.json

3. 다음 코드를 텍스트 상자에 입력합니다.

```
{
  "name": "hello_world",
  "version": "1.0.0",
  "description": "hello world sample for NodeJS",
  "main": "app.js",
  "repository": "https://github.com/aws-labs/aws-sam-cli/tree/develop/samcli/local/init/templates/cookiecutter-aws-sam-hello-nodejs",
  "author": "SAM CLI",
  "license": "MIT",
  "dependencies": {
    "axios": "^0.21.1",
    "nyc": "^15.1.0"
  },
  "scripts": {
    "test": "nyc --reporter=clover mocha hello-world/tests/unit/ --reporter xunit --reporter-option output=junit.xml"
  },
  "devDependencies": {
```

```

    "aws-sdk": "^2.815.0",
    "chai": "^4.2.0",
    "mocha": "^8.2.1"
  }
}

```

4. 커밋을 선택한 다음 커밋을 다시 선택합니다.

이제 리포지토리 루트에서 `package.json` 파일을 생성했습니다.

sam-template.yml 파일

`sam-template.yml` 파일에는 Lambda 함수 및 API Gateway를 배포하고 함께 구성하는 지침이 포함되어 있습니다. [AWS Serverless Application Model 템플릿 사양](#)을 따르는 템플릿 CloudFormation 사양을 따릅니다.

는 유용한 [AWS::Serverless::Function](#) 리소스 유형을 AWS SAM 제공하기 때문에이 자습서 AWS SAM에서는 일반 템플릿 대신 CloudFormation 템플릿을 사용합니다. 이 유형은 기본 CloudFormation 구문을 사용하기 위해 일반적으로 작성해야 하는 보이지 않는 구성을 다수 수행합니다. 예를 들어 `AWS::Serverless::Function`은 Lambda 함수, Lambda 실행 역할 및 함수를 시작하는 이벤트 소스 매핑을 생성합니다. 기본 CloudFormation을 사용하여 작성하려면 이 모든 것을 코딩해야 합니다.

이 자습서에서는 미리 작성된 템플릿을 사용하지만 빌드 작업을 사용하여 워크플로의 일부로 템플릿을 생성할 수 있습니다. 자세한 내용은 [CloudFormation 스택 배포](#) 섹션을 참조하세요.

`sam-template.yml` 파일을 추가하려면

1. 리포지토리의 파일에서 파일 생성을 선택합니다.
2. 파일 이름에 다음과 같이 입력합니다.

sam-template.yml

3. 다음 코드를 텍스트 상자에 입력합니다.

```

AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: >
  serverless-api

  Sample SAM Template for serverless-api

```

```
# More info about Globals: https://github.com/aws-labs/serverless-application-model/blob/master/docs/globals.rst
Globals:
  Function:
    Timeout: 3

Resources:
  HelloWorldFunction:
    Type: AWS::Serverless::Function # For details on this resource type,
    see https://github.com/aws-labs/serverless-application-model/blob/master/versions/2016-10-31.md#awsserverlessfunction
    Properties:
      CodeUri: hello-world/
      Handler: app.lambdaHandler
      Runtime: nodejs12.x
      Events:
        HelloWorld:
          Type: Api # For details on this event source type, see
          https://github.com/aws-labs/serverless-application-model/blob/master/versions/2016-10-31.md#api
          Properties:
            Path: /hello
            Method: get

Outputs:
  # ServerlessRestApi is an implicit API created out of the events key under
  Serverless::Function
  # Find out about other implicit resources you can reference within AWS SAM at
  # https://github.com/aws-labs/serverless-application-model/blob/master/docs/internals/generated_resources.rst#api
  HelloWorldApi:
    Description: "API Gateway endpoint URL for the Hello World function"
    Value: !Sub "https://${ServerlessRestApi}.execute-api.
    ${AWS::Region}.amazonaws.com/Prod/hello/"
  HelloWorldFunction:
    Description: "Hello World Lambda function ARN"
    Value: !GetAtt HelloWorldFunction.Arn
  HelloWorldFunctionIamRole:
    Description: "Implicit Lambda execution role created for the Hello World
    function"
    Value: !GetAtt HelloWorldFunctionRole.Arn
```

4. 커밋을 선택한 다음 커밋을 다시 선택합니다.

이제 리포지토리의 루트 폴더 아래에 `sam-template.yml` 파일을 추가했습니다.

setup-sam.sh 파일

`setup-sam.sh` 파일에는 AWS SAM CLI 유틸리티 다운로드 및 설치 지침이 포함되어 있습니다. 워크플로는 이 유틸리티를 사용하여 `hello-world` 소스를 패키징합니다.

`setup-sam.sh` 파일을 추가하려면

1. 리포지토리의 파일에서 파일 생성을 선택합니다.
2. 파일 이름에 다음과 같이 입력합니다.

```
setup-sam.sh
```

3. 다음 코드를 텍스트 상자에 입력합니다.

```
#!/usr/bin/env bash
echo "Setting up sam"

yum install unzip -y

curl -LO https://github.com/aws/aws-sam-cli/releases/latest/download/aws-sam-cli-linux-x86_64.zip
unzip -qq aws-sam-cli-linux-x86_64.zip -d sam-installation-directory

./sam-installation-directory/install; export AWS_DEFAULT_REGION=us-west-2
```

앞의 코드에서 *us-west-2*를 AWS 리전으로 바꿉니다.

4. 커밋을 선택한 다음 커밋을 다시 선택합니다.

이제 리포지토리 루트에서 `setup-sam.sh` 파일을 생성했습니다.

app.js 파일

`app.js`는 Lambda 함수 코드를 포함합니다. 이 자습서에서는 코드가 `hello world` 텍스트를 반환합니다.

app.js 파일을 추가하려면

1. 리포지토리의 파일에서 파일 생성을 선택합니다.
2. 파일 이름에 다음과 같이 입력합니다.

```
hello-world/app.js
```

3. 다음 코드를 텍스트 상자에 입력합니다.

```
// const axios = require('axios')
// const url = 'http://checkip.amazonaws.com/';
let response;

/**
 *
 * Event doc: https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html#api-gateway-simple-proxy-for-lambda-input-format
 * @param {Object} event - API Gateway Lambda Proxy Input Format
 *
 * Context doc: https://docs.aws.amazon.com/lambda/latest/dg/nodejs-prog-model-context.html
 * @param {Object} context
 *
 * Return doc: https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html
 * @returns {Object} object - API Gateway Lambda Proxy Output Format
 */
exports.lambdaHandler = async (event, context) => {
  try {
    // const ret = await axios(url);
    response = {
      'statusCode': 200,
      'body': JSON.stringify({
        message: 'hello world',
        // location: ret.data.trim()
      })
    }
  } catch (err) {
    console.log(err);
    return err;
  }
}
```

```
    return response
  };
```

4. 커밋을 선택한 다음 커밋을 다시 선택합니다.

이제 hello-world 폴더와 app.js 파일을 생성했습니다.

test-handler.js 파일

test-handler.js 파일에는 Lambda 함수에 대한 단위 테스트가 포함되어 있습니다.

test-handler.js 파일을 추가하려면

1. 리포지토리의 파일에서 파일 생성을 선택합니다.
2. 파일 이름에 다음과 같이 입력합니다.

```
hello-world/tests/unit/test-handler.js
```

3. 다음 코드를 텍스트 상자에 입력합니다.

```
'use strict';

const app = require('../../app.js');
const chai = require('chai');
const expect = chai.expect;
var event, context;

describe('Tests index', function () {
  it('verifies successful response', async () => {
    const result = await app.lambdaHandler(event, context)

    expect(result).to.be.an('object');
    expect(result.statusCode).to.equal(200);
    expect(result.body).to.be.an('string');

    let response = JSON.parse(result.body);

    expect(response).to.be.an('object');
    expect(response.message).to.be.equal("hello world");
    // expect(response.location).to.be.an("string");
  });
});
```

4. 커밋을 선택한 다음 커밋을 다시 선택합니다.

이제 hello-world/tests/unit 폴더 아래에 test-handler.js 파일을 추가했습니다.

이제 모든 소스 파일을 추가했습니다.

잠시 시간을 내어 작업을 다시 확인하고 모든 파일을 올바른 폴더에 넣었는지 확인하세요. 폴더의 구조는 다음과 같습니다.

```
.
|- hello-world
|  |- tests
|     |- unit
|        |- test-handler.js
|  |- app.js
|- .npmignore
|- README.md
|- package.json
|- sam-template.yml
|- setup-sam.sh
```

6단계: 워크플로 생성 및 실행

이 단계에서는 Lambda 소스 코드를 패키징하고 배포하는 워크플로를 생성합니다. 워크플로는 순차적으로 실행되는 다음 구성 요소로 구성됩니다.

- 트리거 - 이 트리거는 소스 리포지토리에 변경 사항을 푸시할 때 워크플로 실행을 자동으로 시작합니다. 트리거에 대한 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 주제를 참조하세요.
- 테스트 작업(Test) - 트리거 시 이 작업은 [노드 패키지 관리자\(npm\)](#)를 설치한 다음 npm run test 명령을 실행합니다. 이 명령은 package.json 파일에 정의된 test 스크립트를 실행하도록 npm에 지시합니다. test 스크립트는 다시 단위 테스트를 실행하고 테스트 보고서(junit.xml)와 코드 적용 범위 보고서(clover.xml)의 두 가지 보고서를 생성합니다. 자세한 내용은 [package.json 파일](#) 섹션을 참조하세요.

다음으로 테스트 작업은 XML 보고서를 CodeCatalyst 보고서로 변환하고 테스트 작업의 보고서 탭 아래 CodeCatalyst 콘솔에 표시합니다.

테스트 작업에 대한 자세한 내용은 [워크플로를 사용한 테스트](#) 섹션을 참조하세요.

- 빌드 작업(BuildBackend) - 테스트 작업이 완료되면 빌드 작업은 AWS SAM CLI를 다운로드하여 설치하고, hello-world 소스를 패키징하고, 패키지를 Lambda 서비스가 예상하는 Amazon

S3 버킷에 복사합니다. 또한 작업은 라는 새 AWS SAM 템플릿 파일을 출력sam-template-packaged.yml하여 라는 출력 아티팩트에 배치합니다buildArtifact.

빌드 작업에 대한 자세한 내용은 [워크플로로 빌드하기](#) 섹션을 참조하세요.

- 배포 작업(DeployCloudFormationStack) - 빌드 작업이 완료되면 배포 작업은 빌드 작업()에서 생성된 출력 아티팩트를 찾고 그 안의 AWS SAM 템플릿을 buildArtifact찾은 다음 템플릿을 실행합니다. AWS SAM 템플릿은 서버리스 애플리케이션을 배포하는 스택을 생성합니다.

워크플로 생성

1. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
2. 워크플로 생성을 선택합니다.
3. 소스 리포지토리에서 codecatalyst-cfn-source-repository를 선택합니다.
4. 브랜치에서 main을 선택합니다.
5. 생성(Create)을 선택합니다.
6. YAML 샘플 코드를 삭제합니다.
7. 다음 YAML 코드를 추가합니다.

Note

다음 YAML 코드에서 원하는 경우 Connections: 섹션을 생략할 수 있습니다. 이 섹션을 생략하는 경우 환경의 기본 IAM 역할 필드에 지정된 역할에 [2단계: AWS 역할 생성](#)에 설명된 두 역할의 권한 및 신뢰 정책이 포함되어 있는지 확인해야 합니다. 기본 IAM 역할이 있는 환경 설정에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.

```
Name: codecatalyst-cfn-workflow
SchemaVersion: 1.0

Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  Test:
    Identifier: aws/managed-test@v1
    Inputs:
```



```

Sources:
  - WorkflowSource
Outputs:
Reports:
  CoverageReport:
    Format: CLOVERXML
    IncludePaths:
      - "coverage/*"
  TestReport:
    Format: JUNITXML
    IncludePaths:
      - junit.xml
Configuration:
  Steps:
    - Run: npm install
    - Run: npm run test
BuildBackend:
  Identifier: aws/build@v1
  DependsOn:
    - Test
  Environment:
    Name: codecatalyst-cfn-environment
  Connections:
    - Name: codecatalyst-account-connection
      Role: codecatalyst-build-role
Inputs:
  Sources:
    - WorkflowSource
Configuration:
  Steps:
    - Run: . ./setup-sam.sh
    - Run: sam package --template-file sam-template.yml --s3-
bucket codecatalyst-cfn-s3-bucket --output-template-file sam-template-packaged.yml
--region us-west-2
  Outputs:
    Artifacts:
      - Name: buildArtifact
      Files:
        - "**/*"
DeployCloudFormationStack:
  Identifier: aws/cfn-deploy@v1
  DependsOn:
    - BuildBackend
  Environment:

```

```

Name: codecatalyst-cfn-environment
Connections:
  - Name: codecatalyst-account-connection
    Role: codecatalyst-deploy-role
Inputs:
  Artifacts:
    - buildArtifact
  Sources: []
Configuration:
  name: codecatalyst-cfn-stack
  region: us-west-2
  role-arn: arn:aws:iam::111122223333:role/StackRole
  template: ./sam-template-packaged.yml
  capabilities: CAPABILITY_IAM,CAPABILITY_AUTO_EXPAND

```

앞의 코드에서

- *codecatalyst-cfn-environment* 인스턴스 두 개를 모두 에서 생성한 환경 이름으로 바꿉니다.
- *codecatalyst-account-connection* 인스턴스 두 개를 모두 계정 연결의 표시 이름으로 바꿉니다. 표시 이름은 숫자일 수 있습니다. 자세한 내용은 [3단계: CodeCatalyst에 AWS 역할 추가](#) 섹션을 참조하세요.
- [2단계: AWS 역할 생성](#)에서 생성한 빌드 역할의 이름을 갖는 *codecatalyst-build-role*.
- [4단계: Amazon S3 버킷 생성](#)에서 생성한 Amazon S3 버킷의 이름을 갖는 *codecatalyst-cfn-s3-bucket*입니다.
- Amazon S3 버킷이 있는 리전(첫 번째 인스턴스)과 스택이 배포될 리전(두 번째 인스턴스)이 있는 *us-west-2*의 두 인스턴스 모두. 이러한 리전은 다를 수 있습니다. 이 자습서에서는 두 리전이 모두 *us-west-2*로 설정되어 있다고 가정합니다. Amazon S3 및에서 지원하는 리전에 대한 자세한 내용은의 서비스 엔드포인트 및 할당량을 CloudFormation참조하세요AWS 일반 참조. <https://docs.aws.amazon.com/general/latest/gr/aws-service-information.html>
- [2단계: AWS 역할 생성](#)에서 생성한 배포 역할의 이름을 갖는 *codecatalyst-deploy-role*.
- [사전 조건](#)에서 생성한 환경 이름을 갖는 *codecatalyst-cfn-environment*.
- [2단계: AWS 역할 생성](#)에서 생성한 스택 역할의 Amazon 리소스 이름(ARN)을 갖는 *arn:aws:iam::111122223333:role/StackRole*.

Note

빌드, 배포 및 스택 역할을 생성하지 않기로 결정한 경우 *codecatalyst-build-role*, *codecatalyst-deploy-role* 및 *arn:aws:iam::111122223333:role/StackRole*을 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할의 이름 또는 ARN으로 바꿉니다. 이에 대한 자세한 내용은 [2단계: AWS 역할 생성](#) 섹션을 참조하세요.

이전에 표시된 코드의 속성에 대한 자세한 내용은 섹션을 참조하세요 ['스 CloudFormation 태크 배포' 작업 YAML](#).

8. (선택 사항) 커밋하기 전에 YAML 코드가 유효한지 확인하려면 검증을 선택합니다.
9. 커밋을 선택합니다.
10. 워크플로 커밋 대화 상자에서 다음을 입력합니다.
 - a. 워크플로 파일 이름의 경우 기본값인 *codecatalyst-cfn-workflow*를 유지합니다.
 - b. 커밋 메시지에 다음을 입력합니다.

add initial workflow file

- c. 리포지토리에서 *codecatalyst-cfn-source-repositor*를 선택합니다.
- d. 브랜치 이름에서 기본을 선택합니다.
- e. 커밋을 선택합니다.

이제 워크플로를 생성했습니다. 워크플로 상단에 정의된 트리거로 인해 워크플로 실행이 자동으로 시작됩니다. 특히 *codecatalyst-cfn-workflow.yaml* 파일을 소스 리포지토리에 커밋(및 푸시)할 때 트리거가 워크플로 실행을 시작했습니다.

진행 중인 워크플로 실행을 보려면

1. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
2. 방금 생성한 *codecatalyst-cfn-workflow* 워크플로를 선택합니다.
3. 실행 탭을 선택합니다.
4. 실행 ID 옆에서 실행 ID를 선택합니다.

5. 테스트를 선택하여 테스트 진행 상황을 확인합니다.
6. BuildBackend를 선택하여 빌드 진행 상황을 확인합니다.
7. DeployCloudFormationStack을 선택하여 배포 진행 상황을 확인합니다.

실행 세부 정보 보기에 대한 자세한 내용은 [워크플로 실행 상태 및 세부 정보 보기](#)를 참조하세요.

8. DeployCloudFormationStack 작업이 완료되면 다음을 수행합니다.
 - 워크플로 실행에 성공한 경우 다음 절차로 이동합니다.
 - 테스트 또는 BuildBackend 작업에서 워크플로 실행이 실패한 경우 로그를 선택하여 문제를 해결합니다.
 - DeployCloudFormationStack 작업에서 워크플로 실행이 실패한 경우 배포 작업을 선택한 다음 요약 탭을 선택합니다. CloudFormation 이벤트 섹션으로 스크롤하여 자세한 오류 메시지를 확인합니다. 롤백이 발생한 경우 워크플로를 다시 실행하기 전에 CloudFormation AWS 콘솔을 통해 codecatalyst-cfn-stack 스택을 삭제합니다.

배포를 확인하려면

1. 성공적으로 배포한 후 상단 근처의 가로 메뉴 모음에서 변수(7)를 선택합니다. (오른쪽 창에서 변수를 선택하지 마세요.)
2. HelloWorldApi 옆에 `https://` URL을 브라우저에 붙여 넣습니다.

Lambda 함수의 hello world JSON 메시지가 표시되며, 이는 워크플로가 Lambda 함수 및 API Gateway를 성공적으로 배포하고 구성했음을 나타냅니다.

Tip

CodeCatalyst가 몇 가지 작은 구성으로 워크플로 다이어그램에 이 URL을 표시하도록 할 수 있습니다. 자세한 내용은 [워크플로 다이어그램에 앱 URL 표시](#) 섹션을 참조하세요.

단위 테스트 결과 및 코드 적용 범위를 확인하려면

1. 워크플로 다이어그램에서 테스트를 선택한 다음 보고서를 선택합니다.
2. TestReport를 선택하여 단위 테스트 결과를 보거나 CoverageReport를 선택하여 테스트 중인 파일의 코드 적용 범위 세부 정보를 봅니다. 이 경우 app.js 및 test-handler.js입니다.

배포된 리소스를 확인하려면

1. 에 로그인 AWS Management Console 하고 <https://console.aws.amazon.com/apigateway/> API Gateway 콘솔을 엽니다.
2. AWS SAM 템플릿이 생성한 codecatalyst-cfn-stack API를 관찰합니다. API 이름은 워크플로 정의 파일(codecatalyst-cfn-workflow.yaml)의 Configuration/name 값에서 가져옵니다.
3. <https://console.aws.amazon.com/lambda/> AWS Lambda 콘솔을 엽니다.
4. 탐색 창에서 함수를 선택합니다.
5. codecatalyst-cfn-stack-HelloWorldFunction-*string* Lambda 함수를 선택합니다.
6. API Gateway가 함수의 트리거인 방법을 확인할 수 있습니다. 이 통합은 리소스 유형에 따라 AWS SAM `AWS::Serverless::Function` 자동으로 구성되었습니다.

7단계: 변경

이 단계에서는 Lambda 소스 코드를 변경하고 커밋합니다. 이 커밋은 새 워크플로 실행을 시작합니다. 이 실행은 Lambda 콘솔에 지정된 기본 트래픽 이동 구성을 사용하는 청록색 체계로 새 Lambda 함수를 배포합니다.

Lambda 소스를 변경하려면

1. CodeCatalyst에서 프로젝트로 이동합니다.
2. 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
3. codecatalyst-cfn-source-repository 소스 리포지토리를 선택합니다.
4. 애플리케이션 파일 변경:
 - a. hello-world 폴더를 선택합니다.
 - b. app.js 파일을 선택합니다.
 - c. 편집을 선택합니다.
 - d. 23번 줄에서 hello world를 **Tutorial complete!**로 변경합니다.
 - e. 커밋을 선택한 다음 커밋을 다시 선택합니다.

커밋으로 인해 워크플로 실행이 시작됩니다. 이름 변경을 반영하도록 단위 테스트를 업데이트하지 않았기 때문에 이 실행은 실패합니다.

5. 단위 테스트를 업데이트합니다.
 - a. hello-world\tests\unit\test-handler.js를 선택합니다.

- b. 편집을 선택합니다.
- c. 19번 줄에서 `hello world`를 **Tutorial complete!**로 변경합니다.
- d. 커밋을 선택한 다음 커밋을 다시 선택합니다.

커밋으로 인해 다른 워크플로 실행이 시작됩니다. 이 실행이 성공합니다.

- 6. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
- 7. `codecatalyst-cfn-workflow`를 선택한 다음 실행을 선택합니다.
- 8. 최신 실행의 실행 ID를 선택합니다. 상태는 진행 중이어야 합니다.
- 9. 테스트, BuildBackend 및 DeployCloudFormationStack을 선택하여 워크플로 실행 진행 상황을 확인합니다.
- 10. 워크플로가 완료되면 상단 근처의 변수(7)를 선택합니다.
- 11. HelloWorldApi 옆에 `https://` URL을 브라우저에 붙여 넣습니다.

새 애플리케이션이 성공적으로 배포되었음을 나타내는 `Tutorial complete!` 메시지가 브라우저에 나타납니다.

정리

요금이 부과되지 않도록 이 자습서에서 사용하는 파일과 서비스를 정리합니다.

CodeCatalyst 콘솔에서 정리하려면

- 1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
- 2. `codecatalyst-cfn-workflow`를 삭제합니다.
- 3. `codecatalyst-cfn-environment`를 삭제합니다.
- 4. `codecatalyst-cfn-source-repository`를 삭제합니다.
- 5. `codecatalyst-cfn-project`를 삭제합니다.

에서 정리하려면 AWS Management Console

- 1. 다음과 같이 CloudFormation에서 정리합니다.
 - a. <https://console.aws.amazon.com/cloudformation> CloudFormation 콘솔을 엽니다.
 - b. `codecatalyst-cfn-stack`를 삭제합니다.

스택을 삭제하면 API Gateway 및 Lambda 서비스에서 모든 자습서 리소스가 제거됩니다.

2. 다음과 같이 Amazon S3에서 정리합니다.
 - a. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.
 - b. `codecatalyst-cfn-s3-bucket`을 선택합니다.
 - c. 버킷 콘텐츠를 삭제합니다.
 - d. 버킷을 삭제합니다.
3. 다음과 같이 IAM에서 정리합니다.
 - a. IAM 콘솔(<https://console.aws.amazon.com/iam/>)을 엽니다.
 - b. `codecatalyst-deploy-policy`를 삭제합니다.
 - c. `codecatalyst-build-policy`를 삭제합니다.
 - d. `codecatalyst-stack-policy`를 삭제합니다.
 - e. `codecatalyst-deploy-role`를 삭제합니다.
 - f. `codecatalyst-build-role`를 삭제합니다.
 - g. `codecatalyst-stack-role`를 삭제합니다.

이 자습서에서는 CodeCatalyst 워크플로 및 스택 배포 작업을 사용하여 서버리스 애플리케이션을 CloudFormation CloudFormation 스택으로 배포하는 방법을 배웠습니다.

'스 CloudFormation 텍 배포' 작업 추가

다음 지침에 따라 CloudFormation 스택 배포 작업을 워크플로에 추가합니다.

Visual

시각적 편집기를 사용하여 '스 CloudFormation 텍 배포' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.

8. 드롭다운 목록에서 Amazon CodeCatalyst를 선택합니다.
9. CloudFormation 스택 배포 작업을 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.

또는

 - CloudFormation 스택 배포를 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 다운로드를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
10. 입력 및 구성 탭에서 필요에 따라 필드를 작성합니다. 각 필드의 설명은 '[스 CloudFormation 템플릿 배포](#)' 작업 YAML 섹션을 참조하세요. 이 참조는 YAML 및 시각적 편집기 모두에 나타나는 각 필드(및 해당 YAML 속성 값)에 대한 자세한 정보를 제공합니다.
11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 '스 CloudFormation 템플릿 배포' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.
8. 드롭다운 목록에서 Amazon CodeCatalyst를 선택합니다.
9. CloudFormation 스택 배포 작업을 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.

또는

- CloudFormation 스택 배포를 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 다운로드를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
- 10. 필요에 따라 YAML 코드의 속성을 수정합니다. 사용 가능한 각 속성에 대한 설명은 '[스 CloudFormation 템플릿 배포](#)' 작업 YAML에서 볼 수 있습니다.
- 11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
- 12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

롤백 구성

기본적으로 CloudFormation 스택 배포 작업이 실패하면 CloudFormation 가 스택을 마지막으로 알려진 안정 상태로 롤백합니다. 작업이 실패할 때뿐만 아니라 지정된 Amazon CloudWatch 경보가 발생할 때 롤백이 발생하도록 동작을 변경할 수 있습니다. CloudWatch 경보에 대한 자세한 내용을 알아보려면 Amazon CloudWatch 사용 설명서의 [Amazon CloudWatch 경보 사용](#)을 참조하세요.

작업이 실패할 때 CloudFormation이 스택을 롤백하지 않도록 기본 동작을 변경할 수도 있습니다.

다음 지침에 따라 롤백을 구성합니다.

Note

롤백을 수동으로 시작할 수 없습니다.

Visual

시작하기 전 준비 사항

1. 작동하는 CloudFormation 스택 배포 작업이 포함된 [워크플로](#)가 있는지 확인합니다. 자세한 내용은 [CloudFormation 스택 배포](#) 단원을 참조하십시오.
2. 스택 배포 작업의 스택 역할 - 선택적 필드에 지정된 역할에 CloudWatchFullAccess 권한을 포함해야 합니다. CloudFormation 적절한 권한이 있는 역할 생성에 대한 자세한 내용은 [2단계: AWS 역할 생성](#)을 참조하세요.

'스 CloudFormation 템플릿 배포' 작업에 대한 롤백 경보를 구성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. CloudFormation 스택 배포 작업을 선택합니다.
8. 세부 정보 창에서 구성을 선택합니다.
9. 하단에서 고급을 확장합니다.
10. 경보 ARN 모니터링에서 경보 추가를 선택합니다.
11. 다음 필드에 정보를 입력합니다.

- 경보 ARN

롤백 트리거를 추가하려면 Amazon CloudWatch 경보의 Amazon 리소스 이름(ARN)을 지정합니다. 예를 들어 `arn:aws:cloudwatch::123456789012:alarm/MyAlarm`입니다. 최대 5개의 롤백 트리거를 가질 수 있습니다.

 Note

CloudWatch 경보 ARN을 지정하는 경우 작업이 CloudWatch에 액세스할 수 있도록 추가 권한도 구성해야 합니다. 자세한 내용은 [롤백 구성](#) 섹션을 참조하세요.

- 모니터링 시간

CloudFormation이 지정된 경보를 모니터링하는 데 걸리는 시간을 0~180분으로 지정합니다. 모니터링은 모든 스택 리소스가 배포된 후 시작됩니다. 지정된 모니터링 시간 내에 경보가 발생하면 배포가 실패하고 CloudFormation이 전체 스택 작업을 롤백합니다.

기본값: 0. CloudFormation은 스택 리소스가 배포되는 동안에만 경보를 모니터링하며 이후는 모니터링하지 않습니다.

YAML

'CloudFormation 스택 배포' 작업에 대한 롤백 트리거를 구성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. CloudFormation 스택 배포 작업이 포함된 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. YAML 코드에 `monitor-alarm-arns` 및 `monitor-timeout-in-minutes` 속성을 추가하여 롤백 트리거를 추가합니다. 각 속성에 대한 설명은 ['스 CloudFormation 스택 배포' 작업 YAML](#) 섹션을 참조하세요.
8. CloudFormation 스택 배포 작업의 `role-arn` 속성에 지정된 역할에 `CloudWatchFullAccess` 권한을 포함해야 합니다. 적절한 권한이 있는 역할 생성에 대한 자세한 내용은 [2단계: AWS 역할 생성](#)을 참조하세요.

Visual

'CloudFormation 스택 배포' 작업에 대한 롤백을 끄려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. CloudFormation 스택 배포 작업이 포함된 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. CloudFormation 스택 배포 작업을 선택합니다.
8. 세부 정보 창에서 구성을 선택합니다.
9. 하단에서 고급을 확장합니다.

10. 롤백 비활성화를 켭니다.

YAML

'CloudFormation 스택 배포' 작업에 대한 롤백을 끄려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. CloudFormation 스택 배포 작업이 포함된 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 롤백을 중지하려면 YAML 코드에 `disable-rollback: 1` 속성을 추가합니다. 이 속성에 대한 설명은 '[스 CloudFormation 스택 배포](#)' 작업 [YAML](#) 섹션을 참조하세요.

'스 CloudFormation 텍 배포' 변수

CloudFormation 스택 배포 작업은 런타임에 다음 변수를 생성하고 설정합니다. 이를 사전 정의된 변수라고 합니다.

워크플로에서 이러한 변수를 참조하는 방법에 대한 자세한 내용은 [사전 정의된 변수 사용](#) 섹션을 참조하세요.

Key(키)	값
deployment-platform	배포 플랫폼의 이름입니다. AWS:CloudFormation 로 하드코딩되었습니다.
리전	워크플로 실행 중에 배포 AWS 리전 된의 리전 코드입니다. 예시: us-west-2

Key(키)	값
stack-id	배포 작업의 Amazon 리소스 이름(ARN)입니다. 예시: arn:aws:cloudformation:us-west-2:111122223333:stack/co decatalyst-cfn-stack/6aad43 80-100a-11ec-a10a-03b8a84d40df

'스 CloudFormation 템플릿 배포' 작업 YAML

다음은 CloudFormation 스택 배포 작업의 YAML 정의입니다. 이러한 작업 사용 방법을 배우려면 [CloudFormation 스택 배포](#) 섹션을 참조하세요.

이 작업 정의는 더 광범위한 워크플로 정의 파일 내의 섹션으로 존재합니다. 이 파일에 대한 자세한 내용은 [워크플로 YAML 정의](#)를 참조합니다.

Note

이어지는 대부분의 YAML 속성에는 시각적 편집기에 해당 UI 요소가 있습니다. UI 요소를 찾으려면 Ctrl+F를 사용합니다. 요소가 연결된 YAML 속성과 함께 나열됩니다.

```
# The workflow definition starts here.
# See ### ## for details.
```

```
Name: MyWorkflow
SchemaVersion: 1.0
Actions:
```

```
# The action definition starts here.
```

```
DeployCloudFormationStack:
```

```
  Identifier: aws/cfn-deploy@v1
```

```
  DependsOn:
```

```
    - build-action
```

```
  Compute:
```

```
    Type: EC2 | Lambda
```

```
    Fleet: fleet-name
```

```
    Timeout: timeout-minutes
```

```
    Environment:
```

```

Name: environment-name
Connections:
  - Name: account-connection-name
    Role: DeployRole
Inputs:
Sources:
  - source-name-1
Artifacts:
  - CloudFormation-artifact
Configuration:
  name: stack-name
  region: us-west-2
  template: template-path
  role-arn: arn:aws:iam::123456789012:role/StackRole
  capabilities: CAPABILITY_IAM,CAPABILITY_NAMED_IAM,CAPABILITY_AUTO_EXPAND
  parameter-overrides: KeyOne=ValueOne,KeyTwo=ValueTwo | path-to-JSON-file
  no-execute-changeset: 1|0
  fail-on-empty-changeset: 1|0
  disable-rollback: 1|0
  termination-protection: 1|0
  timeout-in-minutes: minutes
  notification-arns: arn:aws:sns:us-east-1:123456789012:MyTopic,arn:aws:sns:us-east-1:123456789012:MyOtherTopic
  monitor-alarm-arns: arn:aws:cloudwatch::123456789012:alarm/MyAlarm,arn:aws:cloudwatch::123456789012:alarm/MyOtherAlarm
  monitor-timeout-in-minutes: minutes
  tags: '[{"Key": "MyKey1", "Value": "MyValue1"}, {"Key": "MyKey2", "Value": "MyValue2"}]'

```

DeployCloudFormationStack

(필수)

작업 이름을 지정합니다. 워크플로 내의 모든 작업 이름은 고유해야 합니다. 작업 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 작업 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

기본값: DeployCloudFormationStack_nn.

해당 UI: 구성 탭/작업 표시 이름

Identifier

(*DeployCloudFormationStack/Identifier*)

(필수)

작업을 식별합니다. 버전을 변경하려는 경우가 아니면 이 속성을 변경하지 마세요. 자세한 내용은 [사용할 작업 버전 지정](#) 섹션을 참조하세요.

기본값: aws/cfn-deploy@v1.

해당 UI: 워크플로 다이어그램/DeployCloudFormationStack_nn/aws/cfn-deploy@v1 레이블

DependsOn

(*DeployCloudFormationStack*/DependsOn)

(선택 사항)

이 작업을 실행하기 위해 성공적으로 실행해야 하는 작업, 작업 그룹 또는 게이트를 지정합니다.

'depends on' 함수에 대한 자세한 내용은 [작업 순서 지정](#) 섹션을 참조하세요.

해당 UI: 입력 탭/의존 - 선택 사항

Compute

(*DeployCloudFormationStack*/Compute)

(선택 사항)

워크플로 작업을 실행하는 데 사용되는 컴퓨팅 엔진입니다. 워크플로 수준 또는 작업 수준에서 컴퓨팅을 지정할 수 있지만 둘 다 지정할 수는 없습니다. 워크플로 수준에서 지정하면 컴퓨팅 구성이 워크플로에 정의된 모든 작업에 적용됩니다. 워크플로 수준에서는 동일한 인스턴스에서 여러 작업을 실행할 수도 있습니다. 자세한 내용은 [작업 간에 컴퓨팅 공유](#) 섹션을 참조하세요.

해당 UI: 없음

Type

(*DeployCloudFormationStack*/Compute/Type)

([Compute](#) 포함 시 필수)

컴퓨팅 엔진의 유형입니다. 다음 값 중 하나를 사용할 수 있습니다.

- EC2(시각 편집기) 또는 EC2(YAML 편집기)

작업 실행 중 유연성을 위해 최적화되었습니다.

- Lambda(시각 편집기) 또는 Lambda(YAML 편집기)

작업 시작 속도를 최적화했습니다.

컴퓨팅 유형에 대한 자세한 정보는 [컴퓨팅 유형](#)을 참고하세요.

해당 UI: 구성 탭/고급 - 선택 사항/컴퓨팅 유형

Fleet

(*DeployCloudFormationStack*/Compute/Fleet)

(선택 사항)

워크플로 또는 워크플로 작업을 실행할 시스템 또는 플릿을 지정합니다. 온디맨드 플릿의 경우 작업이 시작되면 워크플로가 필요한 리소스를 프로비저닝하고 작업이 완료되면 시스템이 파괴됩니다. 온디맨드 플릿의 예시: Linux.x86-64.Large, Linux.x86-64.XLarge. 온디맨드 플릿에 대한 자세한 내용은 [온디맨드 플릿 속성](#) 섹션을 참조하세요.

프로비저닝된 플릿을 사용하면 워크플로 작업을 실행하도록 전용 시스템 세트를 구성할 수 있습니다. 이러한 시스템은 유휴 상태로 유지되므로 작업을 즉시 처리할 수 있습니다. 프로비저닝된 플릿에 대한 자세한 내용은 [프로비저닝된 플릿 속성](#) 섹션을 참조하세요.

Fleet 생략 시 기본값은 Linux.x86-64.Large입니다.

해당 UI: 구성 탭/고급 - 선택적/컴퓨팅 플릿

Timeout

(*DeployCloudFormationStack*/Timeout)

(선택 사항)

CodeCatalyst가 작업을 종료하기 전에 작업을 실행할 수 있는 시간을 분(YAML 편집기) 또는 시간 및 분(시각적 편집기) 단위로 지정합니다. 최소값은 5분이고 최대값은 [CodeCatalyst의 워크플로 할당량](#)에 설명되어 있습니다. 기본 제한 시간은 최대 제한 시간과 동일합니다.

해당 UI: 구성 탭/제한 시간을 분 단위로 표시 - 선택 사항

Environment

(*DeployCloudFormationStack*/Environment)

(필수)

작업에 사용할 CodeCatalyst 환경을 지정합니다. 작업은 선택한 환경에 지정된 AWS 계정 및 선택적 Amazon VPC에 연결됩니다. 작업은 환경에 지정된 기본 IAM 역할을 사용하여 연결하고 [Amazon VPC 연결](#)에 지정된 IAM 역할을 AWS 계정으로 사용하여 Amazon VPC에 연결합니다.

Note

기본 IAM 역할에 작업에 필요한 권한이 없는 경우 다른 역할을 사용하도록 작업을 구성할 수 있습니다. 자세한 내용은 [작업의 IAM 역할 변경](#) 섹션을 참조하세요.

환경에 대한 자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 및 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 구성 탭/환경

Name

(*DeployCloudFormationStack*/Environment/Name)

([Environment](#) 포함 시 필수)

작업과 연결하려는 기존 환경의 이름을 지정합니다.

해당 UI: 구성 탭/환경

Connections

(*DeployCloudFormationStack*/Environment/Connections)

(최신 버전의 작업에서는 선택 사항, 이전 버전에서는 필수)

작업과 연결할 계정 연결을 지정합니다. Environment에서 계정 연결을 최대 1개까지 지정할 수 있습니다.

계정 연결을 지정하지 않는 경우:

- 작업은 CodeCatalyst 콘솔의 환경에 지정된 AWS 계정 연결 및 기본 IAM 역할을 사용합니다. 환경에 계정 연결 및 기본 IAM 역할을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.
- 기본 IAM 역할에는 작업에 필요한 정책 및 권한이 포함되어야 합니다. 이러한 정책 및 권한이 무엇인지 확인하려면 작업의 YAML 정의 설명서에서 역할 속성에 대한 설명을 참조하세요.

계정 연결에 대한 자세한 정보는 [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 섹션을 참조하세요. 환경에 계정 연결을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/AWS 계정 연결

Name

(*DeployCloudFormationStack*/Environment/Connections/Name)

([Connections](#) 포함 시 필수)

계정 연결의 이름을 지정합니다.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/AWS 계정 연결

Role

(*DeployCloudFormationStack*/Environment/Connections/Role)

([Connections](#) 포함 시 필수)

CloudFormation 스택 배포 작업이 AWS 및 CloudFormation 서비스에 액세스하는 데 사용하는 IAM 역할의 이름을 지정합니다. [CodeCatalyst 스페이스에 역할을 추가](#)했고 역할에 다음 정책이 포함되어 있는지 확인합니다.

IAM 역할을 지정하지 않으면 작업은 CodeCatalyst 콘솔의 [환경](#)에 나열된 기본 IAM 역할을 사용합니다. 환경에서 기본 역할을 사용하는 경우 다음 정책이 있는지 확인합니다.

- 다음 권한 정책:

Warning

다음 정책에 표시된 대로 권한을 제한합니다. 더 광범위한 권한을 가진 역할을 사용하면 보안 위험이 발생할 수 있습니다.

Note

역할을 처음 사용할 때는 리소스 정책 설명에서 다음 와일드카드를 사용한 다음 사용 가능한 리소스 이름으로 정책의 범위를 좁힙니다.

```
"Resource": "*"

```

- 다음 사용자 지정 신뢰 정책:

Note

원하는 경우 이 작업에서 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 사용할 수 있습니다. 이에 대한 자세한 내용은 [계정 및 스페이스의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 생성](#) 섹션을 참조하세요.

CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할에 보안 위험을 초래할 수 있는 전체 액세스 권한이 있음을 이해합니다. 보안에 대한 우려가 적은 자습서 및 시나리오에서만 이 역할을 사용하는 것이 좋습니다.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/역할

Inputs

(*DeployCloudFormationStack*/Inputs)

(선택 사항)

이 Inputs 섹션에서는 워크플로 실행 중에 `DeployCloudFormationStack` 필요한 데이터를 정의합니다.

Note

CloudFormation 스택 배포 작업당 최대 4개의 입력(소스 1개 및 아티팩트 3개)이 허용됩니다.

서로 다른 입력(소스 및 아티팩트)에 있는 파일을 참조해야 하는 경우 소스 입력은 기본 입력이고 아티팩트는 보조 입력입니다. 보조 입력의 파일에 대한 참조는 특수 접두사를 사용하여 기본 입력에서 파일을 구분합니다. 자세한 내용은 [예시: 여러 아티팩트에서 파일 참조](#)를 참조하세요.

해당 UI: 입력 탭

Sources

(*DeployCloudFormationStack*/Inputs/Sources)

(CloudFormation 또는 AWS SAM 템플릿이 소스 리포지토리에 저장된 경우 필수)

CloudFormation 또는 AWS SAM 템플릿이 소스 리포지토리에 저장되어 있는 경우 해당 소스 리포지토리의 레이블을 지정합니다. 현재, WorkflowSource 레이블만 지원됩니다.

CloudFormation 또는 AWS SAM 템플릿이 소스 리포지토리에 포함되지 않은 경우 다른 작업에서 생성된 아티팩트 또는 Amazon S3 버킷에 있어야 합니다.

소스에 대한 자세한 내용은 [워크플로에 소스 리포지토리 연결](#) 섹션을 참조하세요.

해당 UI: 입력 탭/소스 - 선택 사항

Artifacts - input

(*DeployCloudFormationStack*/Inputs/Artifacts)

(CloudFormation 또는 AWS SAM 템플릿이 이전 작업의 [출력 아티팩트](#)에 저장된 경우 필수)

배포하려는 CloudFormation 또는 AWS SAM 템플릿이 이전 작업에서 생성된 아티팩트에 포함된 경우 여기에 해당 아티팩트를 지정합니다. CloudFormation 템플릿이 아티팩트 내에 포함되어 있지 않은 경우 소스 리포지토리 또는 Amazon S3 버킷에 있어야 합니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

해당 UI: 구성 탭/아티팩트 - 선택 사항

Configuration

(*DeployCloudFormationStack*/Configuration)

(필수)

작업의 구성 속성을 정의할 수 있는 섹션입니다.

해당 UI: 구성 탭

name

(*DeployCloudFormationStack*/Configuration/name)

(필수)

CloudFormation 스택 배포 작업이 생성하거나 업데이트하는 CloudFormation 스택의 이름을 지정합니다.

해당 UI: 구성 탭/스택 이름

region

(*DeployCloudFormationStack*/Configuration/region)

(필수)

스택을 배포할 AWS 리전 를 지정합니다. 리전 코드 목록은 [리전 엔드포인트](#)를 참조하세요.

해당 UI: 구성 탭/스택 리전

template

(*DeployCloudFormationStack*/Configuration/template)

(필수)

CloudFormation 또는 AWS SAM 템플릿 파일의 이름과 경로를 지정합니다. 템플릿은 JSON 또는 YAML 형식일 수 있으며 소스 리포지토리, 이전 작업의 아티팩트 또는 Amazon S3 버킷에 상주할 수 있습니다. 템플릿 파일이 소스 리포지토리 또는 아티팩트에 있는 경우 경로는 소스 또는 아티팩트 루트를 기준으로 합니다. 템플릿이 Amazon S3 버킷에 있는 경우 경로는 템플릿의 객체 URL 값입니다.

예시:

```
./MyFolder/MyTemplate.json
```

```
MyFolder/MyTemplate.yml
```

```
https://MyBucket.s3.us-west-2.amazonaws.com/MyTemplate.yml
```

Note

템플릿의 파일 경로에 접두사를 추가하여 찾을 아티팩트 또는 소스를 나타내야 할 수 있습니다. 자세한 내용은 [소스 리포지토리 파일 참조](#) 및 [아티팩트의 파일 참조](#) 섹션을 참조하세요.

해당 UI: 구성 탭/템플릿

role-arn

(*DeployCloudFormationStack*/Configuration/role-arn)

(필수)

스택 역할의 Amazon 리소스 이름(ARN)을 지정합니다. CloudFormation은 이 역할을 사용하여 스택의 리소스에 액세스하고 수정합니다. 예시: `arn:aws:iam::123456789012:role/StackRole`.

스택 역할에 다음이 포함되어 있는지 확인합니다.

- 하나 이상의 권한 정책. 정책은 스택에 있는 리소스에 따라 달라집니다. 예를 들어 스택에 AWS Lambda 함수가 포함된 경우 Lambda에 대한 액세스 권한을 부여하는 권한을 추가해야 합니다. [자습서: 서버리스 애플리케이션 배포](#)에 설명된 자습서를 따른 경우 일반적인 서버리스 애플리케이션 스택을 배포할 때 스택 역할에 필요한 권한을 나열하는 [스택 역할을 생성하려면](#) 프로시저가 포함됩니다.

Warning

스택의 리소스에 액세스하기 위해 CloudFormation 서비스에 필요한 권한으로 권한을 제한합니다. 더 광범위한 권한을 가진 역할을 사용하면 보안 위험이 발생할 수 있습니다.

- 다음 신뢰 정책:

JSON

```
{
```

```

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

```

필요에 따라 이 역할을 계정 연결에 연결합니다. IAM 역할을 계정 연결과 연결하는 방법에 대한 자세한 내용은 [계정 연결에 IAM 역할 추가](#) 섹션을 참조하세요. 스택 역할을 계정 연결과 연결하지 않으면 스택 역할이 시각적 편집기의 스택 역할 드롭다운 목록에 표시되지 않지만 YAML 편집기를 사용하여 `role-arn` 필드에 역할 ARN을 계속 지정할 수 있습니다.

Note

원하는 경우 이 작업에서 `CodeCatalystWorkflowDevelopmentRole-spaceName` 역할을 사용할 수 있습니다. 이에 대한 자세한 내용은 [계정 및 스페이스의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 생성](#) 섹션을 참조하세요.

`CodeCatalystWorkflowDevelopmentRole-spaceName` 역할에 보안 위험을 초래할 수 있는 전체 액세스 권한이 있음을 이해합니다. 보안에 대한 우려가 적은 자습서 및 시나리오에서만 이 역할을 사용하는 것이 좋습니다.

해당 UI: 구성 탭/스택 역할 - 선택 사항

capabilities

(*DeployCloudFormationStack*/Configuration/capabilities)

(필수)

가 특정 스택을 CloudFormation 생성하도록 허용하는 데 필요한 IAM 기능 목록을 지정합니다. 대부분의 경우 `CAPABILITY_IAM`, `CAPABILITY_NAMED_IAM`, `CAPABILITY_AUTO_EXPAND`의 기본값인 `capabilities`를 그대로 둡니다.

CloudFormation 스택 배포 작업의 로그에 `##[error] requires capabilities:`

`[capability-name]` 오류가 표시되면 문제를 해결하는 방법에 대한 자세한 내용은 [IAM 기능 오류를 해결하려면 어떻게 해야 하나요?](#) 섹션을 참조하세요.

IAM 기능에 대한 자세한 내용은 [IAM 사용 설명서의 CloudFormation 템플릿에서 IAM 리소스 확인을](#) 참조하세요.

해당 UI: 구성 탭/고급/기능

parameter-overrides

(*DeployCloudFormationStack*/Configuration/parameter-overrides)

(선택 사항)

CloudFormation 또는 AWS SAM 템플릿에서 기본값이 없거나 기본값이 아닌 값을 지정하려는 파라미터를 지정합니다. 파라미터에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 [파라미터를](#) 참조하세요.

parameter-overrides 속성은 다음을 허용합니다.

- 파라미터와 값이 포함된 JSON 파일입니다.
- 파라미터 및 값의 쉼표로 구분된 목록입니다.

JSON 파일을 지정하려면

1. JSON 파일이 다음 구문 중 하나를 사용하는지 확인합니다.

```
{
  "Parameters": {
    "Param1": "Value1",
    "Param2": "Value2",
    ...
  }
}
```

또는...

```
[
  {
    "ParameterKey": "Param1",
```



```

    "ParameterValue": "Value1"
  },
  ...
]

```

(다른 구문이 있지만 작성 시 CodeCatalyst에서 지원하지 않습니다.) JSON 파일에서 CloudFormation 파라미터를 지정하는 방법에 대한 자세한 내용은 AWS CLI 명령 참조의 [지원되는 JSON 구문](#)을 참조하세요.

2. 다음 형식 중 하나를 사용하여 JSON 파일의 경로를 지정합니다.

- JSON 파일이 이전 작업의 출력 아티팩트에 있는 경우 다음을 사용합니다.

```
file:///artifacts/current-action-name/output-artifact-name/path-to-json-file
```

자세한 내용은 예시 1을 참조하세요.

- JSON 파일이 소스 리포지토리에 있는 경우 다음을 사용합니다.

```
file:///sources/WorkflowSource/path-to-json-file
```

자세한 내용은 예시 2를 참조하세요.

예시 1 - JSON 파일은 출력 아티팩트에 있습니다.

```

##My workflow YAML
...
Actions:
  MyBuildAction:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ParamArtifact
          Files:
            - params.json
    Configuration:
      ...
  MyDeployCFNStackAction:
    Identifier: aws/cfn-deploy@v1
    Configuration:
      parameter-overrides: file:///artifacts/MyDeployCFNStackAction/
ParamArtifact/params.json

```

예시 2 - JSON 파일은 소스 리포지토리의 my/folder 폴더에 있습니다.

```
##My workflow YAML
...
Actions:
  MyDeployCloudFormationStack:
    Identifier: aws/cfn-deploy@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      parameter-overrides: file:///sources/WorkflowSource/my/folder/params.json
```

쉼표로 구분된 파라미터 목록을 사용하려면

- 다음 형식을 사용하여 parameter-overrides 속성에 파라미터 이름-값 페어를 추가합니다.

param-1=value-1,param-2=value-2

예를 들어 다음 CloudFormation 템플릿을 가정합니다.

```
##My CloudFormation template

Description: My CloudFormation template

Parameters:
  InstanceType:
    Description: Defines the Amazon EC2 compute for the production server.
    Type: String
    Default: t2.micro
    AllowedValues:
      - t2.micro
      - t2.small
      - t3.medium

Resources:
  ...
```

... 다음과 같이 parameter-overrides 속성을 설정할 수 있습니다.

```
##My workflow YAML
...
Actions:
...
  DeployCloudFormationStack:
    Identifier: aws/cfn-deploy@v1
    Configuration:
      parameter-overrides: InstanceType=t3.medium,UseVPC=true
```

Note

undefined 값을 사용하여 해당 값 없이 파라미터 이름을 지정할 수 있습니다. 예:

`parameter-overrides: MyParameter=undefined`

그 효과는 스택 업데이트 중에 CloudFormation이 지정된 파라미터 이름에 기존 파라미터 값을 사용한다는 것입니다.

해당 UI:

- 구성 탭/고급/파라미터 재정의
- 구성 탭/고급/파라미터 재정의/파일을 사용하여 재정의 지정
- 구성 탭/고급/파라미터 재정의/값 세트를 사용하여 재정의 지정

`no-execute-changeset`

(*DeployCloudFormationStack*/Configuration/no-execute-changeset)

(선택 사항)

CodeCatalyst가 CloudFormation 변경 세트를 생성한 다음 실행하기 전에 중지할지 여부를 지정합니다. 이렇게 하면 CloudFormation 콘솔에서 변경 세트를 검토할 수 있습니다. 변경 세트가 양호한 것으로 판단되면 이 옵션을 비활성화한 다음 워크플로를 다시 실행하여 CodeCatalyst가 중지 없이 변경 세트를 생성하고 실행할 수 있도록 합니다. 기본값은 중지 없이 변경 세트를 생성하고 실행하는 것입니다. 자세한 내용은 AWS CLI 명령 참조의 CloudFormation [배포](#) 파라미터를 참조하세요. 변경 세트 보기에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 [변경 세트 보기](#)를 참조하세요.

해당 UI: 구성 탭/고급/변경 세트 실행 안 함

fail-on-empty-changeset

(*DeployCloudFormationStack*/Configuration/fail-on-empty-changeset)

(선택 사항)

CloudFormation 변경 세트가 비어 있는 경우 CodeCatalyst가 CloudFormation 스택 배포 작업에 실패할지 여부를 지정합니다. (변경 세트가 비어 있으면 최신 배포 중에 스택에 변경 사항이 없음을 의미합니다.) 기본값은 변경 세트가 비어 있는 경우 작업을 진행하도록 허용하고 스택이 업데이트되지 않았더라도 UPDATE_COMPLETE 메시지를 반환하는 것입니다.

이 설정에 대한 자세한 내용은 AWS CLI 명령 참조의 CloudFormation [배포](#) 파라미터를 참조하세요. 변경 집합에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 [변경 집합을 사용하여 스택 업데이트](#)를 참조하세요.

해당 UI: 구성 탭/고급/빈 변경 사항 세트 실패

disable-rollback

(*DeployCloudFormationStack*/Configuration/disable-rollback)

(선택 사항)

CodeCatalyst가 스택 배포가 실패할 경우 롤백할지 여부를 지정합니다. 롤백은 스택을 마지막으로 알려진 안정 상태로 반환합니다. 기본값은 롤백을 활성화하는 것입니다. 이 설정에 대한 자세한 내용은 AWS CLI 명령 참조의 CloudFormation [배포](#) 파라미터를 참조하세요.

CloudFormation 스택 배포 작업이 롤백을 처리하는 방법에 대한 자세한 내용은 섹션을 참조하세요 [롤백 구성](#).

스택 롤백에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 [스택 실패 옵션](#)을 참조하세요.

해당 UI: 구성 탭/고급/롤백 비활성화

termination-protection

(*DeployCloudFormationStack*/Configuration/termination-protection)

(선택 사항)

배포 스택이 배포 중인 CloudFormation 스택에 종료 방지 기능을 추가할지 여부를 지정합니다. 종료 방지 기능이 활성화된 스택을 삭제하려고 시도하면 삭제가 실패하고 스택은 상태를 포함하여

변함없이 그대로 유지됩니다. 종료 방지 기능은 기본적으로 비활성화됩니다. 자세한 내용은 AWS CloudFormation 사용 설명서의 [스택이 삭제되지 않도록 보호](#)를 참조하세요.

해당 UI: 구성 탭/고급/종료 보호

timeout-in-minutes

(*DeployCloudFormationStack*/Configuration/timeout-in-minutes)

(선택 사항)

스택 생성 작업 시간이 초과되어 스택 상태를 CREATE_FAILED로 설정하기 전에 CloudFormation이 할당해야 하는 시간(분)을 지정합니다. CloudFormation이 할당된 시간 내에 전체 스택을 생성하지 못할 경우, 시간 초과로 인해 스택 생성이 실패하고 스택이 롤백됩니다.

기본적으로 스택 생성에는 시간 초과가 없습니다. 하지만 개인 리소스에는 해당 리소스가 구현하는 서비스의 특성에 따라 자체 시간 초과가 설정될 수 있습니다. 예를 들어 스택의 개별 리소스가 시간 초과될 경우, 스택 생성에 대해 지정한 시간 초과 시간이 아직 지나지 않았더라도 스택 생성이 시간 초과됩니다.

해당 UI: 구성 탭/고급/CloudFormation 제한 시간

notification-arns

(*DeployCloudFormationStack*/Configuration/notification-arns)

(선택 사항)

CodeCatalyst에서 알림 메시지를 보낼 Amazon SNS 주제의 ARN을 지정합니다. 예를 들어 `arn:aws:sns:us-east-1:111222333:MyTopic`입니다. CloudFormation 스택 배포 작업이 실행되면 CodeCatalyst는 CloudFormation과 조정하여 스택 생성 또는 업데이트 프로세스 중에 발생하는 CloudFormation 이벤트당 하나의 알림을 보냅니다. (이 이벤트는 스택에 대한 CloudFormation 콘솔의 이벤트 탭에 표시됩니다.) 최대 다섯 개의 주제를 지정할 수 있습니다. 자세한 내용은 [Amazon SNS란 무엇인가?](#)를 참조하세요.

해당 UI: 구성 탭/고급/알림 ARN

monitor-alarm-arns

(*DeployCloudFormationStack*/Configuration/monitor-alarm-arns)

(선택 사항)

롤백 트리거를 추가하려면 Amazon CloudWatch 경보의 Amazon 리소스 이름(ARN)을 지정합니다. 예를 들어 `arn:aws:cloudwatch::123456789012:alarm/MyAlarm`입니다. 최대 5개의 롤백 트리거를 가질 수 있습니다.

Note

CloudWatch 경보 ARN을 지정하는 경우 작업이 CloudWatch에 액세스할 수 있도록 추가 권한도 구성해야 합니다. 자세한 내용은 [롤백 구성](#) 섹션을 참조하세요.

해당 UI: 구성 탭/고급/모니터 경보 ARN

`monitor-timeout-in-minutes`

(DeployCloudFormationStack/Configuration/monitor-timeout-in-minutes)

(선택 사항)

CloudFormation이 지정된 경보를 모니터링하는 데 걸리는 시간을 0~180분으로 지정합니다. 모니터링은 모든 스택 리소스가 배포된 후 시작됩니다. 지정된 모니터링 시간 내에 경보가 발생하면 배포가 실패하고 CloudFormation이 전체 스택 작업을 롤백합니다.

기본값: 0. CloudFormation은 스택 리소스가 배포되는 동안에만 경보를 모니터링하며 이후는 모니터링하지 않습니다.

해당 UI: 구성 탭/고급/모니터링 시간

`tags`

(DeployCloudFormationStack/Configuration/tags)

(선택 사항)

CloudFormation 스택에 연결할 태그를 지정합니다. 태그는 비용 할당과 같은 용도를 위해 스택을 식별하는 데 사용할 수 있는 임의의 키-값 페어입니다. 어떤 태그를 어떤 방식으로 사용할지에 대한 자세한 내용을 알아보려면 Amazon EC2 사용 설명서의 [리소스에 태그 지정](#)을 참조하세요. CloudFormation의 태그 지정에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 [CloudFormation 스택 옵션 설정](#)을 참조하세요.

키는 영숫자 문자 또는 공백을 포함할 수 있으며 최대 127자를 포함할 수 있습니다. 값은 영숫자 문자 또는 공백을 포함할 수 있으며 최대 255자를 포함할 수 있습니다.

각 스택에 대해 최대 50개의 고유 태그를 추가할 수 있습니다.

해당 UI: 구성 탭/고급/태그

워크플로를 사용하여 AWS CDK 앱 배포

이 섹션에서는 워크플로를 사용하여 AWS 계정에 AWS Cloud Development Kit (AWS CDK) 앱을 배포하는 방법을 설명합니다. 이렇게 하려면 워크플로에 AWS CDK 배포 작업을 추가해야 합니다. AWS CDK 배포 작업은 앱을 합성하고 배포합니다 AWS Cloud Development Kit (AWS CDK) AWS. 앱 AWS이 이미에 있는 경우 필요한 경우 작업이 앱을 업데이트합니다.

를 사용하여 앱을 작성하는 방법에 대한 일반적인 내용은 AWS Cloud Development Kit (AWS CDK) 개발자 안내서의 [란 무엇입니까 AWS CDK?](#)를 AWS CDK참조하세요.

주제

- ['AWS CDK 배포' 작업을 사용해야 하는 경우](#)
- ['AWS CDK 배포' 작업 작동 방식](#)
- ['AWS CDK 배포' 작업에 사용되는 CDK CLI 버전](#)
- ['AWS CDK 배포' 작업에서 사용하는 런타임 이미지](#)
- [작업이 배포할 수 있는 스택 수는 몇 개입니까?](#)
- [예: AWS CDK 앱 배포](#)
- ['AWS CDK 배포' 작업 추가](#)
- ['AWS CDK 배포' 변수](#)
- ['AWS CDK 배포' 작업 YAML](#)

'AWS CDK 배포' 작업을 사용해야 하는 경우

를 사용하여 앱을 개발했고 AWS CDK이제 자동화된 지속적 통합 및 전송(CI/CD) 워크플로의 일부로 앱을 자동으로 배포하려는 경우이 작업을 사용합니다. 예를 들어 누군가 AWS CDK 앱 소스와 관련된 풀 요청을 병합할 때마다 AWS CDK 앱을 자동으로 배포할 수 있습니다.

'AWS CDK 배포' 작업 작동 방식

AWS CDK 배포는 다음과 같이 작동합니다.

1. 런타임 시 작업 버전 1.0.12 이하를 지정한 경우 작업은 최신 CDK CLI(AWS CDK Toolkit이라고도 함)를 CodeCatalyst [런타임 환경 이미지](#)에 다운로드합니다.

버전 1.0.13 이상을 지정한 경우 작업은 [특정 버전](#)의 CDK CLI와 번들로 제공되므로 다운로드가 발생하지 않습니다.

- 작업은 CDK CLI를 사용하여 `cdk deploy` 명령을 실행합니다. 이 명령은 AWS CDK 앱을 합성하고 배포합니다 AWS. 자세한 내용은 AWS Cloud Development Kit (AWS CDK) 개발자 가이드의 [AWS CDK 툴킷\(cdk 명령\)](#)을 참조하세요.

'AWS CDK 배포' 작업에 사용되는 CDK CLI 버전

다음 표에는 기본적으로 AWS CDK 배포 작업의 여러 버전에서 사용되는 CDK CLI 버전이 나와 있습니다.

Note

기본값을 재정의할 수 있습니다. 자세한 내용은 ['AWS CDK 배포' 작업 YAML](#)의 [CdkCliVersion](#) 섹션을 참조하세요.

'AWS CDK 배포' 작업 버전	AWS CDK CLI 버전
1.0.0~1.0.12	최신
1.0.13 이상	2.99.1

'AWS CDK 배포' 작업에서 사용하는 런타임 이미지

다음 표에는 CodeCatalyst가 AWS CDK 배포 작업의 다른 버전을 실행하는 데 사용하는 런타임 환경 이미지가 나와 있습니다. 이미지에는 사전 설치된 다양한 도구 세트가 포함됩니다. 자세한 내용은 [활성 이미지](#) 섹션을 참조하세요.

Note

2024년 3월 이미지에서 사용할 수 있는 최신 도구를 활용하려면 AWS CDK 배포 작업을 버전 2.x로 업그레이드하는 것이 좋습니다. 작업을 업그레이드하려면 워크플로 정의 파일에서 Identifier 속성을 `aws/cdk-deploy@v2`로 설정합니다. 자세한 내용은 ['AWS CDK 배포' 작업 YAML](#) 단원을 참조하십시오.

'AWS CDK 배포' 작업 버전	런타임 환경 이미지
1.x	2022년 11월 이미지
2.x	2024년 3월 이미지

작업이 배포할 수 있는 스택 수는 몇 개입니까?

AWS CDK 배포는 단일 스택만 배포할 수 있습니다. AWS CDK 앱이 여러 스택으로 구성된 경우 중첩 스택이 있는 상위 스택을 생성하고이 작업을 사용하여 상위 스택을 배포해야 합니다.

예: AWS CDK 앱 배포

다음 워크플로 예시에는 AWS CDK 부트스트랩 작업과 함께 AWS CDK 배포 작업이 포함됩니다. 워크플로는 순차적으로 실행되는 다음 구성 요소로 구성됩니다.

- 트리거 - 이 트리거는 소스 리포지토리에 변경 사항을 푸시할 때 워크플로 실행을 자동으로 시작합니다. 이 리포지토리에는 AWS CDK 앱이 포함되어 있습니다. 트리거에 대한 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 주제를 참조하세요.
- AWS CDK 부트스트랩 작업(CDKBootstrap) - 트리거 시 작업은 CDKToolkit 부트스트랩 스택을 배포합니다 AWS. CDKToolkit 스택이 이미 환경에 있는 경우 필요한 경우 업그레이드됩니다. 그렇지 않으면 아무 일도 발생하지 않고 작업이 성공으로 표시됩니다.
- AWS CDK 배포 작업(AWS CDK Deploy) - AWS CDK 부트스트랩 작업이 완료되면 AWS CDK 배포 작업은 AWS CDK 앱 코드를 CloudFormation 템플릿으로 합성하고 템플릿에 정의된 스택에 배포합니다 AWS.

Note

다음 워크플로 예시는 설명을 돕기 위한 참고용이며 추가 구성 없이는 작동하지 않습니다.

Note

다음 YAML 코드에서 원하는 경우 `Connections:` 섹션을 생략할 수 있습니다. 이러한 섹션을 생략하는 경우 환경의 기본 IAM 역할 필드에 지정된 역할에 AWS CDK 부트스트랩 및 AWS CDK 배포 작업에 필요한 권한 및 신뢰 정책이 포함되어 있는지 확인해야 합니다. 기본 IAM 역할

할이 있는 환경 설정에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요. AWS CDK 부트스트랩 및 AWS CDK 배포 작업에 필요한 권한 및 신뢰 정책에 대한 자세한 내용은 '[AWS CDK 부트스트랩](#)' 작업 YAML 및 '[AWS CDK 배포](#)' 작업 YAML의 Role 속성 설명을 참조하세요.

```
Name: codecatalyst-cdk-deploy-workflow
SchemaVersion: 1.0

Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  CDKBootstrap:
    Identifier: aws/cdk-bootstrap@v2
    Inputs:
      Sources:
        - WorkflowSource
    Environment:
      Name: codecatalyst-cdk-deploy-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-cdk-bootstrap-role
    Configuration:
      Region: us-west-2

  CDKDeploy:
    Identifier: aws/cdk-deploy@v2
    DependsOn:
      - CDKBootstrap
    Environment:
      Name: codecatalyst-cdk-deploy-environment
      Connections:
        - Name: codecatalyst-account-connection
          Role: codecatalyst-cdk-deploy-role
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      StackName: my-app-stack
      Region: us-west-2
```

'AWS CDK 배포' 작업 추가

다음 지침에 따라 AWS CDK 배포 작업을 워크플로에 추가합니다.

시작하기 전에

워크플로에 AWS CDK 배포 작업을 추가하기 전에 다음 작업을 완료합니다.

1. AWS CDK 앱을 준비합니다. 에서 지원하는 모든 프로그래밍 언어로 AWS CDK v1 또는 v2를 사용하여 AWS CDK 앱을 작성할 수 있습니다 AWS CDK. AWS CDK 앱 파일을 다음에서 사용할 수 있는지 확인합니다.
 - CodeCatalyst [소스 리포지토리](#) 또는
 - 다른 워크플로 작업에서 생성된 CodeCatalyst [출력 아티팩트](#)
2. AWS 환경을 부트스트랩합니다. 부트스트랩을 수행하려면 다음을 수행할 수 있습니다.
 - AWS Cloud Development Kit (AWS CDK) 개발자 안내서의 [부트스트랩 방법](#)에 설명된 방법 중 하나를 사용합니다.
 - AWS CDK 부트스트랩 작업을 사용합니다. AWS CDK 배포와 동일한 워크플로 또는 다른 워크플로에서 이 작업을 추가할 수 있습니다. AWS CDK 배포 작업을 실행하기 전에 부트스트랩 작업이 한 번 이상 실행되어 필요한 리소스가 준비되었는지 확인하세요. AWS CDK 부트스트랩 작업에 대한 자세한 내용은 섹션을 참조하세요 [워크플로를 사용하여 AWS CDK 앱 부트스트래핑](#).

부트스트래핑에 대한 자세한 내용은 AWS Cloud Development Kit (AWS CDK) 개발자 안내서의 [부트스트래핑](#)을 참조하세요.

Visual

시각적 편집기를 사용하여 'AWS CDK 배포' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.

8. 드롭다운 목록에서 Amazon CodeCatalyst를 선택합니다.
9. AWS CDK 배포 작업을 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.

또는

 - AWS CDK 배포를 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 다운로드를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
10. 입력 및 구성 탭에서 필요에 따라 필드를 작성합니다. 각 필드의 설명은 '[AWS CDK 배포](#)' 작업 [YAML](#) 섹션을 참조하세요. 이 참조는 YAML 및 시각적 편집기 모두에 나타나는 각 필드(및 해당 YAML 속성 값)에 대한 자세한 정보를 제공합니다.
11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

Note

AWS CDK 배포 작업이 실패하고 `npm install` 오류가 발생하는 경우 오류를 수정하는 방법에 대한 자세한 내용은 '[npm install](#)' 오류를 해결하려면 어떻게 해야 하나요? 섹션을 참조하세요.

YAML

YAML 편집기를 사용하여 'AWS CDK 배포' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.

8. 드롭다운 목록에서 Amazon CodeCatalyst를 선택합니다.
9. AWS CDK 배포 작업을 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.

또는

 - AWS CDK 배포를 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 다운로드를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
10. 필요에 따라 YAML 코드의 속성을 수정합니다. 사용 가능한 각 속성에 대한 설명은 ['AWS CDK 배포' 작업 YAML](#)에서 볼 수 있습니다.
11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

Note

AWS CDK 배포 작업이 실패하고 `npm install` 오류가 발생하는 경우 오류를 수정하는 방법에 대한 자세한 내용은 ['npm install' 오류를 해결하려면 어떻게 해야 하나요?](#) 섹션을 참조하세요.

'AWS CDK 배포' 변수

AWS CDK 배포 작업은 런타임에 다음 변수를 생성하고 설정합니다. 이를 사전 정의된 변수라고 합니다.

워크플로에서 이러한 변수를 참조하는 방법에 대한 자세한 내용은 [사전 정의된 변수 사용](#) 섹션을 참조하세요.

Key(키)	값
stack-id	워크플로 실행 중에 배포된 AWS CDK 애플리케이션 스택의 Amazon 리소스 이름(ARN)입니다.

Key(키)	값
	예시: <code>arn:aws:cloudformation:us-west-2:111122223333:stack/codecatalyst-cdk-app-stack/6aad4380-100a-11ec-a10a-03b8a84d40df</code>
<code>deployment-platform</code>	배포 플랫폼의 이름입니다. <code>AWS::CloudFormation</code> 로 하드코딩되었습니다.
리전	워크플로 실행 중에 배포 AWS 리전 된의 리전 코드입니다. 예시: <code>us-west-2</code>
<code>SKIP-DEPLOYMENT</code>	값이 이면 워크플로 실행 중에 AWS CDK 애플리케이션 스택의 배포를 건너뛰었음을 <code>true</code> 나 타냅니다. 마지막 배포 이후 스택에 변경 사항이 없는 경우 스택 배포를 건너뛵니다. 이 변수는 값이 <code>true</code>인 경우에만 생성됩니다. <code>true</code>로 하드코딩되었습니다.

Key(키)	값
CloudFormation variables	AWS CDK 배포 작업은 이전에 나열된 변수를 생성하는 것 외에도 CloudFormation 출력 변수를 후속 워크플로 작업에 사용할 워크플로 변수로 노출합니다. 기본적으로 작업은 처음 찾은 4개(또는 그 이하)의 CloudFormation 변수만 노출합니다. 어떤 배포 작업이 노출되었는지 확인하려면 AWS CDK 배포 작업을 한 번 실행한 다음 실행 세부 정보 페이지의 변수 탭을 확인합니다. 변수 탭에 나열된 변수가 원하는 변수가 아닌 경우 CfnOutputVariables YAML 속성을 사용하여 다른 변수를 구성할 수 있습니다. 자세한 내용은 'AWS CDK 배포' 작업 YAML의 CfnOutputVariables 작업에 관한 설명을 참조하세요.

'AWS CDK 배포' 작업 YAML

다음은 AWS CDK 배포 작업의 YAML 정의입니다. 이러한 작업 사용 방법을 배우려면 [워크플로를 사용하여 AWS CDK 앱 배포](#) 섹션을 참조하세요.

이 작업 정의는 더 광범위한 워크플로 정의 파일 내의 섹션으로 존재합니다. 이 파일에 대한 자세한 내용은 [워크플로 YAML 정의](#)를 참조합니다.

Note

이어지는 대부분의 YAML 속성에는 시각적 편집기에 해당 UI 요소가 있습니다. UI 요소를 찾으려면 Ctrl+F를 사용합니다. 요소가 연결된 YAML 속성과 함께 나열됩니다.

```
# The workflow definition starts here.
```

```
# See ### ## for details.
```

```
Name: MyWorkflow
```

```
SchemaVersion: 1.0
```

```
Actions:
```

```
# The action definition starts here.
CDKDeploy_nn:
  Identifier: aws/cdk-deploy@v2
  DependsOn:
    - CDKBootstrap
  Compute:
    Type: EC2 | Lambda
    Fleet: fleet-name
  Timeout: timeout-minutes
  Inputs:
    # Specify a source or an artifact, but not both.
    Sources:
      - source-name-1
    Artifacts:
      - artifact-name
  Outputs:
    Artifacts:
      - Name: cdk_artifact
    Files:
      - "cdk.out/**/*"
  Environment:
    Name: environment-name
  Connections:
    - Name: account-connection-name
      Role: iam-role-name
  Configuration:
    StackName: my-cdk-stack
    Region: us-west-2
    Tags: '{"key1": "value1", "key2": "value2"}'
    Context: '{"key1": "value1", "key2": "value2"}'
    CdkCliVersion: version
    CdkRootPath: directory-containing-cdk.json-file
    CfnOutputVariables: '["CfnOutputKey1", "CfnOutputKey2", "CfnOutputKey3"]'
    CloudAssemblyRootPath: path-to-cdk.out
```

CDKDeploy

(필수)

작업 이름을 지정합니다. 워크플로 내의 모든 작업 이름은 고유해야 합니다. 작업 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 작업 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

기본값: CDKDeploy_nn.

해당 UI: 구성 탭/작업 이름

Identifier

(*CDKDeploy*/Identifier)

(필수)

작업을 식별합니다. 버전을 변경하려는 경우가 아니면 이 속성을 변경하지 마세요. 자세한 내용은 [사용할 작업 버전 지정](#) 섹션을 참조하세요.

Note

를 지정aws/cdk-deploy@v2하면 Node.js 18과 같은 최신 도구가 포함된 [2024년 3월 이미지](#)에서 작업이 실행됩니다. 를 지정aws/cdk-deploy@v1하면 Node.js 16과 같은 이전 도구가 포함된 [2022년 11월 이미지](#)에서 작업이 실행됩니다.

기본값: aws/cdk-deploy@v2.

해당 UI: 워크플로 다이어그램/CDKDeploy_nn/aws/cdk-deploy@v2 레이블

DependsOn

(*CDKDeploy*/DependsOn)

(선택 사항)

AWS CDK 배포 작업을 실행하려면 성공적으로 실행해야 하는 작업 또는 작업 그룹을 지정합니다. 다음과 같이 DependsOn 속성에서 AWS CDK 부트스트랩 작업을 지정하는 것이 좋습니다.

```
CDKDeploy:
  Identifier: aws/cdk-deploy@v2
  DependsOn:
    - CDKBootstrap
```

Note

[부트스트래핑](#)은 AWS CDK 앱을 배포하기 위한 필수 사전 조건입니다. 워크플로에 AWS CDK 부트스트랩 작업을 포함하지 않는 경우 AWS CDK 배포 작업을 실행하기 전에 AWS CDK 부트

스트랩 스택을 배포할 다른 방법을 찾아야 합니다. 자세한 설명은 [워크플로를 사용하여 AWS CDK 앱 배포](#)에서 '[AWS CDK 배포](#)' 작업 추가 섹션을 참조하세요.

'depends on' 함수에 대한 자세한 내용은 [작업 순서 지정](#) 섹션을 참조하세요.

해당 UI: 입력 탭/의존 - 선택 사항

Compute

(*CDKDeploy*/Compute)

(선택 사항)

워크플로 작업을 실행하는 데 사용되는 컴퓨팅 엔진입니다. 워크플로 수준 또는 작업 수준에서 컴퓨팅을 지정할 수 있지만 둘 다 지정할 수는 없습니다. 워크플로 수준에서 지정하면 컴퓨팅 구성이 워크플로에 정의된 모든 작업에 적용됩니다. 워크플로 수준에서는 동일한 인스턴스에서 여러 작업을 실행할 수도 있습니다. 자세한 내용은 [작업 간에 컴퓨팅 공유](#) 섹션을 참조하세요.

해당 UI: 없음

Type

(*CDKDeploy*/Compute/Type)

([Compute](#) 포함 시 필수)

컴퓨팅 엔진의 유형입니다. 다음 값 중 하나를 사용할 수 있습니다.

- EC2(시각 편집기) 또는 EC2(YAML 편집기)
작업 실행 중 유연성을 위해 최적화되었습니다.
- Lambda(시각 편집기) 또는 Lambda(YAML 편집기)
작업 시작 속도를 최적화했습니다.

컴퓨팅 유형에 대한 자세한 정보는 [컴퓨팅 유형](#)을 참고하세요.

해당 UI: 구성 탭/고급 - 선택 사항/컴퓨팅 유형

Fleet

(*CDKDeploy*/Compute/Fleet)

(선택 사항)

워크플로 또는 워크플로 작업을 실행할 시스템 또는 플릿을 지정합니다. 온디맨드 플릿의 경우 작업이 시작되면 워크플로가 필요한 리소스를 프로비저닝하고 작업이 완료되면 시스템이 파괴됩니다. 온디맨드 플릿의 예시: `Linux.x86-64.Large`, `Linux.x86-64.XLarge`. 온디맨드 플릿에 대한 자세한 내용은 [온디맨드 플릿 속성](#) 섹션을 참조하세요.

프로비저닝된 플릿을 사용하면 워크플로 작업을 실행하도록 전용 시스템 세트를 구성할 수 있습니다. 이러한 시스템은 유휴 상태로 유지되므로 작업을 즉시 처리할 수 있습니다. 프로비저닝된 플릿에 대한 자세한 내용은 [프로비저닝된 플릿 속성](#) 섹션을 참조하세요.

Fleet 생략 시 기본값은 `Linux.x86-64.Large`입니다.

해당 UI: 구성 탭/고급 - 선택적/컴퓨팅 플릿

Timeout

(*CDKDeploy*/Timeout)

(필수)

CodeCatalyst가 작업을 종료하기 전에 작업을 실행할 수 있는 시간을 분(YAML 편집기) 또는 시간 및 분(시각적 편집기) 단위로 지정합니다. 최소값은 5분이고 최대값은 [CodeCatalyst의 워크플로 할당량](#)에 설명되어 있습니다. 기본 제한 시간은 최대 제한 시간과 동일합니다.

해당 UI: 구성 탭/제한 시간 - 선택 사항

Inputs

(*CDKDeploy*/Inputs)

(선택 사항)

이 Inputs 섹션에서는 워크플로 실행 중에 CDKDeploy 필요한 데이터를 정의합니다.

Note

각 AWS CDK 배포 작업에 대해 하나의 입력(소스 또는 아티팩트)만 허용됩니다.

해당 UI: 입력 탭

Sources

(*CDKDeploy*/Inputs/Sources)

(배포하려는 AWS CDK 앱이 소스 리포지토리에 저장된 경우 필수)

AWS CDK 앱이 소스 리포지토리에 저장된 경우 해당 소스 리포지토리의 레이블을 지정합니다. AWS CDK 배포 작업은 배포 프로세스를 시작하기 전에 이 리포지토리의 앱을 합성합니다. 현재, WorkflowSource 레이블만 지원됩니다.

AWS CDK 앱이 소스 리포지토리에 포함되지 않은 경우 다른 작업에서 생성된 아티팩트에 있어야 합니다.

소스에 대한 자세한 내용은 [워크플로에 소스 리포지토리 연결](#) 섹션을 참조하세요.

해당 UI: 입력 탭/소스 - 선택 사항

Artifacts - input

(*CDKDeploy*/Inputs/Artifacts)

(배포하려는 AWS CDK 앱이 이전 작업의 [출력 아티팩트](#)에 저장된 경우 필수)

AWS CDK 앱이 이전 작업에서 생성된 아티팩트에 포함된 경우 여기에 해당 아티팩트를 지정합니다. AWS CDK 배포 작업은 배포 프로세스를 시작하기 전에 지정된 아티팩트의 앱을 CloudFormation 템플릿으로 합성합니다. AWS CDK 앱이 아티팩트에 포함되지 않은 경우 앱은 소스 리포지토리에 있어야 합니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

해당 UI: 입력 탭/아티팩트 - 선택 사항

Outputs

(*CDKDeploy*/Outputs)

(선택 사항)

워크플로 실행 중에 작업에 의해 출력되는 데이터를 정의합니다.

해당 UI: 출력 탭

Artifacts - output

(*CDKDeploy*/Outputs/Artifacts)

(선택 사항)

작업에서 생성된 아티팩트를 지정합니다. 이러한 아티팩트를 다른 작업의 입력으로 참조할 수 있습니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

해당 UI: 출력 탭/아티팩트

Name

(*CDKDeploy*/Outputs/Artifacts/Name)

([Artifacts - output](#) 포함 시 필수)

런타임 시 AWS CDK 배포 작업에 의해 합성되는 CloudFormation 템플릿을 포함할 아티팩트의 이름을 지정합니다. 기본값은 `cdk_artifact`입니다. 아티팩트를 지정하지 않으면 작업이 템플릿을 합성하지만 아티팩트에 저장하지는 않습니다. 테스트 또는 문제 해결을 위해 합성된 템플릿을 아티팩트에 저장하여 레코드를 보존하는 것이 좋습니다.

해당 UI: 출력 탭/아티팩트/아티팩트 추가/빌드 아티팩트 이름

Files

(*CDKDeploy*/Outputs/Artifacts/Files)

([Artifacts - output](#) 포함 시 필수)

아티팩트에 포함할 파일을 지정합니다. AWS CDK 앱의 합성된 CloudFormation 템플릿을 "`cdk.out/**/*`" 포함하도록 지정해야 합니다.

Note

`cdk.out` 는 합성된 파일이 저장되는 기본 디렉터리입니다. `cdk.json` 파일에 이외의 출력 디렉터리를 지정한 경우 대신 여기에 해당 디렉터리 `cdk.out`를 지정합니다 `cdk.out`.

해당 UI: 출력 탭/아티팩트/아티팩트 추가/빌드에서 생성된 파일

Environment

(*CDKDeploy*/Environment)

(필수)

작업에 사용할 CodeCatalyst 환경을 지정합니다. 작업은 선택한 환경에 지정된 AWS 계정 및 선택적 Amazon VPC에 연결됩니다. 작업은 환경에 지정된 기본 IAM 역할을 사용하여 연결하고 [Amazon VPC 연결](#)에 지정된 IAM 역할을 AWS 계정으로 사용하여 Amazon VPC에 연결합니다.

 Note

기본 IAM 역할에 작업에 필요한 권한이 없는 경우 다른 역할을 사용하도록 작업을 구성할 수 있습니다. 자세한 내용은 [작업의 IAM 역할 변경](#) 섹션을 참조하세요.

환경에 대한 자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 및 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 구성 탭/환경

Name

(*CDKDeploy*/Environment/Name)

([Environment](#) 포함 시 필수)

작업과 연결하려는 기존 환경의 이름을 지정합니다.

해당 UI: 구성 탭/환경

Connections

(*CDKDeploy*/Environment/Connections)

(최신 버전의 작업에서는 선택 사항, 이전 버전에서는 필수)

작업과 연결할 계정 연결을 지정합니다. Environment에서 계정 연결을 최대 1개까지 지정할 수 있습니다.

계정 연결을 지정하지 않는 경우:

- 작업은 CodeCatalyst 콘솔의 환경에 지정된 AWS 계정 연결 및 기본 IAM 역할을 사용합니다. 환경에 계정 연결 및 기본 IAM 역할을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.
- 기본 IAM 역할에는 작업에 필요한 정책 및 권한이 포함되어야 합니다. 이러한 정책 및 권한이 무엇인지 확인하려면 작업의 YAML 정의 설명서에서 역할 속성에 대한 설명을 참조하세요.

계정 연결에 대한 자세한 정보는 [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 섹션을 참조하세요. 환경에 계정 연결을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/AWS 계정 연결

Name

(*CDKDeploy*/Environment/Connections/Name)

([Connections](#) 포함 시 필수)

계정 연결의 이름을 지정합니다.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/AWS 계정 연결

Role

(*CDKDeploy*/Environment/Connections/Role)

([Connections](#) 포함 시 필수)

계정 연결의 이름을 지정합니다.

AWS CDK 배포 작업이 AWS CDK 애플리케이션 스택에 액세스 AWS 하고 배포하는 데 사용하는 IAM 역할의 이름을 지정합니다. [CodeCatalyst 스페이스에 역할을 추가](#)했고 역할에 다음 정책이 포함되어 있는지 확인합니다.

IAM 역할을 지정하지 않으면 작업은 CodeCatalyst 콘솔의 [환경](#)에 나열된 기본 IAM 역할을 사용합니다. 환경에서 기본 역할을 사용하는 경우 다음 정책이 있는지 확인합니다.

- 다음 권한 정책:

⚠ Warning

다음 정책에 표시된 대로 권한을 제한합니다. 더 광범위한 권한을 가진 역할을 사용하면 보안 위험이 발생할 수 있습니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "cloudformation:DescribeStackEvents",
        "cloudformation:DescribeChangeSet",
        "cloudformation:DescribeStacks",
        "cloudformation:ListStackResources"
      ],
      "Resource": "*"
    },
    {
      "Sid": "VisualEditor1",
      "Effect": "Allow",
      "Action": "sts:AssumeRole",
      "Resource": "arn:aws:iam::111122223333:role/cdk-*"
    }
  ]
}
```

- 다음 사용자 지정 신뢰 정책:

i Note

원하는 경우 이 작업에서 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 사용할 수 있습니다. 이에 대한 자세한 내용은 [계정 및 스페이스의](#)

[CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 생성](#) 섹션을 참조하세요.

CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할에 보안 위험을 초래할 수

있는 전체 액세스 권한이 있음을 이해합니다. 보안에 대한 우려가 적은 자습서 및 시나리오에서만 이 역할을 사용하는 것이 좋습니다.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/역할

Configuration

(*CDKDeploy*/Configuration)

(필수)

작업의 구성 속성을 정의할 수 있는 섹션입니다.

해당 UI: 구성 탭

StackName

(*CDKDeploy*/Configuration/StackName)

(필수)

AWS CDK 앱 bin 디렉터리의 진입점 파일에 표시되는 AWS CDK 앱 스택의 이름입니다. 다음 예시에서는 스택 이름이 **### #####**로 강조 표시된 TypeScript 진입점 파일의 내용을 보여줍니다. 진입점 파일이 다른 언어로 되어 있는 경우 비슷해 보일 것입니다.

```
import * as cdk from 'aws-cdk-lib';
import { CdkWorkshopTypescriptStack } from '../lib/cdk_workshop_typescript-stack';

const app = new cdk.App();
new CdkWorkshopTypescriptStack(app, 'CdkWorkshopTypescriptStack');
```

하나의 스택만 지정할 수 있습니다.

Tip

스택이 여러 개 있는 경우 중첩 스택이 있는 상위 스택을 생성할 수 있습니다. 그런 다음 이 작업에서 상위 스택을 지정하여 모든 스택을 배포할 수 있습니다.

해당 UI: 구성 탭/스택 이름

Region

(*CDKDeploy*/Configuration/Region)

(선택 사항)

AWS CDK 애플리케이션 스택을 배포할 AWS 리전 를 지정합니다. 리전 코드 목록은 [리전 엔드포인트](#)를 참조하세요.

리전을 지정하지 않으면 AWS CDK 배포 작업이 AWS CDK 코드에 지정된 리전에 배포됩니다. 자세한 내용을 알아보려면 AWS Cloud Development Kit (AWS CDK) 개발자 안내서의 [환경](#)을 참조하세요.

해당 UI: 구성 탭/리전

Tags

(*CDKDeploy*/Configuration/Tags)

(선택 사항)

AWS CDK 애플리케이션 스택의 AWS 리소스에 적용할 태그를 지정합니다. 태그는 스택 자체뿐만 아니라 스택의 개별 리소스에도 적용됩니다. 태그에 대한 자세한 내용을 알아보려면 AWS Cloud Development Kit (AWS CDK) 개발자 안내서의 [태그 지정](#)을 참조하세요.

해당 UI: 구성 탭/고급 - 선택적/태그

Context

(*CDKDeploy*/Configuration/Context)

(선택 사항)

AWS CDK 애플리케이션 스택과 연결할 컨텍스트를 키-값 페어 형태로 지정합니다. 컨텍스트에 대한 자세한 내용을 알아보려면 AWS Cloud Development Kit (AWS CDK) 개발자 안내서의 [런타임 컨텍스트](#)를 참조하세요.

해당 UI: 구성 탭/고급 - 선택 사항/컨텍스트

CdkCliVersion

(*CDKDeploy*/Configuration/CdkCliVersion)

(선택 사항)

이 속성은 버전 1.0.13 이상의 AWS CDK 배포 작업과 버전 1.0.8 이상의 AWS CDK 부트스트랩 작업에서 사용할 수 있습니다.

다음 중 하나를 지정하세요.

- 이 작업에서 사용할 AWS Cloud Development Kit (AWS CDK) 명령줄 인터페이스(CLI)(도구 키트라고 AWS CDK 도 함)의 전체 버전입니다. 예시: 2.102.1. 애플리케이션을 구축하고 배포할 때 일관성과 안정성을 보장하기 위해 전체 버전을 지정하는 것이 좋습니다.

또는

- latest. CDK CLI의 최신 기능과 수정 사항을 활용하도록 latest를 지정하는 것이 좋습니다.

이 작업은 지정된 CLI 버전(또는 최신 버전)을 CodeCatalyst [빌드 이미지](#) AWS CDK 에 다운로드한 다음이 버전을 사용하여 CDK 애플리케이션을 배포하거나 환경을 부트스트랩 AWS 하는 데 필요한 명령을 실행합니다.

사용할 수 있는 지원되는 CDK CLI 버전 목록은 [AWS CDK 버전](#) 섹션을 참조하세요.

이 속성을 생략하면 작업은 다음 주제 중 하나에 설명된 기본 AWS CDK CLI 버전을 사용합니다.

- ['AWS CDK 배포' 작업에 사용되는 CDK CLI 버전](#)
- ["AWS CDK 부트스트랩" 작업에서 사용하는 CDK CLI 버전](#)

해당 UI: 구성 탭/AWS CDK CLI 버전

CdkRootPath

(*CDKDeploy*/Configuration/CdkRootPath)

(선택 사항)

AWS CDK 프로젝트 cdk.json 파일이 포함된 디렉터리의 경로입니다. AWS CDK 배포 작업은 이 폴더에서 실행되며 작업에서 생성된 모든 출력이 이 디렉터리에 추가됩니다. 지정하지 않으면 AWS CDK 배포 작업은 cdk.json 파일이 AWS CDK 프로젝트의 루트에 있다고 가정합니다.

해당 UI: cdk.json이 있는 구성 탭/디렉터리

CfnOutputVariables

(*CDKDeploy*/Configuration/CfnOutputVariables)

(선택 사항)

AWS CDK 애플리케이션 코드에서 워크플로 출력 변수로 노출하려는 CfnOutput 구문을 지정합니다. 그런 다음 워크플로의 후속 작업에서 워크플로 출력 변수를 참조할 수 있습니다. CodeCatalyst의 변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

예를 들어 AWS CDK 애플리케이션 코드가 다음과 같은 경우:

```
import { Duration, Stack, StackProps, CfnOutput, RemovalPolicy } from 'aws-cdk-lib';
import * as dynamodb from 'aws-cdk-lib/aws-dynamodb';
import * as s3 from 'aws-cdk-lib/aws-s3';
import { Construct } from 'constructs';
import * as cdk from 'aws-cdk-lib';
export class HelloCdkStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
    super(scope, id, props);
    const bucket = new s3.Bucket(this, 'amzn-s3-demo-bucket', {
      removalPolicy: RemovalPolicy.DESTROY,
    });
    new CfnOutput(this, 'bucketName', {
      value: bucket.bucketName,
      description: 'The name of the s3 bucket',
      exportName: 'amzn-s3-demo-bucket',
    });
    const table = new dynamodb.Table(this, 'todos-table', {
      partitionKey: { name: 'todoId', type: dynamodb.AttributeType.NUMBER },
      billingMode: dynamodb.BillingMode.PAY_PER_REQUEST,
      removalPolicy: RemovalPolicy.DESTROY,
    });
    new CfnOutput(this, 'tableName', {
      value: table.tableName,
      description: 'The name of the dynamodb table',
      exportName: 'myDynamoDbTable',
    });
    ...
  }
}
```

...그리고 CfnOutputVariables 속성은 다음과 같습니다.

```
Configuration:
  ...
```

```
CfnOutputVariables: ['bucketName','tableName']
```

...그런 다음 작업은 다음과 같은 워크플로 출력 변수를 생성합니다.

Key(키)	값
bucketName	bucket.bucketName
tableName	table.tableName

그런 다음 후속 작업에서 bucketName 및 tableName 변수를 참조할 수 있습니다. 후속 작업에서 워크플로 출력 변수를 참조하는 방법을 알아보려면 [사전 정의된 변수 참조](#) 섹션을 참조하세요.

CfnOutputVariables 속성에서 CfnOutput 구문을 지정하지 않으면 작업에서 워크플로 출력 변수로 찾는 처음 4개(또는 그 이하)의 CloudFormation 출력 변수가 표시됩니다. 자세한 내용은 ['AWS CDK 배포' 변수](#) 섹션을 참조하세요.

Tip

작업이 생성하는 모든 CloudFormation 출력 변수 목록을 가져오려면 AWS CDK 배포 작업이 포함된 워크플로를 한 번 실행한 다음 작업의 로그 탭을 확인합니다. 로그에는 AWS CDK 앱과 연결된 모든 CloudFormation 출력 변수 목록이 포함됩니다. 모든 CloudFormation 변수가 무엇인지 알고 나면 CfnOutputVariables 속성을 사용하여 워크플로 출력 변수로 변환할 변수를 지정할 수 있습니다.

CloudFormation 출력 변수에 대한 자세한 내용은 AWS Cloud Development Kit (AWS CDK) API 참조의 [클래스 CfnOutput\(construct\)](#)에서 제공되는 CfnOutput 구문 설명서를 참조하세요.

해당 UI: 구성 탭/CloudFormation 출력 변수

CloudAssemblyRootPath

(*CDKDeploy*/Configuration/CloudAssemblyRootPath)

(선택 사항)

AWS CDK 앱 스택을 클라우드 어셈블리로 이미 합성한 경우(cdk synth 작업 사용) 클라우드 어셈블리 디렉터리의 루트 경로()를 지정합니다 cdk.out. 지정된 클라우드 어셈블리 디렉터리에 있는

CloudFormation 템플릿은 `cdk deploy --app` 명령을 AWS 계정 사용하여 AWS CDK 배포 작업에 의해 배포됩니다. `--app` 옵션이 있으면 `cdk synth` 작업이 발생하지 않습니다.

클라우드 어셈블리 디렉토리를 지정하지 않으면 AWS CDK 배포 작업이 `--app` 옵션 없이 `cdk deploy` 명령을 실행합니다. `--app` 옵션이 없으면 `cdk deploy` 작업은 (`cdk synth`)를 합성하고 AWS CDK 앱에 배포합니다 AWS 계정.

"AWS CDK 배포" 작업이 런타임에 합성을 수행할 수 있는 경우 합성된 기존 클라우드 어셈블리를 지정하는 이유는 무엇입니까?

기존 합성 클라우드 어셈블리를 지정하여 다음을 수행할 수 있습니다.

- "AWS CDK 배포" 작업이 실행될 때마다 정확히 동일한 리소스 세트가 배포되는지 확인합니다.

클라우드 어셈블리를 지정하지 않으면 AWS CDK 배포 작업이 실행 시점에 따라 다른 파일을 합성하고 배포할 수 있습니다. 예를 들어, AWS CDK 배포 작업은 테스트 단계 동안 하나의 종속성 세트와 프로덕션 단계 동안 다른 종속성 세트(단계 간에 종속성이 변경된 경우)를 사용하여 클라우드 어셈블리를 합성할 수 있습니다. 테스트 대상과 배포 대상 간의 정확한 패리티를 보장하려면 한 번 합성한 다음 클라우드 어셈블리 디렉터리 경로 필드(시각적 편집기) 또는 `CloudAssemblyRootPath` 속성(YAML 편집기)을 사용하여 이미 합성된 클라우드 어셈블리를 지정하는 것이 좋습니다.

- AWS CDK 앱에서 비표준 패키지 관리자 및 도구 사용

`synth` 작업 중에 AWS CDK 배포 작업은 `npm` 또는 `pip`와 같은 표준 도구를 사용하여 앱을 실행하려고 시도합니다. 이러한 도구를 사용하여 작업을 성공적으로 실행할 수 없는 경우 합성이 발생하지 않고 작업이 실패합니다. 이 문제를 해결하려면 앱 `cdk.json` 파일에서 AWS CDK 앱을 성공적으로 실행하는 데 필요한 정확한 명령을 지정한 다음 AWS CDK 배포 작업이 포함되지 않은 방법을 사용하여 앱을 합성할 수 있습니다. 클라우드 어셈블리가 생성된 후 AWS CDK 배포 작업의 클라우드 어셈블리 디렉터리 경로 필드(시각 편집기) 또는 `CloudAssemblyRootPath` 속성(YAML 편집기)에서 지정할 수 있습니다.

AWS CDK 앱을 설치하고 실행하기 위한 명령을 포함하도록 `cdk.json` 파일을 구성하는 방법에 대한 자세한 내용은 [앱 명령 지정](#)을 참조하세요.

`cdk deploy` 및 `cdk synth` 명령과 `--app` 옵션에 대한 자세한 내용은 AWS Cloud Development Kit (AWS CDK) 개발자 안내서의 [스택 배포](#), [스택 합성](#) 및 [합성 건너뛰기](#)를 참조하세요.

클라우드 어셈블리에 대한 자세한 내용은 AWS Cloud Development Kit (AWS CDK) API 참조의 [클라우드 어셈블리](#)를 참조하세요.

해당 UI: 구성 탭/클라우드 어셈블리 디렉터리 경로

워크플로를 사용하여 AWS CDK 앱 부트스트래핑

이 섹션에서는 CodeCatalyst 워크플로를 사용하여 AWS CDK 애플리케이션을 부트스트랩하는 방법을 설명합니다. 이렇게 하려면 워크플로에 AWS CDK 부트스트랩 작업을 추가해야 합니다. AWS CDK 부트스트랩 작업은 [최신 템플릿](#)을 사용하여 AWS 환경의 부트스트랩 스택을 프로비저닝합니다. 부트스트랩 스택이 이미 있는 경우 필요한 경우 작업이 해당 스택을 업데이트합니다. 예 부트스트랩 스택 AWS 이 있으면 AWS CDK 앱을 배포하기 위한 사전 조건입니다.

부트스트래핑에 대한 자세한 내용은 AWS Cloud Development Kit (AWS CDK) 개발자 안내서의 [부트스트래핑](#)을 참조하세요.

주제

- ['AWS CDK bootstrap' 작업을 사용해야 하는 경우](#)
- ['AWS CDK 부트스트랩' 작업 작동 방식](#)
- [“AWS CDK 부트스트랩” 작업에서 사용하는 CDK CLI 버전](#)
- ['AWS CDK 부트스트랩' 작업에서 사용하는 런타임 이미지](#)
- [예: AWS CDK 앱 부트스트래핑](#)
- ['AWS CDK bootstrap' 작업 추가](#)
- ['AWS CDK 부트스트랩' 변수](#)
- ['AWS CDK 부트스트랩' 작업 YAML](#)

'AWS CDK bootstrap' 작업을 사용해야 하는 경우

AWS CDK 앱을 배포하는 워크플로가 있고 부트스트랩 스택을 동시에 배포(및 필요한 경우 업데이트)하려는 경우 이 작업을 사용합니다. 이 경우 AWS CDK 앱을 배포하는 워크플로와 동일한 워크플로에 AWS CDK 부트스트랩 작업을 추가합니다.

다음 중 하나가 적용되는 경우 이 작업을 사용하지 마세요.

- 다른 메커니즘을 사용하여 부트스트랩 스택을 이미 배포했으며 그대로 유지하고자 합니다(업데이트 없음).
- AWS CDK 부트스트랩 작업에서 지원되지 않는 [사용자 지정 부트스트랩 템플릿](#)을 사용하려고 합니다.

'AWS CDK 부트스트랩' 작업 작동 방식

AWS CDK 부트스트랩은 다음과 같이 작동합니다.

1. 런타임 시 작업 버전 1.0.7 이하를 지정한 경우 작업은 최신 CDK CLI(AWS CDK Toolkit이라고도 함)를 CodeCatalyst [빌드 이미지](#)에 다운로드합니다.

버전 1.0.8 이상을 지정한 경우 작업은 [특정 버전](#)의 CDK CLI와 번들로 제공되므로 다운로드가 발생하지 않습니다.

2. 작업은 CDK CLI를 사용하여 `cdk bootstrap` 명령을 실행합니다. 이 명령은 AWS Cloud Development Kit (AWS CDK) 개발자 안내서의 부트스트래핑 주제에 설명된 [부트스트래핑](#) 작업을 수행합니다.

“AWS CDK 부트스트랩” 작업에서 사용하는 CDK CLI 버전

다음 표에는 AWS CDK 부트스트랩 작업의 다양한 버전에서 기본적으로 사용되는 CDK CLI 버전이 나와 있습니다.

Note

기본값을 재정의할 수 있습니다. 자세한 내용은 ['AWS CDK 부트스트랩' 작업 YAML](#)의 [CdkCliVersion](#) 섹션을 참조하세요.

'AWS CDK 부트스트랩' 작업 버전	AWS CDK CLI 버전
1.0.0 – 1.0.7	최신
1.0.8 이상	2.99.1

'AWS CDK 부트스트랩' 작업에서 사용하는 런타임 이미지

다음 표에는 CodeCatalyst가 AWS CDK 부트스트랩 작업의 다른 버전을 실행하는 데 사용하는 런타임 환경 이미지가 나와 있습니다. 이미지에는 사전 설치된 다양한 도구 세트가 포함됩니다. 자세한 내용은 [활성 이미지](#) 섹션을 참조하세요.

Note

2024년 3월 이미지에서 사용할 수 있는 최신 도구를 활용하려면 AWS CDK 부트스트랩 작업을 버전 2.x로 업그레이드하는 것이 좋습니다. 작업을 업그레이드하려면 워크플로 정의 파일에서 Identifier 속성을 aws/cdk-bootstrapv2로 설정합니다. 자세한 내용은 ['AWS CDK 배포' 작업 YAML](#) 단원을 참조하십시오.

'AWS CDK 부트스트랩' 작업 버전	런타임 환경 이미지
1.x	2022년 11월 이미지
2.x	2024년 3월 이미지

예: AWS CDK 앱 부트스트래핑

AWS CDK 부트스트랩 작업이 포함된 워크플로는 [워크플로를 사용하여 AWS CDK 앱 배포](#)의 [예: AWS CDK 앱 배포](#)를 참조하세요.

'AWS CDK bootstrap' 작업 추가

다음 지침에 따라 AWS CDK 부트스트랩 작업을 워크플로에 추가합니다.

시작하기 전에

AWS CDK 부트스트랩 작업을 사용하려면 먼저 AWS CDK 앱이 준비되었는지 확인하세요. 부트스트랩 작업은 부트스트랩 전에 AWS CDK 앱을 합성합니다. AWS CDK에서 지원하는 모든 프로그래밍 언어로 앱을 작성할 수 있습니다.

AWS CDK 앱 파일을 다음에서 사용할 수 있는지 확인합니다.

- CodeCatalyst [소스 리포지토리](#) 또는
- 다른 워크플로 작업에서 생성된 CodeCatalyst [출력 아티팩트](#)

Visual

시각적 편집기를 사용하여 'AWS CDK 부트스트랩' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.

2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.
8. 드롭다운 목록에서 Amazon CodeCatalyst를 선택합니다.
9. AWS CDK 부트스트랩 작업을 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.

또는

 - AWS CDK 부트스트랩을 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 소스 보기를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
10. 입력, 구성 및 출력 탭에서 필요에 따라 필드를 작성합니다. 각 필드의 설명은 '[AWS CDK 부트스트랩' 작업 YAML](#) 섹션을 참조하세요. 이 참조는 YAML 및 시각적 편집기 모두에 나타나는 각 필드(및 해당 YAML 속성 값)에 대한 자세한 정보를 제공합니다.
11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

Note

AWS CDK 부트스트랩 작업이 실패하고 `npm install` 오류가 발생하는 경우 오류를 수정하는 방법에 대한 자세한 내용은 '[npm install](#) 오류를 해결하려면 어떻게 해야 하나요?' 섹션을 참조하세요.

YAML

YAML 편집기를 사용하여 'AWS CDK bootstrap' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.

2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.
8. 드롭다운 목록에서 Amazon CodeCatalyst를 선택합니다.
9. AWS CDK 부트스트랩 작업을 검색하고 +를 선택하여 워크플로 다이어그램에 추가하고 구성 창을 엽니다.
10. 필요에 따라 YAML 코드의 속성을 수정합니다. 사용 가능한 각 속성에 대한 설명은 ['AWS CDK 부트스트랩' 작업 YAML](#)에서 볼 수 있습니다.
11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

 Note

AWS CDK 부트스트랩 작업이 실패하고 `npm install` 오류가 발생하는 경우 오류를 수정하는 방법에 대한 자세한 내용은 ['npm install' 오류를 해결하려면 어떻게 해야 하나요?](#) 섹션을 참조하세요.

'AWS CDK 부트스트랩' 변수

AWS CDK 부트스트랩 작업은 런타임에 다음 변수를 생성하고 설정합니다. 이를 사전 정의된 변수라고 합니다.

워크플로에서 이러한 변수를 참조하는 방법에 대한 자세한 내용은 [사전 정의된 변수 사용](#) 섹션을 참조하세요.

Key(키)	값
deployment-platform	배포 플랫폼의 이름입니다. AWS:CloudFormation 로 하드코딩되었습니다.

Key(키)	값
리전	워크플로 실행 중에 AWS CDK 부트스트랩 스택 AWS 리전 이 배포된 리전 코드입니다. 예시: us-west-2
stack-id	배포된 AWS CDK 부트스트랩 스택의 Amazon 리소스 이름(ARN)입니다. 예시: arn:aws:cloudformation:us-west-2:111122223333:stack/codecatalyst-cdk-bootstrap-stack/6aad4380-100a-11ec-a10a-03b8a84d40df
SKIP-DEPLOYMENT	값이 이면 워크플로 실행 중에 AWS CDK 부트스트랩 스택 배포를 건너뛰었음을 true 나타냅니다. 마지막 배포 이후 스택에 변경 사항이 없는 경우 스택 배포를 건너뛵니다. 이 변수는 값이 true인 경우에만 생성됩니다. true로 하드코딩되었습니다.

'AWS CDK 부트스트랩' 작업 YAML

다음은 AWS CDK 부트스트랩 작업의 YAML 정의입니다. 이러한 작업 사용 방법을 배우려면 [워크플로를 사용하여 AWS CDK 앱 부트스트래핑](#) 섹션을 참조하세요.

이 작업 정의는 더 광범위한 워크플로 정의 파일 내의 섹션으로 존재합니다. 이 파일에 대한 자세한 내용은 [워크플로 YAML 정의](#)를 참조합니다.

Note

이어지는 대부분의 YAML 속성에는 시각적 편집기에 해당 UI 요소가 있습니다. UI 요소를 찾으려면 Ctrl+F를 사용합니다. 요소가 연결된 YAML 속성과 함께 나열됩니다.

```

# The workflow definition starts here.
# See ### ## for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
CDKBootstrapAction\_nn:
  Identifier: aws/cdk-bootstrap@v2
  DependsOn:
    - action-name
  Compute:
    Type: EC2 | Lambda
    Fleet: fleet-name
    Timeout: timeout-minutes
  Inputs:
    # Specify a source or an artifact, but not both.
    Sources:
      - source-name-1
    Artifacts:
      - artifact-name
  Outputs:
    Artifacts:
      - Name: cdk_bootstrap_artifacts
    Files:
      - "cdk.out/**/*"
  Environment:
    Name: environment-name
  Connections:
    - Name: account-connection-name
      Role: iam-role-name
  Configuration:
    Region: us-west-2
    CdkCliVersion: version

```

CDKBootstrapAction

(필수)

작업 이름을 지정합니다. 워크플로 내의 모든 작업 이름은 고유해야 합니다. 작업 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 작업 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

기본값: `CDKBootstrapAction_nn`.

해당 UI: 구성 탭/작업 표시 이름

Identifier

(*CDKBootstrapAction*/Identifier)

(필수)

작업을 식별합니다. 버전을 변경하려는 경우가 아니면 이 속성을 변경하지 마세요. 자세한 내용은 [사용할 작업 버전 지정](#) 섹션을 참조하세요.

 Note

를 지정aws/cdk-bootstrap@v2하면 Node.js 18과 같은 최신 도구가 포함된 [2024년 3월 이미지](#)에서 작업이 실행됩니다. 를 지정aws/cdk-bootstrap@v1하면 Node.js 16과 같은 이전 도구가 포함된 [2022년 11월 이미지](#)에서 작업이 실행됩니다.

기본값: `aws/cdk-bootstrap@v2`.

해당 UI: 워크플로 다이어그램/CDKBootstrapAction_nn/aws/cdk-bootstrap@v2 레이블

DependsOn

(*CDKBootstrapAction*/DependsOn)

(선택 사항)

이 작업을 실행하기 위해 성공적으로 실행해야 하는 작업, 작업 그룹 또는 게이트를 지정합니다.

'depends on' 함수에 대한 자세한 내용은 [작업 순서 지정](#) 섹션을 참조하세요.

해당 UI: 입력 탭/의존 - 선택 사항

Compute

(*CDKBootstrapAction*/Compute)

(선택 사항)

워크플로 작업을 실행하는 데 사용되는 컴퓨팅 엔진입니다. 워크플로 수준 또는 작업 수준에서 컴퓨팅을 지정할 수 있지만 둘 다 지정할 수는 없습니다. 워크플로 수준에서 지정하면 컴퓨팅 구성이 워크플

로에 정의된 모든 작업에 적용됩니다. 워크플로 수준에서는 동일한 인스턴스에서 여러 작업을 실행할 수도 있습니다. 자세한 내용은 [작업 간에 컴퓨팅 공유](#) 섹션을 참조하세요.

해당 UI: 없음

Type

(*CDKBootstrapAction*/Compute/Type)

([Compute](#) 포함 시 필수)

컴퓨팅 엔진의 유형입니다. 다음 값 중 하나를 사용할 수 있습니다.

- EC2(시각 편집기) 또는 EC2(YAML 편집기)
작업 실행 중 유연성을 위해 최적화되었습니다.
- Lambda(시각 편집기) 또는 Lambda(YAML 편집기)
작업 시작 속도를 최적화했습니다.

컴퓨팅 유형에 대한 자세한 정보는 [컴퓨팅 유형](#)을 참고하세요.

해당 UI: 구성 탭/고급 - 선택 사항/컴퓨팅 유형

Fleet

(*CDKBootstrapAction*/Compute/Fleet)

(선택 사항)

워크플로 또는 워크플로 작업을 실행할 시스템 또는 플릿을 지정합니다. 온디맨드 플릿의 경우 작업이 시작되면 워크플로가 필요한 리소스를 프로비저닝하고 작업이 완료되면 시스템이 파괴됩니다. 온디맨드 플릿의 예시: Linux.x86-64.Large, Linux.x86-64.XLarge. 온디맨드 플릿에 대한 자세한 내용은 [온디맨드 플릿 속성](#) 섹션을 참조하세요.

프로비저닝된 플릿을 사용하면 워크플로 작업을 실행하도록 전용 시스템 세트를 구성할 수 있습니다. 이러한 시스템은 유휴 상태로 유지되므로 작업을 즉시 처리할 수 있습니다. 프로비저닝된 플릿에 대한 자세한 내용은 [프로비저닝된 플릿 속성](#) 섹션을 참조하세요.

Fleet 생략 시 기본값은 Linux.x86-64.Large입니다.

해당 UI: 구성 탭/고급 - 선택적/컴퓨팅 플릿

Timeout

(*CDKBootstrapAction*/Timeout)

(필수)

CodeCatalyst가 작업을 종료하기 전에 작업을 실행할 수 있는 시간을 분(YAML 편집기) 또는 시간 및 분(시각적 편집기) 단위로 지정합니다. 최소값은 5분이고 최대값은 [CodeCatalyst의 워크플로 할당량](#)에 설명되어 있습니다. 기본 제한 시간은 최대 제한 시간과 동일합니다.

해당 UI: 구성 탭/제한 시간 - 선택 사항

Inputs

(*CDKBootstrapAction*/Inputs)

(선택 사항)

이 Inputs 섹션에서는 워크플로 실행 중에 AWS CDK 부트스트랩 작업에 필요한 데이터를 정의합니다.

해당 UI: 입력 탭

Note

각 AWS CDK 부트스트랩 작업에 대해 하나의 입력(소스 또는 아티팩트)만 허용됩니다.

Sources

(*CDKBootstrapAction*/Inputs/Sources)

(AWS CDK 앱이 소스 리포지토리에 저장된 경우 필수)

AWS CDK 앱이 소스 리포지토리에 저장된 경우 해당 소스 리포지토리의 레이블을 지정합니다. AWS CDK 부트스트랩 작업은 부트스트랩 프로세스를 시작하기 전에 이 리포지토리의 앱을 합성합니다. 현재, WorkflowSource 리포지토리 레이블만 지원됩니다.

AWS CDK 앱이 소스 리포지토리에 포함되지 않은 경우 다른 작업에서 생성된 아티팩트에 있어야 합니다.

소스에 대한 자세한 내용은 [워크플로에 소스 리포지토리 연결](#) 섹션을 참조하세요.

해당 UI: 입력 탭/소스 - 선택 사항

Artifacts - input

(*CDKBootstrapAction*/Inputs/Artifacts)

(AWS CDK 앱이 이전 작업의 [출력 아티팩트](#)에 저장된 경우 필수)

AWS CDK 앱이 이전 작업에서 생성된 아티팩트에 포함된 경우 여기에 해당 아티팩트를 지정합니다. AWS CDK 부트스트랩 작업은 부트스트랩 프로세스를 시작하기 전에 지정된 아티팩트의 앱을 CloudFormation 템플릿으로 합성합니다. AWS CDK 앱이 아티팩트 내에 포함되어 있지 않은 경우 앱은 소스 리포지토리에 있어야 합니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

해당 UI: 입력 탭/아티팩트 - 선택 사항

Outputs

(*CDKBootstrapAction*/Outputs)

(선택 사항)

워크플로 실행 중에 작업에 의해 출력되는 데이터를 정의합니다.

해당 UI: 출력 탭

Artifacts - output

(*CDKBootstrapAction*/Outputs/Artifacts)

(선택 사항)

작업에서 생성된 아티팩트를 지정합니다. 이러한 아티팩트를 다른 작업의 입력으로 참조할 수 있습니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

해당 UI: 출력 탭/아티팩트

Name

(*CDKBootstrapAction*/Outputs/Artifacts/Name)

([Artifacts - output](#) 포함 시 필수)

런타임 시 AWS CDK 부트스트랩 작업으로 합성되는 CloudFormation 템플릿을 포함할 아티팩트의 이름을 지정합니다. 기본값은 `cdk_bootstrap_artifacts`입니다. 아티팩트를 지정하지 않으면 작업이 템플릿을 합성하지만 아티팩트에 저장하지는 않습니다. 테스트 또는 문제 해결을 위해 합성된 템플릿을 아티팩트에 저장하여 레코드를 보존하는 것이 좋습니다.

해당 UI: 출력 탭/아티팩트/아티팩트 추가/빌드 아티팩트 이름

Files

(*CDKBootstrapAction*/Outputs/Artifacts/Files)

([Artifacts - output](#) 포함 시 필수)

아티팩트에 포함할 파일을 지정합니다. AWS CDK 앱의 합성된 CloudFormation 템플릿을 `"cdk.out/**/*"` 포함하도록 지정해야 합니다.

Note

`cdk.out` 는 합성된 파일이 저장되는 기본 디렉터리입니다. `cdk.json` 파일에 이외의 출력 디렉터리를 지정한 경우 대신 여기에 해당 디렉터리 `cdk.out`를 지정합니다 `cdk.out`.

해당 UI: 출력 탭/아티팩트/아티팩트 추가/빌드에서 생성된 파일

Environment

(*CDKBootstrapAction*/Environment)

(필수)

작업에 사용할 CodeCatalyst 환경을 지정합니다. 작업은 선택한 환경에 지정된 AWS 계정 및 선택적 Amazon VPC에 연결됩니다. 작업은 환경에 지정된 기본 IAM 역할을 사용하여 연결하고 [Amazon VPC 연결](#)에 지정된 IAM 역할을 AWS 계정으로 사용하여 Amazon VPC에 연결합니다.

Note

기본 IAM 역할에 작업에 필요한 권한이 없는 경우 다른 역할을 사용하도록 작업을 구성할 수 있습니다. 자세한 내용은 [작업의 IAM 역할 변경](#) 섹션을 참조하세요.

환경에 대한 자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 및 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 구성 탭/환경

Name

(*CDKBootstrapAction*/Environment/Name)

([Environment](#) 포함 시 필수)

작업과 연결하려는 기존 환경의 이름을 지정합니다.

해당 UI: 구성 탭/환경

Connections

(*CDKBootstrapAction*/Environment/Connections)

(최신 버전의 작업에서는 선택 사항, 이전 버전에서는 필수)

작업과 연결할 계정 연결을 지정합니다. Environment에서 계정 연결을 최대 1개까지 지정할 수 있습니다.

계정 연결을 지정하지 않는 경우:

- 작업은 CodeCatalyst 콘솔의 환경에 지정된 AWS 계정 연결 및 기본 IAM 역할을 사용합니다. 환경에 계정 연결 및 기본 IAM 역할을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.
- 기본 IAM 역할에는 작업에 필요한 정책 및 권한이 포함되어야 합니다. 이러한 정책 및 권한이 무엇인지 확인하려면 작업의 YAML 정의 설명서에서 역할 속성에 대한 설명을 참조하세요.

계정 연결에 대한 자세한 정보는 [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 섹션을 참조하세요. 환경에 계정 연결을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/AWS 계정 연결

Name

(*CDKBootstrapAction*/Environment/Connections/Name)

([Connections](#) 포함 시 필수)

계정 연결의 이름을 지정합니다.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/AWS 계정 연결

Role

(*CDKBootstrapAction*/Environment/Connections/Role)

([Connections](#) 포함 시 필수)

AWS CDK 부트스트랩 작업이 부트스트랩 스택에 액세스 AWS 하고 추가하는 데 사용하는 IAM 역할의 이름을 지정합니다. [CodeCatalyst 스페이스에 역할을 추가](#)했고 역할에 다음 정책이 포함되어 있는지 확인합니다.

IAM 역할을 지정하지 않으면 작업은 CodeCatalyst 콘솔의 [환경](#)에 나열된 기본 IAM 역할을 사용합니다. 환경에서 기본 역할을 사용하는 경우 적절한 정책이 있는지 확인합니다.

원하는 경우 이 작업에서 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 사용할 수 있습니다. 이에 대한 자세한 내용은 [계정 및 스페이스의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 생성](#) 섹션을 참조하세요.

CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할에 보안 위험을 초래할 수 있는 전체 액세스 권한이 있음을 이해합니다. 보안에 대한 우려가 적은 자습서 및 시나리오에서만 이 역할을 사용하는 것이 좋습니다.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/역할

Configuration

(*CDKBootstrapAction*/Configuration)

(필수)

작업의 구성 속성을 정의할 수 있는 섹션입니다.

해당 UI: 구성 탭

Region

(*CDKBootstrapAction*/Configuration/Region)

(필수)

부트스트랩 스택 AWS 리전을 배포할 리전을 지정합니다. 이 리전은 AWS CDK 앱이 배포된 리전과 일치해야 합니다. 리전 코드 목록은 [리전 엔드포인트](#)를 참조하세요.

해당 UI: 구성 탭/리전

CdkCliVersion

(*CDKBootstrapAction*/Configuration/CdkCliVersion)

(선택 사항)

이 속성은 버전 1.0.13 이상의 AWS CDK 배포 작업과 버전 1.0.8 이상의 AWS CDK 부트스트랩 작업에서 사용할 수 있습니다.

다음 중 하나를 지정하세요.

- 이 작업에서 사용할 AWS Cloud Development Kit (AWS CDK) 명령줄 인터페이스(CLI)(도구 키트라고 AWS CDK 포함)의 전체 버전입니다. 예시: 2.102.1. 애플리케이션을 구축하고 배포할 때 일관성과 안정성을 보장하기 위해 전체 버전을 지정하는 것이 좋습니다.

또는

- latest. CDK CLI의 최신 기능과 수정 사항을 활용하도록 latest를 지정하는 것이 좋습니다.

작업은 지정된 CLI 버전(또는 최신 버전)을 CodeCatalyst [빌드 이미지](#) AWS CDK 에 다운로드한 다음 이 버전을 사용하여 CDK 애플리케이션을 배포하거나 환경을 부트스트랩 AWS 하는 데 필요한 명령을 실행합니다.

사용할 수 있는 지원되는 CDK CLI 버전 목록은 [AWS CDK 버전](#) 섹션을 참조하세요.

이 속성을 생략하면 작업은 다음 주제 중 하나에 설명된 기본 AWS CDK CLI 버전을 사용합니다.

- ['AWS CDK 배포' 작업에 사용되는 CDK CLI 버전](#)
- ["AWS CDK 부트스트랩" 작업에서 사용하는 CDK CLI 버전](#)

해당 UI: 구성 탭/AWS CDK CLI 버전

워크플로를 사용하여 Amazon S3에 파일 게시

이 섹션에서는 CodeCatalyst 워크플로를 사용하여 Amazon S3에 파일을 게시하는 방법을 설명합니다. 이렇게 하려면, 워크플로에 Amazon S3 게시 작업을 추가해야 합니다. Amazon S3 게시 작업은 소스 디렉터리의 파일을 Amazon S3 버킷으로 복사합니다. 소스 디렉터리는 다음 위치에 있을 수 있습니다.

- [소스 리포지토리](#) 또는
- 다른 워크플로 작업에서 생성된 [출력 아티팩트](#)

주제

- ['Amazon S3 게시' 작업을 사용하는 경우](#)
- ['Amazon S3 게시' 작업에 사용되는 런타임 이미지](#)
- [예시: Amazon S3에 파일 게시](#)
- ['Amazon S3 게시' 작업 추가](#)
- ['Amazon S3 게시' 작업 YAML](#)

'Amazon S3 게시' 작업을 사용하는 경우

다음과 같은 경우 이 작업을 사용합니다.

- Amazon S3에 저장하려는 파일을 생성하는 워크플로가 있습니다.

예를 들어, Amazon S3에서 호스팅하려는 정적 웹 사이트를 빌드하는 워크플로가 있을 수 있습니다. 이 경우 워크플로에는 사이트의 HTML 및 지원 파일을 빌드하기 위한 빌드 [작업](#)과 파일을 Amazon S3에 복사하기 위한 Amazon S3 게시 작업이 포함됩니다.

- Amazon S3에 저장하려는 파일이 포함된 소스 리포지토리가 있습니다.

예를 들어, Amazon S3에 매일 밤 보관하려는 애플리케이션 소스 파일이 있는 소스 리포지토리가 있을 수 있습니다.

'Amazon S3 게시' 작업에 사용되는 런타임 이미지

Amazon S3 게시 작업은 [2022년 11월 이미지](#)에서 실행됩니다. 자세한 내용은 [활성 이미지](#) 단원을 참조하십시오.

예시: Amazon S3에 파일 게시

다음 예시 워크플로에는 빌드 작업과 함께 Amazon S3 게시 작업이 포함됩니다. 워크플로는 정적 설명서 웹 사이트를 빌드한 다음 Amazon S3에 게시하여 호스팅합니다. 워크플로는 순차적으로 실행되는 다음 구성 요소로 구성됩니다.

- 트리거 - 이 트리거는 소스 리포지토리에 변경 사항을 푸시할 때 워크플로 실행을 자동으로 시작합니다. 트리거에 대한 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 주제를 참조하세요.
- 빌드 작업(BuildDocs) - 트리거 시 작업은 정적 설명서 웹 사이트(mkdocs build)를 빌드하고 연결된 HTML 파일과 지원 메타데이터를 MyDocsSite라는 아티팩트에 추가합니다. 빌드 작업에 대한 자세한 내용은 [워크플로로 빌드하기](#) 섹션을 참조하세요.
- Amazon S3 게시 작업(PublishToS3) - 빌드 작업이 완료되면, 이 작업은 호스팅을 위해 MyDocsSite 아티팩트의 사이트를 Amazon S3에 복사합니다.

Note

다음 워크플로 예시는 설명을 돕기 위한 참고용이며 추가 구성 없이는 작동하지 않습니다.

Note

다음 YAML 코드에서 원하는 경우 Connections: 섹션을 생략할 수 있습니다. 이 섹션을 생략하는 경우, 환경의 기본 IAM 역할 필드에서 지정된 역할에 Amazon S3 게시 작업에 필요한 권한 및 신뢰 정책이 포함되어 있는지 확인해야 합니다. 기본 IAM 역할이 있는 환경 설정에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요. Amazon S3 게시 작업에 필요한 권한 및 신뢰 정책에 대한 자세한 내용은 ['Amazon S3 게시' 작업 YAML](#)의 [Role](#) 속성 설명을 참조하세요.

Name: codecatalyst-s3-publish-workflow

SchemaVersion: 1.0

Triggers:

- Type: PUSH
- Branches:
 - main

Actions:

BuildDocs:
Identifier: aws/build@v1

```

Inputs:
  Sources:
    - WorkflowSource
Configuration:
  Steps:
    - Run: echo BuildDocs started on `date`
    - Run: pip install --upgrade pip
    - Run: pip install mkdocs
    - Run: mkdocs build
    - Run: echo BuildDocs completed on `date`
Outputs:
  Artifacts:
    - Name: MyDocsSite
    Files:
      - "site/**/*"

```

```

PublishToS3:
  Identifier: aws/s3-publish@v1
  Environment:
    Name: codecatalyst-s3-publish-environment
  Connections:
    - Name: codecatalyst-account-connection
      Role: codecatalyst-s3-publish-build-role
  Inputs:
    Sources:
      - WorkflowSource
    Artifacts:
      - MyDocsSite
  Configuration:
    DestinationBucketName: amzn-s3-demo-bucket
    SourcePath: /artifacts/PublishToS3/MyDocSite/site
    TargetPath: my/docs/site

```

'Amazon S3 게시' 작업 추가

다음 지침에 따라 Amazon S3 게시 작업을 워크플로에 추가합니다.

Visual

시각적 편집기를 사용하여 'Amazon S3 게시' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.

3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.
8. 드롭다운 목록에서 Amazon CodeCatalyst를 선택합니다.
9. Amazon S3 게시 작업을 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
- 또는
 - Amazon S3 게시를 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 소스 보기를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
10. 입력, 구성 및 출력 탭에서 필요에 따라 필드를 작성합니다. 각 필드의 설명은 '[Amazon S3 게시](#)' 작업 YAML 섹션을 참조하세요. 이 참조는 YAML 및 시각적 편집기 모두에 나타나는 각 필드(및 해당 YAML 속성 값)에 대한 자세한 정보를 제공합니다.
11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 'Amazon S3 게시' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.

8. 드롭다운 목록에서 Amazon CodeCatalyst를 선택합니다.
9. Amazon S3 게시 작업을 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.

또는

 - Amazon S3 게시를 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 소스 보기를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
10. 필요에 따라 YAML 코드의 속성을 수정합니다. 사용 가능한 각 속성에 대한 설명은 '[Amazon S3 게시](#)' 작업 [YAML](#)에서 볼 수 있습니다.
11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

'Amazon S3 게시' 작업 YAML

다음은 Amazon S3 게시 작업의 YAML 정의입니다. 이러한 작업 사용 방법을 배우려면 [워크플로를 사용하여 Amazon S3에 파일 게시](#) 섹션을 참조하세요.

이 작업 정의는 더 광범위한 워크플로 정의 파일 내의 섹션으로 존재합니다. 이 파일에 대한 자세한 내용은 [워크플로 YAML 정의](#)를 참조합니다.

Note

이어지는 대부분의 YAML 속성에는 시각적 편집기에 해당 UI 요소가 있습니다. UI 요소를 찾으려면 Ctrl+F를 사용합니다. 요소가 연결된 YAML 속성과 함께 나열됩니다.

```
# The workflow definition starts here.
# See #### ## for details.
```

```
Name: MyWorkflow
SchemaVersion: 1.0
Actions:
```

```
# The action definition starts here.
S3Publish\_nn:
```

```

Identifier: aws/s3-publish@v1
DependsOn:
  - build-action
Compute:
  Type: EC2 | Lambda
  Fleet: fleet-name
Timeout: timeout-minutes
Inputs:
  Sources:
    - source-name-1
  Artifacts:
    - artifact-name
  Variables:
    - Name: variable-name-1
      Value: variable-value-1
    - Name: variable-name-2
      Value: variable-value-2
Environment:
  Name: environment-name
Connections:
  - Name: account-connection-name
    Role: iam-role-name
Configuration:
  SourcePath: my/source
  DestinationBucketName: amzn-s3-demo-bucket
  TargetPath: my/target

```

S3Publish

(필수)

작업 이름을 지정합니다. 워크플로 내의 모든 작업 이름은 고유해야 합니다. 작업 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 작업 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

기본값: S3Publish_nn.

해당 UI: 구성 탭/작업 이름

Identifier

(*S3Publish*/Identifier)

(필수)

작업을 식별합니다. 버전을 변경하려는 경우가 아니면 이 속성을 변경하지 마세요. 자세한 내용은 [사용할 작업 버전 지정](#) 섹션을 참조하세요.

기본값: aws/s3-publish@v1.

해당 UI: 워크플로 다이어그램/S3Publish_nn/aws/s3-publish@v1 레이블

DependsOn

(*S3Publish*/DependsOn)

(선택 사항)

이 작업을 실행하기 위해 성공적으로 실행해야 하는 작업, 작업 그룹 또는 게이트를 지정합니다.

'depends on' 함수에 대한 자세한 내용은 [작업 순서 지정](#) 섹션을 참조하세요.

해당 UI: 입력 탭/의존 - 선택 사항

Compute

(*S3Publish*/Compute)

(선택 사항)

워크플로 작업을 실행하는 데 사용되는 컴퓨팅 엔진입니다. 워크플로 수준 또는 작업 수준에서 컴퓨팅을 지정할 수 있지만 둘 다 지정할 수는 없습니다. 워크플로 수준에서 지정하면 컴퓨팅 구성이 워크플로에 정의된 모든 작업에 적용됩니다. 워크플로 수준에서는 동일한 인스턴스에서 여러 작업을 실행할 수도 있습니다. 자세한 내용은 [작업 간에 컴퓨팅 공유](#) 섹션을 참조하세요.

해당 UI: 없음

Type

(*S3Publish*/Compute/Type)

([Compute](#) 포함 시 필수)

컴퓨팅 엔진의 유형입니다. 다음 값 중 하나를 사용할 수 있습니다.

- EC2(시각 편집기) 또는 EC2(YAML 편집기)

작업 실행 중 유연성을 위해 최적화되었습니다.

- Lambda(시각 편집기) 또는 Lambda(YAML 편집기)

작업 시작 속도를 최적화했습니다.

컴퓨팅 유형에 대한 자세한 정보는 [컴퓨팅 유형](#)을 참고하세요.

해당 UI: 구성 탭/컴퓨팅 유형

Fleet

(*S3Publish*/Compute/Fleet)

(선택 사항)

워크플로 또는 워크플로 작업을 실행할 시스템 또는 플릿을 지정합니다. 온디맨드 플릿의 경우 작업이 시작되면 워크플로가 필요한 리소스를 프로비저닝하고 작업이 완료되면 시스템이 파괴됩니다. 온디맨드 플릿의 예시: Linux.x86-64.Large, Linux.x86-64.XLarge. 온디맨드 플릿에 대한 자세한 내용은 [온디맨드 플릿 속성](#) 섹션을 참조하세요.

프로비저닝된 플릿을 사용하면 워크플로 작업을 실행하도록 전용 시스템 세트를 구성할 수 있습니다. 이러한 시스템은 유휴 상태로 유지되므로 작업을 즉시 처리할 수 있습니다. 프로비저닝된 플릿에 대한 자세한 내용은 [프로비저닝된 플릿 속성](#) 섹션을 참조하세요.

Fleet 생략 시 기본값은 Linux.x86-64.Large입니다.

해당 UI: 구성 탭/컴퓨팅 플릿

Timeout

(*S3Publish*/Timeout)

(필수)

CodeCatalyst가 작업을 종료하기 전에 작업을 실행할 수 있는 시간을 분(YAML 편집기) 또는 시간 및 분(시각적 편집기) 단위로 지정합니다. 최소값은 5분이고 최대값은 [CodeCatalyst의 워크플로 할당량](#)에 설명되어 있습니다. 기본 제한 시간은 최대 제한 시간과 동일합니다.

해당 UI: 구성 탭/제한 시간 - 선택 사항

Inputs

(*S3Publish*/Inputs)

(선택 사항)

이 Inputs 섹션에서는 워크플로 실행 중에 S3Publish에 필요한 데이터를 정의합니다.

Note

각 AWS CDK 배포 작업에 대해 최대 4개의 입력(소스 1개 및 아티팩트 3개)이 허용됩니다. 변수는 이 합계에 포함되지 않습니다.

서로 다른 입력(소스 및 아티팩트)에 있는 파일을 참조해야 하는 경우 소스 입력은 기본 입력이고 아티팩트는 보조 입력입니다. 보조 입력의 파일에 대한 참조는 특수 접두사를 사용하여 기본 입력에서 파일을 구분합니다. 자세한 내용은 [예시: 여러 아티팩트에서 파일 참조](#)를 참조하세요.

해당 UI: 입력 탭

Sources

(*S3Publish*/Inputs/Sources)

(Amazon S3에 게시하려는 파일이 소스 리포지토리에 저장된 경우 필수)

Amazon S3에 게시하려는 파일이 소스 리포지토리에 저장된 경우, 해당 소스 리포지토리의 레이블을 지정합니다. 현재, WorkflowSource 레이블만 지원됩니다.

Amazon S3에 게시하려는 파일이 소스 리포지토리에 포함되어 있지 않은 경우, 다른 작업에서 생성된 아티팩트에 있어야 합니다.

소스에 대한 자세한 내용은 [워크플로에 소스 리포지토리 연결](#) 섹션을 참조하세요.

해당 UI: 입력 탭/소스 - 선택 사항

Artifacts - input

(*S3Publish*/Inputs/Artifacts)

(Amazon S3에 게시하려는 파일이 이전 작업의 [출력 아티팩트](#)에 저장되는 경우 필수)

Amazon S3에 게시하려는 파일이 이전 작업에서 생성된 아티팩트에 포함된 경우, 여기에 해당 아티팩트를 지정합니다. 아티팩트 내에 파일이 포함되어 있지 않은 경우 파일은 소스 리포지토리에 있어야 합니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

해당 UI: 구성 탭/아티팩트 - 선택 사항

Variables - input

(*S3Publish*/Inputs/Variables)

(선택 사항)

작업에 사용할 수 있도록 하려는 입력 변수를 정의하는 이름/값 페어의 시퀀스를 지정합니다. 변수 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스은 허용되지 않습니다. 변수 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

예시를 비롯한 변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

해당 UI: 입력 탭/변수 - 선택 사항

Environment

(*S3Publish*/Environment)

(필수)

작업에 사용할 CodeCatalyst 환경을 지정합니다. 작업은 선택한 환경에 지정된 AWS 계정 및 선택적 Amazon VPC에 연결됩니다. 작업은 환경에 지정된 기본 IAM 역할을 사용하여 연결하고 [Amazon VPC 연결](#)에 지정된 IAM 역할을 AWS 계정으로 사용하여 Amazon VPC에 연결합니다.

Note

기본 IAM 역할에 작업에 필요한 권한이 없는 경우 다른 역할을 사용하도록 작업을 구성할 수 있습니다. 자세한 내용은 [작업의 IAM 역할 변경](#) 섹션을 참조하세요.

환경에 대한 자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 및 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 구성 탭/환경

Name

(*S3Publish*/Environment/Name)

([Environment](#) 포함 시 필수)

작업과 연결하려는 기존 환경의 이름을 지정합니다.

해당 UI: 구성 탭/환경

Connections

(*S3Publish*/Environment/Connections)

(최신 버전의 작업에서는 선택 사항, 이전 버전에서는 필수)

작업과 연결할 계정 연결을 지정합니다. Environment에서 계정 연결을 최대 1개까지 지정할 수 있습니다.

계정 연결을 지정하지 않는 경우:

- 작업은 CodeCatalyst 콘솔의 환경에 지정된 AWS 계정 연결 및 기본 IAM 역할을 사용합니다. 환경에 계정 연결 및 기본 IAM 역할을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.
- 기본 IAM 역할에는 작업에 필요한 정책 및 권한이 포함되어야 합니다. 이러한 정책 및 권한이 무엇인지 확인하려면 작업의 YAML 정의 설명서에서 역할 속성에 대한 설명을 참조하세요.

계정 연결에 대한 자세한 정보는 [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 섹션을 참조하세요. 환경에 계정 연결을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/*# ##*의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/AWS 계정 연결

Name

(*S3Publish*/Environment/Connections/Name)

([Connections](#) 포함 시 필수)

계정 연결의 이름을 지정합니다.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/*# ##*의 내용/점 3개 메뉴/역할 전환

- (이전 버전) 구성 탭/'환경/계정/역할'/AWS 계정 연결

Role

(*S3Publish*/Environment/Connections/Role)

([Connections](#) 포함 시 필수)

Amazon S3 게시 작업이 Amazon S3에 액세스 AWS 하고 파일을 복사하는 데 사용하는 IAM 역할의 이름을 지정합니다 Amazon S3. [CodeCatalyst 스페이스에 역할을 추가](#)했고 역할에 다음 정책이 포함되어 있는지 확인합니다.

IAM 역할을 지정하지 않으면 작업은 CodeCatalyst 콘솔의 [환경](#)에 나열된 기본 IAM 역할을 사용합니다. 환경에서 기본 역할을 사용하는 경우 다음 정책이 있는지 확인합니다.

- 다음 권한 정책:

Warning

다음 정책에 표시된 대로 권한을 제한합니다. 더 광범위한 권한을 가진 역할을 사용하면 보안 위험이 발생할 수 있습니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor0",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:ListBucket",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::bucket-name",
        "arn:aws:s3:::bucket-name/*"
      ]
    }
  ]
}
```

```
    ]
  }
```

- 다음 사용자 지정 신뢰 정책:

Note

원하는 경우 이 작업에서 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 사용할 수 있습니다. 이에 대한 자세한 내용은 [계정 및 스페이스의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 생성](#) 섹션을 참조하세요.

CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할에 보안 위험을 초래할 수 있는 전체 액세스 권한이 있음을 이해합니다. 보안에 대한 우려가 적은 자습서 및 시나리오에서만 이 역할을 사용하는 것이 좋습니다.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/역할

Configuration

(*S3Publish*/Configuration)

(필수)

작업의 구성 속성을 정의할 수 있는 섹션입니다.

해당 UI: 구성 탭

SourcePath

(*S3Publish*/Configuration/SourcePath)

(필수)

Amazon S3에 게시하려는 디렉터리 또는 파일의 이름과 경로를 지정합니다. 디렉터리 또는 파일은 이전 작업의 소스 리포지토리 또는 아티팩트에 상주할 수 있으며 소스 리포지토리 또는 아티팩트 루트를 기준으로 합니다.

예시:

./myFolder/를 지정하면 /myFolder의 내용이 Amazon S3에 복사되고, 기본 디렉터리 구조가 보존됩니다.

Amazon S3에만 myfile.txt ./myFolder/myfile.txt 복사본 지정 (디렉토리 구조가 제거됩니다.)

와일드카드는 사용할 수 없습니다.

 Note

디렉터리 또는 파일 경로에 접두사를 추가하여 찾을 아티팩트 또는 소스를 나타내야 할 수 있습니다. 자세한 내용은 [소스 리포지토리 파일 참조](#) 및 [아티팩트의 파일 참조](#) 섹션을 참조하세요.

해당 UI: 구성 탭/소스 경로

DestinationBucketName

(*S3Publish*/Configuration/DestinationBucketName)

(필수)

파일을 게시하려는 Amazon S3 버킷의 이름을 지정합니다.

해당 UI: 구성 탭/대상 버킷 - 선택 사항

TargetPath

(*S3Publish*/Configuration/TargetPath)

(선택 사항)

Amazon S3에서 파일을 게시할 디렉터리의 이름과 경로를 지정합니다. 디렉터리가 존재하지 않으면 새로 생성됩니다. 디렉터리 경로에는 버킷 이름이 포함되어서는 안 됩니다.

예시:

myS3Folder

./myS3Folder/myS3Subfolder

해당 UI: 구성 탭/대상 디렉터리 - 선택 사항

AWS 계정 및 VPCs에 배포

[CodeCatalyst 워크플로](#)를 사용하면 애플리케이션 및 기타 리소스를 AWS 클라우드의 대상 AWS 계정 및 Amazon VPCs에 배포할 수 있습니다. 이러한 배포를 활성화하려면 CodeCatalyst 환경을 설정해야 합니다.

[개발](#) 환경과 혼동해서는 안 되는 CodeCatalyst 환경은 CodeCatalyst [워크플로](#)가 연결되는 대상 AWS 계정 및 선택적 Amazon VPC를 정의합니다. 또한 환경은 워크플로가 대상 계정 내의 AWS 서비스 및 리소스에 액세스하는 데 필요한 [IAM 역할을](#) 정의합니다.

여러 환경을 설정하고 개발, 테스트, 스테이징 및 프로덕션과 같은 이름을 지정할 수 있습니다. 이러한 환경에 배포하면 배포에 대한 정보가 환경의 CodeCatalyst 배포 활동 및 배포 대상 탭에 표시됩니다.

환경을 시작하려면 어떻게 해야 하나요?

CodeCatalyst 환경을 추가하고 사용하는 상위 단계는 다음과 같습니다.

1. CodeCatalyst 스페이스에서 하나 이상의 AWS 계정을 연결합니다. 이 프로세스 중에 워크플로가 AWS 계정의 리소스에 액세스하는 데 필요한 IAM 역할을 추가합니다. 자세한 내용은 [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 단원을 참조하십시오.
2. CodeCatalyst 프로젝트에서 AWS 계정 1단계의 및 IAM 역할 중 하나를 포함하는 환경을 생성합니다. 자세한 내용은 [환경 생성](#) 단원을 참조하십시오.
3. CodeCatalyst 프로젝트의 워크플로에서 2단계에서 생성한 환경을 가리키는 [작업](#)을 추가합니다. 자세한 내용은 [워크플로에 작업 추가](#) 섹션을 참조하십시오.

이제 환경을 구성했습니다. 이제 작업은 환경에서 지정된 AWS 계정 에 리소스를 배포할 수 있습니다.

Note

Amazon VPC를 환경에 추가할 수도 있습니다. 자세한 내용은 CodeCatalyst 관리 안내서 및 [VPC를 환경과 연결의 스페이스에 대한 VPC 연결 추가](#)를 참조하십시오.

단일 워크플로 내에 여러 환경이 존재할 수 있습니까?

예. 워크플로에 여러 작업이 포함된 경우 각 작업에 환경을 할당할 수 있습니다. 예를 들어 두 개의 배포 작업이 포함된 워크플로가 있을 때, 하나는 my-staging-environment 환경이 할당되고 다른 하나는 my-production-environment 환경이 할당될 수 있습니다.

환경을 지원하는 워크플로 작업은 무엇입니까?

AWS 클라우드에 리소스를 배포하거나 다른 이유(예: 모니터링 및 보고)로 AWS 서비스와 통신하는 모든 워크플로 작업은 환경을 지원합니다.

CodeCatalyst에 배포 정보가 표시되는 것을 지원하는 작업은 무엇입니까?

환경을 지원하는 워크플로 작업 중 CodeCatalyst 콘솔의 배포 활동 및 배포 대상 페이지에 배포 정보가 표시되는 작업은 몇 개뿐입니다.

다음 워크플로 작업은 배포 정보가 표시되는 것을 지원합니다.

- CloudFormation 스택 배포 - 자세한 내용은 섹션을 참조하세요. [CloudFormation 스택 배포](#)
- Amazon ECS에 배포 - 자세한 내용은 [워크플로를 사용하여 Amazon ECS에 배포](#) 섹션을 참조하세요.
- Kubernetes 클러스터에 배포 - 자세한 내용은 [워크플로를 사용하여 Amazon EKS에 배포](#) 섹션을 참조하세요.
- AWS CDK 배포 - 자세한 내용은 섹션을 참조하세요. [워크플로를 사용하여 AWS CDK 앱 배포](#)

지원되는 리전:

환경 페이지에서는 모든 AWS 리전의 리소스를 표시할 수 있습니다.

환경은 필수입니까?

할당된 워크플로 작업이 AWS 클라우드에 리소스를 배포하거나 다른 이유(예: 모니터링 및 보고)로 AWS 서비스와 통신하는 경우 환경은 필수입니다.

예를 들어 애플리케이션을 빌드하지만 AWS 계정 또는 Amazon VPC와 통신할 필요가 없는 빌드 작업이 있는 경우 작업에 환경을 할당할 필요가 없습니다. 그러나 빌드 작업이 AWS 계정의 Amazon CloudWatch 서비스로 로그를 전송하는 경우 작업에 환경이 할당되어 있어야 합니다.

주제

- [환경 생성](#)
- [작업과 환경 연결](#)
- [VPC를 환경과 연결](#)
- [환경 AWS 계정 과 연결](#)
- [작업의 IAM 역할 변경](#)

환경 생성

다음 지침을 사용하여 나중에 워크플로 작업과 연결할 수 있는 환경을 생성합니다.

시작하기 전 준비 사항

다음 항목이 필요합니다.

- CodeCatalyst 스페이스. 자세한 내용은 [CodeCatalyst 설정 및 로그인](#) 섹션을 참조하세요.
- CodeCatalyst 프로젝트. 자세한 내용은 [블루프린트를 사용하여 프로젝트 생성](#) 단원을 참조하십시오.
- 워크플로 작업이 액세스해야 하는 IAM 역할을 포함하는 AWS 계정 연결입니다. AWS 계정 연결 생성에 대한 자세한 내용은 [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 섹션을 참조하세요. 환경당 최대 하나의 계정 연결을 사용할 수 있습니다.

Note

계정 연결 없이 환경을 생성할 수 있지만 나중에 다시 돌아와 연결을 추가해야 합니다.

- 다음 CodeCatalyst 역할 중 하나:
 - 스페이스 관리자
 - 프로젝트 관리자
 - 기고자

Note

기고자 역할이 있는 경우 환경을 생성할 수 있지만 AWS 계정 연결에는 연결할 수 없습니다. 스페이스 관리자 또는 프로젝트 관리자 역할을 가진 사람에게 환경을 AWS 계정 연결에 연결하도록 요청해야 합니다.

역할과 권한에 대한 자세한 내용은 [사용자에게 프로젝트 권한 부여](#) 섹션을 참조하세요.

환경을 생성하는 방법

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 환경을 선택합니다.

4. 환경 이름에 **Production** 또는 **Staging** 같은 이름을 입력합니다.

5. 환경 유형에서 다음 중 하나를 선택합니다.

- 비프로덕션 - 애플리케이션을 프로덕션으로 이동하기 전에 애플리케이션을 테스트하여 의도한 대로 작동하는지 확인할 수 있는 환경입니다.
- 프로덕션 - 공개적으로 사용할 수 있고 최종 애플리케이션을 호스팅하는 '라이브' 환경입니다.

프로덕션을 선택하면 환경이 연결된 작업 옆으로 프로덕션 배지가 UI에 나타납니다. 배지를 사용하면 프로덕션에 배포되는 작업을 빠르게 확인할 수 있습니다. 배지 모양 외에는 프로덕션 환경과 비프로덕션 환경 간에 차이가 없습니다.

6. (선택 사항) 설명: 설명을 입력합니다. 예시: **Production environment for the hello-world app.**

7. AWS 계정 연결 - 선택 사항에서 이 환경과 연결할 AWS 계정 연결을 선택합니다. 이 환경에 할당된 워크플로 작업은 연결된 AWS 계정에 연결할 수 있습니다. CodeCatalyst에서 AWS 계정 연결을 생성하는 방법에 대한 자세한 내용은 섹션을 참조하세요 [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#).

사용하려는 AWS 계정 연결이 나열되지 않은 경우 프로젝트에서 허용되지 않기 때문일 수 있습니다. 자세한 내용은 Amazon CodeCatalyst 관리자 안내서의 [프로젝트 제한 계정 연결 구성](#)을 참조하세요.

8. 기본 IAM 역할에서 이 환경과 연결할 IAM 역할을 선택합니다. 이 환경에 할당된 워크플로 작업은 이 IAM 역할을 상속하며 이를 사용하여의 서비스 및 리소스에 연결할 수 있습니다 AWS 계정.

여러 작업에 환경을 할당해야 하는데 이러한 작업에 여기에 지정된 기본 역할과 다른 IAM 역할이 필요한 경우, 역할 전환 옵션을 사용하여 각 작업의 구성 탭에서 서로 다른 역할을 지정할 수 있습니다. 자세한 내용은 [작업의 IAM 역할 변경](#) 단원을 참조하십시오.

기본값으로 사용하려는 IAM 역할이 나열되지 않은 경우 아직 AWS 계정 연결에 추가하지 않았기 때문일 수 있습니다. 계정 연결에 IAM 역할을 추가하려면 [계정 연결에 IAM 역할 추가](#) 섹션을 참조하세요.

9. (선택 사항) VPC 연결에서 이 환경과 연결할 VPC 연결을 선택합니다. VPC 연결 생성에 대한 자세한 내용은 [Amazon CodeCatalyst 관리자 안내서](#)의 Amazon Virtual Private Clouds 관리를 참조하세요.

사용하려는 VPC 연결이 나열되지 않은 경우 프로젝트에 허용되지 않는 AWS 계정 연결이 포함되어 있기 때문일 수 있습니다. 자세한 내용은 Amazon CodeCatalyst 관리자 안내서의 [프로젝트 제한 계정 연결 구성](#)을 참조하세요.

10. 환경 생성을 선택합니다. CodeCatalyst는 빈 환경을 생성합니다.

다음 단계

- 이제 환경을 생성했으므로 워크플로 작업과 연결할 준비가 되었습니다. 자세한 내용은 [작업과 환경 연결](#) 섹션을 참조하세요.

작업과 환경 연결

환경을 [지원되는 워크플로 작업](#)과 연결하면 환경의 , AWS 계정 기본 IAM 역할 및 선택적 Amazon VPC가 작업에 할당됩니다. 그런 다음 IAM 역할을 사용하여 작업을 AWS 계정에 연결 및 배포하고 선택적 Amazon VPC에도 연결할 수 있습니다.

다음 지침을 사용하여 환경을 작업과 연결합니다.

1단계: 환경을 워크플로 작업과 연결

다음 절차에 따라 환경을 워크플로 작업과 연결합니다.

Visual

시각적 편집기를 사용하여 환경을 워크플로 작업과 연결하려면

- <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
- 프로젝트를 선택합니다.
- 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
- 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
- 편집을 선택합니다.
- 비주얼을 선택합니다.
- 워크플로 다이어그램에서 환경에서 지원되는 작업을 선택합니다. 자세한 내용은 [CodeCatalyst에 배포 정보가 표시되는 것을 지원하는 작업은 무엇입니까?](#) 섹션을 참조하세요.
- 구성 탭을 선택하고 다음과 같이 환경 필드에 정보를 지정합니다.

환경

작업에 사용할 CodeCatalyst 환경을 지정합니다. 작업은 선택한 환경에 지정된 AWS 계정 및 선택적 Amazon VPC에 연결됩니다. 작업은 환경에 지정된 기본 IAM 역할을 사용하여 연결

하고 [Amazon VPC 연결](#)에 지정된 IAM 역할을 AWS 계정으로 사용하여 Amazon VPC에 연결합니다.

 Note

기본 IAM 역할에 작업에 필요한 권한이 없는 경우 다른 역할을 사용하도록 작업을 구성할 수 있습니다. 자세한 내용은 [작업의 IAM 역할 변경](#) 섹션을 참조하세요.

환경에 대한 자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 및 [환경 생성](#) 섹션을 참조하세요.

9. (선택 사항) 작업과 연결된 IAM 역할을 변경합니다. 작업에 대한 잘못된 권한 집합이 포함된 경우 역할을 변경할 수 있습니다.

역할을 생성하려면:

1. **# ##**에 무엇이 있나요? 상자에서 세로 줄임표 아이콘



을 선택합니다.

2. 다음 중 하나를 선택합니다.

- 역할을 전환합니다. 이 작업에서 사용하는 IAM 역할을 변경하고 이 작업만 변경하려면 이 옵션을 선택합니다. 다른 작업은 연결된 환경에 지정된 기본 IAM 역할을 계속 사용합니다. 자세한 내용은 [작업의 IAM 역할 변경](#) 섹션을 참조하세요.
- 환경을 편집합니다. 환경에 나열된 기본 IAM 역할을 변경하려면 이 옵션을 선택합니다. 이 옵션을 선택하면 작업과 동일한 환경과 연결된 다른 작업이 새 기본 IAM 역할을 사용하여 시작됩니다.

 Important

기본 IAM 역할을 업데이트할 때 주의하세요. 역할의 권한이 환경을 공유하는 모든 작업에 충분하지 않은 경우 역할을 변경하면 작업 실패가 발생할 수 있습니다.

10. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
11. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 환경을 워크플로 작업과 연결하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 환경과 연결하려는 워크플로 작업에서 다음과 유사한 코드를 추가합니다.

```
action-name:
  Environment:
    Name: environment-name
```

자세한 내용은 [작업 유형](#) 항목을 참조하세요. 이 주제에는 YAML 참조를 포함하여 각 작업에 대한 설명서로 연결되는 링크가 있습니다.

8. (선택 사항) 작업에 환경에 나열된 기본 IAM 역할과 다른 역할을 사용하려면 사용하려는 역할이 포함된 Connections: 섹션을 추가합니다. 자세한 내용은 [작업의 IAM 역할 변경](#) 섹션을 참조하세요.
9. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
10. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

2단계: 배포 활동 페이지 채우기

환경을 워크플로 작업과 연결한 후 CodeCatalyst 콘솔의 환경 섹션에 있는 배포 활동 및 배포 대상 페이지를 배포 정보로 채울 수 있습니다. 다음 지침을 사용하여 이러한 페이지를 채웁니다.

Note

CodeCatalyst 콘솔에 배포 정보가 표시되는 작업은 몇 가지뿐입니다. 자세한 내용은 [CodeCatalyst에 배포 정보가 표시되는 것을 지원하는 작업은 무엇입니까?](#) 섹션을 참조하세요.

CodeCatalyst에 배포 정보를 추가하려면

1. [1단계: 환경을 워크플로 작업과 연결](#)에서 변경 사항을 커밋할 때 워크플로 실행이 자동으로 시작되지 않은 경우 다음과 같이 수동으로 실행을 시작합니다.
 - a. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
 - b. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
 - c. 실행을 선택합니다.

워크플로 실행은 새 배포를 시작하며, 이로 인해 CodeCatalyst는 CodeCatalyst에 배포 정보를 추가합니다.

2. CodeCatalyst 콘솔에 배포 활동이 추가되었는지 확인합니다.
 - a. 탐색 창에서 CI/CD를 선택한 다음 환경을 선택합니다.
 - b. 환경을 선택합니다(예: Production).
 - c. 배포 활동 탭을 선택하고 배포가 SUCCEEDED 상태로 나타나는지 확인합니다. 이는 워크플로 실행이 애플리케이션 리소스를 성공적으로 배포했음을 나타냅니다.
 - d. 배포 대상 탭을 선택하고 애플리케이션 리소스가 나타나는지 확인합니다.

VPC를 환경과 연결

작업이 VPC 연결이 있는 환경으로 구성된 경우 연결된 VPC에서 지정한 네트워크 규칙 및 액세스 리소스에 따라 작업이 VPC에 연결된 상태로 실행됩니다. 하나 이상의 환경에서 동일한 VPC 연결을 사용할 수 있습니다.

다음 지침에 따라 VPC 연결을 환경과 연결합니다.

VPC 연결을 환경에 연결하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 환경을 선택합니다.
4. 환경을 선택합니다(예: Production).
5. 환경 속성 탭을 선택합니다.

6. VPC 연결 관리를 선택하고 원하는 VPC 연결을 선택한 다음 확인을 선택합니다. 이렇게 하면 선택한 VPC 연결이 이 환경과 연결됩니다.

Note

사용하려는 VPC 연결이 나열되지 않은 경우 프로젝트에 허용되지 않는 AWS 계정 연결이 포함되어 있기 때문일 수 있습니다. 자세한 내용은 Amazon CodeCatalyst 관리자 안내서의 [프로젝트 제한 계정 연결 구성](#)을 참조하세요.

자세한 내용은 CodeCatalyst Cloud 관리자 안내서의 [Amazon Virtual Private Clouds 관리하기](#)를 참조하세요.

환경 AWS 계정 과 연결

다음 지침에 따라 환경 AWS 계정 과 연결합니다. AWS 계정 를 환경에 연결하면 환경에 할당된 워크폴로 작업이 연결할 수 있습니다 AWS 계정.

계정 연결에 대한 자세한 정보는 [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 섹션을 참조하세요.

시작하기 전 준비 사항

다음 항목이 필요합니다.

- 워크폴로 작업이 액세스해야 하는 IAM 역할을 포함하는 AWS 계정 연결입니다 AWS. 계정 연결 생성에 대한 자세한 내용은 [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 섹션을 참조하세요. 환경당 최대 하나의 계정 연결을 사용할 수 있습니다.
- 스페이스 관리자 또는 프로젝트 관리자 CodeCatalyst 역할 중 하나를 선택합니다. 자세한 내용은 [사용자에게 프로젝트 권한 부여](#) 단원을 참조하십시오.

AWS 계정 를 환경에 연결하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 환경을 선택합니다.
4. 환경을 선택합니다(예: Production).
5. 환경 편집을 선택합니다.

6. 환경 속성의 AWS 계정 연결 - 선택 사항 드롭다운 목록에서 원하는 AWS 계정을 선택합니다.

사용하려는 AWS 계정 연결이 나열되지 않은 경우 프로젝트에서 허용되지 않기 때문일 수 있습니다. 자세한 내용은 Amazon CodeCatalyst 관리자 안내서의 [프로젝트 제한 계정 연결 구성](#)을 참조하세요.

7. 기본 IAM 역할에서 이 환경과 연결할 IAM 역할을 선택합니다. 이 환경에 할당된 워크플로 작업은 이 IAM 역할을 상속하며 이를 사용하여의 서비스 및 리소스에 연결할 수 있습니다 AWS 계정.

기본값으로 사용하려는 IAM 역할이 나열되지 않은 경우 아직 AWS 계정 연결에 추가하지 않았기 때문일 수 있습니다. 계정 연결에 IAM 역할을 추가하려면 [계정 연결에 IAM 역할 추가](#) 섹션을 참조하세요.

작업의 IAM 역할 변경

기본적으로 [환경](#)을 워크플로 [작업](#)과 연결하면 작업은 환경에 지정된 기본 IAM 역할을 상속합니다. 작업이 다른 역할을 사용하도록 이 동작을 변경할 수 있습니다. 작업이 AWS 클라우드에서 작동하는 데 필요한 권한이 기본 IAM 역할에 없는 경우 다른 역할을 사용할 수 있습니다.

작업에 다른 IAM 역할을 할당하려면 시각적 편집기에서 역할 전환 옵션을 사용하거나 YAML 편집기에서 Connections: 속성을 사용할 수 있습니다. 새 역할은 환경에 지정된 기본 IAM 역할을 재정의하므로 기본 IAM 역할을 그대로 유지할 수 있습니다. 기본 IAM 역할을 사용하는 다른 작업이 있는 경우 기본 IAM 역할을 그대로 유지할 수 있습니다.

다음 지침을 사용하여 환경에 지정된 것과 다른 IAM 역할을 사용하도록 작업을 구성합니다.

Visual

작업에 다른 IAM 역할을 할당하려면(시각적 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. IAM 역할을 업데이트하려는 작업을 나타내는 상자를 선택합니다.
7. 구성 탭을 선택합니다.

8. **# ##**에 무엇이 있나요? 상자에서 세로 줄임표 아이콘

(;

을 선택합니다.

9. 역할 전환을 선택합니다.
10. 역할 전환 대화 상자의 IAM 역할 드롭다운 목록에서 작업을 사용할 IAM 역할을 선택합니다. 이 역할은 환경의 기본 IAM 역할을 재정의합니다. 사용하려는 역할이 목록에 없는 경우 스페이스에 추가했는지 확인합니다. 자세한 내용은 [계정 연결에 IAM 역할 추가](#) 섹션을 참조하세요.

이제 선택한 역할이 워크플로에서 정의된 배치와 함께 **# ##**에 무엇이 있나요? 상자에 나타납니다. 역할은 Connections: 섹션의 워크플로 정의 파일에도 표시됩니다.

11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

작업에 다른 IAM 역할을 할당하려면(YAML 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 다른 IAM 역할을 사용하려는 워크플로 작업에서 다음과 유사한 Connections: 섹션을 추가합니다.

```
action-name:
  Environment:
    Name: environment-name
  Connections:
    - Name: account-connection-name
      Role: iam-role-name
```

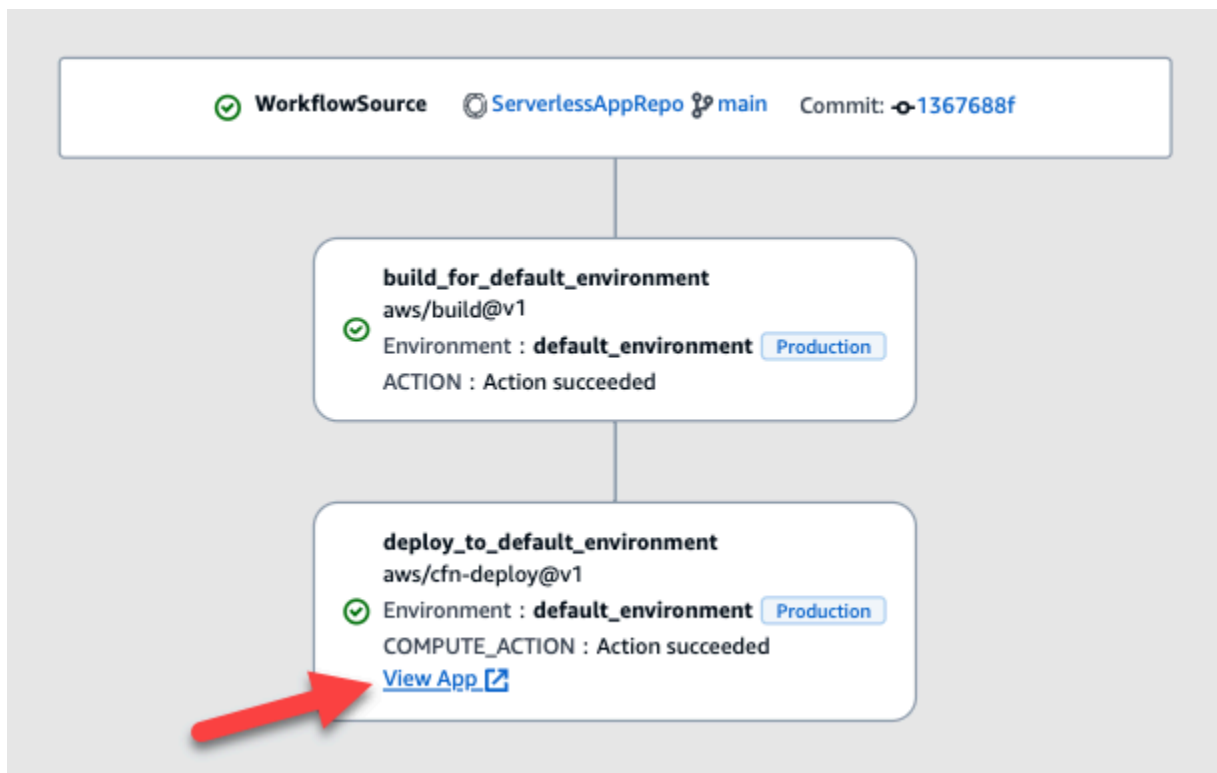
앞의 코드에서 *account-connection-name*을 IAM 역할이 포함된 [계정 연결](#)의 이름으로 바꾸고, *iam-role-name*을 작업을 사용할 IAM 역할의 이름으로 바꿉니다. 이 역할은 환경의 기

본 IAM 역할을 재정의합니다. 스페이스에 역할을 추가했는지 확인합니다. 자세한 내용은 [계정 연결에 IAM 역할 추가](#) 섹션을 참조하세요.

자세한 내용은 [작업 유형](#) 항목을 참조하세요. 이 주제에는 YAML 참조를 포함하여 각 작업에 대한 설명서로 연결되는 링크가 있습니다.

워크플로 다이어그램에 앱 URL 표시

워크플로가 애플리케이션을 배포하는 경우 애플리케이션의 URL을 클릭 가능한 링크로 표시하도록 Amazon CodeCatalyst를 구성할 수 있습니다. 이 링크는 CodeCatalyst 콘솔에서 배포한 작업 내부에 나타납니다. 다음 워크플로 다이어그램은 작업 하단에 나타나는 앱 보기 URL을 보여줍니다.



CodeCatalyst 콘솔에서 이 URL을 클릭하여 애플리케이션 배포를 빠르게 확인할 수 있습니다.

Note

앱 URL은 Amazon ECS에 배포 작업에서는 지원되지 않습니다.

이 기능을 활성화하려면 `appurl` 또는 `endpointurl`이 포함된 이름으로 작업에 출력 변수를 추가합니다. 대시(-), 밑줄(_) 또는 공백()과 함께 또는 없이 이름을 사용할 수 있습니다. 문자열은 대/소문자를 구분하지 않습니다. 변수 값을 배포된 애플리케이션의 `http` 또는 `https` URL로 설정합니다.

Note

`app url`, 또는 `endpoint url` 문자열을 포함하도록 기존 출력 변수를 업데이트하는 경우 새 변수 이름을 사용하도록 이 변수에 대한 모든 참조를 업데이트합니다.

자세한 단계는 다음 절차를 참조하세요.

- ["AWS CDK 배포" 작업에 앱 URL을 표시하려면](#)
- ["CloudFormation 스택 배포" 작업에 앱 URL을 표시하려면](#)
- [다른 모든 작업에 앱 URL을 표시하려면](#)

URL 구성을 완료했으면 다음 지침에 따라 URL이 예상대로 표시되는지 확인합니다.

- [애플리케이션 URL이 추가되었는지 확인하려면](#)

"AWS CDK 배포" 작업에 앱 URL을 표시하려면

1. AWS CDK 배포 작업을 사용하는 경우 AWS CDK 애플리케이션 코드에 `CfnOutput` 구문(키-값 페어)을 추가합니다.
 - 키 이름에는 대시(-), 밑줄(_) 또는 공백()이 있거나 없는 `appurl` 또는 `endpointurl`이 포함되어야 합니다. 문자열은 대/소문자를 구분하지 않습니다.
 - 값은 배포된 애플리케이션의 `http` 또는 `https` URL이어야 합니다.

예를 들어 AWS CDK 코드는 다음과 같을 수 있습니다.

```
import { Duration, Stack, StackProps, CfnOutput, RemovalPolicy } from 'aws-cdk-lib';
import * as dynamodb from 'aws-cdk-lib/aws-dynamodb';
import * as s3 from 'aws-cdk-lib/aws-s3';
import { Construct } from 'constructs';
import * as cdk from 'aws-cdk-lib';
export class HelloCdkStack extends Stack {
  constructor(scope: Construct, id: string, props?: StackProps) {
```



```

    super(scope, id, props);
    const bucket = new s3.Bucket(this, 'amzn-s3-demo-bucket', {
        removalPolicy: RemovalPolicy.DESTROY,
    });
    new CfnOutput(this, 'APP-URL', {
        value: https://mycompany.myapp.com,
        description: 'The URL of the deployed application',
        exportName: 'myApp',
    });
    ...
}
}

```

CfnOutput 구문에 대한 자세한 내용은 AWS Cloud Development Kit (AWS CDK) API 참조의 [인터페이스 CfnOutputProps](#)를 참조하세요.

2. 코드를 저장하고 커밋합니다.
3. [애플리케이션 URL이 추가되었는지 확인하려면](#)로 이동합니다.

"CloudFormation 스택 배포" 작업에 앱 URL을 표시하려면

1. CloudFormation 스택 배포 작업을 사용하는 경우 다음과 같은 특성을 가진 CloudFormation 템플릿 또는 AWS SAM 템플릿의 Outputs 섹션에 출력을 추가합니다.
 - 키(논리적 ID라고도 함)에는 대시(-), 밑줄(_) 또는 공백()이 있거나 없는 appurl 또는 endpointurl이 포함되어야 합니다. 문자열은 대/소문자를 구분하지 않습니다.
 - 값은 배포된 애플리케이션의 http 또는 https URL이어야 합니다.

예를 들어 CloudFormation 템플릿은 다음과 같습니다.

```

"Outputs" : {
  "APP-URL" : {
    "Description" : "The URL of the deployed app",
    "Value" : "https://mycompany.myapp.com",
    "Export" : {
      "Name" : "My App"
    }
  }
}

```

CloudFormation에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 [출력](#)을 참조하세요.

2. 코드를 저장하고 커밋합니다.
3. [애플리케이션 URL이 추가되었는지 확인하려면](#)로 이동합니다.

다른 모든 작업에 앱 URL을 표시하려면

빌드 작업 또는 GitHub Actions와 같은 다른 작업을 사용하여 애플리케이션을 배포하는 경우 다음을 수행하여 앱 URL을 표시합니다.

1. 워크플로 정의 파일의 작업 Inputs 또는 Steps 섹션에서 환경 변수를 정의합니다. 변수는 다음과 같은 특성을 가져야 합니다.
 - name에는 대시(-), 밑줄(_) 또는 공백()이 있거나 없는 appurl 또는 endpointurl이 포함되어야 합니다. 문자열은 대/소문자를 구분하지 않습니다.
 - 값은 배포된 애플리케이션의 http 또는 https URL이어야 합니다.

예를 들어 빌드 작업은 다음과 같을 수 있습니다.

```
Build-action:
  Identifier: aws/build@v1
  Inputs:
    Variables:
      - Name: APP-URL
        Value: https://mycompany.myapp.com
```

아니면 다음과 같습니다.

```
Actions:
  Build:
    Identifier: aws/build@v1
    Configuration:
      Steps:
        - Run: APP-URL=https://mycompany.myapp.com
```

환경 변수 정의에 대한 자세한 내용은 [변수 정의](#) 섹션을 참조하세요.

2. 변수를 내보냅니다.

예를 들어 빌드 작업은 다음과 같습니다.

```
Build-action:
...
Outputs:
  Variables:
    - APP-URL
```

변수에 대한 자세한 내용은 [다른 작업에서 사용할 수 있도록 변수 내보내기](#) 섹션을 참조하세요.

3. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
4. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.
5. [애플리케이션 URL이 추가되었는지 확인하려면](#)로 이동합니다.

애플리케이션 URL이 추가되었는지 확인하려면

- 워크플로가 자동으로 시작되지 않은 경우 워크플로 실행을 시작합니다. 새 실행에는 워크플로 다이어그램에 클릭 가능한 링크로 표시되는 앱 URL이 있어야 합니다. 분석 시작하기에 대한 자세한 내용은 [워크플로 수동 실행 시작](#) 섹션을 참조하세요.

배포 대상 제거

CodeCatalyst 콘솔의 배포 대상 페이지에서 Amazon ECS 클러스터 또는 CloudFormation 스택과 같은 배포 대상을 제거할 수 있습니다.

Important

배포 대상을 제거하면 CodeCatalyst 콘솔에서 제거되지만 해당 배포 대상을 호스팅하는 AWS 서비스에서 계속 사용할 수 있습니다(있는 경우).

CodeCatalyst에서 대상이 오래된 경우 배포 대상을 제거하는 것이 좋습니다. 다음과 같은 경우 대상이 오래될 수 있습니다.

- 대상에 배포된 워크플로를 삭제했습니다.
- 배포하려는 스택 또는 클러스터를 변경했습니다.
- AWS 콘솔의 CloudFormation 또는 Amazon ECS 서비스에서 스택 또는 클러스터를 삭제했습니다.

배포 대상을 제거하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 환경을 선택합니다.
4. 제거하려는 배포 대상이 포함된 환경의 이름을 선택합니다. 환경에 대한 자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 섹션을 참조하세요.
5. 배포 대상 탭을 선택합니다.
6. 제거하려는 배포 대상 옆의 라디오 버튼을 선택합니다.
7. 제거를 선택합니다.

대상이 페이지에서 제거됩니다.

커밋별 배포 상태 추적

개발 수명 주기 중 언제든지 버그 수정, 새로운 특성 또는 기타 영향력 있는 변경 사항과 같은 특정 커밋의 배포 상태를 파악하는 것이 중요합니다. 배포 상태 추적 기능이 개발팀에 도움이 되는 다음 시나리오를 고려해 보세요.

- 개발자로서 버그를 해결하기 위해 수정 사항을 만들었고 팀의 배포 환경 전체에 배포된 상태를 보고하려고 합니다.
- 릴리스 관리자는 배포된 커밋 목록을 보고 배포 상태를 추적하고 보고하려고 합니다.

CodeCatalyst는 개별 커밋 또는 변경 사항이 배포된 위치와 환경을 한 눈에 확인할 수 있는 보기를 제공합니다. 이 보기에는 다음이 포함됩니다.

- 커밋 목록입니다.
- 커밋을 포함하는 배포의 상태입니다.
- 커밋이 성공적으로 배포되는 환경입니다.
- CI/CD 워크플로에서 커밋에 대해 실행되는 모든 테스트의 상태입니다.

다음 절차에서는 이 보기로 이동하고 이 보기를 사용하여 프로젝트의 변경 사항을 추적하는 방법을 자세히 설명합니다.

Note

커밋별 배포 상태 추적은 [CodeCatalyst 리포지토리](#)에서만 지원됩니다. 이 특성은 [GitHub 리포지토리](#), [Bitbucket 리포지토리](#) 또는 [GitLab 프로젝트 리포지토리](#)에서는 사용할 수 없습니다.

커밋별 배포 상태 추적

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 추적 변경을 선택합니다.
4. 메인 창 상단에 있는 두 개의 드롭다운 목록에서 릴리스 상태를 보려는 커밋이 포함된 소스 리포지토리와 브랜치를 선택합니다.
5. 변경 사항 보기를 선택합니다.

커밋 목록이 나타납니다.

각 클러스터에 대해 다음을 볼 수 있습니다.

- ID, 작성자, 메시지 및 커밋 시기와 같은 정보를 커밋합니다. 자세한 내용은 [CodeCatalyst의 소스 리포지토리로 코드 저장 및 협업](#) 섹션을 참조하세요.
- 각 환경에 대한 배포 상태입니다. 자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 섹션을 참조하세요.
- 테스트 및 코드 적용 범위 결과입니다. 자세한 내용은 [워크플로를 사용한 테스트](#) 섹션을 참조하세요.

Note

소프트웨어 구성 분석(SCA) 결과는 표시되지 않습니다.

6. 최신 배포, 자세한 코드 커버리지 및 단위 테스트 정보 등 특정 커밋과 관련된 변경 사항에 대한 자세한 정보를 보려면 해당 커밋에 대한 세부 정보 보기를 선택합니다.

배포 로그 보기

Amazon CodeCatalyst의 문제를 해결하기 위해 특정 배포 작업과 관련된 로그를 볼 수 있습니다.

[워크플로](#) 또는 [환경](#)에서 시작하는 로그를 볼 수 있습니다.

워크플로에서 시작하는 배포 작업의 로그를 보려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 실행을 선택합니다.
6. 애플리케이션을 배포한 워크플로 실행을 선택합니다.
7. 워크플로 다이어그램에서 로그를 보려는 작업을 선택합니다.
8. 로그 탭을 선택하고 섹션을 확장하여 로그 메시지를 표시합니다.
9. 더 많은 로그를 보려면 요약 탭을 선택한 다음 CloudFormation에서 보기(사용 가능한 경우)를 선택하여 더 많은 로그를 봅니다. AWS에 로그인해야 할 수도 있습니다.

환경에서 시작하는 배포 작업의 로그를 보려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 환경을 선택합니다.
4. 애플리케이션이 배포된 환경을 선택합니다.
5. 배포 활동에서 워크플로 실행 ID 열을 찾고 스택을 배포한 워크플로 실행을 선택합니다.
6. 워크플로 다이어그램에서 로그를 보려는 작업을 선택합니다.
7. 로그 탭을 선택하고 섹션을 확장하여 로그 메시지를 표시합니다.
8. 더 많은 로그를 보려면 요약 탭을 선택한 다음 CloudFormation에서 보기(사용 가능한 경우)를 선택하여 더 많은 로그를 봅니다. AWS에 로그인해야 할 수도 있습니다.

배포 정보 보기

Amazon CodeCatalyst에서 배포에 대한 다음 정보를 볼 수 있습니다.

- 배포 상태, 시작 시간, 종료 시간, 기록 및 이벤트 기간을 포함한 배포 활동입니다.
- 스택 이름, AWS 리전마지막 업데이트 시간 및 관련 워크플로.
- 요청을 커밋하고 가져옵니다.

- CloudFormation 이벤트 및 출력과 같은 작업별 정보입니다.

[워크플로](#), [환경](#) 또는 워크플로 [작업](#)부터 배포 정보를 볼 수 있습니다.

워크플로에서 시작하는 배포 정보를 보려면

- 애플리케이션을 배포한 워크플로 실행으로 이동합니다. 지침은 [워크플로 실행 상태 및 세부 정보 보기](#) 섹션을 참조하세요.

환경에서 시작하는 배포 정보를 보려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 환경을 선택합니다.
4. Production 같이 스택이 배포된 환경을 선택합니다.
5. 배포 활동을 선택하여 스택의 배포 기록, 배포 상태(예: 성공 또는 실패) 및 기타 배포 관련 정보를 확인합니다.
6. 환경에 배포된 스택, 클러스터 또는 기타 대상에 대한 정보를 보려면 배포 대상을 선택합니다. 스택 이름, 리전, 공급자 및 식별자와 같은 정보를 볼 수 있습니다.

작업에서 시작하는 배포 정보를 보려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 워크플로 다이어그램에서 애플리케이션을 배포한 워크플로 작업을 선택합니다. 예를 들어 DeployCloudFormationStack을 선택할 수 있습니다.
6. 작업별 배포 정보는 오른쪽 창의 내용을 검토하세요.

워크플로 생성

워크플로는 지속적 통합 및 지속적 전송(CI/CD) 시스템의 일부로 코드를 빌드, 테스트 및 배포하는 방법을 설명하는 자동화된 절차입니다. 워크플로는 워크플로 실행 중에 수행할 일련의 단계 또는 작업을 정의합니다. 또한 워크플로는 워크플로를 시작하게 하는 이벤트 또는 트리거를 정의합니다. 워크플로를 설정하려면 CodeCatalyst 콘솔의 [시각적 또는 YAML 편집기](#)를 사용하여 워크플로 정의 파일을 생성합니다.

Tip

프로젝트에서 워크플로를 사용하는 방법을 간략하게 알아보려면 [블루프린트가 있는 프로젝트를 생성](#)합니다. 각 블루프린트는 검토, 실행 및 실험할 수 있는 기능 워크플로를 배포합니다.

CodeCatalyst에서 워크플로를 생성하려면 다음 절차를 따르세요. 워크플로는 선택한 소스 리포지토리의 `~/.codecatalyst/workflows/` 폴더에 YAML 파일로 저장됩니다. 선택적으로 워크플로를 커밋할 때 워크플로 파일 이름 앞에 폴더 이름을 붙여 워크플로를 `~/.codecatalyst/workflows/`의 하위 폴더에 저장할 수 있습니다. 자세한 내용은 다음 지침을 참조하세요.

워크플로에 대한 자세한 내용은 [워크플로를 사용하여 빌드, 테스트 및 배포](#) 섹션을 참조하세요.

Visual

시각적 편집기를 사용하여 워크플로 생성

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로 생성을 선택합니다.

워크플로 생성 대화 상자가 표시됩니다.

5. 소스 리포지토리 필드에서 워크플로 정의 파일이 상주할 소스 리포지토리를 선택합니다. 소스 리포지토리가 없는 경우 [하나를 생성](#)합니다.
6. 브랜치 필드에서 워크플로 정의 파일이 상주할 브랜치를 선택합니다.
7. 생성(Create)을 선택합니다.

Amazon CodeCatalyst는 리포지토리 및 브랜치 정보를 메모리에 저장하지만 워크플로는 아직 커밋되지 않았습니다.

8. 비주얼을 선택합니다.
9. 워크플로 빌드:
 - a. (선택 사항) 워크플로 다이어그램에서 소스 및 트리거 상자를 선택합니다. 트리거 창이 나타납니다. 트리거에 연결하려면 트리거 추가를 선택합니다. 자세한 내용은 [워크플로에 트리거 추가](#) 섹션을 참조하세요.
 - b. + 작업(상단 왼쪽)을 선택합니다. 작업 카탈로그가 나타납니다.
 - c. 작업 내에서 더하기 기호(+)를 선택하여 워크플로에 추가합니다. 오른쪽에 있는 창을 사용하여 작업을 구성합니다. 자세한 내용은 [워크플로에 작업 추가](#) 섹션을 참조하세요.
 - d. (선택 사항) 워크플로 속성(오른쪽 상단)을 선택합니다. 워크플로 속성 창이 나타납니다. 워크플로 이름 실행 모드 및 컴퓨팅을 구성합니다. 자세한 내용은 [실행의 대기열 동작 구성](#) 및 [컴퓨팅 및 런타임 이미지 구성](#) 섹션을 참조하세요.
10. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
11. 커밋을 선택하고 커밋 워크플로 대화 상자에서 다음을 수행합니다.
 - a. 워크플로 파일 이름에 이름을 입력하거나 기본 이름을 그대로 유지합니다. 파일은 선택한 소스 리포지토리 및 브랜치의 `~/.codecatalyst/workflows/` 폴더에 저장됩니다. 파일 이름 앞에 폴더 또는 하위 폴더를 붙일 수 있습니다. 예시:
 - my-workflow(폴더 없음)를 지정하면 파일을 `~/.codecatalyst/workflows/my-workflow.yaml`로 저장
 - folder/subfolder/my-workflow를 지정하면 파일을 `~/.codecatalyst/workflows/folder/subfolder/my-workflow.yaml`로 저장
 - b. 커밋 메시지의 경우 기본 메시지를 그대로 두거나 직접 입력합니다.
 - c. 리포지토리 및 브랜치에서 워크플로 정의 파일의 소스 리포지토리 및 브랜치를 선택합니다. 이러한 필드는 워크플로 생성 대화 상자에서 앞서 지정한 리포지토리 및 브랜치로 설정해야 합니다. 원하는 경우 지금 리포지토리와 브랜치를 변경할 수 있습니다.

 Note

워크플로 정의 파일을 커밋한 후에는 다른 리포지토리 또는 브랜치와 연결할 수 없으므로 신중하게 선택해야 합니다.

- d. 커밋을 선택하여 워크플로 정의 파일을 커밋합니다.

YAML

YAML 편집기를 사용하여 워크플로 생성

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로 생성을 선택합니다.

워크플로 생성 대화 상자가 표시됩니다.

5. 소스 리포지토리 필드에서 워크플로 정의 파일이 상주할 소스 리포지토리를 선택합니다. 소스 리포지토리가 없는 경우 [하나를 생성](#)합니다.
6. 브랜치 필드에서 워크플로 정의 파일이 상주할 브랜치를 선택합니다.
7. 생성(Create)을 선택합니다.

Amazon CodeCatalyst는 리포지토리 및 브랜치 정보를 메모리에 저장하지만 워크플로는 아직 커밋되지 않았습니다.

8. YAML을 선택합니다.
9. 워크플로 빌드:
 - a. (선택 사항) YAML 코드에 트리거를 추가합니다. 자세한 내용은 [워크플로에 트리거 추가](#) 섹션을 참조하세요.
 - b. + 작업(상단 왼쪽)을 선택합니다. 작업 카탈로그가 나타납니다.
 - c. 작업 내에서 더하기 기호(+)를 선택하여 워크플로에 추가합니다. 오른쪽에 있는 창을 사용하여 작업을 구성합니다. 자세한 내용은 [워크플로에 작업 추가](#) 섹션을 참조하세요.
 - d. (선택 사항) 워크플로 속성(오른쪽 상단)을 선택합니다. 워크플로 속성 창이 나타납니다. 워크플로 이름, 실행 모드 및 컴퓨팅을 구성합니다. 자세한 내용은 [실행의 대기열 동작 구성](#) 및 [컴퓨팅 및 런타임 이미지 구성](#) 섹션을 참조하세요.
10. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
11. 커밋을 선택하고 커밋 워크플로 대화 상자에서 다음을 수행합니다.
 - a. 워크플로 파일 이름에 이름을 입력하거나 기본 이름을 그대로 유지합니다. 파일은 선택한 소스 리포지토리 및 브랜치의 `~/.codecatalyst/workflows/` 폴더에 저장됩니다. 파일 이름 앞에 폴더 또는 하위 폴더를 붙일 수 있습니다. 예시:

- my-workflow(폴더 없음)를 지정하면 파일을 ~/.codecatalyst/workflows/my-workflow.yaml로 저장
 - folder/subfolder/my-workflow를 지정하면 파일을 ~/.codecatalyst/workflows/folder/subfolder/my-workflow.yaml로 저장
- b. 커밋 메시지의 경우 기본 메시지를 그대로 두거나 직접 입력합니다.
 - c. 리포지토리 및 브랜치에서 워크플로 정의 파일의 소스 리포지토리 및 브랜치를 선택합니다. 이러한 필드는 워크플로 생성 대화 상자에서 앞서 지정한 리포지토리 및 브랜치로 설정해야 합니다. 원하는 경우 지금 리포지토리와 브랜치를 변경할 수 있습니다.

Note

워크플로 정의 파일을 커밋한 후에는 다른 리포지토리 또는 브랜치와 연결할 수 없으므로 신중하게 선택해야 합니다.

- d. 커밋을 선택하여 워크플로 정의 파일을 커밋합니다.

워크플로 실행

실행은 워크플로의 단일 반복입니다. 실행하는 동안 CodeCatalyst는 워크플로 구성 파일에 정의된 작업을 수행하고 관련 로그, 아티팩트 및 변수를 출력합니다.

워크플로 트리거를 통해 수동으로 실행을 시작하거나 자동으로 실행할 수 있습니다. 워크플로 트리거의 예시로는 기본 브랜치에 커밋을 푸시하는 소프트웨어 개발자가 있습니다.

실수로 워크플로를 시작한 경우 처리 도중에 실행 중인 워크플로를 수동으로 중지할 수도 있습니다.

여러 워크플로 실행이 거의 동시에 시작되는 경우 이러한 실행을 대기열에 넣는 방법을 구성할 수 있습니다. 실행이 시작된 순서대로 차례로 대기하는 기본 대기 동작을 사용하거나, 나중에 실행된 실행이 이전 실행을 대체(또는 '인수')하도록 하여 전체적으로 실행 속도를 높일 수 있습니다. 워크플로 실행을 병렬로 실행하여 다른 실행을 기다리지 않도록 설정하는 것도 가능합니다.

워크플로 실행을 수동 또는 자동으로 시작한 후에는 실행 상태 및 기타 세부 정보를 볼 수 있습니다. 예를 들어 언제 시작되었는지, 누가 시작했는지, 아직 실행 중인지 등을 확인할 수 있습니다.

주제

- [워크플로 수동 실행 시작](#)
- [트리거를 사용하여 워크플로 실행 자동 시작](#)

- [수동 전용 트리거 구성](#)
- [워크플로 실행 중지](#)
- [워크플로 게이팅](#)
- [워크플로 실행에 대한 승인 요구](#)
- [실행의 대기열 동작 구성](#)
- [워크플로 실행 간 파일 캐싱](#)
- [워크플로 실행 상태 및 세부 정보 보기](#)

워크플로 수동 실행 시작

Amazon CodeCatalyst에서 CodeCatalyst 콘솔에서 수동으로 워크플로 실행을 시작할 수 있습니다.

워크플로 실행에 대한 자세한 내용은 [워크플로 실행](#) 섹션을 참조하세요.

Note

[트리거를 구성](#)하여 워크플로 실행을 자동으로 시작할 수도 있습니다.

워크플로 실행 수동 시작

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 실행을 선택합니다.

트리거를 사용하여 워크플로 실행 자동 시작

워크플로 트리거를 사용하여 Amazon CodeCatalyst 워크플로가 자동으로 실행되도록 시작할 수 있습니다.

워크플로 트리거 또는 간단히 트리거를 사용하면 코드 푸시와 같은 특정 이벤트가 발생하면 워크플로 실행을 자동으로 시작할 수 있습니다. 트리거를 구성하여 소프트웨어 개발자가 CodeCatalyst 콘솔을 통해 워크플로 실행을 수동으로 시작하지 않아도 되도록 할 수 있습니다.

세 가지 유형의 트리거를 사용할 수 있습니다.

- **푸시** - 코드 푸시 트리거는 커밋이 푸시될 때마다 워크플로 실행을 시작하도록 합니다.
- **풀 요청** - 풀 요청 트리거는 풀 요청이 생성, 수정 또는 종료될 때마다 워크플로 실행을 시작하게 합니다.
- **일정** - 일정 트리거는 정의한 일정에 따라 워크플로 실행이 시작되도록 합니다. 소프트웨어 개발자가 다음 날 아침에 작업할 수 있도록 최신 빌드를 준비할 수 있도록 일정 트리거를 사용하여 소프트웨어의 야간 빌드를 실행하는 것을 고려하세요.

푸시, 풀 요청 및 예약 트리거를 단독으로 사용하거나 동일한 워크플로에서 조합하여 사용할 수 있습니다.

트리거는 선택 사항입니다. 구성하지 않으면 워크플로를 수동으로만 시작할 수 있습니다.

Tip

트리거가 실제로 작동하는지 확인하려면 블루프린트가 있는 프로젝트를 시작합니다. 대부분의 블루프린트에는 트리거가 있는 워크플로가 포함되어 있습니다. 블루프린트의 워크플로 정의 파일에서 Trigger 속성을 찾습니다. 청사진에 대한 자세한 내용은 [블루프린트를 사용하여 프로젝트 생성](#)을 참조하세요.

주제

- [예시: 워크플로의 트리거](#)
- [트리거 및 브랜치 사용 지침](#)
- [워크플로에 트리거 추가](#)

예시: 워크플로의 트리거

다음 예시에서는 Amazon CodeCatalyst 워크플로 정의 파일에 다양한 유형의 트리거를 추가하는 방법을 보여줍니다.

트리거에 대한 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 주제를 참조하세요.

주제

- [예시: 간단한 코드 푸시 트리거](#)
- [예시: 간단한 '메인으로 푸시' 트리거](#)

- [예시: 간단한 풀 요청 트리거](#)
- [예시: 간단한 일정 트리거](#)
- [예시: 일정 및 브랜치가 있는 트리거](#)
- [예시: 일정, 푸시 및 브랜치가 있는 트리거](#)
- [예시: 풀 및 브랜치가 있는 트리거](#)
- [예시: 풀, 브랜치 및 'CLOSED' 이벤트가 있는 트리거](#)
- [예시: 푸시, 브랜치 및 파일이 있는 트리거](#)
- [예시: 수동 트리거](#)
- [예시: CI/CD 다중 워크플로 설정의 트리거](#)

예시: 간단한 코드 푸시 트리거

다음 예시는 소스 리포지토리의 모든 브랜치로 코드가 푸시될 때마다 워크플로 실행을 시작하는 트리거를 보여줍니다.

이 트리거가 활성화되면 CodeCatalyst는 푸시하려는 브랜치(즉, 대상 브랜치)의 파일을 사용하여 워크플로 실행을 시작합니다.

예를 들어 커밋을 main에 푸시하면 CodeCatalyst는 main의 워크플로 정의 파일 및 기타 소스 파일을 사용하여 워크플로 실행을 시작합니다.

또 다른 예를 들면 커밋을 feature-branch-123에 푸시하면 CodeCatalyst는 feature-branch-123의 워크파우 정의 파일 및 기타 소스 파일을 사용하여 워크플로 실행을 시작합니다.

Triggers:

- Type: PUSH

Note

main으로 푸시할 때만 워크플로 실행을 시작하려면 [예시: 간단한 '메인으로 푸시' 트리거](#) 섹션을 참조하세요.

예시: 간단한 '메인으로 푸시' 트리거

다음 예시는 소스 리포지토리에서 코드가 main 브랜치(및 main 브랜치만)에 푸시될 때마다 워크플로 실행을 시작하는 트리거를 보여줍니다.

```
Triggers:
- Type: PUSH
  Branches:
  - main
```

예시: 간단한 풀 요청 트리거

다음 예시는 소스 리포지토리에서 풀 요청이 생성되거나 수정될 때마다 워크플로 실행을 시작하는 트리거를 보여줍니다.

이 트리거가 활성화되면 CodeCatalyst는 워크플로 정의 파일과 가져오려는 브랜치(즉, 소스 브랜치)의 다른 소스 파일을 사용하여 워크플로 실행을 시작합니다.

예를 들어, feature-123 소스 브랜치와 main 대상 브랜치가 있는 풀 요청을 만들면 CodeCatalyst는 feature-123에서 워크플로 정의 파일 및 기타 소스 파일을 사용하여 워크플로 실행을 시작합니다.

```
Triggers:
- Type: PULLREQUEST
  Events:
  - OPEN
  - REVISION
```

예시: 간단한 일정 트리거

다음 예시는 매주 월요일부터 금요일까지 자정(UTC+0)에 워크플로 실행을 시작하는 트리거를 보여줍니다.

이 트리거가 활성화되면 CodeCatalyst는 이 트리거가 포함된 워크플로 정의 파일이 포함된 소스 리포지토리의 각 브랜치에 대해 단일 워크플로 실행을 시작합니다.

예를 들어 소스 리포지토리에 main, release-v1, feature-123라는 세 개의 브랜치가 있고 각 브랜치에 트리거가 다음과 같은 워크플로 정의 파일이 포함된 경우 CodeCatalyst는 세 개의 워크플로 실행을 시작합니다. 하나는 main의 파일을 사용하고, 다른 하나는 release-v1의 파일을 사용하며, 다른 하나는 feature-123의 파일을 사용합니다.

```
Triggers:
- Type: SCHEDULE
  Expression: "0 0 ? * MON-FRI *"
```

Expression 속성에서 사용할 수 있는 cron 표현식의 자세한 예시는 [Expression](#) 섹션을 참조하세요.

예시: 일정 및 브랜치가 있는 트리거

다음 예시는 매일 오후 6시 15분(UTC+0)에 워크플로 실행을 시작하는 트리거를 보여줍니다.

이 트리거가 활성화되면 CodeCatalyst는 main 브랜치의 파일을 사용하여 워크플로 실행을 시작하고 release-로 시작하는 각 브랜치에 대해 추가 실행을 시작합니다.

예를 들어 소스 리포지토리에 main, release-v1, bugfix-1, bugfix-2라는 브랜치가 있는 경우 CodeCatalyst는 두 개의 워크플로 실행을 시작합니다. 하나는 main의 파일을 사용하고 다른 하나는 release-v1의 파일을 사용합니다. bugfix-1 및 bugfix-1 브랜치에 대한 워크플로 실행을 시작하지 않습니다.

```
Triggers:
- Type: SCHEDULE
  Expression: "15 18 * * ? *"
  Branches:
    - main
    - release\-.*
```

Expression 속성에서 사용할 수 있는 cron 표현식의 자세한 예시는 [Expression](#) 섹션을 참조하세요.

예시: 일정, 푸시 및 브랜치가 있는 트리거

다음 예시에서는 매일 자정(UTC+0)에 워크플로 실행을 시작하는 트리거와 코드가 main 브랜치로 푸시될 때마다 트리거를 보여줍니다.

이 예시에서는 다음이 적용됩니다.

- 워크플로 실행은 매일 자정에 시작됩니다. 워크플로 실행은 main 브랜치의 워크플로 정의 파일 및 기타 소스 파일을 사용합니다.
- 워크플로 실행은 또한 main 브랜치에 커밋을 푸시할 때마다 시작됩니다. 워크플로 실행은 대상 브랜치(main)의 워크플로 정의 파일 및 기타 소스 파일을 사용합니다.

```
Triggers:
- Type: SCHEDULE
  Expression: "0 0 * * ? *"
  Branches:
    - main
- Type: PUSH
```


Branches:

- main

Expression 속성에서 사용할 수 있는 cron 표현식의 자세한 예시는 [Expression](#) 섹션을 참조하세요.

예시: 풀 및 브랜치가 있는 트리거

다음 예시는 누군가 main 대상 브랜치로 풀 요청을 열거나 수정할 때마다 워크플로 실행을 시작하는 트리거를 보여줍니다. Triggers 구성에 지정된 브랜치는 main이지만 워크플로 실행은 소스 브랜치 (가져오는브랜치)의 워크플로 정의 파일 및 기타 소스 파일을 사용합니다.

Triggers:

- Type: PULLREQUEST

Branches:

- main

Events:

- OPEN
- REVISION

예시: 풀, 브랜치 및 'CLOSED' 이벤트가 있는 트리거

다음 예시는 main로 시작하는 브랜치에서 풀 요청이 종료될 때마다 워크플로 실행을 시작하는 트리거를 보여줍니다.

이 예시에서는 다음이 적용됩니다.

- main으로 시작하는 대상 브랜치로 풀 요청을 닫으면 워크플로 정의 파일과 (현재 달힌) 소스 브랜치의 기타 소스 파일을 사용하여 워크플로 실행이 자동으로 시작됩니다.
- 풀 요청이 병합된 후 소스 리포지토리가 브랜치를 자동으로 삭제하도록 구성한 경우 이러한 브랜치는 CLOSED 상태로 들어갈 기회가 없습니다. 즉, 병합된 브랜치는 풀 요청 CLOSED 트리거를 활성화하지 않습니다. 이 시나리오에서 CLOSED 트리거를 활성화하는 유일한 방법은 병합하지 않고 풀 요청을 닫는 것입니다.

Triggers:

- Type: PULLREQUEST

Branches:

- main.*

Events:

- CLOSED

예시: 푸시, 브랜치 및 파일이 있는 트리거

다음 예시는 main 브랜치의 filename.txt 파일 또는 src 디렉터리에 있는 파일을 변경할 때마다 워크플로 실행을 시작하는 트리거를 보여줍니다.

이 트리거가 활성화되면 CodeCatalyst는 main 브랜치의 워크플로 정의 파일 및 기타 소스 파일을 사용하여 워크플로 실행을 시작합니다.

```
Triggers:
- Type: PUSH
  Branches:
    - main
  FilesChanged:
    - filename.txt
    - src\/*.*
```

예시: 수동 트리거

수동 트리거를 구성하려면 워크플로 정의 파일에서 Triggers 섹션을 생략합니다. 이 섹션이 없으면 사용자는 CodeCatalyst 콘솔에서 실행 버튼을 선택하여 워크플로를 수동으로 시작해야 합니다. 자세한 내용은 [워크플로 수동 실행 시작](#) 섹션을 참조하세요.

예시: CI/CD 다중 워크플로 설정의 트리거

이 예시에서는 지속적 통합(CI) 및 지속적 전송(CD)을 위해 별도의 Amazon CodeCatalyst 워크플로를 사용하려는 경우 트리거를 설정하는 방법을 설명합니다.

이 시나리오에서는 두 가지 워크플로를 설정합니다.

- CI 워크플로 - 이 워크플로는 풀 요청이 생성되거나 수정될 때 애플리케이션을 빌드하고 테스트합니다.
- CD 워크플로 - 이 워크플로는 풀 요청이 병합될 때 애플리케이션을 빌드하고 배포합니다.

CI 워크플로의 정의 파일은 다음과 비슷합니다.

```
Triggers:
- Type: PULLREQUEST
  Branches:
    - main
  Events:
```

- OPEN
- REVISION

Actions:

BuildAction:

instructions-for-building-the-app

TestAction:

instructions-for-test-the-app

Triggers 코드는 소프트웨어 개발자가 특성 브랜치를 main브랜치에 병합하도록 요청하는 풀 요청을 생성할 때마다(또는 [수정](#)할 때마다) 워크플로 실행을 자동으로 시작하도록 나타냅니다. CodeCatalyst는 소스 브랜치(기능 브랜치)의 소스 코드를 사용하여 워크플로 실행을 시작합니다.

CD 워크플로의 정의 파일은 다음과 비슷합니다.

Triggers:

- Type: PUSH
- Branches:
 - main

Actions:

BuildAction:

instructions-for-building-the-app

DeployAction:

instructions-for-deploying-the-app

Triggers 코드는 main에 병합이 발생할 때 워크플로를 자동으로 시작하도록 나타냅니다. CodeCatalyst는 main 브랜치의 소스 코드를 사용하여 워크플로 실행을 시작합니다.

트리거 및 브랜치 사용 지침

이 섹션에서는 브랜치를 포함하는 Amazon CodeCatalyst 트리거를 설정할 때의 몇 가지 주요 지침에 대해 설명합니다.

트리거에 대한 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 주제를 참조하세요.

- 지침 1: 푸시 및 풀 요청 트리거 모두에서 브랜치를 지정하려면 트리거 구성에서 대상(또는 '대상') 브랜치를 지정해야 합니다. 소스(또는 'from') 브랜치를 지정하지 마세요.

다음 예시에서는 브랜치에서 main을 눌러 워크플로를 활성화합니다.

Triggers:

- Type: PUSH

```
Branches:
  - main
```

다음 예시에서는 main으로 브랜치에서 요청을 가져오면 워크플로가 활성화됩니다.

```
Triggers:
  - Type: PULLREQUEST
    Branches:
      - main
    Events:
      - OPEN
      - REVISION
```

- 지침 2: 푸시 트리거의 경우 워크플로가 활성화되면 destination 브랜치에 있는 워크플로 정의 파일과 소스 파일을 사용하여 워크플로가 실행됩니다.
- 지침 3: 풀 요청 트리거의 경우 워크플로가 활성화되면 트리거 구성에서 대상 브랜치를 지정했다더라도 source 브랜치에 있는 워크플로 정의 파일과 소스 파일을 사용하여 워크플로가 실행됩니다.
- 지침 4: 한 브랜치에서 정확히 동일한 트리거가 다른 브랜치에서는 실행되지 않을 수 있습니다.

다음 푸시 트리거를 고려합니다.

```
Triggers:
  - Type: PUSH
    Branches:
      - main
```

이 트리거가 포함된 워크플로 정의 파일이 main에 존재하고 test에 복제된 경우 워크플로는 test의 파일을 사용하여 자동으로 시작되지 않습니다(test의 파일을 사용하도록 워크플로를 수동으로 시작할 수는 있지만). 지침 2를 검토하여 test의 파일을 사용하여 워크플로가 자동으로 실행되지 않는 이유를 알아봅니다.

다음과 같은 풀 요청 트리거도 고려해 보세요.

```
Triggers:
  - Type: PULLREQUEST
    Branches:
      - main
    Events:
      - OPEN
      - REVISION
```

이 트리거가 포함된 워크플로 정의 파일이 main에 있는 경우 워크플로는 main의 파일을 사용하여 실행되지 않습니다. (단, main의 test 브랜치를 생성하면 워크플로는 test의 파일을 사용하여 실행됩니다.) 지침 3을 검토하여 이유를 이해합니다.

워크플로에 트리거 추가

다음 지침에 따라 Amazon CodeCatalyst 워크플로에 푸시, 풀 또는 일정 트리거를 추가합니다.

트리거에 대한 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 주제를 참조하세요.

Visual

트리거 추가(시각 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 워크플로 다이어그램에서 소스 및 트리거 상자를 선택합니다.
8. 구성 창에서 트리거 추가를 선택합니다.
9. 트리거 추가 대화 상자에서 다음과 같이 필드에 정보를 입력합니다.

트리거 유형

트리거 유형을 지정합니다. 다음 값 중 하나를 사용할 수 있습니다.

- 푸시(시각적 편집기) 또는 PUSH(YAML 편집기)

푸시 트리거는 변경 사항이 소스 리포지토리로 푸시될 때 워크플로 실행을 시작합니다. 워크플로 실행은 푸시 대상 브랜치(즉, 대상 브랜치)의 파일을 사용합니다.

- 풀 요청(시각적 편집기) 또는 PULLREQUEST(YAML 편집기)

풀 요청 트리거는 소스 리포지토리에서 풀 요청이 열리거나 업데이트되거나 닫힐 때 워크플로로 실행을 시작합니다. 워크플로로 실행은 가져오려는 브랜치(즉, 소스 브랜치)의 파일을 사용합니다.

- 일정(시각적 편집기) 또는 SCHEDULE(YAML 편집기)

일정 트리거는 지정한 cron 표현식으로 정의된 일정에 따라 워크플로를 실행합니다. 브랜치 파일을 사용하여 소스 리포지토리의 각 브랜치에 대해 별도의 워크플로 실행이 시작됩니다. (트리거가 활성화되는 브랜치를 제한하려면 브랜치 필드(시각적 편집기) 또는 Branches 속성(YAML 편집기)을 사용합니다.)

일정 트리거를 구성할 때는 다음 지침을 따르세요.

- 워크플로당 하나의 일정 트리거만 사용합니다.
- CodeCatalyst 스페이스에 여러 워크플로를 정의한 경우 동시에 시작하도록 10개 이하로 예약하는 것이 좋습니다.
- 실행 사이에 충분한 시간을 두고 트리거의 cron 표현식을 구성해야 합니다. 자세한 내용은 [Expression](#) 섹션을 참조하세요.

예시는 [예시: 워크플로의 트리거](#) 섹션을 참조하세요.

풀 요청에 대한 이벤트

이 필드는 풀 요청 트리거 유형을 선택한 경우에만 나타납니다.

워크플로 실행을 시작할 풀 요청 이벤트의 유형을 지정합니다. 유효한 값은 다음과 같습니다.

- 풀 요청이 생성됨(시각적 편집기) 또는 OPEN(YAML 편집기)

풀 요청이 생성되면 워크플로 실행이 시작됩니다.

- 풀 요청이 닫힘(시각적 편집기) 또는 CLOSED(YAML 편집기)

풀 요청이 종료되면 워크플로 실행이 시작됩니다. CLOSED 이벤트의 동작은 까다롭고 예시를 통해 가장 잘 이해됩니다. 자세한 정보는 [예시: 풀, 브랜치 및 'CLOSED' 이벤트가 있는 트리거](#)를 참조하세요.

- 풀 요청에 대한 새 개정(시각적 편집기) 또는 REVISION(YAML 편집기)

워크플로 실행은 풀 요청에 대한 개정이 생성될 때 시작됩니다. 풀 요청이 생성되면 첫 번째 개정이 생성됩니다. 그런 다음 풀 요청에 지정된 소스 브랜치에 새 커밋을 푸시할 때마다

새 개정이 생성됩니다. 풀 요청 트리거에 REVISION 이벤트를 포함하는 경우 REVISION은 OPEN의 수퍼 세트이므로 OPEN 이벤트를 생략할 수 있습니다.

동일한 풀 요청 트리거에서 여러 이벤트를 지정할 수 있습니다.

예시는 [예시: 워크플로의 트리거](#) 섹션을 참조하세요.

Schedule

이 필드는 일정 트리거 유형을 선택한 경우에만 나타납니다.

예약된 워크플로 실행 시기를 설명하는 cron 표현식을 지정합니다.

CodeCatalyst의 Cron 표현식은 다음과 같은 6개 필드 구문을 사용합니다. 여기서 각 필드는 공백으로 구분됩니다.

minutes hours days-of-month month days-of-week year

cron 표현식의 예

분	시간	일	월	요일	연도	의미
0	0	?	*	월-금	*	매주 월요일부터 금요일까지 자정(UTC +0)에 워크플로를 실행합니다.
0	2	*	*	?	*	매일 오전 2시(UTC +0)에 워크플로를 실행합니다.

분	시간	일	월	요일	연도	의미
15	22	*	*	?	*	매일 오후 10:15(UTC+0)에 워크플로를 실행합니다.
0/30	22-2	?	*	토-일	*	시작일 오후 10시부터 다음 날 오전 2시(UTC+0) 사이에 토요일부터 일요일까지 30분마다 워크플로를 실행합니다.
45	13	L	*	?	2023-2027	2023년부터 2027년까지 매월 마지막 날 오후 1시 45분(UTC+0)에 워크플로를 실행합니다.

CodeCatalyst에서 cron 표현식을 지정할 때는 다음 지침을 따라야 합니다.

- SCHEDULE 트리거당 단일 cron 표현식을 지정합니다.

- YAML 편집기에서 cron 표현식을 큰따옴표(")로 묶습니다.
- 시간은 협정 세계시(UTC)로 지정합니다. 다른 시간대는 지원되지 않습니다.
- 실행 간격은 최소 30분으로 구성합니다. 더 빠른 주기는 지원되지 않습니다.
- *days-of-month* 또는 *days-of-week* 필드를 지정하되 둘 다 지정하지는 않습니다. 필드 중 하나에 값 또는 별표(*)를 지정하는 경우 다른 필드에는 물음표(?)를 사용해야 합니다. 별표는 '모두'를 의미하고 물음표는 '모두'를 의미합니다.

cron 표현식의 더 많은 예시와 ?, *, L과 같은 와일드카드에 대한 정보는 Amazon EventBridge 사용 설명서의 [cron 표현식 참조](#)를 참조하세요. EventBridge 및 CodeCatalyst의 Cron 표현식은 정확히 동일한 방식으로 작동합니다.

일정 트리거의 예시는 [예시: 워크플로의 트리거](#) 섹션을 참조하세요.

브랜치 및 브랜치 패턴

(선택 사항)

워크플로 실행을 시작할 시기를 알기 위해 트리거가 모니터링하는 소스 리포지토리의 브랜치를 지정합니다. 정규식 패턴을 사용하여 브랜치 이름을 정의할 수 있습니다. 예를 들어 `main.*`을 사용하여 `main`으로 시작하는 모든 브랜치를 일치시킵니다.

지정할 브랜치는 트리거 유형에 따라 다릅니다.

- 푸시 트리거의 경우 푸시 대상 브랜치, 즉 대상 브랜치를 지정합니다. 일치하는 브랜치의 파일을 사용하여 일치하는 브랜치당 하나의 워크플로 실행이 시작됩니다.

예시: `main.*,mainline`

- 풀 요청의 경우 푸시할 브랜치, 즉 대상 브랜치를 지정합니다. 워크플로 정의 파일과 소스 브랜치(일치하는 브랜치 아님)의 소스 파일을 사용하여 일치하는 브랜치당 하나의 워크플로 실행이 시작됩니다.

예시: `main.*,mainline,v1\-.*(v1-로 시작하는 브랜치와 일치)`

- 일정 트리거의 경우 예약된 실행에서 사용할 파일이 포함된 브랜치를 지정합니다. 워크플로 정의 파일과 일치하는 브랜치의 소스 파일을 사용하여 일치하는 브랜치당 하나의 워크플로 실행이 시작됩니다.

예시: `main.*,version\ -1\ .0`

Note

브랜치를 지정하지 않으면 트리거는 소스 리포지토리의 모든 브랜치를 모니터링하고 다음에서 워크플로 정의 파일 및 소스 파일을 사용하여 워크플로 실행을 시작합니다.

- 푸시 대상 브랜치(푸시 트리거용). 자세한 내용은 [예시: 간단한 코드 푸시 트리거](#) 섹션을 참조하세요.
- 풀 원본 브랜치(풀 요청 트리거용). 자세한 내용은 [예시: 간단한 풀 요청 트리거](#) 섹션을 참조하세요.
- 모든 브랜치(일정 트리거용). 소스 리포지토리의 브랜치당 워크플로 실행이 한 번 시작됩니다. 자세한 내용은 [예시: 간단한 일정 트리거](#) 섹션을 참조하세요.

브랜치 및 트리거에 대한 자세한 내용은 [트리거 및 브랜치 사용 지침](#) 섹션을 참조하세요.

더 많은 예시는 [예시: 워크플로의 트리거](#)를 참조합니다.

파일 변경됨

이 필드는 푸시 또는 풀 요청 트리거 유형을 선택한 경우에만 나타납니다.

워크플로 실행을 시작해야 하는 시기를 알 수 있도록 트리거가 모니터링하는 소스 리포지토리의 파일 또는 폴더를 지정합니다. 정규식을 사용하여 파일 이름 또는 경로와 일치시킬 수 있습니다.

예시는 [예시: 워크플로의 트리거](#) 섹션을 참조하세요.

10. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
11. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML**트리거 추가(YAML 편집기)**

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.

4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 다음 예시를 안내서로 사용하여 Triggers 섹션 및 기본 속성을 추가합니다. 자세한 내용은 [워크플로 YAML 정의](#)의 [Triggers](#)를 참조하세요.

코드 푸시 트리거는 다음과 같을 수 있습니다.

```
Triggers:
  - Type: PUSH
    Branches:
      - main
```

풀 요청 트리거는 다음과 같을 수 있습니다.

```
Triggers:
  - Type: PULLREQUEST
    Branches:
      - main.*
    Events:
      - OPEN
      - REVISION
      - CLOSED
```

일정 트리거는 다음과 같을 수 있습니다.

```
Triggers:
  - Type: SCHEDULE
    Branches:
      - main.*
    # Run the workflow at 1:15 am (UTC+0) every Friday until the end of 2023
    Expression: "15 1 ? * FRI 2022-2023"
```

Expression 속성에서 사용할 수 있는 cron 표현식의 자세한 예시는 [Expression](#) 섹션을 참조하세요.

푸시, 풀 요청 및 일정 트리거의 자세한 예시는 [예시: 워크플로의 트리거](#) 섹션을 참조하세요.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.

9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

수동 전용 트리거 구성

CodeCatalyst 콘솔의 실행 버튼을 사용하여 팀에서 수동으로만 워크플로를 시작할 수 있도록 워크플로를 제한할 수 있습니다. 이 기능을 구성하려면 워크플로 정의 파일에서 Triggers 섹션을 제거해야 합니다. Triggers 섹션은 워크플로를 생성할 때 기본적으로 포함되지만, 섹션은 선택 사항이며 제거할 수 있습니다.

워크플로를 수동으로만 시작할 수 있도록 다음 지침에 따라 워크플로 정의 파일의 Triggers 섹션을 제거합니다.

트리거에 대한 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 주제를 참조하세요.

워크플로 실행에 대한 자세한 내용은 [워크플로 실행](#) 섹션을 참조하세요.

Visual

'Triggers' 섹션 제거(시각적 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 워크플로 다이어그램에서 소스 상자를 선택합니다.
8. 트리거에서 휴지통 아이콘을 선택하여 워크플로에서 Triggers 섹션을 제거합니다.
9. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
10. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

'Triggers' 섹션 제거(YAML 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.

2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. Triggers 섹션을 찾아 제거합니다.
8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

워크플로 실행 중지

진행 중인 워크플로 실행을 중지하려면 다음 절차를 따르세요. 실수로 시작된 경우 실행을 중지할 수 있습니다.

워크플로 실행을 중지하면 CodeCatalyst는 진행 중인 작업이 완료될 때까지 기다렸다가 CodeCatalyst 콘솔에서 실행을 중지됨으로 표시합니다. 시작할 기회를 얻지 못한 모든 작업은 시작되지 않고 중단됨으로 표시됩니다.

Note

실행이 대기열에 있는 경우(즉, 진행 중인 작업이 없는 경우) 실행이 즉시 중지됩니다.

워크플로 실행에 대한 자세한 내용은 [워크플로 실행](#) 섹션을 참조하세요.

워크플로 실행 중지

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로에서 실행을 선택하고 목록에서 진행 중인 실행을 선택합니다.
5. 중지를 선택합니다.

워크플로 게이팅

게이트는 특정 조건이 충족되지 않으면 워크플로 실행이 진행되지 않도록 하는 데 사용할 수 있는 워크플로 구성 요소입니다. 게이트의 예시로는 사용자가 워크플로 실행을 계속하기 전에 CodeCatalyst 콘솔에서 승인을 제출해야 하는 승인 게이트를 들 수 있습니다.

워크플로의 일련의 작업 사이에 또는 첫 번째 작업(소스 다운로드 직후에 실행되는) 전에 게이트를 추가할 수 있습니다. 필요한 경우 마지막 작업 후 게이트를 추가할 수도 있습니다.

워크플로 실행에 대한 자세한 내용은 [워크플로 실행](#) 섹션을 참조하세요.

주제

- [게이트 유형](#)
- [다른 작업과 병렬로 실행되도록 게이트를 설정할 수 있나요?](#)
- [게이트를 사용하여 워크플로 실행이 시작되지 않도록 할 수 있나요?](#)
- [게이트의 제한 사항](#)
- [워크플로에 게이트 추가](#)
- [게이트 및 작업 순서 지정](#)
- [게이트의 버전 지정](#)

게이트 유형

현재 Amazon CodeCatalyst는 승인 게이트라는 한 가지 유형의 게이트를 지원합니다. 자세한 내용은 [워크플로 실행에 대한 승인 요구](#) 섹션을 참조하세요.

다른 작업과 병렬로 실행되도록 게이트를 설정할 수 있나요?

아니요. 게이트는 작업 전후에만 실행할 수 있습니다. 자세한 내용은 [게이트 및 작업 순서 지정](#) 섹션을 참조하세요.

게이트를 사용하여 워크플로 실행이 시작되지 않도록 할 수 있나요?

예, 자격 조건이 있습니다.

워크플로 실행이 태스크를 수행하지 못하도록 할 수 있습니다. 이는 워크플로 실행이 시작하지 못하도록 하는 것과 약간 다릅니다.

워크플로가 작업을 수행하지 못하도록 워크플로의 첫 번째 작업 앞에 게이트를 추가합니다. 이 시나리오에서는 워크플로 실행이 시작됩니다. 즉, 소스 리포지토리 파일을 다운로드하지만 게이트가 잠금 해제될 때까지 태스크를 수행할 수 없습니다.

Note

워크플로가 시작되었다가 게이트에 의해 차단된 경우에도 스페이스당 최대 동시 워크플로 실행 수 할당량 및 기타 할당량에 포함됩니다. 워크플로 할당량을 초과하지 않도록 하려면 게이트를 사용하는 대신 워크플로 트리거를 사용하여 워크플로를 조건부로 시작하는 것이 좋습니다. 게이트 대신 풀 요청 승인 규칙을 사용하는 것도 좋습니다. 할당량, 트리거 및 풀 요청 승인 규칙에 대한 자세한 내용은 [CodeCatalyst의 워크플로 할당량](#), [트리거를 사용하여 워크플로 실행 자동 시작](#) 및 [풀 요청을 승인 규칙과 병합하기 위한 요구 사항 관리](#) 섹션을 참조하세요.

게이트의 제한 사항

게이트에는 다음과 같은 제한 사항이 있습니다.

- 게이트는 컴퓨팅 공유 특성과 함께 사용할 수 없습니다. 이 기능에 대한 자세한 내용은 [작업 간에 컴퓨팅 공유](#)를 참조하세요.
- 게이트는 작업 그룹 내에서 사용할 수 없습니다. 작업 그룹에 대한 자세한 내용은 [작업을 작업 그룹으로 그룹화](#) 섹션을 참조하세요.

워크플로에 게이트 추가

Amazon CodeCatalyst에서는 워크플로에 게이트를 추가하여 특정 조건이 충족되지 않으면 진행되지 않도록 할 수 있습니다. 다음 안내에 따라 워크플로에 게이트를 추가합니다.

게이트에 대한 자세한 내용은 [워크플로 게이팅](#) 섹션을 참조하세요.

게이트 추가 및 구성

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.

5. Edit(편집)을 선택합니다.
6. 비주얼을 선택합니다.
7. 왼쪽에서 게이트를 선택합니다.
8. 게이트 카탈로그에서 게이트를 검색한 다음 더하기 기호(+)를 선택하여 워크플로에 게이트를 추가합니다.
9. 게이스 구성 시각적 편집기를 사용하려면 비주얼을 선택하고, YAML 편집기를 사용하려면 YAML을 선택합니다. 자세한 지침은 다음을 참조하세요.

- ['승인' 게이트 추가](#)

10. (선택 사항) YAML 코드가 유효한지 확인하려면 검증을 선택합니다.
11. 커밋을 선택하여 변경 사항을 커밋합니다.

게이트 및 작업 순서 지정

Amazon CodeCatalyst에서는 워크플로 작업, 작업 그룹 또는 게이트의 전후에 실행되도록 게이트를 설정할 수 있습니다. 예를 들어 Deploy 작업 전에 실행할 Approval 게이트를 설정할 수 있습니다. 이 경우 Deploy 작업은 Approval 게이트에 종속된다고 합니다.

게이트와 작업 간의 종속성을 설정하려면 게이트 또는 작업의 Depends on 속성을 구성합니다. 지침은 [작업 간 종속성 설정](#) 섹션을 참조하세요. 참조된 지침은 워크플로 작업을 참조하지만 게이트에도 동일하게 적용됩니다.

게이트에서 Depends on 속성을 설정하는 방법의 예시는 [예시: '승인' 게이트](#) 섹션을 참조하세요.

게이트에 대한 자세한 내용은 [워크플로 게이팅](#) 섹션을 참조하세요.

워크플로 작업에 대한 자세한 내용은 [워크플로 작업 구성](#) 섹션을 참조하세요.

게이트의 버전 지정

기본적으로 워크플로에 게이트를 추가하면 CodeCatalyst는 형식을 사용하여 워크플로 정의 파일에 전체 버전을 추가합니다.

vmajor.minor.patch

예:

My-Gate:

Identifier: aws/approval@v1

워크플로가 특정 메이저 또는 마이너 버전의 게이트를 사용하도록 버전을 늘릴 수 있습니다. 지침은 [사용할 작업 버전 지정](#) 섹션을 참조하세요. 참조된 주제는 워크플로 작업을 참조하지만 게이트에도 동일하게 적용됩니다.

CodeCatalyst의 게이트에 대한 자세한 내용은 [워크플로 게이팅](#) 섹션을 참조하세요.

워크플로 실행에 대한 승인 요구

워크플로 실행을 구성하여 진행하기 전에 승인이 필요하도록 할 수 있습니다. 이렇게 하려면 워크플로에 승인 [게이트](#)를 추가해야 합니다. 승인 게이트는 사용자 또는 사용자 집합이 CodeCatalyst 콘솔에서 하나 이상의 승인을 제출할 때까지 워크플로가 진행되지 않도록 합니다. 모든 승인이 제공되면 게이트가 '잠금 해제됨' 상태가 되고 워크플로 실행을 재개할 수 있습니다.

워크플로의 승인 게이트를 사용하여 개발, 운영 및 리더십 팀이 변경 사항을 더 많은 대상에 배포하기 전에 검토할 수 있는 기회를 제공합니다.

워크플로 실행에 대한 자세한 내용은 [워크플로 실행](#) 섹션을 참조하세요.

주제

- [승인 게이트를 잠금 해제하려면 어떻게 해야 하나요?](#)
- ['승인' 게이트를 사용해야 하는 경우](#)
- [누가 승인을 제공할 수 있나요?](#)
- [사용자에게 승인이 필요함을 알려려면 어떻게 해야 하나요?](#)
- ['승인' 게이트를 사용하여 워크플로 실행이 시작되지 않도록 할 수 있나요?](#)
- [워크플로 승인은 대기 중 실행, 대체된 실행 및 병렬 실행 모드에서 어떻게 작동하나요?](#)
- [예시: '승인' 게이트](#)
- ['승인' 게이트 추가](#)
- [승인 알림 구성](#)
- [워크플로 실행 승인 또는 거부](#)
- ['승인' 게이트 YAML](#)

승인 게이트를 잠금 해제하려면 어떻게 해야 하나요?

승인 게이트를 잠금 해제하려면 다음 조건을 모두 충족해야 합니다.

- 조건 1: 필요한 승인 수를 제출해야 합니다. 필요한 승인 수는 구성 가능하며 각 사용자는 단일 승인을 제출할 수 있습니다.
- 조건 2: 게이트 제한 시간 전에 모든 승인을 제출해야 합니다. 게이트는 활성화되고 14일 후에 시간 초과됩니다. 이 기간은 구성할 수 없습니다.
- 조건 3: 아무도 워크플로 실행을 거부해서는 안 됩니다. 단일 거부로 인해 워크플로 실행이 실패합니다.
- 조건 4: (대체된 실행 모드를 사용하는 경우에만 적용됩니다.) 이후 실행으로 실행을 대체해서는 안 됩니다. 자세한 내용은 [워크플로 승인은 대기 중 실행, 대체된 실행 및 병렬 실행 모드에서 어떻게 작동하나요?](#) 섹션을 참조하세요.

조건이 충족되지 않으면 CodeCatalyst는 워크플로를 중지하고 실행 상태를 실패(조건 1~3의 경우) 또는 대체됨(조건 4의 경우)으로 설정합니다.

'승인' 게이트를 사용해야 하는 경우

일반적으로 애플리케이션 및 기타 리소스를 프로덕션 서버 또는 품질 표준을 검증해야 하는 환경에 배포하는 워크플로에서 승인 게이트를 사용합니다. 프로덕션으로 배포하기 전에 게이트를 배치하면 검토자가 새 소프트웨어 개정을 공개적으로 사용할 수 있게 되기 전에 검증할 수 있습니다.

누가 승인을 제공할 수 있나요?

프로젝트의 멤버이고 기여자 또는 프로젝트 관리자 역할이 있는 모든 사용자는 승인을 제공할 수 있습니다. 프로젝트 스페이스에 속한 스페이스 관리자 역할을 가진 사용자도 승인을 제공할 수 있습니다.

Note

검토자 역할을 가진 사용자는 승인을 제공할 수 없습니다.

사용자에게 승인이 필요함을 알려려면 어떻게 해야 하나요?

사용자에게 승인이 필요함을 알려려면 다음을 수행해야 합니다.

- CodeCatalyst가 Slack 알림을 보내도록 합니다. 자세한 내용은 [승인 알림 구성](#) 섹션을 참조하세요.
- 승인 및 거부 버튼이 있는 CodeCatalyst 콘솔의 페이지로 이동하여 해당 페이지의 URL을 승인자에게 전달되는 이메일 또는 메시징 애플리케이션에 붙여넣습니다. 이 페이지로 이동하는 방법에 대한 자세한 내용은 [워크플로 실행 승인 또는 거부](#) 섹션을 참조하세요.

'승인' 게이트를 사용하여 워크플로 실행이 시작되지 않도록 할 수 있나요?

예, 자격 조건이 있습니다. 자세한 내용은 [게이트를 사용하여 워크플로 실행이 시작되지 않도록 할 수 있나요?](#) 섹션을 참조하세요.

워크플로 승인은 대기 중 실행, 대체된 실행 및 병렬 실행 모드에서 어떻게 작동하나요?

대기 중, 대체된, 병렬로 실행 모드를 사용하는 경우 승인 게이트는 [작업](#)과 유사한 방식으로 작동합니다. 이러한 실행 모드를 숙지하려면 [대기 중 실행 모드 정보](#), [대체된 실행 모드 정보](#), [병렬 실행 모드 정보](#) 섹션을 읽는 것이 좋습니다. 이러한 실행 모드에 대한 기본적인 이해가 되면 이 섹션으로 돌아가서 승인 게이트가 있을 때 이러한 실행 모드가 어떻게 작동하는지 알아봅니다.

승인 게이트가 있으면 다음과 같이 실행이 처리됩니다.

- [대기 중 실행 모드](#)를 사용하는 경우 실행은 현재 게이트에서 승인을 기다리고 있는 실행 뒤에 대기합니다. 해당 게이트가 잠금 해제되면(즉, 모든 승인이 제공됨) 대기열의 다음 실행이 게이트로 진행되고 승인을 기다립니다. 이 프로세스는 대기 중 실행이 게이트를 통해 하나씩 처리되는 동안 계속됩니다. [Figure 1](#)는 이 프로세스를 보여줍니다.
- [대체된 실행 모드](#)를 사용하는 경우 동작은 대기열에 있는 실행 모드의 동작과 동일합니다. 단, 게이트에서 대기열에 파일을 쌓는 대신 새 실행이 이전 실행을 대체(인수)합니다. 대기열이 없으며 현재 게이트에서 승인을 기다리고 있는 모든 실행은 취소되고 최신 실행으로 대체됩니다. [Figure 2](#)는 이 프로세스를 보여줍니다.
- [병렬 실행 모드](#)를 사용하는 경우 실행이 병렬로 시작되며 대기열이 형성되지 않습니다. 앞에 실행이 없으므로 각 실행은 게이트에서 즉시 처리됩니다. [Figure 3](#)는 이 프로세스를 보여줍니다.

그림 1: '대기 중 실행 모드' 및 승인 게이트

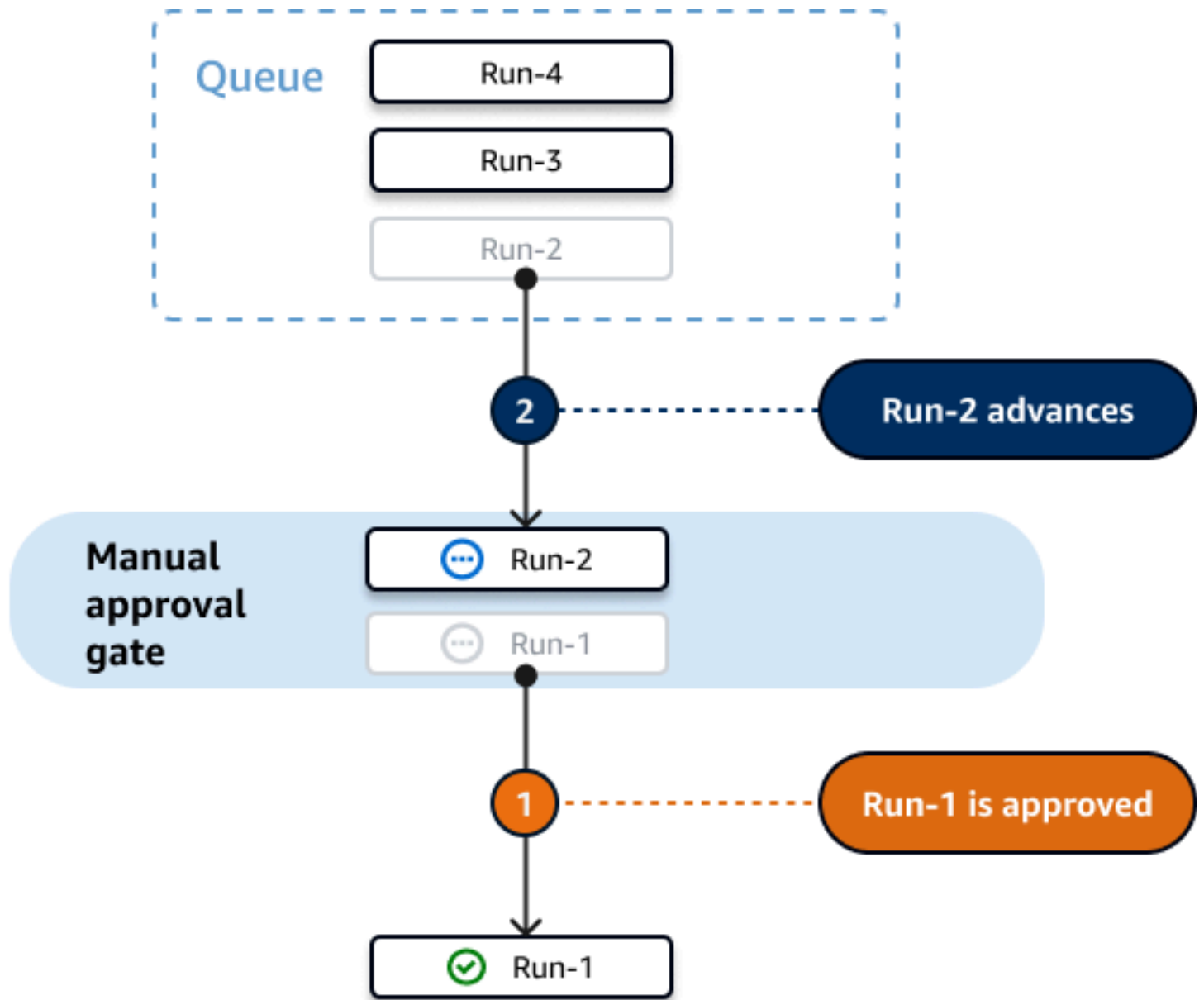


그림 2: '대체된 실행 모드' 및 승인 게이트

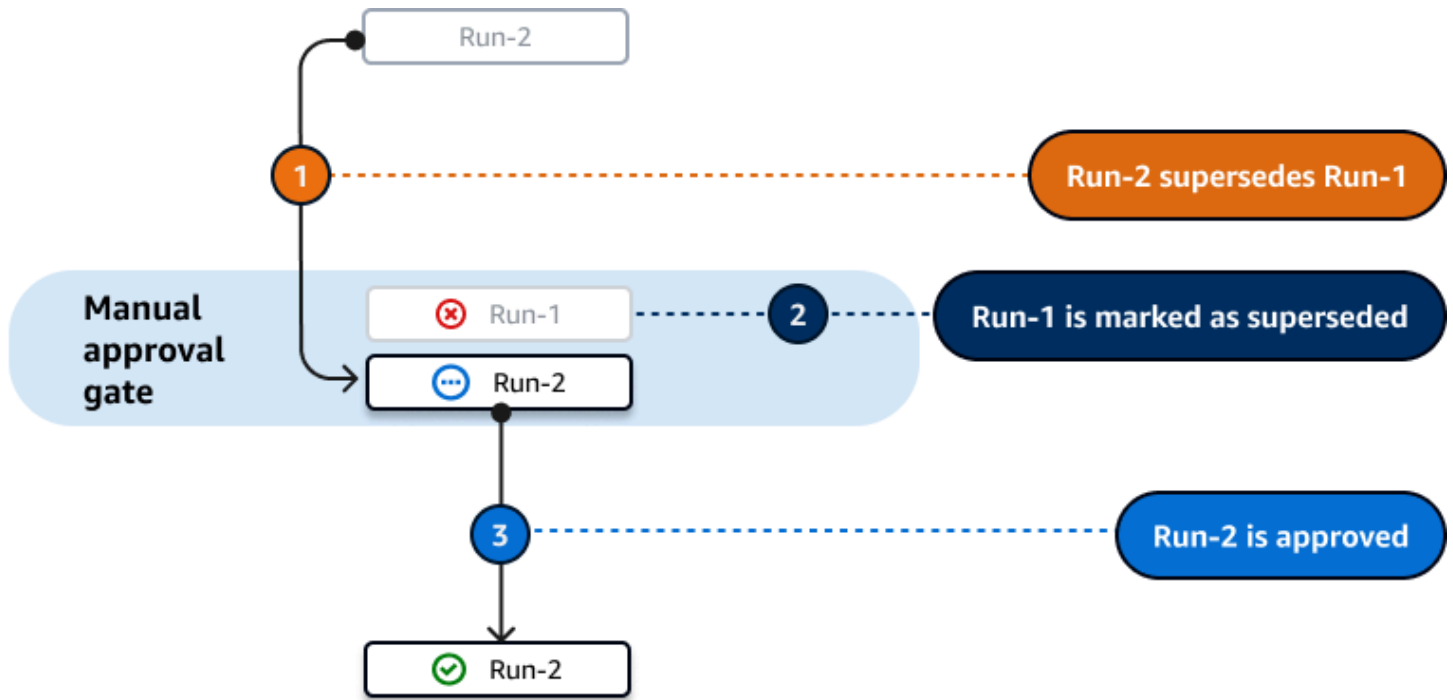
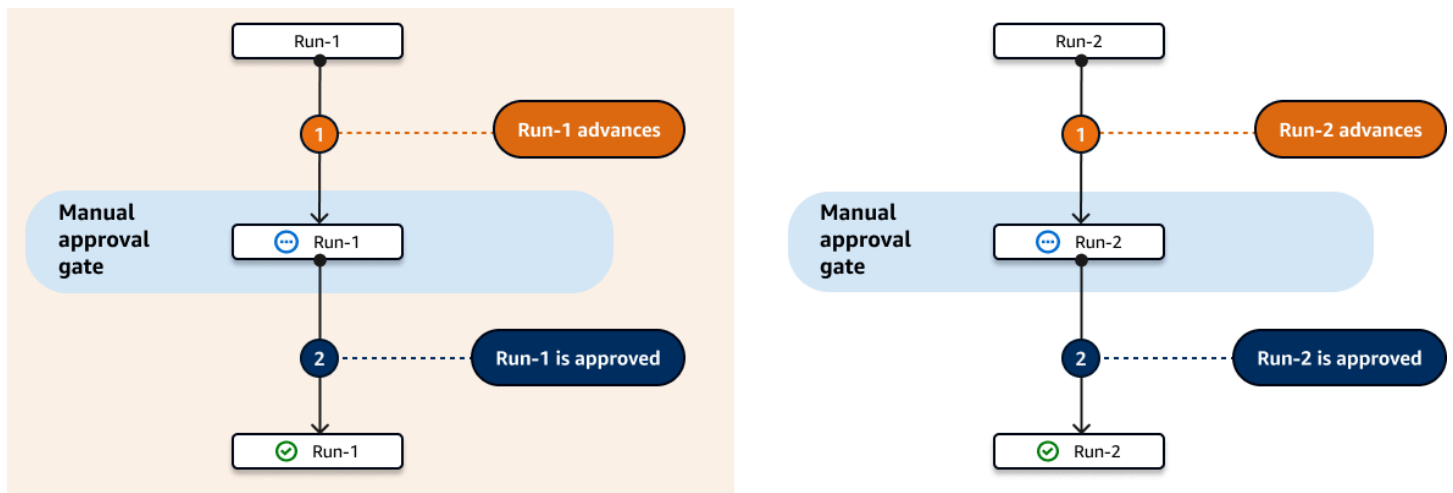


그림 3: '병렬 실행 모드' 및 승인 게이트



예시: '승인' 게이트

다음 예시는 Staging과 Production이라는 두 작업 사이에 Approval_01라는 승인 게이트를 추가하는 방법을 보여줍니다. Staging 작업이 먼저 실행되고, Approval_01 게이트가 두 번째로 실행되며, Production 작업이 마지막으로 실행됩니다. Production 작업은 Approval_01 게이트가 잠금 해제된 경우에만 실행됩니다. DependsOn 속성은 Staging, Approval_01 및 Production 단계가 순차적으로 실행되도록 합니다.

승인 게이트에 대한 자세한 내용은 [워크플로 실행에 대한 승인 요구](#) 섹션을 참조하세요.

Actions:

```

Staging: # Deploy to a staging server
  Identifier: aws/ecs-deploy@v1
  Configuration:
    ...
Approval_01:
  Identifier: aws/approval@v1
  DependsOn:
    - Staging
  Configuration:
    ApprovalsRequired: 2
Production: # Deploy to a production server
  Identifier: aws/ecs-deploy@v1
  DependsOn:
    - Approval_01
  Configuration:
    ...

```

'승인' 게이트 추가

승인이 필요하도록 워크플로를 구성하려면 워크플로에 승인 게이트를 추가해야 합니다. 다음 지침에 따라 워크플로에 승인 게이트를 추가합니다.

이 게이트에 대한 자세한 내용은 [워크플로 실행에 대한 승인 요구](#) 섹션을 참조하세요.

Visual

워크플로에 '승인' 게이트 추가(시각적 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 왼쪽 상단에서 게이트를 선택합니다.
7. 게이트 카탈로그의 승인 에서 더하기 기호(+)를 선택합니다.
8. 입력을 선택하고 종속 대상 필드에서 다음을 수행합니다.

이 게이트를 실행하기 위해 성공적으로 실행해야 하는 작업, 작업 그룹 또는 게이트를 지정합니다. 기본적으로 워크플로에 게이트를 추가할 때 게이트는 워크플로의 마지막 작업에 따라 달라지도록 설정됩니다. 이 속성을 제거하면 게이트가 다른 작업에 종속되지 않고 다른 작업보다 먼저 실행됩니다.

 Note

작업, 작업 그룹 또는 게이트 전후에 실행되도록 게이트를 구성해야 합니다. 다른 작업, 작업 그룹 및 게이트와 동시에 실행되도록 설정할 수 없습니다.

Depends on 함수에 대한 자세한 내용은 [게이트 및 작업 순서 지정](#) 섹션을 참조하세요.

9. 구성 탭을 선택합니다.
10. 게이트 이름 필드에서 다음을 수행합니다.

게이트에 부여할 이름을 지정합니다. 모든 게이트 이름은 워크플로 내에서 고유해야 합니다. 게이트 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 게이트 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

11. (선택 사항) 승인 수 필드에서 다음을 수행합니다.

승인 게이트를 잠금 해제하는 데 필요한 최소 승인 수를 지정합니다. 최소값은 1입니다. 최대값은 2입니다. 생략 시 기본값은 1입니다.

 Note

ApprovalsRequired 속성을 생략하려면 워크플로 정의 파일에서 게이트 Configuration 섹션을 제거합니다.

12. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
13. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

워크플로에 '승인' 게이트 추가(YAML 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.

2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 다음 예시를 가이드로 사용하여 Approval 섹션 및 기본 속성을 추가합니다. 자세한 내용은 [워크플로 YAML 정의의 '승인' 게이트 YAML](#)를 참조하세요.

```

Actions:
  MyApproval_01:
    Identifier: aws/approval@v1
    DependsOn:
      - PreviousAction
    Configuration:
      ApprovalsRequired: 2

```

다른 예시는 [예시: '승인' 게이트](#)를 참조하세요.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

승인 알림 구성

CodeCatalyst가 워크플로 실행에 승인이 필요하다는 알림을 사용자에게 Slack 채널로 보내도록 할 수 있습니다. 사용자는 알림을 보고 해당 알림 내에 있는 링크를 클릭합니다. 이 링크는 워크플로를 승인하거나 거부할 수 있는 CodeCatalyst 승인 페이지로 이동합니다.

워크플로가 승인되었거나 거부되었거나 승인 요청이 만료되었음을 사용자에게 알리도록 알림을 구성할 수도 있습니다.

다음 지침에 따라 Slack 알림을 설정합니다.

시작하기 전 준비 사항

워크플로에 승인 게이트를 추가했는지 확인합니다. 자세한 내용은 ['승인' 게이트 추가](#) 섹션을 참조하세요.

Slack 채널로 워크플로 승인 알림 보내기

1. Slack을 사용하여 CodeCatalyst를 구성합니다. 자세한 내용은 [스택 알림 시작하기](#) 섹션을 참조하세요.
2. 승인이 필요한 워크플로가 포함된 CodeCatalyst 프로젝트에서 알림이 아직 활성화되어 있지 않은 경우 활성화합니다. 알림을 활성화하려면:
 - a. 프로젝트로 이동하고 탐색 창에서 프로젝트 설정을 선택합니다.
 - b. 상단에서 알림을 선택합니다.
 - c. 알림 이벤트에서 알림 편집을 선택합니다.
 - d. 워크플로 승인 보류를 켜고 CodeCatalyst가 알림을 보낼 Slack 채널을 선택합니다.
 - e. (선택 사항) 승인, 거부 및 만료된 승인에 대해 사용자에게 알려려면 추가 알림을 복사합니다. 워크플로 실행 승인됨, 워크플로 실행 거부됨, 워크플로 승인 대체됨 및 워크플로 승인 제한 시간 초과됨을 설정할 수 있습니다. 각 알림 옆에서 CodeCatalyst가 알림을 보낼 Slack 채널을 선택합니다.
 - f. 저장을 선택합니다.

워크플로 실행 승인 또는 거부

승인 게이트가 포함된 워크플로 실행은 승인 또는 거부되어야 합니다. 사용자는 다음 시점부터 승인 또는 거부를 제공할 수 있습니다.

- CodeCatalyst 콘솔
- 팀원이 제공하는 링크
- 자동 Slack 알림

사용자가 승인 또는 거부의 의사를 밝힌 후에는 이 결정을 되돌릴 수 없습니다.

Note

특정 사용자만 워크플로 실행을 승인하거나 거부할 수 있습니다. 자세한 내용은 [누가 승인을 제공할 수 있나요?](#) 섹션을 참조하세요.

시작하기 전 준비 사항

워크플로에 승인 게이트를 추가했는지 확인합니다. 자세한 내용은 '[승인](#)' [게이트 추가](#) 섹션을 참조하세요.

CodeCatalyst 콘솔에서 시작하는 워크플로 실행 승인 또는 거부

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 워크플로 다이어그램에서 승인 게이트를 나타내는 상자를 선택합니다.

측면 패널이 나타납니다.

 Note

이때 원한다면 이 페이지의 URL을 다른 승인자에게 보낼 수 있습니다.

6. 검토 결정에서 승인 또는 거부를 선택합니다.
7. (선택 사항) 설명 - 선택 사항에 워크플로 실행을 승인하거나 거부한 이유를 나타내는 설명을 입력합니다.
8. 제출을 선택합니다.

팀원이 제공한 링크에서 시작하는 워크플로 실행 승인 또는 거부

1. 팀원이 전송한 링크를 선택합니다. (팀원에게 위의 절차를 읽고 링크를 받도록 할 수 있습니다.)
2. 메시지가 표시되면 CodeCatalyst에 로그인합니다.

워크플로 실행 승인 페이지로 리디렉션됩니다.

3. 검토 결정에서 승인 또는 거부를 선택합니다.
4. (선택 사항) 설명 - 선택 사항에 워크플로 실행을 승인하거나 거부한 이유를 나타내는 설명을 입력합니다.
5. 제출을 선택합니다.

자동 Slack 알림에서 시작하는 워크플로 실행 승인 또는 거부

1. Slack 알림이 설정되어 있는지 확인합니다. [승인 알림 구성](#)(를) 참조하세요.
2. Slack의 승인 알림이 전송된 채널에서 승인 알림의 링크를 선택합니다.
3. 메시지가 표시되면 CodeCatalyst에 로그인합니다.

워크플로 실행 페이지로 리디렉션됩니다.

4. 워크플로 다이어그램에서 승인 게이트를 선택합니다.
5. 검토 결정에서 승인 또는 거부를 선택합니다.
6. (선택 사항) 설명 - 선택 사항에 워크플로 실행을 승인하거나 거부한 이유를 나타내는 설명을 입력합니다.
7. 제출을 선택합니다.

'승인' 게이트 YAML

다음은 승인 게이트의 YAML 정의입니다. 이 게이트에 대해 자세히 알아보려면 [워크플로 실행에 대한 승인 요구](#) 섹션을 참조하세요.

이 작업 정의는 더 광범위한 워크플로 정의 파일 내의 섹션으로 존재합니다. 이 파일에 대한 자세한 내용은 [워크플로 YAML 정의](#)을 참조합니다.

Note

이어지는 대부분의 YAML 속성에는 시각적 편집기에 해당 UI 요소가 있습니다. UI 요소를 찾으려면 Ctrl+F를 사용합니다. 요소가 연결된 YAML 속성과 함께 나열됩니다.

```
# The workflow definition starts here.
# See ### ## for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The 'Approval' gate definition starts here.
Approval:
  Identifier: aws/approval@v1
  DependsOn:
```

```
- another-action
Configuration:
  ApprovalsRequired: number
```

Approval

(필수)

게이트에 부여할 이름을 지정합니다. 모든 게이트 이름은 워크플로 내에서 고유해야 합니다. 게이트 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 게이트 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

기본값: Approval_nn.

해당 UI: 구성 탭/게이트 이름

Identifier

(*Approval*/Identifier)

(필수)

게이트를 식별합니다. 승인 게이트는 버전 1.0.0을 지원합니다. 버전을 단축하려는 경우가 아니면 이 속성을 변경하지 마세요. 자세한 내용은 [사용할 작업 버전 지정](#) 섹션을 참조하세요.

기본값: aws/approval@v1.

해당 UI: 워크플로 다이어그램/Approval_nn/aws/approval@v1 레이블

DependsOn

(*Approval*/DependsOn)

(선택 사항)

이 게이트를 실행하기 위해 성공적으로 실행해야 하는 작업, 작업 그룹 또는 게이트를 지정합니다. 기본적으로 워크플로에 게이트를 추가할 때 게이트는 워크플로의 마지막 작업에 따라 달라지도록 설정됩니다. 이 속성을 제거하면 게이트가 다른 작업에 종속되지 않고 다른 작업보다 먼저 실행됩니다.

Note

작업, 작업 그룹 또는 게이트 전후에 실행되도록 게이트를 구성해야 합니다. 다른 작업, 작업 그룹 및 게이트와 동시에 실행되도록 설정할 수 없습니다.

Depends on 함수에 대한 자세한 내용은 [게이트 및 작업 순서 지정](#) 섹션을 참조하세요.

해당 UI: 입력 탭/의존

Configuration

(*Approval*/Configuration)

(선택 사항)

게이트의 구성 속성을 정의할 수 있는 섹션입니다.

해당 UI: 구성 탭

ApprovalsRequired

(*Approval*/Configuration/ApprovalsRequired)

(선택 사항)

승인 게이트를 잠금 해제하는 데 필요한 최소 승인 수를 지정합니다. 최소값은 1입니다. 최대값은 2입니다. 생략 시 기본값은 1입니다.

Note

ApprovalsRequired 속성을 생략하려면 워크플로 정의 파일에서 게이트 Configuration 섹션을 제거합니다.

해당 UI: 구성 탭/승인 수

실행의 대기열 동작 구성

Amazon CodeCatalyst에서 기본적으로 여러 워크플로가 동시에 실행되면 CodeCatalyst는 워크플로를 대기열에 넣고 시작 순서대로 하나씩 처리합니다. 실행 모드를 지정하여 이 기본 동작을 변경할 수 있습니다. 몇 가지 실행 모드가 있습니다.

- (기본값) 대기 중 실행 모드 - CodeCatalyst 프로세스가 하나씩 실행됩니다.
- 대체된 실행 모드 - CodeCatalyst 프로세스는 하나씩 실행되며, 최신 실행은 이전 실행보다 우선합니다.

- 병렬 실행 모드 - CodeCatalyst 프로세스는 병렬로 실행됩니다.

워크플로 실행에 대한 자세한 내용은 [워크플로 실행](#) 섹션을 참조하세요.

주제

- [대기 중 실행 모드 정보](#)
- [대체된 실행 모드 정보](#)
- [병렬 실행 모드 정보](#)
- [실행 모드 구성](#)

대기 중 실행 모드 정보

대기 중 실행 모드 에서 실행은 직렬로 수행되며 대기열을 구성하는 대기 실행이 있습니다.

대기열은 작업 및 작업 그룹에 대한 진입점에 형성되므로 동일한 워크플로 내에 여러 대기열을 가질 수 있습니다([Figure 1](#) 참조). 대기 중인 실행이 작업에 들어가면 작업이 잠기고 다른 실행은 입력할 수 없습니다. 실행이 완료되고 작업을 종료하면 작업이 잠금 해제되고 다음 실행을 위한 준비가 됩니다.

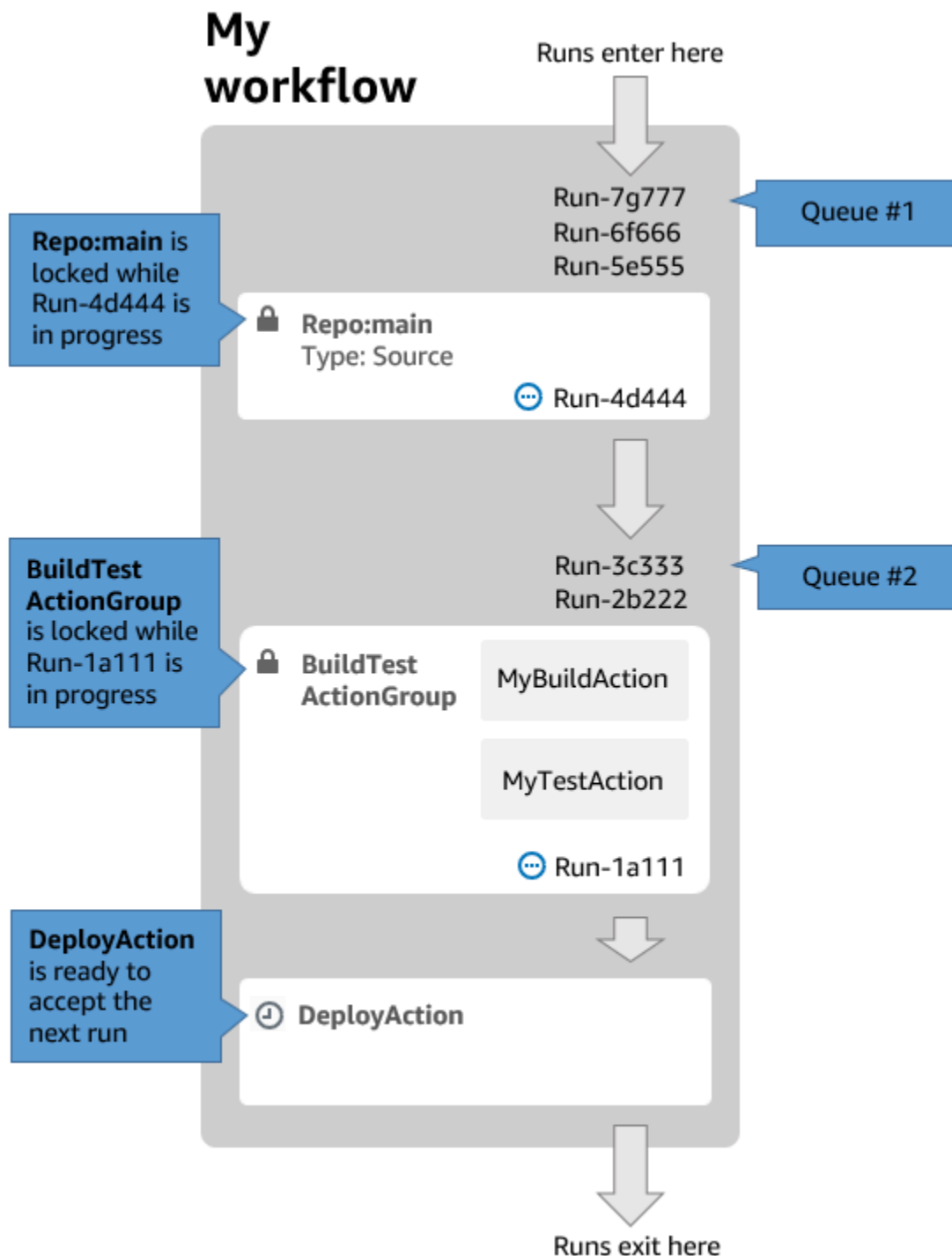
[Figure 1](#) 는 대기 중 실행 모드로 구성된 워크플로를 보여줍니다. 표시되는 정보는 다음과 같습니다.

- 워크플로를 통해 7번의 실행이 수행됩니다.
- 두 개의 대기열: 입력 소스에 대한 항목 외부의 대기열(Repo:main)과 BuildTestActionGroup 작업에 대한 항목 외부의 대기열.
- 두 개의 잠긴 블록: 입력 소스(Repo:main) 및 BuildTestActionGroup.

워크플로가 처리 완료를 실행함에 따라 상황이 어떻게 트랜스피어링되는지는 다음과 같습니다.

- Run-4d444가 소스 리포지토리 복제를 완료하면 입력 소스를 종료하고 Run-3c333 뒤에 있는 대기열에 조인합니다. 그런 다음 Run-5e555가 입력 소스를 입력합니다.
- Run-1a111이 빌드 및 테스트를 완료하면 BuildTestActionGroup 작업을 종료하고 DeployAction 작업에 들어갑니다. 그런 다음 Run-2b222는 BuildTestActionGroup 작업에 들어갑니다.

그림 1: '대기 중 실행 모드'로 구성된 워크플로



다음과 같은 경우 대기 중 실행 모드를 사용합니다.

- 기능과 실행 간에 일대일 관계를 유지하려는 경우 대체된 모드를 사용할 때 이러한 기능을 그룹화할 수 있습니다. 예를 들어 커밋 1에서 기능 1을 병합하면 1이 시작되고 커밋 2에서 기능 2를 병합하면 2가 시작되는 등의 작업이 수행됩니다. 대기중 모드 대신 대체된 모드를 사용하는 경우 다른 기능(및 커밋)을 대체하는 기능(및 커밋)이 실행에서 함께 그룹화됩니다.

- 병렬 모드 를 사용할 때 발생할 수 있는 레이스 조건과 예상치 못한 문제를 피하려고 합니다. 예를 들어 Wang과 Saanvi라는 두 소프트웨어 개발자가 Amazon ECS 클러스터에 배포하기 위해 워크플로를 거의 동시에 실행하는 경우 Wang의 실행은 클러스터에서 통합 테스트를 시작할 수 있지만 Saanvi의 실행은 클러스터에 새 애플리케이션 코드를 배포하여 Wang의 테스트가 실패하거나 잘못된 코드를 테스트합니다. 또 다른 예로 잠금 메커니즘이 없는 대상이 있을 수 있습니다. 이 경우 두 실행이 예상치 못한 방식으로 서로의 변경 사항을 덮어쓸 수 있습니다.
- CodeCatalyst가 실행을 처리하는 데 사용하는 컴퓨팅 리소스에 대한 부하를 제한하려고 합니다. 예를 들어 워크플로에 작업이 3개 있는 경우 동시에 최대 3회 실행할 수 있습니다. 한 번에 발생할 수 있는 실행 수에 제한을 가하면 실행 처리량을 더 예측할 수 있습니다.
- 워크플로를 통해 타사 서비스에 대한 요청 수를 제한하려고 합니다. 예를 들어 워크플로에 Docker Hub에서 이미지를 가져오는 지침이 포함된 빌드 작업이 있을 수 있습니다. [Docker Hub는 계정별로 시간당 특정 수로 수행할 수 있는 풀 요청 수를 제한](#)하며, 한도를 초과하면 잠깁니다. 대기 중 실행 모드를 사용하여 실행 처리량을 줄이면 시간당 Docker Hub에 대한 요청을 더 적게 생성할 수 있으므로 잠금 및 그에 따른 빌드 및 실행 실패의 가능성이 제한됩니다.

최대 대기열 크기: 50

최대 대기열 크기에 대한 참고 사항:

- 최대 대기열 크기는 워크플로의 모든 대기열에 허용되는 최대 실행 수를 나타냅니다.
- 대기열이 50회 이상 실행되면 CodeCatalyst는 51번째 및 후속 실행을 삭제합니다.

실패 동작:

작업으로 처리되는 동안 실행이 응답하지 않으면 작업 시간이 초과될 때까지 대기열에 실행이 보류됩니다. 1시간 후 작업 제한 시간입니다.

작업 내에서 실행이 실패하면 작업 뒤에 있는 첫 번째 대기 중 실행을 계속 진행할 수 있습니다.

대체된 실행 모드 정보

대체된 실행 모드는 다음을 제외하고 대기 중 실행 모드와 동일합니다.

- 대기열에서 대기 중인 실행이 대기열의 다른 실행을 따라잡으면 이후 실행이 이전 실행을 대체(인계)하고 이전 실행이 취소되고 '대체됨'으로 표시됩니다.
- 첫 번째 글머리 기호에 설명된 동작의 결과로 대체된 실행 모드를 사용하는 경우 대기열에 하나의 실행만 포함될 수 있습니다.

[Figure 1](#)의 워크플로를 가이드로 사용하면 대체된 실행 모드를 이 워크플로에 적용하면 다음과 같은 결과가 발생합니다.

- Run-7g777은 대기열의 다른 두 실행을 대체하며 Queue #1에 남아 있는 유일한 실행입니다. Run-6f666 및 Run-5e555가 취소됩니다.
- Run-3c333은 Run-2b222를 대체하며 Queue #2에 남아 있는 유일한 실행입니다. Run-2b222가 취소됩니다.

원하는 경우 대체된 실행 모드를 사용합니다.

- 대기 중 모드보다 처리량 향상
- 대기 중 모드보다 타사 서비스에 대한 요청이 훨씬 적음. 이는 타사 서비스에 Docker Hub와 같은 속도 제한이 있는 경우 유용합니다.

병렬 실행 모드 정보

병렬 실행 모드에서 실행은 서로 독립적이며 시작하기 전에 다른 실행이 완료될 때까지 기다리지 않습니다. 대기열이 없으며 실행 처리량은 워크플로 내 작업이 완료되는 데 걸리는 속도에 따라 제한됩니다.

각 사용자에게 자체 기능 브랜치가 있고 다른 사용자가 공유하지 않는 대상에 배포하는 개발 환경에서 병렬 실행 모드를 사용합니다.

Important

프로덕션 환경의 Lambda 함수와 같이 여러 사용자가 배포할 수 있는 공유 대상이 있는 경우 레이스 조건이 발생할 수 있으므로 병렬 모드를 사용하지 마세요. 레이스 조건은 병렬 워크플로 실행이 공유 리소스를 동시에 변경하려고 시도할 때 발생하며, 이로 인해 예측할 수 없는 결과가 발생합니다.

최대 병렬 실행 수: CodeCatalyst 스페이스당 1,000개

실행 모드 구성

실행 모드를 대기 중, 대체된 또는 병렬로 설정할 수 있습니다. 기본값은 대기 중입니다.

실행 모드를 대기 중이나 대체된에서 병렬로 변경하면 CodeCatalyst는 대기열에서 대기 중인 실행을 취소하고 현재 작업으로 처리 중인 실행을 취소하기 전에 완료할 수 있도록 허용합니다.

실행 모드를 병렬에서 대기 중 또는 대체된으로 변경하면 CodeCatalyst는 현재 실행 중인 모든 병렬 실행을 완료하도록 허용합니다. 실행 모드를 대기 중 또는 대체된으로 변경한 후 시작하는 모든 실행은 새 모드를 사용합니다.

Visual

시각적 편집기를 사용하여 실행 모드 변경

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 오른쪽 상단에서 워크플로 속성을 선택합니다.
7. 고급을 확장하고 실행 모드에서 다음 중 하나를 선택합니다.
 - a. 대기 중 - 참조 [대기 중 실행 모드 정보](#)
 - b. 대체됨 - [대체된 실행 모드 정보](#) 참조
 - c. 병렬 - [병렬 실행 모드 정보](#) 참조
8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 실행 모드 변경

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.

- 다음과 같이 RunMode 속성을 추가합니다.

```
Name: Workflow_6d39
SchemaVersion: "1.0"
RunMode: QUEUED/SUPERSEDED/PARALLEL
```

자세한 내용은 [워크플로 YAML 정의의 최상위 속성](#) 섹션에서 RunMode 속성에 관한 설명을 참조하세요.

- (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
- 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

워크플로 실행 간 파일 캐싱

파일 캐싱이 활성화되면 빌드 및 테스트 작업은 온디스크 파일을 캐시에 저장하고 후속 워크플로 실행 시 해당 캐시에서 복원합니다. 캐싱은 실행 간에 변경되지 않은 종속성을 구축하거나 다운로드하여 발생하는 지연 시간을 줄입니다. CodeCatalyst는 필요한 종속성 중 일부를 포함하는 부분 캐시를 복원하는 데 사용할 수 있는 대체 캐시도 지원합니다. 이렇게 하면 캐시 누락의 지연 시간 영향을 줄일 수 있습니다.

Note

파일 캐싱은 Amazon CodeCatalyst [빌드](#) 및 [테스트](#) 작업과 EC2 [컴퓨팅 유형](#)을 사용하도록 구성된 경우에만 사용할 수 있습니다.

주제

- [파일 캐싱 정보](#)
- [캐시 생성](#)
- [파일 캐싱 제약 조건](#)

파일 캐싱 정보

파일 캐싱을 사용하면 데이터를 FileCaching 속성에서 각각 참조되는 여러 캐시로 구성할 수 있습니다. 각 캐시는 지정된 경로로 지정된 디렉터리를 저장합니다. 지정된 디렉터리는 향후 워크플로 실행 시 복원됩니다. 다음은 cacheKey1 및 cacheKey2라는 여러 캐시를 사용하여 캐싱하기 위한 YAML 조각의 예입니다.

Actions:**BuildMyNpmApp:**

Identifier: aws/build@v1

Inputs:**Sources:**

- WorkflowSource

Configuration:**Steps:**

- Run: npm install
- Run: npm run test

Caching:**FileCaching:****cacheKey1:**

Path: file1.txt

RestoreKeys:

- restoreKey1

cacheKey2:

Path: /root/repository

RestoreKeys:

- restoreKey2
- restoreKey3

Note

CodeCatalyst는 로컬 캐시와 원격 캐시로 구성된 다층 캐싱을 사용합니다. 프로비저닝된 플릿 또는 온디맨드 시스템에서 로컬 캐시에서 캐시 누락이 발생하면 원격 캐시에서 종속성이 복원됩니다. 따라서 일부 작업 실행에서 원격 캐시 다운로드 지연이 발생할 수 있습니다.

CodeCatalyst는 캐시 액세스 제한을 적용하여 한 워크플로의 작업이 다른 워크플로의 캐시를 수정할 수 없도록 합니다. 이렇게 하면 빌드 또는 배포에 영향을 미치는 잘못된 데이터를 푸시할 수 있는 다른 워크플로로부터 각 워크플로를 보호할 수 있습니다. 제한은 모든 워크플로 및 브랜치 페어링에 캐시를 격리하는 캐시 범위에 적용됩니다. 예를 들어, 브랜치 feature-A의 workflow-A는 형제 브랜치 feature-B의 workflow-A와 다른 파일 캐시를 가지고 있습니다.

캐시 누락은 워크플로가 지정된 파일 캐시를 찾아 찾을 수 없을 때 발생합니다. 이는 새 브랜치가 생성되거나 새 캐시가 참조되고 아직 생성되지 않은 경우와 같이 여러 가지 이유로 발생할 수 있습니다. 캐시가 만료될 때도 발생할 수 있으며, 기본적으로 캐시는 마지막으로 사용된 후 14일 후에 발생합니다. 캐시 누락을 완화하고 캐시 적중률을 높이기 위해 CodeCatalyst는 대체 캐시를 지원합니다. 대체 캐시는 대체 캐시이며 캐시의 이전 버전일 수 있는 부분 캐시를 복원할 수 있는 기회를 제공합니다. 캐

시는 먼저 FileCaching에서 속성 이름에 대한 일치 항목을 검색하여 복원되며, 찾을 수 없는 경우 RestoreKeys를 평가합니다. 속성 이름과 모든 RestoreKeys 모두에 캐시가 누락된 경우 캐싱은 최선의 노력이며 보장되지 않으므로 워크플로가 계속 실행됩니다.

캐시 생성

다음 지침에 따라 워크플로에 캐시를 추가할 수 있습니다.

Visual

시각적 편집기를 사용하여 캐시 추가

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 워크플로 다이어그램에서 캐시를 추가하려는 작업을 선택합니다.
8. 구성을 선택합니다.
9. 파일 캐싱 - 선택 사항에서 캐시 추가를 선택하고 다음과 같이 필드에 정보를 입력합니다.

Key(키)

기본 캐시 속성 이름의 이름을 지정합니다. 캐시 속성 이름은 워크플로 내에서 고유해야 합니다. 각 작업은 FileCaching에서 최대 5개의 항목을 가질 수 있습니다.

경로

캐시의 관련 경로를 지정합니다.

키 복원 - 선택 사항

기본 캐시 속성을 찾을 수 없을 때 대체로 사용할 복원 키를 지정합니다. 복원 키 이름은 워크플로 내에서 고유해야 합니다. 각 캐시는 RestoreKeys에서 최대 5개의 항목을 가질 수 있습니다.

10. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.

11. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 캐시 추가

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 워크플로 작업에서 다음과 유사한 코드를 추가합니다.

```
action-name:
  Configuration:
    Steps: ...
  Caching:
    FileCaching:
      key-name:
        Path: file-path
        # # Specify any additional fallback caches
        # RestoreKeys:
        #   - restore-key
```

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

파일 캐싱 제약 조건

다음은 속성 이름 및 RestoreKeys에 대한 제약 조건입니다.

- 이름은 워크플로 내에서 고유해야 합니다.
- 이름은 영숫자 문자(A~Z, a~z, 0~9), 하이픈(-) 및 밑줄(_)로 제한됩니다.
- 이름은 최대 180자까지 가능합니다.
- 각 작업에는 FileCaching에서 최대 5개의 캐시가 있을 수 있습니다.

- 각 캐시는 RestoreKeys에서 최대 5개의 항목을 가질 수 있습니다.

다음은 경로에 대한 제약 조건입니다.

- 별표(*)는 허용되지 않습니다.
- 이름은 최대 255자까지 가능합니다.

워크플로 실행 상태 및 세부 정보 보기

Amazon CodeCatalyst에서 단일 워크플로 실행 또는 여러 실행의 상태 및 세부 정보를 동시에 볼 수 있습니다.

가능한 실행 상태 목록은 [워크플로 실행 상태](#) 섹션을 참조하세요.

Note

워크플로 실행 상태와 다른 워크플로 상태를 볼 수도 있습니다. 자세한 내용은 [워크플로의 상태 보기](#) 섹션을 참조하세요.

워크플로 실행에 대한 자세한 내용은 [워크플로 실행](#) 섹션을 참조하세요.

주제

- [단일 실행의 상태 및 세부 정보 보기](#)
- [프로젝트의 모든 실행 상태 및 세부 정보 보기](#)
- [특정 워크플로의 모든 실행 상태 및 세부 정보 보기](#)
- [워크플로 다이어그램에서 워크플로 실행 보기](#)

단일 실행의 상태 및 세부 정보 보기

단일 워크플로 실행의 상태 및 세부 정보를 보고 성공 여부를 확인하거나, 완료된 시간을 확인하거나, 누가 또는 무엇을 시작했는지 확인할 수 있습니다.

단일 실행의 상태 및 세부 정보 보기

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.

2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 워크플로 이름에서 실행을 선택합니다.
6. 실행 기록의 실행 ID 옆에서 실행을 선택합니다. 예를 들어 Run-95a4d입니다.
7. 실행 이름에서 다음 중 하나를 수행합니다.
 - 워크플로 실행의 작업과 상태를 보여주는 워크플로 다이어그램을 시각적으로 볼 수 있습니다 ([워크플로 실행 상태](#) 참조). 이 보기에는 실행 중에 사용되는 소스 리포지토리와 브랜치도 표시됩니다.

워크플로 다이어그램에서 작업을 선택하여 실행 중에 작업에 의해 생성된 로그, 보고서 및 출력과 같은 세부 정보를 확인합니다. 표시되는 정보는 선택한 작업 유형에 따라 달라집니다. 빌드 또는 배포 로그 보기에 대한 자세한 내용은 [빌드 작업 결과 보기](#) 또는 [배포 로그 보기](#) 섹션을 참조하세요.

- YAML을 사용하여 실행에 사용된 워크플로 정의 파일을 확인합니다.
- 워크플로 실행에서 생성된 아티팩트를 볼 수 있는 아티팩트입니다. 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.
- 워크플로 실행에서 생성된 테스트 보고서 및 기타 유형의 보고서를 볼 수 있는 보고서입니다. 보고서에 대한 자세한 내용은 [품질 보고서 유형](#)의 내용을 참조하세요.
- 워크플로 실행에서 생성된 출력 변수를 볼 수 있는 변수입니다. 변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

Note

실행의 상위 워크플로가 삭제된 경우 이 사실을 나타내는 메시지가 실행 세부 정보 페이지 상단에 나타납니다.

프로젝트의 모든 실행 상태 및 세부 정보 보기

프로젝트 내 모든 워크플로 실행의 상태와 세부 정보를 확인하여 프로젝트에서 진행 중인 워크플로 활동의 양을 파악하고 워크플로의 전반적인 건강 상태를 알아보고 싶을 수 있습니다.

프로젝트의 모든 실행 상태 및 세부 정보 보기

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로에서 실행을 선택합니다.

프로젝트의 모든 리포지토리에 걸쳐 모든 브랜치의 모든 워크플로에 대한 모든 실행이 표시됩니다.

페이지에 다음 열이 포함됩니다.

- 실행 ID - 실행의 고유 식별자입니다. 실행 ID 링크를 선택하여 실행에 대한 자세한 정보를 확인합니다.
- 상태 - 워크플로 실행의 처리 상태입니다. 실행 상태에 대한 자세한 내용은 [워크플로 실행 상태](#) 섹션을 참조하세요.
- 트리거 - 워크플로 실행을 시작한 사람, 커밋, 풀 요청(PR) 또는 일정입니다. 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 섹션을 참조하세요.
- 워크플로 - 실행이 시작된 워크플로의 이름과 워크플로 정의 파일이 있는 소스 리포지토리 및 브랜치입니다. 이 정보를 보려면 열 너비를 확장해야 할 수 있습니다.

Note

이 열이 사용 불가로 설정된 경우 일반적으로 연결된 워크플로가 삭제되거나 이동되었기 때문입니다.

- 시작 시간 - 워크플로 실행이 시작된 시간입니다.
- 기간 - 워크플로 실행이 처리되는 데 걸린 시간입니다. 매우 길거나 매우 짧은 지속 시간은 문제를 나타낼 수 있습니다.
- 종료 시간 - 워크플로 실행이 종료된 시간입니다.

특정 워크플로의 모든 실행 상태 및 세부 정보 보기

특정 워크플로와 연결된 모든 실행의 상태 및 세부 정보를 확인하여 워크플로 내에서 병목 현상이 발생하는 실행이 있는지 확인하거나 현재 진행 중이거나 완료된 실행을 확인할 수 있습니다.

특정 워크플로의 모든 실행 상태 및 세부 정보 보기

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 워크플로 이름에서 실행을 선택합니다.

선택한 워크플로와 연결된 실행이 나타납니다.

이 페이지는 두 섹션으로 나뉘어 있습니다.

- 활성 실행 - 진행 중인 실행을 표시합니다. 이러한 실행은 진행 중 상태 중 하나입니다.
- 실행 기록 - 완료된 실행(즉, 진행 중이 아님)을 표시합니다.

실행 상태에 대한 자세한 내용은 [워크플로 실행 상태](#) 섹션을 참조하세요.

워크플로 다이어그램에서 워크플로 실행 보기

워크플로를 통해 함께 진행됨에 따라 워크플로의 모든 실행 상태를 볼 수 있습니다. 실행은 워크플로 다이어그램 내에 표시됩니다(목록 보기에는 표시되지 않음). 이를 통해 어떤 실행이 어떤 작업으로 처리되고 어떤 실행이 대기열에서 대기 중인지 시각적으로 표현할 수 있습니다.

워크플로를 통해 함께 진행하면서 여러 실행의 상태 보기

Note

이 절차는 워크플로가 대기 중, 대체된 실행 모드를 사용하는 경우에만 적용됩니다. 자세한 내용은 [실행의 대기열 동작 구성](#) 섹션을 참조하세요.

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.

Note

실행 페이지가 아닌 워크플로 페이지를 보고 있는지 확인합니다.

5. 왼쪽 상단의 최신 상태 탭을 선택합니다.

워크플로 다이어그램이 나타납니다.

6. 워크플로 다이어그램을 검토합니다. 다이어그램은 워크플로 내에서 현재 진행 중인 모든 실행과 완료된 최신 실행을 보여줍니다. 보다 구체적으로는 다음과 같습니다.

- 소스 앞에 맨 위에 나타나는 실행은 대기열에 있고 시작을 기다립니다.
- 작업 사이에 나타나는 실행은 대기 중 상태이며 다음 작업에서 처리되기를 기다립니다.
- 작업 내에 나타나는 실행은 1. 현재 작업에서 처리 중, 2. 작업에서 처리 완료 또는 3. 작업에서 처리되지 않았습니다(일반적으로 이전 작업이 실패했기 때문).

워크플로 작업 구성

작업은 워크플로의 기본 구성 요소이며 워크플로 실행 중에 수행할 논리적 작업 단위 또는 태스크를 정의합니다. 일반적으로 워크플로에는 구성 방식에 따라 순차적으로 또는 병렬로 실행되는 여러 작업이 포함됩니다.

주제

- [작업 유형](#)
- [워크플로에 작업 추가](#)
- [워크플로에서 작업 제거](#)
- [사용자 지정 작업 개발](#)
- [작업을 작업 그룹으로 그룹화](#)
- [작업 순서 지정](#)
- [작업 간 아티팩트 및 파일 공유](#)
- [사용할 작업 버전 지정](#)
- [사용 가능한 작업 버전 나열](#)
- [작업의 소스 코드 보기](#)
- [GitHub Actions와 통합](#)

작업 유형

Amazon CodeCatalyst 워크플로 내에서 다음 유형의 작업을 사용할 수 있습니다.

작업 유형

- [CodeCatalyst 작업](#)
- [CodeCatalyst Labs 작업](#)
- [GitHub Actions](#)
- [타사 작업](#)

CodeCatalyst 작업

CodeCatalyst 작업은 CodeCatalyst 개발 팀이 작성, 유지 및 완벽하게 지원하는 작업입니다.

애플리케이션 구축, 테스트 및 배포는 물론 AWS Lambda 함수 호출과 같은 기타 작업을 수행하기 위한 CodeCatalyst 작업이 있습니다.

다음 CodeCatalyst 작업을 사용할 수 있습니다.

- 빌드

이 작업은 아티팩트를 빌드하고 Docker 컨테이너에서 단위 테스트를 실행합니다. 자세한 내용은 [빌드 작업 추가](#) 섹션을 참조하세요.

- 테스트

이 작업은 애플리케이션 또는 아티팩트에 대해 통합 및 시스템 테스트를 실행합니다. 자세한 내용은 [테스트 작업 추가](#) 섹션을 참조하세요.

- Amazon S3 게시

이 작업은 애플리케이션 아티팩트를 Amazon S3 버킷에 복사합니다. 자세한 내용은 [워크플로를 사용하여 Amazon S3에 파일 게시](#) 단원을 참조하십시오.

- AWS CDK 부트스트랩

이 작업은 AWS CDK 앱을 배포하는 데 AWS CDK 필요한 리소스를 프로비저닝합니다. 자세한 내용은 [워크플로를 사용하여 AWS CDK 앱 부트스트래핑](#) 단원을 참조하십시오.

- AWS CDK 배포

이 작업은 AWS Cloud Development Kit (AWS CDK) 앱을 합성하고 배포합니다. 자세한 내용은 [워크플로를 사용하여 AWS CDK 앱 배포](#) 단원을 참조하십시오.

- AWS Lambda 간접 호출

이 작업은 AWS Lambda 함수를 호출합니다. 자세한 내용은 [워크플로를 사용하여 Lambda 함수 호출](#) 단원을 참조하십시오.

- GitHub Actions

이 작업은 CodeCatalyst 워크플로 내에서 GitHub Actions를 실행할 수 있는 CodeCatalyst 작업입니다. 자세한 내용은 [워크플로를 사용하여 Lambda 함수 호출](#) 단원을 참조하십시오.

- CloudFormation 스택 배포

이 작업은 CloudFormation 스택을 배포합니다. 자세한 내용은 [CloudFormation 스택 배포](#) 단원을 참조하십시오.

- Amazon EKS에 배포

이 작업은 Amazon ECS 작업 정의를 등록하고 Amazon ECS 서비스에 배포합니다. 자세한 내용은 [워크플로를 사용하여 Amazon ECS에 배포](#) 섹션을 참조하세요.

- Kubernetes 클러스터에 배포

이 작업은 애플리케이션을 Kubernetes 클러스터에 배포합니다. 자세한 내용은 [워크플로를 사용하여 Amazon EKS에 배포](#) 섹션을 참조하세요.

- Amazon ECS 태스크 정의 렌더링

이 작업은 컨테이너 이미지 URI를 Amazon ECS 작업 정의 JSON 파일에 삽입하여 새 태스크 정의 파일을 생성합니다. 자세한 내용은 [Amazon ECS 작업 정의 수정](#) 섹션을 참조하세요.

CodeCatalyst 작업에 대한 설명서는 이 안내서와 각 작업의 읽어보기에서 확인할 수 있습니다.

사용 가능한 CodeCatalyst 작업과 워크플로에 추가하는 방법에 대한 자세한 내용은 [워크플로에 작업 추가](#) 섹션을 참조하세요.

CodeCatalyst Labs 작업

CodeCatalyst Labs 작업은 실험 애플리케이션의 입증 근거인 Amazon CodeCatalyst Labs의 일부인 작업입니다. CodeCatalyst Labs 작업은 AWS 서비스와의 통합을 보여주기 위해 개발되었습니다.

다음 CodeCatalyst Labs 작업을 사용할 수 있습니다.

- AWS Amplify 호스팅에 배포

이 작업은 Amplify Hosting에 애플리케이션을 배포합니다.

- 예 배포 AWS App Runner

이 작업은 소스 이미지 리포지토리의 최신 이미지를 App Runner에 배포합니다.

- Amazon CloudFront 및 Amazon S3에 배포

이 작업은 애플리케이션을 CloudFront 및 Amazon S3에 배포합니다.

- 를 사용하여 배포 AWS SAM

이 작업은 AWS Serverless Application Model (AWS SAM)를 사용하여 서버리스 애플리케이션을 배포합니다.

- Amazon CloudFront 캐시 무효화

이 작업은 지정된 경로 집합에 대한 CloudFront 캐시를 무효화합니다.

- 발신 웹훅

이 작업을 통해 사용자는 HTTPS 요청을 사용하여 워크플로 내의 메시지를 임의의 웹 서버로 보낼 수 있습니다.

- 예 게시 AWS CodeArtifact

이 작업은 패키지를 CodeArtifact 리포지토리에 게시합니다.

- Amazon SNS에 게시

이 작업을 통해 사용자는 주제를 생성하거나, 주제에 게시하거나, 주제를 구독하여 Amazon SNS와 통합할 수 있습니다.

- Amazon ECR에 게시

이 작업은 Docker 이미지를 빌드하고 Amazon Elastic Container Registry(Amazon ECR) 리포지토리에 게시합니다.

- Amazon CodeGuru Security로 스캔

이 작업은 구성된 코드 경로의 zip 아카이브를 생성하고 CodeGuru Security를 사용하여 코드 스캔을 실행합니다.

- Terraform Community Edition

이 작업은 Terraform Community Edition plan 및 apply 작업을 실행합니다.

CodeCatalyst Labs 작업에 대한 설명서는 각 작업의 읽어보기에서 사용할 수 있습니다.

워크플로에 CodeCatalyst Labs 작업을 추가하고 해당 읽어보기를 보는 방법에 대한 자세한 내용은 [워크플로에 작업 추가](#) 섹션을 참조하세요.

GitHub Actions

GitHub 작업은 GitHub 워크플로와 함께 사용하도록 개발되었다는 점을 제외하면 [CodeCatalyst 작업](#)과 매우 유사합니다. GitHub Actions에 대한 자세한 내용은 [GitHub Actions](#) 설명서를 참조하세요.

CodeCatalyst 워크플로에서 기본 CodeCatalyst 작업과 함께 GitHub Actions를 사용할 수 있습니다.

편의를 위해 CodeCatalyst 콘솔은 인기 있는 여러 GitHub Actions에 대한 액세스를 제공합니다. [GitHub Marketplace](#)에 나열된 모든 GitHub 작업을 사용할 수도 있습니다(몇 가지 제한 사항이 적용됨).

GitHub Actions에 대한 설명서는 각 작업의 readme에서 사용할 수 있습니다.

자세한 내용은 [GitHub Actions와 통합](#) 섹션을 참조하세요.

타사 작업

타사 작업은 타사 공급업체에서 작성하고 CodeCatalyst 콘솔에서 사용할 수 있는 작업입니다. 타사 작업의 예로는 각각 Mend 및 Sonar에서 작성한 Mend SCA 및 SonarCloud Scan 작업이 있습니다.

타사 작업에 대한 설명서는 각 작업의 readme에서 확인할 수 있습니다. 타사 공급업체에서 추가 문서를 제공할 수도 있습니다.

워크플로에 타사 작업을 추가하고 해당 읽어보기를 보는 방법에 대한 자세한 내용은 [워크플로에 작업 추가](#) 섹션을 참조하세요.

워크플로에 작업 추가

다음 지침에 따라 워크플로에 작업을 추가한 다음 구성합니다.

작업 추가 및 구성

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.

6. 왼쪽 상단에 + 작업을 선택하면 작업 카탈로그가 나타납니다.
7. 드롭다운 목록에서 다음 중 하나를 선택합니다.
 - Amazon CodeCatalyst를 선택하여 [CodeCatalyst](#), [CodeCatalyst Labs](#) 또는 [타사](#) 작업을 봅니다.
 - CodeCatalyst 작업에는 AWS레이블이 있습니다.
 - CodeCatalyst Labs 작업에는 by CodeCatalyst Labs 레이블이 있습니다.
 - 타사 작업에는 by **vendor** 레이블이 있으며, 여기서 **vendor**는 타사 공급업체의 이름입니다.
 - GitHub를 선택하여 [큐레이션된 GitHub Actions 목록](#)을 봅니다.
8. 작업 카탈로그에서 작업을 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로에 작업을 추가합니다.
 - 작업의 이름을 선택하여 읽어보기를 봅니다.
9. 작업을 구성합니다. 시각적 편집기를 사용하려면 비주얼을 선택하고, YAML 편집기를 사용하려면 YAML을 선택합니다. 자세한 지침은 다음 링크를 참조하세요.

[CodeCatalyst 작업](#) 추가에 대한 지침은 다음을 참조하세요.

- [빌드 작업 추가](#)
- [테스트 작업 추가](#)
- ['Amazon ECS에 배포' 작업 추가](#)
- ['Kubernetes 클러스터에 배포' 작업 추가](#)
- ['스 CloudFormation 템플릿 배포' 작업 추가](#)
- ['AWS CDK 배포' 작업 추가](#)
- ['AWS CDK bootstrap' 작업 추가](#)
- ['Amazon S3 게시' 작업 추가](#)
- ['AWS Lambda 호출' 작업 추가](#)
- ['Amazon ECS 작업 정의 렌더링' 작업 추가](#)

[CodeCatalyst Labs 작업](#) 추가에 대한 지침은 다음을 참조하세요.

- 작업의 readme입니다. 작업 카탈로그에서 작업의 이름을 선택하면 조사 결과를 확인할 수 있습니다.

- [GitHub Actions와 통합](#)

[타사 작업](#) 추가에 대한 지침은 다음을 참조하세요.

- 작업의 readme입니다. 작업 카탈로그에서 작업의 이름을 선택하면 조사 결과를 확인할 수 있습니다.

10. (선택 사항) YAML 코드가 유효한지 확인하려면 검증을 선택합니다.

11. 커밋을 선택하여 변경 사항을 커밋합니다.

워크플로에서 작업 제거

다음 지침에 따라 워크플로에서 작업을 제거합니다.

Visual

시각적 편집기를 사용하여 작업 제거

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 워크플로 다이어그램에서 제거하려는 작업에서 세로 줄임표 아이콘을 선택하고 제거를 선택합니다.
8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 작업 제거

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.

3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 제거하려는 작업이 포함된 YAML 섹션을 조사합니다.

섹션을 선택하고 키보드에서 delete 키를 누릅니다.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

사용자 지정 작업 개발

CodeCatalyst 작업 개발 키트(ADK)를 사용하여 워크플로에서 사용할 사용자 지정 작업을 개발할 수 있습니다. 그런 다음 해당 작업을 CodeCatalyst 작업 카탈로그에 게시하여 다른 CodeCatalyst 사용자가 워크플로에서 보고 사용할 수 있도록 할 수 있습니다.

작업 개발, 테스트 및 게시(고급 태스크)

1. 작업 개발에 필요한 도구와 패키지를 설치합니다.
2. 작업 코드를 저장할 CodeCatalyst 리포지토리를 만듭니다.
3. 작업을 초기화합니다. 이렇게 하면 자체 코드로 업데이트할 수 있는 작업 정의 파일(action.yml)을 포함하여 작업에 필요한 소스 파일이 정리됩니다.
4. 작업 코드를 부트스트랩하여 작업 프로젝트를 빌드, 테스트 및 릴리스하는 데 필요한 도구와 라이브러리를 가져옵니다.
5. 로컬 컴퓨터에서 작업을 빌드하고 변경 사항을 CodeCatalyst 리포지토리로 푸시합니다.
6. 로컬에서 단위 테스트를 통해 작업을 테스트하고, ADK에서 생성된 워크플로를 CodeCatalyst에서 실행합니다.
7. CodeCatalyst 콘솔에서 게시 버튼을 선택하여 CodeCatalyst 작업 카탈로그에 작업을 게시합니다.

자세한 단계는 [Amazon CodeCatalyst 작업 개발 키트 개발자 안내서](#)를 참조하세요.

작업을 작업 그룹으로 그룹화

작업 그룹에는 하나 이상의 작업이 포함됩니다. 작업을 작업 그룹으로 그룹화하면 워크플로를 체계적으로 관리할 수 있고, 서로 다른 그룹 간의 종속성을 구성할 수도 있습니다.

Note

다른 작업 그룹이나 작업 내에 작업 그룹을 중첩할 수 없습니다.

주제

- [작업 그룹 정의](#)
- [예시: 두 작업 그룹 정의](#)

작업 그룹 정의

다음 지침에 따라 CodeCatalyst 작업 그룹을 정의합니다.

Visual

사용할 수 없습니다. YAML을 선택하여 YAML 지침을 봅니다.

YAML

그룹 정의

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. Actions에서 다음과 유사한 코드를 추가합니다.

```
Actions:
```

```

action-group-name:
  Actions:
    action-1:
      Identifier: aws/build@v1
      Configuration:
        ...
    action-2:
      Identifier: aws/build@v1
      Configuration:
        ...

```

다른 예시는 [예시: 두 작업 그룹 정의](#)를 참조하세요. 자세한 내용은 [워크플로 YAML 정의의 작업](#)에서 action-group-name 속성에 관한 설명을 참조하세요.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

예시: 두 작업 그룹 정의

다음 예시에서는 두 개의 Amazon CodeCatalyst 작업 그룹인 BuildAndTest 및 Deploy를 정의하는 방법을 보여줍니다. BuildAndTest 그룹에는 두 개의 작업(Build 및 Test)이 포함되며 Deploy 그룹에는 두 개의 작업(DeployCloudFormationStack 및 DeployToECS)도 포함됩니다.

```

Actions:
  BuildAndTest: # Action group 1
    Actions:
      Build:
        Identifier: aws/build@v1
        Configuration:
          ...
      Test:
        Identifier: aws/managed-test@v1
        Configuration:
          ...
  Deploy: #Action group 2
    Actions:
      DeployCloudFormationStack:
        Identifier: aws/cfn-deploy@v1
        Configuration:
          ...
      DeployToECS:
        Identifier: aws/ecs-deploy@v1
        Configuration:
          ...

```

...

작업 순서 지정

기본적으로 워크플로에 작업을 추가하면 [시각적 편집기](#)에서 나란히 추가됩니다. 즉, 워크플로 실행을 시작할 때 작업이 병렬로 실행됩니다. 작업이 순차적으로 실행되고 시각적 편집기에서 세로로 표시되도록 하려면 작업 간에 종속성을 설정해야 합니다. 예를 들어, 테스트 작업이 빌드 작업 이후에 실행되도록 TestA 작업이 Build 작업에 종속되도록 설정할 수 있습니다.

작업과 작업 그룹 간에 종속성을 설정할 수 있습니다. 하나의 작업이 시작하기 위해 다른 여러 작업에 종속되도록 일대다 종속성을 구성할 수도 있습니다. [작업 간 종속성 설정](#)의 가이드라인을 참조하여 종속성 설정이 워크플로의 YAML 구문을 준수하는지 확인하세요.

주제

- [작업 간의 종속성을 구성하는 방법의 예시](#)
- [작업 간 종속성 설정](#)

작업 간의 종속성을 구성하는 방법의 예시

다음 예시에서는 워크플로 정의 파일의 작업과 그룹 간에 종속성을 구성하는 방법을 보여줍니다.

주제

- [예시: 단순 종속성 구성](#)
- [예시: 작업에 종속되도록 작업 그룹 구성](#)
- [예시: 다른 작업 그룹에 종속되도록 작업 그룹 구성](#)
- [예시: 여러 작업에 종속되도록 작업 그룹 구성](#)

예시: 단순 종속성 구성

다음 예시에서는 DependsOn 속성을 사용하여 Build 작업에 따라 Test 작업을 구성하는 방법을 보여줍니다.

```
Actions:
  Build:
    Identifier: aws/build@v1
    Configuration:
      ...
```

```
Test:
  DependsOn:
    - Build
  Identifier: aws/managed-test@v1
  Configuration:
    ...
```

예시: 작업에 종속되도록 작업 그룹 구성

다음 예시에서는 FirstAction 작업에 따라 DeployGroup 작업 그룹을 구성하는 방법을 보여줍니다. 작업과 작업 그룹은 동일한 수준에 있습니다.

```
Actions:
  FirstAction: #An action outside an action group
    Identifier: aws/github-actions-runner@v1
    Configuration:
      ...
  DeployGroup: #An action group containing two actions
    DependsOn:
      - FirstAction
    Actions:
      DeployAction1:
        ...
      DeployAction2:
        ...
```

예시: 다른 작업 그룹에 종속되도록 작업 그룹 구성

다음 예시에서는 BuildAndTestGroup 작업 그룹에 따라 DeployGroup 작업 그룹을 구성하는 방법을 보여줍니다. 작업 그룹은 동일한 수준에 있습니다.

```
Actions:
  BuildAndTestGroup: # Action group 1
    Actions:
      BuildAction:
        ...
      TestAction:
        ...
  DeployGroup: #Action group 2
    DependsOn:
      - BuildAndTestGroup
    Actions:
```

```

DeployAction1:
...
DeployAction2:
...

```

예시: 여러 작업에 종속되도록 작업 그룹 구성

다음 예시에서는 FirstAction 작업, SecondAction 작업 및 BuildAndTestGroup 작업 그룹에 따라 DeployGroup 작업 그룹을 구성하는 방법을 보여줍니다. DeployGroup은 FirstAction, SecondAction 및 BuildAndTestGroup과 동일한 수준에 있습니다.

```

Actions:
  FirstAction: #An action outside an action group
  ...
  SecondAction: #Another action
  ...
  BuildAndTestGroup: #Action group 1
    Actions:
      Build:
      ...
      Test:
      ...
  DeployGroup: #Action group 2
    DependsOn:
      - FirstAction
      - SecondAction
      - BuildAndTestGroup
    Actions:
      DeployAction1:
      ...
      DeployAction2:
      ...

```

작업 간 종속성 설정

다음 지침에 따라 워크플로의 작업 간 종속성을 설정합니다.

종속성을 구성할 때는 다음 지침을 따르세요.

- 작업이 그룹 내에 있는 경우 해당 작업은 동일한 그룹 내의 다른 작업에만 의존할 수 있습니다.
- 작업 및 작업 그룹은 YAML 계층 구조에서 동일한 수준의 다른 작업 및 작업 그룹에 의존할 수 있지만 다른 수준에서는 의존할 수 없습니다.

Visual

시각적 편집기를 사용하여 종속성 설정

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 워크플로 다이어그램에서 다른 작업에 종속될 작업을 선택합니다.
8. 입력 탭을 선택합니다.
9. 종속 대상 - 선택 사항에서 다음을 수행합니다.

이 작업을 실행하기 위해 성공적으로 실행해야 하는 작업, 작업 그룹 또는 게이트를 지정합니다.

'depends on' 함수에 대한 자세한 내용은 [작업 순서 지정](#) 섹션을 참조하세요.

10. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
11. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 종속성 설정

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 다른 항목에 종속되는 작업에서 다음과 유사한 코드를 추가합니다


```
action-name:
  DependsOn:
    - action-1
```

더 많은 예시는 [작업 간의 종속성을 구성하는 방법의 예시](#)를 참조합니다. 일반 지침은 [작업 간 종속성 설정](#) 섹션을 참조하세요. 자세한 내용은 작업에 대한 [워크플로 YAML 정의](#)의 DependsOn 속성 설명을 참조하세요.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

작업 간 아티팩트 및 파일 공유

아티팩트는 워크플로 작업의 출력이며 일반적으로 폴더 또는 파일 아카이브로 구성됩니다. 아티팩트를 사용하면 작업 간에 파일과 정보를 공유할 수 있기 때문에 아티팩트가 중요합니다.

예를 들어, sam-template.yml 파일을 생성 하는 빌드 작업이 있을 수 있지만 배포 작업에서 파일을 사용하려는 경우가 있습니다. 이 시나리오에서는 아티팩트를 사용하여 빌드 작업이 배포 작업과 sam-template.yml 파일을 공유하도록 허용합니다. 코드는 다음과 같을 것입니다.

```
Actions:
  BuildAction:
    Identifier: aws/build@v1
    Steps:
      - Run: sam package --output-template-file sam-template.yml
    Outputs:
      Artifacts:
        - Name: MYARTIFACT
          Files:
            - sam-template.yml
  DeployAction:
    Identifier: aws/cfn-deploy@v1
    Inputs:
      Artifacts:
        - MYARTIFACT
    Configuration:
      template: sam-template.yml
```

이전 코드에서 빌드 작업(BuildAction)은 sam-template.yml 파일을 생성한 다음 라는 출력 아티팩트 MYARTIFACT에 추가합니다. 후속 배포 작업(DeployAction)은 MYARTIFACT를 입력으로 지정 하여 sam-template.yml 파일에 대한 액세스 권한을 부여합니다.

주제

- [아티팩트를 출력 및 입력으로 지정하지 않고 공유할 수 있나요?](#)
- [워크플로 간에 아티팩트를 공유할 수 있나요?](#)
- [아티팩트 예시](#)
- [출력 아티팩트 정의](#)
- [입력 아티팩트 정의](#)
- [아티팩트의 파일 참조](#)
- [아티팩트 다운로드](#)

아티팩트를 출력 및 입력으로 지정하지 않고 공유할 수 있나요?

예, 작업의 YAML 코드의 Outputs 및 Inputs 섹션에서 아티팩트를 지정하지 않고 작업 간에 아티팩트를 공유할 수 있습니다. 이렇게 하려면 컴퓨팅 공유를 켜야 합니다. 컴퓨팅 공유 및 아티팩트가 켜져 있을 때 아티팩트를 지정하는 방법에 대한 자세한 내용은 [작업 간에 컴퓨팅 공유](#) 섹션을 참조하세요.

Note

컴퓨팅 공유 기능을 사용하면 Outputs 및 Inputs 섹션이 필요 없어 워크플로의 YAML 코드를 간소화할 수 있지만, 이 기능을 켜기 전에 알아두어야 할 제한 사항이 있습니다. 이러한 제한에 대한 자세한 내용은 [컴퓨팅 공유 고려 사항](#) 섹션을 참조하세요.

워크플로 간에 아티팩트를 공유할 수 있나요?

아니요. 서로 다른 워크플로 간에 아티팩트를 공유할 수 없지만 동일한 워크플로 내의 작업 간에 아티팩트를 공유할 수 있습니다.

아티팩트 예시

다음 예시에서는 Amazon CodeCatalyst 워크플로 정의 파일에서 아티팩트를 출력, 입력 및 참조하는 방법을 보여줍니다.

주제

- [예시: 아티팩트 출력](#)
- [예시: 다른 작업에서 생성된 아티팩트 입력](#)
- [예시: 여러 아티팩트에서 파일 참조](#)
- [예시: 단일 아티팩트에서 파일 참조](#)
- [예시: WorkflowSource가 있을 때 아티팩트의 파일 참조](#)
- [예시: 작업 그룹이 있을 때 아티팩트의 파일 참조](#)

예시: 아티팩트 출력

다음 예시에서는 두 개의 .jar 파일이 포함된 아티팩트를 출력하는 방법을 보여줍니다.

```
Actions:
  Build:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ARTIFACT1
          Files:
            - build-output/file1.jar
            - build-output/file2.jar
```

예시: 다른 작업에서 생성된 아티팩트 입력

다음 예시에서는 BuildActionA의 ARTIFACT4 아티팩트를 출력하고 BuildActionB에 입력하는 방법을 보여줍니다.

```
Actions:
  BuildActionA:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ARTIFACT4
          Files:
            - build-output/file1.jar
            - build-output/file2.jar
  BuildActionB:
    Identifier: aws/build@v1
```

```

Inputs:
  Artifacts:
    - ARTIFACT4
Configuration:

```

예시: 여러 아티팩트에서 파일 참조

다음 예시에서는 BuildActionC의 ART5 및 ART6라는 두 개의 아티팩트를 출력한 다음 BuildActionD(Steps 아래)의 file5.txt(ART5 아티팩트) 및 file6.txt(ART6 아티팩트)라는 두 개의 파일을 참조하는 방법을 보여줍니다.

Note

파일 참조에 대한 자세한 내용은 [아티팩트의 파일 참조](#) 섹션을 참조하세요.

Note

예시는 사용 중인 \$CATALYST_SOURCE_DIR_ART5 접두사를 보여주지만 이를 생략할 수 있습니다. 이는 ART5가 기본 입력이기 때문입니다. 기본 입력에 대한 자세한 내용은 [아티팩트의 파일 참조](#) 섹션을 참조하세요.

```

Actions:
  BuildActionC:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ART5
          Files:
            - build-output/file5.txt
        - Name: ART6
          Files:
            - build-output/file6.txt
  BuildActionD:
    Identifier: aws/build@v1
    Inputs:
      Artifacts:
        - ART5
        - ART6

```

Configuration:

Steps:

- run: cd \$CATALYST_SOURCE_DIR_ART5/build-output && cat file5.txt
- run: cd \$CATALYST_SOURCE_DIR_ART6/build-output && cat file6.txt

예시: 단일 아티팩트에서 파일 참조

다음 예시에서는 BuildActionE의 ART7 아티팩트를 출력한 다음 BuildActionF(Steps 아래)의 file7.txt(ART7 아티팩트) 파일을 참조하는 방법을 보여줍니다.

[예시: 여러 아티팩트에서 파일 참조](#)에서와 같이 참조에 build-output 디렉터리 앞에 \$CATALYST_SOURCE_DIR_*artifact-name* 접두사가 필요하지 않은 것을 확인할 수 있습니다. 이는 Inputs에 지정된 항목이 하나뿐이기 때문입니다.

 Note

파일 참조에 대한 자세한 내용은 [아티팩트의 파일 참조](#) 섹션을 참조하세요.

Actions:

BuildActionE:

Identifier: aws/build@v1

Outputs:

Artifacts:

- Name: ART7

Files:

- build-output/file7.txt

BuildActionF:

Identifier: aws/build@v1

Inputs:

Artifacts:

- ART7

Configuration:

Steps:

- run: cd build-output && cat file7.txt

예시: WorkflowSource가 있을 때 아티팩트의 파일 참조

다음 예시에서는 BuildActionG의 ART8 아티팩트를 출력한 다음 BuildActionH(Steps 아래)의 file8.txt(ART8 아티팩트) 파일을 참조하는 방법을 보여줍니다.

참조에 \$CATALYST_SOURCE_DIR_*artifact-name* 접두사가 [예시: 여러 아티팩트에서 파일 참조](#)에서와 같이 어떻게 필요한지 확인합니다. 이는 Inputs(소스 및 아티팩트)에 여러 항목이 지정되어 있기 때문에 파일을 찾을 위치를 나타내는 접두사가 필요합니다.

Note

파일 참조에 대한 자세한 내용은 [아티팩트의 파일 참조](#) 섹션을 참조하세요.

```
Actions:
  BuildActionG:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ART8
          Files:
            - build-output/file8.txt
  BuildActionH:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
      Artifacts:
        - ART8
    Configuration:
      Steps:
        - run: cd $CATALYST_SOURCE_DIR_ART8/build-output && cat file8.txt
```

예시: 작업 그룹이 있을 때 아티팩트의 파일 참조

다음 예시에서는 ActionGroup1, ActionI의 ART9 아티팩트를 출력한 다음 ActionJ의 file9.txt(ART9 아티팩트) 파일을 참조하는 방법을 보여줍니다.

파일 참조에 대한 자세한 내용은 [아티팩트의 파일 참조](#) 섹션을 참조하세요.

```
Actions:
  ActionGroup1:
    Actions:
      ActionI:
        Identifier: aws/build@v1
        Outputs:
```

```

    Artifacts:
      - Name: ART9
        Files:
          - build-output/file9.yml
  ActionJ:
    Identifier: aws/cfn-deploy@v1
    Inputs:
      Sources:
        - WorkflowSource
      Artifacts:
        - ART9
    Configuration:
      template: /artifacts/ActionGroup1@ActionJ/ART9/build-output/file9.yml

```

출력 아티팩트 정의

다음 지침에 따라 Amazon CodeCatalyst 작업이 출력할 아티팩트를 정의합니다. 그러면 이 아티팩트를 다른 작업에서 사용할 수 있게 됩니다.

Note

모든 작업이 출력 아티팩트를 지원하는 것은 아닙니다. 작업이 이를 지원하는지 확인하려면 다음에 나오는 시각적 편집기 지침을 실행하고 출력 탭에서 작업에 출력 아티팩트 버튼이 포함되어 있는지 확인합니다. 포함되어 있다면 출력 아티팩트가 지원됩니다.

Visual

시각적 편집기를 사용하여 출력 아티팩트 정의

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 워크플로 다이어그램에서 아티팩트를 생성할 작업을 선택합니다.
8. 출력 탭을 선택합니다.

9. 아티팩트에서 아티팩트 추가를 선택합니다.
10. 아티팩트 추가를 선택하고 다음과 같이 필드에 정보를 입력합니다.

빌드 아티팩트 이름

작업에서 생성된 아티팩트의 이름을 지정합니다. 아티팩트 이름은 워크플로 내에서 고유해야 하며 영숫자 문자(a-z, A-Z, 0-9) 및 밑줄(_)로 제한됩니다. 공백, 하이픈(-) 및 특수 문자는 허용되지 않습니다. 출력 아티팩트 이름에서 공백, 하이픈 및 기타 특수 문자를 활성화하는 데 따옴표를 사용할 수 없습니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

빌드로 생성한 파일

CodeCatalyst가 작업으로 출력되는 아티팩트에 포함하는 파일을 지정합니다. 이러한 파일은 실행 시 워크플로 작업에 의해 생성되며 소스 리포지토리에서도 사용할 수 있습니다. 파일 경로는 소스 리포지토리 또는 이전 작업의 아티팩트에 상주할 수 있으며 소스 리포지토리 또는 아티팩트 루트와 관련이 있습니다. glob 패턴을 사용하여 경로를 지정할 수 있습니다. 예시:

- 빌드 위치 또는 소스 리포지토리 위치의 루트에 있는 단일 파일을 지정하려면 `my-file.jar`를 사용합니다..
- 하위 디렉터리에 단일 파일을 지정하려면 `directory/my-file.jar` 또는 `directory/subdirectory/my-file.jar`를 사용합니다.
- 모든 파일을 지정하려면 `"**/*"`를 사용합니다. `**` glob 패턴은 임의의 수의 하위 디렉터리와 일치함을 나타냅니다.
- `directory`라는 디렉터리에 있는 모든 파일 및 디렉터를 지정하려면 `"directory/**/*"`를 사용합니다. `**` glob 패턴은 임의의 수의 하위 디렉터리와 일치함을 나타냅니다.
- `directory`라는 디렉터리의 모든 파일을 지정하되 해당 하위 디렉터리는 지정하지 않으려면 `"directory/*"`를 사용합니다.

Note

파일 경로에 별표(*) 또는 기타 특수 문자가 하나 이상 포함된 경우 경로를 큰따옴표(" ")로 묶습니다. 특수 문자에 대한 자세한 내용은 [구문 지침 및 규칙](#) 섹션을 참조하세요.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

 Note

파일 경로에 접두사를 추가하여 찾을 아티팩트 또는 소스를 나타내야 할 수 있습니다. 자세한 내용은 [소스 리포지토리 파일 참조](#) 및 [아티팩트의 파일 참조](#) 섹션을 참조하세요.

11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 출력 아티팩트 정의

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 워크플로 작업에서 다음과 유사한 코드를 추가합니다.

```
action-name:
  Outputs:
    Artifacts:
      - Name: artifact-name
        Files:
          - file-path-1
          - file-path-2
```

더 많은 예시는 [아티팩트 예시](#)를 참조합니다. 자세한 내용은 작업에 해당하는 [워크플로 YAML 정의](#) 섹션을 참조하세요.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

입력 아티팩트 정의

다른 Amazon CodeCatalyst 작업에서 생성된 아티팩트를 사용하려면 현재 작업의 입력으로 지정해야 합니다. 여러 아티팩트를 입력으로 지정할 수 있습니다. 이는 작업에 따라 다릅니다. 자세한 내용은 작업에 해당하는 [워크플로 YAML 정의](#) 섹션을 참조하세요.

Note

다른 워크플로의 아티팩트를 참조할 수 없습니다.

다음 지침에 따라 다른 작업의 아티팩트를 현재 작업에 대한 입력으로 지정합니다.

사전 조건

시작하기 전에 다른 작업에서 아티팩트가 출력되어 있는지 확인하세요. 자세한 내용은 [출력 아티팩트 정의](#) 섹션을 참조하세요. 아티팩트를 출력하면 다른 작업에서 사용할 수 있습니다.

Visual

아티팩트를 작업에 대한 입력으로 지정(시각적 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 워크플로 다이어그램에서 아티팩트를 입력으로 지정하려는 작업을 선택합니다.
8. 입력을 선택합니다.
9. 아티팩트 - 선택 사항에서 다음을 수행합니다.

이 작업에 대한 입력으로 제공하려는 이전 작업의 아티팩트를 지정합니다. 이러한 아티팩트는 이전 작업에서 출력 아티팩트로 이미 정의되어 있어야 합니다.

입력 아티팩트를 지정하지 않으면 *action-name*/Inputs/Sources 아래에 소스 리포지토리를 하나 이상 지정해야 합니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

Note

아티팩트 - 선택적 드롭다운 목록을 사용할 수 없거나(시각적 편집기) YAML(YAML 편집기)을 검증할 때 오류가 발생하는 경우 작업이 하나의 입력만 지원하기 때문일 수 있습니다. 이 경우 소스 입력을 제거해 보세요.

10. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
11. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

아티팩트를 작업에 대한 입력으로 지정(YAML 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 아티팩트를 입력으로 지정하려는 작업에서 다음과 유사한 코드를 추가합니다.

```
action-name:
  Inputs:
    Artifacts:
      - artifact-name
```

더 많은 예시는 [아티팩트 예시](#)를 참조합니다.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

아티팩트의 파일 참조

아티팩트 내에 파일이 있고 Amazon CodeCatalyst 워크플로 작업 중 하나에서 이 파일을 참조해야 하는 경우 다음 절차를 수행하세요.

Note

또한 [소스 리포지토리 파일 참조](#) 섹션도 참조하세요.

Visual

사용할 수 없습니다. YAML을 선택하여 YAML 지침을 봅니다.

YAML

아티팩트의 파일 참조(YAML 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 파일을 참조하려는 작업에서 다음과 유사한 코드를 추가합니다.

```
Actions:
  My-action:
    Inputs:
      Sources:
        - WorkflowSource
      Artifacts:
        - artifact-name
    Configuration:
      template: artifact-path/path/to/file.yml
```

이전 코드에서 다음을 바꿉니다.

- ***artifact-name***: 아티팩트 이름으로 바꿉니다.
- ***artifact-path***: 다음 테이블의 값으로 바꿉니다.

참조 추가 대상	<i>artifact-path</i> 의 대체 대상
빌드 작업 또는 테스트 작업	<code>\$CATALYST_SOURCE_DIR_ <i>artifact-name</i> /</code>
다른 모든 작업	<code>\$CATALYST_SOURCE_DIR_ <i>artifact-name</i> /</code> or <code>/artifacts/ <i>current-action-name</i> /<i>artifact-name</i> /</code> or 현재 작업이 작업 그룹 내에 있는 경우: <code>/artifacts/ <i>current-action-group@current-action-name</i> /<i>artifact-name</i> /</code>

예시는 [아티팩트 예시](#) 섹션을 참조하세요.

Note

다음과 같은 경우 ***artifact-path***를 생략하고 아티팩트 루트 디렉터리를 기준으로 파일 경로를 지정할 수 있습니다.

- 참조를 포함하려는 작업이 Inputs 아래에 하나의 항목만 포함합니다(예: 입력 아티팩트 하나만 포함되고 소스는 포함되지 않음).
- 참조하려는 파일은 기본 입력에 있습니다. 기본 입력은 WorkflowSource이거나 WorkflowSource가 없는 경우 나열된 첫 번째 입력 아티팩트입니다.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.

9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

아티팩트 다운로드

문제 해결을 위해 Amazon CodeCatalyst 워크플로 작업에서 생성된 아티팩트를 다운로드하여 검사할 수 있습니다. 다운로드할 수 있는 아티팩트에는 다음과 같이 두 가지 유형이 있습니다.

- 소스 아티팩트 - 실행이 시작될 때 존재했던 소스 리포지토리 콘텐츠의 스냅샷이 포함된 아티팩트입니다.
- 워크플로 아티팩트 - 워크플로 구성 파일의 Outputs 속성에 정의된 아티팩트입니다.

워크플로에서 아티팩트 출력 다운로드

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 워크플로 이름에서 실행을 선택합니다.
6. 실행 기록의 실행 ID 옆에서 실행을 선택합니다. 예를 들어 Run-95a4d입니다.
7. 실행 이름에서 아티팩트를 선택합니다.
8. 아티팩트 옆에 있는 다운로드를 선택합니다. 아카이브 파일이 다운로드됩니다. 파일 이름은 7개의 무작위 문자로 구성됩니다.
9. 원하는 아카이브 추출 유틸리티를 사용하여 아카이브를 추출합니다.

사용할 작업 버전 지정

기본적으로 워크플로우에 작업을 추가하면 Amazon CodeCatalyst는 해당 형식을 사용하여 워크플로우 정의 파일에 정식 버전을 추가합니다.

vmajor.minor.patch

예제:

```
My-Build-Action:
  Identifier: aws/build@v1.0.0
```

워크플로에서 항상 최신 마이너 또는 패치 버전의 작업을 사용하도록 Identifier 속성에서 정식 버전을 단축할 수 있습니다.

예를 들어, 다음을 지정해야 합니다.

```
My-CloudFormation-Action:
  Identifier: aws/cfn-deploy@v1.0
```

...그리고 최신 패치 버전은 이며 1.0.4, 그러면 작업은 1.0.4 버전을 사용합니다. 이후 버전 1.0.5가 릴리스되면 작업은 1.0.5 버전을 사용합니다. 마이너 버전 1.1.0이 릴리스되면 작업은 1.0.5 버전을 사용합니다.

버전 지정에 대한 자세한 지침은 다음 주제 중 하나를 참조하세요.

다음 지침에 따라 워크플로에서 사용할 작업 버전을 지정합니다. 최신 메이저 또는 마이너 버전 또는 특정 패치 버전을 지정할 수 있습니다.

작업의 최신 마이너 또는 패치 버전을 사용하는 것이 좋습니다.

Visual

사용할 수 없습니다. YAML을 선택하여 YAML 지침을 봅니다.

YAML

작업의 최신 버전 또는 특정 패치 버전을 사용하도록 워크플로 구성

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 버전을 편집하려는 작업을 찾습니다.
8. 작업의 Identifier 속성을 찾아 버전을 다음 중 하나로 설정합니다.
 - `action-identifier@vmajor` - 이 구문을 사용하면 워크플로가 특정 메이저 버전을 사용하도록 하고 최신 마이너 및 패치 버전을 자동으로 선택할 수 있습니다.

- `action-identifier@vmajor.minor` - 이 구문을 사용하면 워크플로가 특정 마이너 버전을 사용하도록 하고 최신 패치 버전을 자동으로 선택할 수 있습니다.
- `action-identifier@vmajor.minor.patch` - 워크플로가 특정 패치 버전을 사용하도록 하려면 이 구문을 사용합니다.

 Note

사용 가능한 버전이 확실하지 않은 경우 [사용 가능한 작업 버전 나열](#) 섹션을 참조하세요.

 Note

메이저 버전은 생략할 수 없습니다.

9. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
10. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

사용 가능한 작업 버전 나열

다음 지침에 따라 워크플로에서 사용할 수 있는 작업의 버전을 확인하세요.

Visual

사용 가능한 작업 버전 확인

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 확인하려는 버전의 작업을 찾습니다.
 - a. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
 - b. 워크플로의 이름을 선택하거나 새로 워크플로를 생성합니다. 워크플로 생성에 대한 자세한 내용은 [워크플로 생성](#) 섹션을 참조하세요.
 - c. 편집을 선택합니다.
 - d. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.

- e. 드롭다운 목록에서 Amazon CodeCatalyst를 선택하여 CodeCatalyst, CodeCatalyst Labs 및 타사 작업을 보거나 GitHub를 선택하여 선별된 GitHub Actions를 볼 수 있습니다.
- f. 작업을 검색하고 해당 이름을 선택합니다. +(더하기 부호)를 선택합니다.

작업에 대한 세부 정보가 표시됩니다.

4. 작업 세부 정보 대화 상자의 오른쪽 상단에서 버전 드롭다운 목록을 선택하여 사용 가능한 작업의 버전 목록을 확인합니다.

YAML

사용할 수 없습니다. 시각적 편집기 지침을 보려면 '비주얼'을 선택합니다.

작업의 소스 코드 보기

작업의 소스 코드를 보고 위험한 코드, 보안 취약성 또는 기타 결함이 있는지 확인할 수 있습니다.

다음 지침에 따라 [CodeCatalyst](#), [CodeCatalyst Labs](#) 또는 [타사](#) 작업의 소스 코드를 확인합니다.

Note

[GitHub 작업](#)의 소스 코드를 보려면 [GitHub Marketplace](#)의 작업 페이지로 이동합니다. 페이지에는 작업의 소스 코드를 찾을 수 있는 작업의 리포지토리에 대한 링크가 포함되어 있습니다.

Note

[빌드](#), [테스트](#), [GitHub Actions](#) 등의 CodeCatalyst 작업의 소스 코드는 볼 수 없습니다.

Note

AWS 는 GitHub Actions 또는 타사 작업의 작업 코드를 지원하지 않거나 보장하지 않습니다.

작업의 소스 코드 보기

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.

3. 코드를 보려는 작업을 찾습니다.

- a. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
- b. 워크플로의 이름을 선택하거나 새로 워크플로를 생성합니다. 워크플로 생성에 대한 자세한 내용은 [워크플로 생성](#) 섹션을 참조하세요.
- c. 편집을 선택합니다.
- d. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.
- e. 드롭다운 목록에서 Amazon CodeCatalyst를 선택하여 CodeCatalyst, CodeCatalyst Labs 및 타사 작업을 확인합니다.
- f. 작업을 검색하고 해당 이름을 선택합니다. +(더하기 부호)를 선택합니다.

작업에 대한 세부 정보가 표시됩니다.

4. 작업 세부 정보 대화 상자 하단에서 다운로드를 선택합니다.

작업의 소스 코드가 있는 Amazon S3 버킷을 보여주는 페이지가 나타납니다. 자세한 내용은 Amazon Simple Storage Service 사용 안내서의 [Amazon S3란 무엇인가요?](#)를 참조하세요.

5. 코드를 검사하여 품질 및 보안에 대한 기대치를 충족하는지 확인합니다.

GitHub Actions와 통합

GitHub 작업은 GitHub 워크플로와 함께 사용하도록 개발되었다는 점을 제외하면 [CodeCatalyst 작업](#)과 매우 유사합니다. GitHub Actions에 대한 자세한 내용은 [GitHub Actions](#) 설명서를 참조하세요.

CodeCatalyst 워크플로에서 기본 CodeCatalyst 작업과 함께 GitHub Actions를 사용할 수 있습니다.

CodeCatalyst 워크플로에 GitHub 작업을 추가하는 방법은 두 가지가 있습니다.

- CodeCatalyst 콘솔의 큐레이션된 목록에서 GitHub 작업을 선택할 수 있습니다. 몇 가지 인기 있는 GitHub Actions를 사용할 수 있습니다. 자세한 내용은 [큐레이션된 GitHub 작업 추가](#) 섹션을 참조하세요.
- CodeCatalyst 콘솔에서 사용하려는 GitHub Actions를 사용할 수 없는 경우 GitHub Actions 작업을 사용하여 추가할 수 있습니다.

GitHub Actions 작업은 GitHub Actions를 래핑하고 CodeCatalyst 워크플로와 호환되는 CodeCatalyst 작업입니다.

다음은 [Super-Linter](#) GitHub Actions를 래핑하는 GitHub Actions의 예입니다.

```

Actions:
  GitHubAction:
    Identifier: aws/github-actions-runner@v1
    Configuration:
      Steps:
        - name: Lint Code Base
          uses: github/super-linter@v4
          env:
            VALIDATE_ALL_CODEBASE: "true"
            DEFAULT_BRANCH: main

```

이전 코드에서 CodeCatalyst GitHub Actions 작업(aws/github-actions-runner@v1로 식별됨)은 Super-Linter OSS 작업(github/super-linter@v4로 식별됨)을 래핑하여 CodeCatalyst 워크플로에서 작동하도록 합니다.

자세한 내용은 ['GitHub Actions' 작업 추가](#) 섹션을 참조하세요.

이전 예시와 같이 큐레이션된 작업과 큐레이션되지 않은 모든 GitHub Actions는 GitHub Actions(aws/github-actions-runner@v1)으로 래핑해야 합니다. 작업이 제대로 작동하려면 래퍼가 필요합니다.

주제

- [GitHub Actions는 CodeCatalyst 작업과 어떻게 다른니까?](#)
- [GitHub Actions는 워크플로의 다른 CodeCatalyst 작업과 상호 작용할 수 있습니까?](#)
- [어떤 GitHub Actions를 사용할 수 있나요?](#)
- [CodeCatalyst의 GitHub Actions 제한 사항](#)
- [GitHub 작업\(상위 단계\)을 추가하려면 어떻게 해야 하나요?](#)
- [GitHub 작업은 GitHub에서 실행되나요?](#)
- [GitHub 워크플로도 사용할 수 있나요?](#)
- ['GitHub Actions' 작업에 사용되는 런타임 이미지](#)
- [자습서: GitHub 작업을 사용한 린트 코드](#)
- ['GitHub Actions' 작업 추가](#)
- [큐레이션된 GitHub 작업 추가](#)
- [GitHub 출력 파라미터 내보내기](#)
- [GitHub 출력 파라미터 참조](#)

- ['GitHub Actions' 작업 YAML](#)

GitHub Actions는 CodeCatalyst 작업과 어떻게 다른니까?

CodeCatalyst 워크플로 내에서 사용되는 GitHub Actions는 CodeCatalyst 작업 AWS 과 동일한 수준의 액세스 및 CodeCatalyst 기능(예: [환경](#) 및 [문제](#))과의 통합을 제공하지 않습니다. CodeCatalyst

GitHub Actions는 워크플로의 다른 CodeCatalyst 작업과 상호 작용할 수 있습니까?

예. 예를 들어 GitHub Actions는 다른 CodeCatalyst 작업에서 생성된 변수를 입력으로 사용할 수 있으며, 출력 파라미터 및 아티팩트를 CodeCatalyst 작업과 공유할 수도 있습니다. 자세한 내용은 [GitHub 출력 파라미터 내보내기](#) 및 [GitHub 출력 파라미터 참조](#) 섹션을 참조하세요.

어떤 GitHub Actions를 사용할 수 있나요?

CodeCatalyst 콘솔을 통해 사용 가능한 모든 GitHub 작업과 [GitHub Marketplace](#)에서 사용 가능한 모든 GitHub 작업을 사용할 수 있습니다. Marketplace에서 GitHub 작업을 사용하기로 결정한 경우 다음 [제한 사항](#)에 유의하세요.

CodeCatalyst의 GitHub Actions 제한 사항

- GitHub Actions는 CodeCatalyst [Lambda 컴퓨팅 유형](#)에 사용할 수 없습니다.
- GitHub Actions는 이전 도구를 포함하는 [2022년 11월](#) 런타임 환경 Docker 이미지에서 실행됩니다. 사용할 이미지 및 도구에 대한 자세한 내용은 [런타임 환경 이미지 지정](#) 섹션을 참조하세요.
- 내부적으로 [github 컨텍스트](#)를 사용하거나 GitHub 관련 리소스를 참조하는 GitHub Actions는 CodeCatalyst에서 지원되지 않습니다. 예를 들어 다음 작업은 CodeCatalyst에서 작동하지 않습니다.
 - GitHub 리소스를 추가, 변경 또는 업데이트하려는 작업입니다. 예를 들어 풀 요청을 업데이트하거나 GitHub에서 문제를 생성하는 작업이 있습니다.
 - <https://github.com/actions> 나열된 거의 모든 작업이 이에 해당합니다.
- [Docker 컨테이너 작업](#)인 GitHub Actions는 작동하지만, 빌드 프로젝트는 기본 Docker 사용자(루트)가 실행해야 합니다. 작업을 user 1001로 실행하지 않습니다. (작성 시점에서 사용자 user 1001은 GitHub 에서 작동하지만 CodeCatalyst에서는 작동하지 않습니다.) 자세한 내용은 [GitHub Actions에 대한 Dockerfile 지원의 사용자](#) 항목을 참조하세요.

CodeCatalyst 콘솔을 통해 사용할 수 있는 GitHub Actions 목록은 [큐레이션된 GitHub 작업 추가](#) 섹션을 참조하세요.

GitHub 작업(상위 단계)을 추가하려면 어떻게 해야 하나요?

GitHub 작업을 CodeCatalyst 워크플로에 추가하는 간략한 단계는 다음과 같습니다.

1. CodeCatalyst 프로젝트에서 워크플로를 생성합니다. 워크플로에서는 애플리케이션을 빌드, 테스트 및 배포하는 방법을 정의합니다. 자세한 내용은 [워크플로 시작하기](#) 섹션을 참조하세요.
2. 워크플로에서 큐레이션된 GitHub Actions를 추가하거나 GitHub Actions 작업을 추가합니다.
3. 다음 중 하나를 수행할 수 있습니다.
 - 큐레이션된 작업을 추가하기로 선택한 경우 구성합니다. 자세한 내용은 [큐레이션된 GitHub 작업 추가](#) 섹션을 참조하세요.
 - 큐레이션되지 않은 작업을 추가하기로 선택한 경우 GitHub Actions 작업 내에서 GitHub Actions의 YAML 코드를 붙여넣습니다. 이 코드는 [GitHub Marketplace](#)에서 선택한 GitHub 작업의 세부 정보 페이지에서 찾을 수 있습니다. CodeCatalyst에서 작동하려면 코드를 약간 수정해야 할 수 있습니다. 자세한 내용은 ['GitHub Actions' 작업 추가](#) 섹션을 참조하세요.
4. (선택 사항) 워크플로 내에 빌드 및 테스트 작업과 같은 다른 작업을 추가합니다. 자세한 내용은 [워크플로를 사용하여 빌드, 테스트 및 배포](#) 섹션을 참조하세요.
5. 트리거를 통해 워크플로를 수동으로 또는 자동으로 시작합니다. 워크플로는 GitHub 작업과 워크플로의 기타 작업을 실행합니다. 자세한 내용은 [워크플로 수동 실행 시작](#) 섹션을 참조하세요.

자세한 단계는 다음을 참조하세요.

- [큐레이션된 GitHub 작업 추가](#).
- ['GitHub Actions' 작업 추가](#).

GitHub 작업은 GitHub에서 실행되나요?

아니요. GitHub 작업은 CodeCatalyst의 [컨타임 환경 이미지](#)를 사용하여 CodeCatalyst에서 실행됩니다.

GitHub 워크플로도 사용할 수 있나요?

아니요.

'GitHub Actions' 작업에 사용되는 런타임 이미지

CodeCatalyst GitHub Actions 작업은 [2022년 11월 이미지](#)에서 실행됩니다. 자세한 내용은 [활성 이미지](#) 섹션을 참조하세요.

자습서: GitHub 작업을 사용한 린트 코드

이 자습서에서는 Amazon CodeCatalyst 워크플로에 [Super-Linter GitHub 작업](#)을 추가합니다. Super-Linter 작업은 코드를 검사하고, 코드에 오류가 있는 영역, 서식 문제, 의심스러운 구문을 찾아 다음, 결과를 CodeCatalyst 콘솔에 출력합니다. 워크플로에 린터를 추가한 후 워크플로를 실행하여 샘플 Node.js 애플리케이션(app.js)을 린트합니다. 그런 다음 보고된 문제를 수정하고 워크플로를 다시 실행하여 수정 사항이 제대로 작동하는지 확인합니다.

Tip

[CloudFormation 템플릿](#)과 같은 YAML 파일을 린트하기 위해 Super-Linter를 사용하는 것이 좋습니다.

주제

- [사전 조건](#)
- [1단계: 소스 리포지토리 생성](#)
- [2단계: app.js 파일 추가](#)
- [3단계: Super-Linter 작업을 실행하는 워크플로 생성](#)
- [4단계: Super-Linter에서 발견한 문제 해결](#)
- [정리](#)

사전 조건

시작하려면 다음이 필요합니다.

- AWS 계정이 연결된 CodeCatalyst 스페이스. 자세한 내용은 [스페이스 생성](#) 섹션을 참조하세요.
- CodeCatalyst 스페이스의 이름이 codecatalyst-linter-project인 빈 프로젝트로. 처음부터 시작 옵션을 선택하여 이 프로젝트를 생성합니다.

자세한 내용은 [Amazon CodeCatalyst에서 빈 프로젝트 생성](#) 섹션을 참조하세요.

1단계: 소스 리포지토리 생성

이 단계에서는 CodeCatalyst에 소스 리포지토리를 생성합니다. 이 리포지토리를 사용하여 이 자습서의 샘플 애플리케이션 소스 파일인 `app.js`를 저장합니다.

소스 리포지토리에 대한 자세한 정보는 [소스 리포지토리 생성](#) 섹션을 참조하세요.

소스 리포지토리를 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. `codecatalyst-linter-project` 프로젝트로 이동합니다.
3. 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
4. 리포지토리 추가를 선택하고 리포지토리 생성을 선택합니다.
5. 리포지토리 이름에 다음과 같이 입력합니다.

```
codecatalyst-linter-source-repository
```

6. 생성(Create)을 선택합니다.

2단계: app.js 파일 추가

이 단계에서는 원본 리포지토리에 `app.js` 파일을 추가합니다. `app.js`에는 린터에서 찾을 수 있는 몇 가지 실수가 있는 함수 코드가 포함되어 있습니다.

`app.js` 파일을 추가하려면

1. CodeCatalyst 콘솔에서 `codecatalyst-linter-project` 프로젝트를 선택합니다.
2. 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
3. 소스 리포지토리 목록에서 `codecatalyst-linter-source-repository` 리포지토리를 선택합니다.
4. 파일에서 파일 생성을 선택합니다.
5. 다음 코드를 텍스트 상자에 입력합니다.

```
// const axios = require('axios')  
// const url = 'http://checkip.amazonaws.com/';
```

```

let response;
/**
 *
 * Event doc: https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html#api-gateway-simple-proxy-for-lambda-input-format
 * @param {Object} event - API Gateway Lambda Proxy Input Format
 *
 * Context doc: https://docs.aws.amazon.com/lambda/latest/dg/nodejs-prog-model-context.html
 * @param {Object} context
 *
 * Return doc: https://docs.aws.amazon.com/apigateway/latest/developerguide/set-up-lambda-proxy-integrations.html
 * @returns {Object} object - API Gateway Lambda Proxy Output Format
 */
exports.lambdaHandler = async (event, context) => {
  try {
    // const ret = await axios(url);
    response = {
      statusCode: 200,
      'body': JSON.stringify({
        message: 'hello world'
        // location: ret.data.trim()
      })
    }
  } catch (err) {
    console.log(err)
    return err
  }

  return response
}

```

6. 파일 이름에 app.js를 입력합니다. 다른 기본 옵션을 유지합니다.
7. 커밋을 선택합니다.

app.js 새 역할이 생성되었습니다.

3단계: Super-Linter 작업을 실행하는 워크플로 생성

이 단계에서는 코드를 소스 리포지토리로 푸시할 때 Super-Linter 작업을 실행하는 워크플로를 생성합니다. 워크플로는 YAML 파일에 정의한 다음 구성 요소로 구성됩니다.

- 트리거 - 이 트리거는 소스 리포지토리에 변경 사항을 푸시할 때 워크플로 실행을 자동으로 시작합니다. 트리거에 대한 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 주제를 참조하세요.
- 'GitHub Actions' 작업 - 트리거 시 GitHub Actions 작업은 Super-Linter 작업을 실행하여 소스 리포지토리의 모든 파일을 검사합니다. 리터에서 문제를 발견하면 워크플로 작업이 실패합니다.

Super-Linter 작업을 실행하는 워크플로를 생성하려면

1. CodeCatalyst 콘솔에서 codecatalyst-linter-project 프로젝트를 선택합니다.
2. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
3. 워크플로 생성을 선택합니다.
4. 소스 리포지토리에서 codecatalyst-linter-source-repository를 선택합니다.
5. 브랜치에서 main을 선택합니다.
6. 생성(Create)을 선택합니다.
7. YAML 샘플 코드를 삭제합니다.
8. 다음 YAML을 추가합니다:

```
Name: codecatalyst-linter-workflow
SchemaVersion: "1.0"
Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  SuperLinterAction:
    Identifier: aws/github-actions-runner@v1
    Configuration:
      Steps:
        github-action-code
```

이 절차의 다음 단계에 설명된 대로 앞의 코드에서 *github-action-code*를 Super-Linter 작업 코드로 바꿉니다.

9. GitHub Marketplace의 [Super-Linter 페이지](#)로 이동합니다.
10. steps:(소문자)에서 코드를 찾아 CodeCatalyst 워크플로의 Steps:(대문자)에 붙여 넣습니다.

다음 코드와 같이 CodeCatalyst 표준에 맞게 GitHub 작업 코드를 조정합니다.

이제 CodeCatalyst 워크플로는 다음과 같습니다.

```

Name: codecatalyst-linter-workflow
SchemaVersion: "1.0"
Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  SuperLinterAction:
    Identifier: aws/github-actions-runner@v1
    Configuration:
      Steps:
        - name: Lint Code Base
          uses: github/super-linter@v4
          env:
            VALIDATE_ALL_CODEBASE: "true"
            DEFAULT_BRANCH: main

```

11. (선택 사항) 커밋하기 전에 YAML 코드가 유효한지 확인하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 codecatalyst-linter-source-repository 리포지토리를 선택하고 커밋을 다시 선택합니다.

이제 워크플로를 생성했습니다. 워크플로 상단에 정의된 트리거로 인해 워크플로 실행이 자동으로 시작됩니다.

진행 중인 워크플로 실행을 보려면

1. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
2. 방금 생성한 codecatalyst-linter-workflow 워크플로를 선택합니다.
3. 워크플로 다이어그램에서 SuperLinterAction을 선택합니다.
4. 작업이 실패할 때까지 기다립니다. 이 실패는 코드에서 linter가 문제를 발견했기 때문에 예상됩니다.
5. CodeCatalyst 콘솔을 열어 두고 [4단계: Super-Linter에서 발견한 문제 해결](#)로 이동합니다.

4단계: Super-Linter에서 발견한 문제 해결

Super-Linter는 app.js 코드와 소스 리포지토리에 포함된 README.md 파일에서 문제를 발견했어야 합니다.

린터에서 발견된 문제를 해결하려면

1. CodeCatalyst 콘솔에서 로그 탭을 선택한 다음 Lint Code Base를 선택합니다.

생성된 슈퍼 린터 작업이 표시되는 로그입니다.

2. 슈퍼 린터 로그에서 90행을 따라 아래로 스크롤하면 문제의 시작을 찾을 수 있습니다. 다음처럼 보일 것입니다.

```
/github/workspace/hello-world/app.js:3:13: Extra semicolon.
/github/workspace/hello-world/app.js:9:92: Trailing spaces not allowed.
/github/workspace/hello-world/app.js:21:7: Unnecessarily quoted property 'body'
found.
/github/workspace/hello-world/app.js:31:1: Expected indentation of 2 spaces but
found 4.
/github/workspace/hello-world/app.js:32:2: Newline required at end of file but not
found.
```

3. 소스 리포지토리에서 app.js 및 README.md를 수정하고 변경 사항을 커밋합니다.

Tip

README.md를 수정하려면 다음과 같이 코드 블록에 markdown을 추가합니다.

```
```markdown
Setup examples:
...
```
```

변경하면 다른 워크플로가 자동으로 실행됩니다. 워크플로가 완료될 때까지 기다립니다. 모든 문제를 해결한 경우 워크플로가 성공해야 합니다.

정리

CodeCatalyst에서 정리하여 환경에서 이 자습서의 트레이스를 제거합니다.

CodeCatalyst에서 정리하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. codecatalyst-linter-source-repository를 삭제합니다.

3. codecatalyst-linter-workflow를 삭제합니다.

이 자습서에서는 일부 코드를 린트하기 위해 CodeCatalyst 워크플로에 Super-Linter GitHub 작업을 추가하는 방법을 배웠습니다.

'GitHub Actions' 작업 추가

GitHub Actions 작업은 GitHub Actions를 래핑하고 CodeCatalyst 워크플로와 호환되는 CodeCatalyst 작업입니다.

자세한 내용은 [GitHub Actions와 통합](#) 섹션을 참조하세요.

워크플로에 GitHub Actions 작업을 추가하려면 다음 단계를 따르세요.

Tip

GitHub Actions 작업을 사용하는 방법을 보여주는 자습서는 [자습서: GitHub 작업을 사용한 린트 코드](#) 섹션을 참조하세요.

Visual

시각적 편집기를 사용하여 'GitHub Actions' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.
8. 드롭다운 목록에서 GitHub를 선택합니다.
9. GitHub Actions 작업을 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.

또는

- GitHub Actions를 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 소스 보기를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
- 10. 입력 및 구성 탭에서 필요에 따라 필드를 작성합니다. 각 필드의 설명은 '[GitHub Actions](#)' 작업 [YAML](#) 섹션을 참조하세요. 이 참조는 YAML 및 시각적 편집기 모두에 나타나는 각 필드(및 해당 YAML 속성 값)에 대한 자세한 정보를 제공합니다.
- 11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
- 12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 'GitHub Actions' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
 2. 프로젝트를 선택합니다.
 3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
 4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
 5. 편집을 선택합니다.
 6. YAML을 선택합니다.
 7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.
 8. 드롭다운 목록에서 GitHub를 선택합니다.
 9. GitHub Actions 작업을 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
- 또는
- GitHub Actions를 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 소스 보기를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
 - 10. 필요에 따라 YAML 코드의 속성을 수정합니다. 사용 가능한 각 속성에 대한 설명은 '[GitHub Actions](#)' 작업 [YAML](#)에서 볼 수 있습니다.

11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

'GitHub Actions' 작업 정의

GitHub Actions 작업은 워크플로 정의 파일 내의 YAML 속성 집합으로 정의됩니다. 이러한 속성에 대한 자세한 내용은 [워크플로 YAML 정의](#)의 '[GitHub Actions' 작업 YAML](#) 섹션을 참조하세요.

큐레이션된 GitHub 작업 추가

큐레이션된 GitHub 작업은 CodeCatalyst 콘솔에서 사용할 수 있는 GitHub 작업이며 CodeCatalyst 워크플로 내에서 GitHub 작업을 사용하는 방법의 예입니다.

큐레이션된 GitHub Actions는 CodeCatalyst에서 만든 [GitHub Actions 작업](#)에 래핑되며 aws/github-actions-runner@v1 식별자로 식별됩니다. 예를 들어 큐레이션된 GitHub 작업인 [TruffleHog OSS](#)는 다음과 같습니다.

```
Actions:
  TruffleHogOSS_e8:
    Identifier: aws/github-actions-runner@v1
    Inputs:
      Sources:
        - WorkflowSource # This specifies that the action requires this Workflow as a
source
    Configuration:
      Steps:
        - uses: truffelsecurity/trufflehog@v3.16.0
          with:
            path: ' ' # Required; description: Repository path
            base: ' ' # Required; description: Start scanning from here (usually main
branch).
            head: ' ' # Optional; description: Scan commits until here (usually dev
branch).
            extra_args: ' ' # Optional; description: Extra args to be passed to the
trufflehog cli.
```

이전 코드에서 CodeCatalyst GitHub Actions 작업(aws/github-actions-runner@v1로 식별됨)은 TruffleHog OSS 작업(truffelsecurity/trufflehog@v3.16.0로 식별됨)을 래핑하여 CodeCatalyst 워크플로에서 작동하도록 합니다.

이 작업을 구성하려면 with:의 빈 문자열을 고유한 값으로 바꿉니다. 예:

Actions:**TruffleHogOSS_e8:**

Identifier: aws/github-actions-runner@v1

Inputs:**Sources:**

- WorkflowSource # This specifies that the action requires this Workflow as a source

Configuration:**Steps:**

- uses: trufflesecurity/trufflehog@v3.16.0
 - with:
 - path: ./
 - base: main # Required; description: Start scanning from here (usually main branch).
 - head: HEAD # Optional; description: Scan commits until here (usually dev branch).
 - extra_args: '--debug --only-verified' # Optional; description: Extra args to be passed to the trufflehog cli.

큐레이션된 GitHub 작업을 워크플로에 추가하려면 다음 절차를 사용합니다. CodeCatalyst 워크플로에서 GitHub Actions를 사용하는 방법에 대한 일반적인 내용은 [GitHub Actions와 통합](#) 섹션을 참조하세요.

Note

큐레이션된 작업 목록 중에 GitHub Actions가 표시되지 않는 경우에도 GitHub Actions 작업을 사용하여 워크플로에 추가할 수 있습니다. 자세한 내용은 '[GitHub Actions' 작업 추가](#) 섹션을 참조하세요.

Visual

시각적 편집기를 사용하여 큐레이션된 GitHub 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.

5. 편집을 선택합니다.
 6. 비주얼을 선택합니다.
 7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.
 8. 드롭다운 목록에서 GitHub를 선택합니다.
 9. GitHub 작업을 찾아보거나 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.

또는

 - GitHub 작업의 이름을 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 소스 보기를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
 10. 입력, 구성 및 출력 탭에서 필요에 따라 필드를 작성합니다. 각 필드의 설명은 '[GitHub Actions](#)' [작업 YAML](#) 섹션을 참조하세요. 이 참조는 GitHub Actions 작업에 사용할 수 있는 각 필드(및 해당 YAML 속성 값)에 대한 자세한 정보를 제공합니다. 이는 YAML 및 시각적 편집기 모두에 표시됩니다.
- 큐레이션된 GitHub 작업에서 사용할 수 있는 구성 옵션에 대한 자세한 내용은 해당 설명서를 참조하세요.
11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
 12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 큐레이션된 GitHub 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.

8. 드롭다운 목록에서 GitHub를 선택합니다.
 9. GitHub 작업을 찾아보거나 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
 - 또는
 - GitHub 작업의 이름을 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 소스 보기를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
 10. 필요에 따라 YAML 코드의 속성을 수정합니다. GitHub Actions 작업에 사용할 수 있는 각 속성에 대한 설명은 '[GitHub Actions' 작업 YAML](#)에 나와 있습니다.
- 큐레이션된 GitHub 작업에서 사용할 수 있는 구성 옵션에 대한 자세한 내용은 해당 설명서를 참조하세요.
11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
 12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

GitHub 출력 파라미터 내보내기

CodeCatalyst 워크플로에서 [GitHub 출력 파라미터](#)를 사용할 수 있습니다.

Note

출력 파라미터의 또 다른 단어는 변수입니다. GitHub는 설명서에서 용어 출력 파라미터를 사용하기 때문에 이 용어도 사용할 것입니다.

다른 CodeCatalyst 워크플로 작업에서 사용할 수 있도록 GitHub 작업에서 GitHub 출력 파라미터를 내보내려면 다음 지침을 사용합니다.

GitHub 출력 파라미터를 내보내려면

1. 워크플로를 열고 편집을 선택합니다. 자세한 내용은 [워크플로 생성](#) 섹션을 참조하세요.
2. 내보내려는 출력 파라미터를 생성하는 GitHub Actions 작업에서 다음과 같은 기본 Variables 속성이 있는 Outputs 섹션을 추가합니다.

```

Actions:
  MyGitHubAction:
    Identifier: aws/github-actions-runner@v1
    Outputs:
      Variables:
        - 'step-id_output-name'

```

다음과 같이 바꿉니다.

- **step-id**를 GitHub 작업 steps 섹션의 id: 속성 값으로 바꿉니다.
- **output-name**을 GitHub 출력 파라미터의 이름으로 바꿉니다.

예시

다음 예시에서는 SELECTEDCOLOR GitHub 출력 파라미터를 내보내는 방법을 보여줍니다.

```

Actions:
  MyGitHubAction:
    Identifier: aws/github-actions-runner@v1
    Outputs:
      Variables:
        - 'random-color-generator_SELECTEDCOLOR'
    Configuration:
      Steps:
        - name: Set selected color
          run: echo "SELECTEDCOLOR=green" >> $GITHUB_OUTPUT
          id: random-color-generator

```

GitHub 출력 파라미터 참조

다음 지침을 사용하여 GitHub 출력 파라미터를 참조합니다.

GitHub 출력 파라미터를 참조하려면

1. [GitHub 출력 파라미터 내보내기](#)의 단계를 수행하세요.

이제 GitHub 출력 파라미터를 다른 작업에 사용할 수 있습니다.

2. 출력 파라미터의 Variables 값을 기록해 둡니다. 밑줄(_)이 포함됩니다.

3. 다음 구문을 사용하여 출력 파라미터를 참조하세요.

```
${action-name.output-name}
```

다음과 같이 바꿉니다.

- *action-name*을 출력 파라미터를 생성하는 CodeCatalyst GitHub 작업 이름으로 바꿉니다 (GitHub 작업 name 또는 id는 사용하지 않습니다).
- *output-name* 을 앞서 적어 둔 출력 파라미터의 Variables 값으로 바꿉니다.

예시

```
BuildActionB:
  Identifier: aws/build@v1
  Configuration:
    Steps:
      - Run: echo ${MyGitHubAction.random-color-generator_SELECTEDCOLOR}
```

컨텍스트가 있는 예시

다음 예시에서는 GitHubActionA에서 SELECTEDCOLOR 변수를 설정하고 출력한 다음 BuildActionB에서 참조하는 방법을 보여줍니다.

```
Actions:
  GitHubActionA:
    Identifier: aws/github-actions-runner@v1
    Configuration:
      Steps:
        - name: Set selected color
          run: echo "SELECTEDCOLOR=green" >> $GITHUB_OUTPUT
          id: random-color-generator
      Outputs:
        Variables:
          - 'random-color-generator_SELECTEDCOLOR'

  BuildActionB:
    Identifier: aws/build@v1
    Configuration:
      Steps:
```

```
- Run: echo ${GitHubActionA.random-color-generator_SELECTEDCOLOR}
```

'GitHub Actions' 작업 YAML

다음은 GitHub Actions 작업의 YAML 정의입니다.

이 작업 정의는 더 광범위한 워크플로 정의 파일 내의 섹션으로 존재합니다. 이 파일에 대한 자세한 내용은 [워크플로 YAML 정의](#)를 참조합니다.

다음 코드에서 YAML 속성을 선택하면 설명이 표시됩니다.

Note

이어지는 대부분의 YAML 속성에는 시각적 편집기에 해당 UI 요소가 있습니다. UI 요소를 찾으려면 Ctrl+F를 사용합니다. 요소가 연결된 YAML 속성과 함께 나열됩니다.

```
# The workflow definition starts here.
# See ### ## for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
action-name:
  Identifier: aws/github-actions-runner@v1
  DependsOn:
    - dependent-action-name-1
  Compute:
    Fleet: fleet-name
    Timeout: timeout-minutes
  Environment:
    Name: environment-name
  Connections:
    - Name: account-connection-name
    Role: iam-role-name
  Inputs:
    Sources:
      - source-name-1
      - source-name-2
```

Artifacts:

- *artifact-name*

Variables:

- Name: *variable-name-1*
Value: *variable-value-1*
- Name: *variable-name-2*
Value: *variable-value-2*

Outputs:Artifacts:

- Name: *output-artifact-1*

Files:

- *github-output/artifact-1.jar*
- *"github-output/build*"*

- Name: *output-artifact-2*

Files:

- *github-output/artifact-2.1.jar*
- *github-output/artifact-2.2.jar*

Variables:

- *variable-name-1*
- *variable-name-2*

AutoDiscoverReports:

Enabled: *true | false*

ReportNamePrefix: *AutoDiscovered*

IncludePaths:

- *"**/*"*

ExcludePaths:

- *node_modules/cdk/junit.xml*

SuccessCriteria:

PassRate: *percent*

LineCoverage: *percent*

BranchCoverage: *percent*

Vulnerabilities:

Severity: *CRITICAL|HIGH|MEDIUM|LOW|INFORMATIONAL*

Number: *whole-number*

Reports:*report-name-1*:

Format: *format*

IncludePaths:

- *"*.xml"*

ExcludePaths:

- *report2.xml*
- *report3.xml*

SuccessCriteria:

PassRate: *percent*

```

    LineCoverage: percent
    BranchCoverage: percent
    Vulnerabilities:
      Severity: CRITICAL|HIGH|MEDIUM|LOW|INFORMATIONAL
      Number: whole-number

```

Configuration

Steps:

- *github-actions-code*

action-name

(필수)

작업 이름을 지정합니다. 워크플로 내의 모든 작업 이름은 고유해야 합니다. 작업 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 작업 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

해당 UI: 구성 탭/*action-name*

Identifier

(*action-name*/Identifier)

작업을 식별합니다. 버전을 변경하려는 경우가 아니면 이 속성을 변경하지 마세요. 자세한 내용은 [사용할 작업 버전 지정](#) 섹션을 참조하세요.

aws/github-actions-runner@v1를 GitHub Actions 작업에 사용합니다.

해당 UI: 워크플로 다이어그램/*action-name* /aws/github-actions-runner@v1 레이블

DependsOn

(*action-name*/DependsOn)

(선택 사항)

이 작업을 실행하기 위해 성공적으로 실행해야 하는 작업, 작업 그룹 또는 게이트를 지정합니다.

'depends on' 함수에 대한 자세한 내용은 [작업 순서 지정](#) 섹션을 참조하세요.

해당 UI: 입력 탭/의존 - 선택 사항

Compute

(*action-name*/Compute)

(선택 사항)

워크플로 작업을 실행하는 데 사용되는 컴퓨팅 엔진입니다. 워크플로 수준 또는 작업 수준에서 컴퓨팅을 지정할 수 있지만 둘 다 지정할 수는 없습니다. 워크플로 수준에서 지정하면 컴퓨팅 구성이 워크플로에 정의된 모든 작업에 적용됩니다. 워크플로 수준에서는 동일한 인스턴스에서 여러 작업을 실행할 수도 있습니다. 자세한 내용은 [작업 간에 컴퓨팅 공유](#) 섹션을 참조하세요.

해당 UI: 없음

Fleet

(*action-name*/Compute/Fleet)

(선택 사항)

워크플로 또는 워크플로 작업을 실행할 시스템 또는 플릿을 지정합니다. 온디맨드 플릿의 경우 작업이 시작되면 워크플로가 필요한 리소스를 프로비저닝하고 작업이 완료되면 시스템이 파괴됩니다. 온디맨드 플릿의 예시: Linux.x86-64.Large, Linux.x86-64.XLarge. 온디맨드 플릿에 대한 자세한 내용은 [온디맨드 플릿 속성](#) 섹션을 참조하세요.

프로비저닝된 플릿을 사용하면 워크플로 작업을 실행하도록 전용 시스템 세트를 구성할 수 있습니다. 이러한 시스템은 유휴 상태로 유지되므로 작업을 즉시 처리할 수 있습니다. 프로비저닝된 플릿에 대한 자세한 내용은 [프로비저닝된 플릿 속성](#) 섹션을 참조하세요.

Fleet 생략 시 기본값은 Linux.x86-64.Large입니다.

해당 UI: 구성 탭/컴퓨팅 플릿 - 선택 사항

Timeout

(*action-name*/Timeout)

(선택 사항)

CodeCatalyst가 작업을 종료하기 전에 작업을 실행할 수 있는 시간을 분(YAML 편집기) 또는 시간 및 분(시각적 편집기) 단위로 지정합니다. 최소값은 5분이고 최대값은 [CodeCatalyst의 워크플로 할당량](#)에 설명되어 있습니다. 기본 제한 시간은 최대 제한 시간과 동일합니다.

해당 UI: 구성 탭/제한 시간 - 선택 사항

Environment

(*action-name*/Environment)

(선택 사항)

작업에 사용할 CodeCatalyst 환경을 지정합니다. 작업은 선택한 환경에 지정된 AWS 계정 및 선택적 Amazon VPC에 연결됩니다. 작업은 환경에 지정된 기본 IAM 역할을 사용하여 연결하고 [Amazon VPC 연결](#)에 지정된 IAM 역할을 AWS 계정으로 사용하여 Amazon VPC에 연결합니다.

Note

기본 IAM 역할에 작업에 필요한 권한이 없는 경우 다른 역할을 사용하도록 작업을 구성할 수 있습니다. 자세한 내용은 [작업의 IAM 역할 변경](#) 섹션을 참조하세요.

환경에 대한 자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 및 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 구성 탭/환경

Name

(*action-name*/Environment/Name)

([Environment](#) 포함 시 필수)

작업과 연결하려는 기존 환경의 이름을 지정합니다.

해당 UI: 구성 탭/환경

Connections

(*action-name*/Environment/Connections)

(선택 사항)

작업과 연결할 계정 연결을 지정합니다. Environment에서 계정 연결을 최대 1개까지 지정할 수 있습니다.

계정 연결을 지정하지 않는 경우:

- 작업은 CodeCatalyst 콘솔의 환경에 지정된 AWS 계정 연결 및 기본 IAM 역할을 사용합니다. 환경에 계정 연결 및 기본 IAM 역할을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.

- 기본 IAM 역할에는 작업에 필요한 정책 및 권한이 포함되어야 합니다. 이러한 정책 및 권한이 무엇인지 확인하려면 작업의 YAML 정의 설명서에서 역할 속성에 대한 설명을 참조하세요.

계정 연결에 대한 자세한 정보는 [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 섹션을 참조하세요. 환경에 계정 연결을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환

Name

(*action-name*/Environment/Connections/Name)

([Connections](#) 포함 시 필수)

계정 연결의 이름을 지정합니다.

해당 UI: 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환

Role

(*action-name*/Environment/Connections/Role)

([Connections](#) 포함 시 필수)

Amazon S3 및 Amazon ECR과 같은 AWS 서비스에 액세스하고 운영하기 위해 이 작업이 사용하는 IAM 역할의 이름을 지정합니다. 이 역할이 스페이스의 AWS 계정 연결에 추가되었는지 확인합니다. 계정 연결에 IAM 역할을 추가하려면 [계정 연결에 IAM 역할 추가](#) 섹션을 참조하세요.

IAM 역할을 지정하지 않으면 작업은 CodeCatalyst 콘솔의 [환경](#)에 나열된 기본 IAM 역할을 사용합니다. 환경에서 기본 역할을 사용하는 경우 다음 정책이 있는지 확인합니다.

Note

이 작업에서 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 사용할 수 있습니다. 이에 대한 자세한 내용은 [계정 및 스페이스의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 생성](#) 섹션을 참조하세요.

CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할에 보안 위험을 초래할 수 있는 전체 액세스 권한이 있음을 이해합니다. 보안에 대한 우려가 적은 자습서 및 시나리오에서만 이 역할을 사용하는 것이 좋습니다.

⚠ Warning

권한을 GitHub 작업 작업에 필요한 권한으로 제한합니다. 더 광범위한 권한을 가진 역할을 사용하면 보안 위험이 발생할 수 있습니다.

해당 UI: 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환

Inputs

(*action-name*/Inputs)

(선택 사항)

이 Inputs 섹션에서는 워크플로 실행 중에 작업에 필요한 데이터를 정의합니다.

i Note

GitHub Actions 작업당 최대 4개의 입력(소스 1개 및 아티팩트 3개)이 허용됩니다. 변수는 이 합계에 포함되지 않습니다.

서로 다른 입력(소스 및 아티팩트)에 있는 파일을 참조해야 하는 경우 소스 입력은 기본 입력이고 아티팩트는 보조 입력입니다. 보조 입력의 파일에 대한 참조는 특수 접두사를 사용하여 기본 입력에서 파일을 구분합니다. 자세한 내용은 [예시: 여러 아티팩트에서 파일 참조](#)를 참조하세요.

해당 UI: 입력 탭

Sources

(*action-name*/Inputs/Sources)

(선택 사항)

작업에 필요한 소스 리포지토리를 나타내는 레이블을 지정합니다. 현재 지원되는 유일한 레이블은 워크플로 정의 파일이 저장되는 소스 리포지토리를 나타내는 WorkflowSource입니다.

소스를 생략하는 경우 *action-name*/Inputs/Artifacts 아래에 하나 이상의 입력 아티팩트를 지정해야 합니다.

소스에 대한 자세한 내용은 [워크플로에 소스 리포지토리 연결](#) 섹션을 참조하세요.

해당 UI: 입력 탭/소스 - 선택 사항

Artifacts - input

(*action-name*/Inputs/Artifacts)

(선택 사항)

이 작업에 대한 입력으로 제공하려는 이전 작업의 아티팩트를 지정합니다. 이러한 아티팩트는 이전 작업에서 출력 아티팩트로 이미 정의되어 있어야 합니다.

입력 아티팩트를 지정하지 않으면 *action-name*/Inputs/Sources 아래에 소스 리포지토리를 하나 이상 지정해야 합니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

Note

아티팩트 - 선택적 드롭다운 목록을 사용할 수 없거나(시각적 편집기) YAML(YAML 편집기)을 검증할 때 오류가 발생하는 경우 작업이 하나의 입력만 지원하기 때문일 수 있습니다. 이 경우 소스 입력을 제거해 보세요.

해당 UI: 입력 탭/아티팩트 - 선택 사항

Variables - input

(*action-name*/Inputs/Variables)

(선택 사항)

작업에 사용할 수 있도록 하려는 입력 변수를 정의하는 이름/값 페어의 시퀀스를 지정합니다. 변수 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스은 허용되지 않습니다. 변수 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

예시를 비롯한 변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

해당 UI: 입력 탭/변수 - 선택 사항

Outputs

(*action-name*/Outputs)

(선택 사항)

워크플로 실행 중에 작업에 의해 출력되는 데이터를 정의합니다.

해당 UI: 출력 탭

Artifacts - output

(*action-name*/Outputs/Artifacts)

(선택 사항)

작업에서 생성된 아티팩트의 이름을 지정합니다. 아티팩트 이름은 워크플로 내에서 고유해야 하며 영숫자 문자(a-z, A-Z, 0-9) 및 밑줄(_)로 제한됩니다. 공백, 하이픈(-) 및 특수 문자는 허용되지 않습니다. 출력 아티팩트 이름에서 공백, 하이픈 및 기타 특수 문자를 활성화하는 데 따옴표를 사용할 수 없습니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

해당 UI: 출력 탭/아티팩트

Name

(*action-name*/Outputs/Artifacts/Name)

([Artifacts - output](#) 포함 시 필수)

작업에서 생성된 아티팩트의 이름을 지정합니다. 아티팩트 이름은 워크플로 내에서 고유해야 하며 영숫자 문자(a-z, A-Z, 0-9) 및 밑줄(_)로 제한됩니다. 공백, 하이픈(-) 및 특수 문자는 허용되지 않습니다. 출력 아티팩트 이름에서 공백, 하이픈 및 기타 특수 문자를 활성화하는 데 따옴표를 사용할 수 없습니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

해당 UI: 출력 탭/아티팩트/아티팩트 추가/빌드 아티팩트 이름

Files

(*action-name*/Outputs/Artifacts/Files)

([Artifacts - output](#) 포함 시 필수)

CodeCatalyst가 작업으로 출력되는 아티팩트에 포함하는 파일을 지정합니다. 이러한 파일은 실행 시 워크플로 작업에 의해 생성되며 소스 리포지토리에서도 사용할 수 있습니다. 파일 경로는 소스 리포지토리 또는 이전 작업의 아티팩트에 상주할 수 있으며 소스 리포지토리 또는 아티팩트 루트와 관련이 있습니다. glob 패턴을 사용하여 경로를 지정할 수 있습니다. 예시:

- 빌드 위치 또는 소스 리포지토리 위치의 루트에 있는 단일 파일을 지정하려면 `my-file.jar`를 사용합니다..
- 하위 디렉터리에 단일 파일을 지정하려면 `directory/my-file.jar` 또는 `directory/subdirectory/my-file.jar`를 사용합니다.
- 모든 파일을 지정하려면 `"/**/*"`를 사용합니다. `** glob` 패턴은 임의의 수의 하위 디렉터리와 일치함을 나타냅니다.
- `directory`라는 디렉터리에 있는 모든 파일 및 디렉터를 지정하려면 `"directory/**/*"`를 사용합니다. `** glob` 패턴은 임의의 수의 하위 디렉터리와 일치함을 나타냅니다.
- `directory`라는 디렉터리의 모든 파일을 지정하되 해당 하위 디렉터리는 지정하지 않으려면 `"directory/*"`를 사용합니다.

Note

파일 경로에 별표(*) 또는 기타 특수 문자가 하나 이상 포함된 경우 경로를 큰따옴표(" ")로 묶습니다. 특수 문자에 대한 자세한 내용은 [구문 지침 및 규칙](#) 섹션을 참조하세요.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

Note

파일 경로에 접두사를 추가하여 찾을 아티팩트 또는 소스를 나타내야 할 수 있습니다. 자세한 내용은 [소스 리포지토리 파일 참조](#) 및 [아티팩트의 파일 참조](#) 섹션을 참조하세요.

해당 UI: 출력 탭/아티팩트/아티팩트 추가/빌드에서 생성된 파일

Variables - output

(*action-name*/Outputs/Variables)

(선택 사항)

후속 작업에서 사용할 수 있도록 작업을 내보낼 변수를 지정합니다.

예시를 비롯한 변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

해당 UI: 출력 탭/변수/변수 추가

variable-name-1

(*action-name*/Outputs/Variablesvariable-name-1)

(선택 사항)

작업을 내보낼 변수의 이름을 지정합니다. 이 변수는 동일한 작업의 Inputs 또는 Steps 섹션에 이미 정의되어 있어야 합니다.

예시를 비롯한 변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

해당 UI: 출력 탭/변수/변수 추가/이름

AutoDiscoverReports

(*action-name*/Outputs/AutoDiscoverReports)

(선택 사항)

자동 검색 기능의 구성을 정의합니다.

자동 검색을 활성화하면 CodeCatalyst는 작업에 전달된 모든 Inputs와 작업 자체에서 생성된 모든 파일을 검색하여 테스트, 코드 적용 범위 및 소프트웨어 구성 분석(SCA) 보고서를 찾습니다. 발견된 각 보고서에 대해 CodeCatalyst는 이를 CodeCatalyst 보고서로 변환합니다. CodeCatalyst 보고서는 CodeCatalyst 서비스에 완전히 통합되고 CodeCatalyst 콘솔을 통해 보고 조작할 수 있는 보고서입니다.

 Note

기본적으로 자동 검색 기능은 모든 파일을 검사합니다. [IncludePaths](#) 또는 [ExcludePaths](#) 속성을 사용하여 검사할 파일을 제한할 수 있습니다.

해당 UI: 없음

Enabled

(*action-name*/Outputs/AutoDiscoverReports/Enabled)

(선택 사항)

자동 검색 기능을 활성화하거나 비활성화합니다.

유효한 값은 true 또는 false입니다.

Enabled 생략 시 기본값은 true입니다.

해당 UI: 출력 탭/보고서/자동 검색 보고서

ReportNamePrefix

(*action-name*/Outputs/AutoDiscoverReports/ReportNamePrefix)

([AutoDiscoverReports](#) 포함 및 활성화 시 필수)

연결된 CodeCatalyst 보고서의 이름을 지정하기 위해 CodeCatalyst가 찾는 모든 보고서에 우선하는 접두사를 지정합니다. 예를 들어 접두사를 AutoDiscovered로 지정하고 CodeCatalyst가 두 개의 테스트 보고서, TestSuiteOne.xml 및 TestSuiteTwo.xml를 자동으로 검색하면 연결된 CodeCatalyst 보고서의 이름이 AutoDiscoveredTestSuiteOne 및 AutoDiscoveredTestSuiteTwo로 지정됩니다.

해당 UI: 출력 탭/보고서/보고서 자동 검색/보고서 접두사

IncludePaths

(*action-name*/Outputs/AutoDiscoverReports/IncludePaths)

또는

(*action-name*/Outputs/Reports/*report-name-1*/IncludePaths)

([AutoDiscoverReports](#) 포함 및 활성화 시 또는 [Reports](#) 포함 시 필수)

원시 보고서를 검색할 때 CodeCatalyst에 포함되는 파일 및 파일 경로를 지정합니다. 예를 들어 `/test/report/*` 지정 시 CodeCatalyst는 작업에서 `/test/report/*` 디렉터리를 찾는 데 사용되는 전체 [빌드 이미지](#)를 검색합니다. 해당 디렉터리를 찾으면 CodeCatalyst는 해당 디렉터리에서 보고서를 찾습니다.

Note

파일 경로에 별표(*) 또는 기타 특수 문자가 하나 이상 포함된 경우 경로를 큰따옴표(" ")로 묶습니다. 특수 문자에 대한 자세한 내용은 [구문 지침 및 규칙](#) 섹션을 참조하세요.

이 속성이 생략되면 기본값은 `**/*`입니다. 즉, 검색에 모든 경로의 모든 파일이 포함됩니다.

Note

수동으로 구성된 보고서의 경우 IncludePaths는 단일 파일과 일치하는 글로브 패턴이어야 합니다.

해당 UI:

- 출력 탭/보고서/보고서 자동 검색/'경로 포함/제외/'경로 포함
- 출력 탭/보고서/보고서 수동 구성/*report-name-1*'경로 포함/제외/'경로 포함

ExcludePaths

(*action-name*/Outputs/AutoDiscoverReports/ExcludePaths)

또는

(*action-name*/Outputs/Reports/ *report-name-1*/ExcludePaths)

(선택 사항)

원시 보고서를 검색할 때 CodeCatalyst에서 제외하는 파일 및 파일 경로를 지정합니다. 예를 들어 "/test/my-reports/**/*" 지정 시 CodeCatalyst는 /test/my-reports/ 디렉터리의 파일을 검색하지 않습니다. 디렉터리의 모든 파일을 무시하려면 **/* glob 패턴을 사용합니다.

Note

파일 경로에 별표(*) 또는 기타 특수 문자가 하나 이상 포함된 경우 경로를 큰따옴표("")로 묶습니다. 특수 문자에 대한 자세한 내용은 [구문 지침 및 규칙](#) 섹션을 참조하세요.

해당 UI:

- 출력 탭/보고서/보고서 자동 검색/'경로 포함/제외/'경로 제외
- 출력 탭/보고서/보고서 수동 구성/*report-name-1*'경로 포함/제외/'경로 제외

SuccessCriteria

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria)

또는

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria)

(선택 사항)

테스트, 코드 적용 범위, 소프트웨어 구성 분석(SCA) 및 정적 분석(SA) 보고서의 성공 기준을 지정합니다.

자세한 내용은 [보고서의 성공 기준 구성](#) 섹션을 참조하세요.

해당 UI:

- 출력 탭/보고서/보고서 자동 검색/성공 기준
- 출력 탭/보고서/보고서 수동 구성/*report-name-1*/성공 기준

PassRate

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/PassRate)

또는

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/PassRate)

(선택 사항)

관련 CodeCatalyst 보고서를 통과로 표시하기 위해 통과해야 하는 테스트 보고서의 테스트 비율을 지정합니다. 유효한 값에는 십진수가 포함됩니다. 예시: 50, 60.5. 통과율 기준은 테스트 보고서에만 적용됩니다. 테스트 보고서에 대한 자세한 내용은 [테스트 보고서](#)의 내용을 참조하세요.

해당 UI:

- 출력 탭/보고서/보고서 자동 검색/성공 기준/통과율
- 출력 탭/보고서/보고서 수동 구성/*report-name-1*/성공 기준/통과율

LineCoverage

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/LineCoverage)

또는

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/LineCoverage)

(선택 사항)

연결된 CodeCatalyst 보고서를 통과로 표시하기 위해 다루어야 하는 코드 적용 범위 보고서의 줄 비율을 지정합니다. 유효한 값에는 십진수가 포함됩니다. 예시: 50, 60.5. 라인 적용 범위 기준은 코드 적용 범위 보고서에만 적용됩니다. 코드 적용 범위 보고서에 대한 자세한 내용은 [코드 적용 범위 보고서](#) 섹션을 참조하세요.

해당 UI:

- 출력 탭/보고서/보고서 자동 검색/성공 기준/라인 적용 범위
- 출력 탭/보고서/보고서 수동 구성/*report-name-1*/성공 기준/라인 적용 범위

BranchCoverage

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/BranchCoverage)

또는

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/BranchCoverage)

(선택 사항)

연결된 CodeCatalyst 보고서를 통과로 표시하기 위해 다루어야 하는 코드 적용 범위 보고서의 브랜치 비율을 지정합니다. 유효한 값에는 십진수가 포함됩니다. 예시: 50, 60.5. 브랜치 적용 범위 기준은 코드 적용 범위 보고서에만 적용됩니다. 코드 적용 범위 보고서에 대한 자세한 내용은 [코드 적용 범위 보고서](#) 섹션을 참조하세요.

해당 UI:

- 출력 탭/보고서/보고서 자동 검색/성공 기준/브랜치 적용 범위
- 출력 탭/보고서/보고서 수동 구성/*report-name-1*/성공 기준/브랜치 적용 범위

Vulnerabilities

(*action-name*/Outputs/AutoDiscoverReports/SuccessCriteria/Vulnerabilities)

또는

(*action-name*/Outputs/Reports/*report-name-1*/SuccessCriteria/Vulnerabilities)

(선택 사항)

SCA 보고서에서 허용되는 취약성의 최대 수와 심각도를 지정하여 관련 CodeCatalyst 보고서를 통과로 표시합니다. 취약성을 지정하려면 다음을 지정해야 합니다.

- 개수에 포함하려는 취약성의 최소 심각도입니다. 최대부터 최소 심각도까지 유효한 값은 CRITICAL, HIGH, MEDIUM, LOW 및 INFORMATIONAL입니다.

예를 들어 HIGH 선택 시 HIGH 및 CRITICAL 취약성이 집계됩니다.

- 허용하려는 지정된 심각도의 최대 취약성 수입니다. 이 수를 초과하면 CodeCatalyst 보고서가 실패로 표시됩니다. 유효한 값은 정수입니다.

취약성 기준은 SCA 보고서에만 적용됩니다. SCA 보고서에 대한 자세한 내용은 [소프트웨어 구성 분석 보고서](#)의 내용을 참조하세요.

최소 심각도를 지정하려면 Severity 속성을 사용합니다. 최대 취약성 수를 지정하려면 Number 속성을 사용합니다.

SCA 보고서에 대한 자세한 내용은 [품질 보고서 유형](#)의 내용을 참조하세요.

해당 UI:

- 출력 탭/보고서/보고서 자동 검색/성공 기준/취약성
- 출력 탭/보고서/보고서 수동 구성/*report-name-1*/성공 기준/취약성

Reports

(*action-name*/Outputs/Reports)

(선택 사항)

테스트 보고서의 구성을 지정하는 섹션입니다.

해당 UI: 출력 탭/보고서

report-name-1

(*action-name*/Outputs/Reports/report-name-1)

([Reports](#) 포함 시 필수)

원시 보고서에서 생성될 CodeCatalyst 보고서에 부여할 이름입니다.

해당 UI: 출력 탭/보고서 수동 구성/보고서 이름

Format

(*action-name*/Outputs/Reports/*report-name-1*/Format)

([Reports](#) 포함 시 필수)

보고서에 사용할 파일 형식을 지정합니다. 가능한 값은 다음과 같습니다.

- 테스트 보고서의 경우:
 - Cucumber JSON에서 Cucumber(시각 편집기) 또는 CUCUMBERJSON(YAML 편집기)를 지정합니다.
 - JUnit XML에 JUnit(시각 편집기) 또는 JUNITXML(YAML 편집기)를 지정합니다.
 - NUnit XML에 NUnit(시각 편집기) 또는 NUNITXML(YAML 편집기)를 지정합니다.
 - NUnit 3 XML에서 NUnit3(시각 편집기) 또는 NUNIT3XML(YAML 편집기)를 지정합니다.
 - Visual Studio TRX에서 Visual Studio TRX(시각 편집기) 또는 VISUALSTUDIOTRX(YAML 편집기)를 지정합니다.
 - TestNG XML에서 TestNG(시각 편집기) 또는 TESTNGXML(YAML 편집기)를 지정합니다.
- 코드 적용 범위 보고서의 경우:
 - Clover XML에서 Clover(시각 편집기) 또는 CLOVERXML(YAML 편집기)를 지정합니다.
 - Cobertura XML에서 Cobertura(시각 편집기) 또는 COBERTURAXML(YAML 편집기)를 지정합니다.
 - JaCoCo XML의 경우 JaCoCo(시각 편집기) 또는 JACOCOXML(YAML 편집기)를 지정합니다.
 - [simplecov-json](#)이 아닌 [simplecov](#)에서 생성한 SimpleCov JSON의 경우 Simplecov(시각 편집기) 또는 SIMPLECOV(YAML 편집기)를 지정합니다.
- 소프트웨어 구성 분석(SCA) 보고서의 경우:
 - SARIF에서 SARIF(시각 편집기) 또는 SARIFSCA(YAML 편집기)를 지정합니다.

해당 UI: 출력 탭/보고서/보고서 수동 구성/보고서 추가/*report-name-1*/보고서 유형 및 보고서 형식

Configuration

(*action-name*/Configuration)

(필수) 작업의 구성 속성을 정의할 수 있는 섹션입니다.

해당 UI: 구성 탭

Steps

(*action-name*/Configuration/Steps)

(필수)

[GitHub Marketplace](#)의 작업 세부 정보 페이지에 표시되는 대로 GitHub 작업 코드를 지정합니다. 다음 지침에 따라 코드를 추가합니다.

1. GitHub 작업 steps: 섹션의 코드를 CodeCatalyst 워크플로의 Steps: 섹션에 붙여 넣습니다. 코드는 대시(-)로 시작하며 다음과 비슷합니다.

붙여넣을 GitHub 코드:

```
- name: Lint Code Base
  uses: github/super-linter@v4
  env:
    VALIDATE_ALL_CODEBASE: false
    DEFAULT_BRANCH: master
    GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
```

2. 방금 붙여넣은 코드를 검토하고 CodeCatalyst 표준을 준수하도록 필요에 따라 수정합니다. 예를 들어 앞의 코드 블록을 사용하면 **###** **###** 끝로 된 코드를 제거하고 굵은 글씨체로 된 코드를 추가할 수 있습니다.

CodeCatalyst 워크플로 yaml:

```
Steps:
- name: Lint Code Base
  uses: github/super-linter@v4
  env:
    VALIDATE_ALL_CODEBASE: false
    DEFAULT_BRANCH: mastermain
    GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
```

3. GitHub 작업에 포함되지만 steps: 섹션 내에 존재하지 않는 추가 코드는 CodeCatalyst에서 해당하는 코드를 사용하여 CodeCatalyst 워크플로에 추가합니다. [워크플로 YAML 정의](#)를 검토하여 GitHub 코드를 CodeCatalyst에 이식하는 방법에 대한 인사이트를 얻을 수 있습니다. 자세한 마이그레이션 단계는 이 가이드의 범위를 벗어납니다.

다음은 GitHub Actions 작업에서 파일 경로를 지정하는 방법의 예입니다.

Steps:

- name: Lint Code Base
uses: github/super-linter@v4
...
- run: cd /sources/WorkflowSource/MyFolder/ && cat file.txt
- run: cd /artifacts/MyGitHubAction/MyArtifact/MyFolder/ && cat file2.txt

파일 경로 지정에 대한 자세한 내용은 [소스 리포지토리 파일 참조](#) 및 [아티팩트의 파일 참조](#)를 참조하세요.

해당 UI: 구성 탭/GitHub Actions YAML

컴퓨팅 및 런타임 이미지 구성

CodeCatalyst 워크플로에서 CodeCatalyst가 워크플로 작업을 실행하는 데 사용하는 컴퓨팅 및 런타임 환경 이미지를 지정할 수 있습니다.

컴퓨팅은 워크플로 작업을 실행하기 위해 CodeCatalyst에서 관리 및 유지 관리하는 컴퓨팅 엔진(CPU, 메모리 및 운영 체제)을 의미합니다.

Note

컴퓨팅이 워크플로의 속성으로 정의된 경우 해당 워크플로의 작업 속성으로 정의할 수 없습니다. 마찬가지로 컴퓨팅이 작업의 속성으로 정의된 경우 워크플로에서 정의할 수 없습니다.

런타임 환경 이미지는 CodeCatalyst가 워크플로 작업을 실행하는 Docker 컨테이너입니다. Docker 컨테이너는 선택한 컴퓨팅 플랫폼을 기반으로 실행되며, 운영 체제와 AWS CLI Node.js, .tar 등 워크플로 작업에 필요할 수 있는 추가 도구를 포함합니다.

주제

- [컴퓨팅 유형](#)
- [컴퓨팅 플릿](#)
- [온디맨드 플릿 속성](#)
- [프로비저닝된 플릿 속성](#)
- [프로비저닝된 플릿 생성](#)
- [프로비저닝된 플릿 편집](#)

- [프로비저닝된 플릿 삭제](#)
- [작업에 플릿 또는 컴퓨팅 할당](#)
- [작업 간에 컴퓨팅 공유](#)
- [런타임 환경 이미지 지정](#)

컴퓨팅 유형

CodeCatalyst는 다음과 같은 컴퓨팅 유형을 제공합니다.

- Amazon EC2
- AWS Lambda

Amazon EC2는 작업 실행 중에 최적화된 유연성을 제공하고 Lambda는 최적화된 작업 시작 속도를 제공합니다. Lambda는 시작 지연 시간이 짧아 워크플로 작업 실행 속도가 빨라집니다. Lambda를 사용하면 일반적인 런타임으로 서버리스 애플리케이션을 빌드, 테스트 및 배포할 수 있는 기본 워크플로를 실행할 수 있습니다. 이러한 런타임에는 Node.js, Python, Java, .NET 및 Go가 포함됩니다. 그러나 Lambda가 지원하지 않는 일부 사용 사례가 있으며, 이러한 사용 사례가 영향을 미치는 경우 Amazon EC2 컴퓨팅 유형을 사용하세요.

- Lambda는 지정된 레지스트리의 런타임 환경 이미지를 지원하지 않습니다.
- Lambda에서 루트 권한이 필요한 도구는 지원하지 않습니다. yum 또는 rpm 등의 도구에는 Amazon EC2 컴퓨팅 유형이나 루트 권한이 필요하지 않은 기타 도구를 사용하세요.
- Lambda에서는 Docker 빌드 또는 실행을 지원하지 않습니다. Docker 이미지를 사용하는 다음 작업은 지원되지 않습니다. AWS CloudFormation 스택 배포, Amazon ECS에 배포, Amazon S3 게시, AWS CDK 부트스트랩, AWS CDK 배포, AWS Lambda 간접 호출 및 GitHub 작업. CodeCatalyst GitHub Actions 작업 내에서 실행되는 Docker 기반 GitHub Actions도 Lambda 컴퓨팅에서 지원되지 않습니다. Podman과 같이 루트 권한이 필요하지 않은 대안을 사용할 수 있습니다.
- Lambda에서는 /tmp 이외 파일에 쓰는 것을 지원하지 않습니다. 워크플로 작업을 구성할 때 설치하거나 /tmp에 쓸 수 있는 도구를 재구성합니다. npm을 설치하는 빌드 작업이 있는 경우 /tmp에 설치하도록 구성해야 합니다.
- Lambda는 15분보다 긴 런타임을 지원하지 않습니다.

컴퓨팅 플릿

CodeCatalyst는 다음과 같은 컴퓨팅 플릿을 제공합니다.

- 온디맨드 플릿
- 프로비저닝된 플릿

온디맨드 플릿의 경우 워크플로 작업이 시작되면 워크플로가 필요한 리소스를 프로비저닝합니다. 작업이 완료되면 머신이 파괴됩니다. 작업을 실행하는 분 수에 대해서만 비용을 지불합니다. 온디맨드 플릿은 완전 관리형이며, 수요 급증을 처리할 수 있는 자동 규모 조정 기능이 포함되어 있습니다.

또한 CodeCatalyst는 CodeCatalyst에서 관리하는 Amazon EC2 기반 시스템을 포함하는 프로비저닝된 플릿을 제공합니다. 프로비저닝된 플릿을 사용하면 워크플로 작업을 실행하도록 전용 시스템 세트를 구성할 수 있습니다. 이러한 시스템은 유휴 상태로 유지되므로 작업을 즉시 처리할 수 있습니다. 프로비저닝된 플릿을 사용하면 머신이 상시 가동되므로 프로비저닝하는 비용이 발생합니다.

플릿을 생성하거나 업데이트 또는 삭제하려면 스페이스 관리자 역할 또는 프로젝트 관리자 역할이 있어야 합니다.

온디맨드 플릿 속성

CodeCatalyst는 다음과 같은 온디맨드 플릿을 제공합니다.

이름	운영 체제	아키텍처	vCPU	메모리 (GiB)	디스크 공간	지원되는 컴퓨팅 유형
Linux.Arm64.Large	Amazon Linux 2	Arm64	2	4	64GB	Amazon EC2
					10GB	Lambda
Linux.Arm64.XLarge	Amazon Linux 2	Arm64	4	8	128GB	Amazon EC2
					10GB	Lambda
Linux.Arm64.2XLarge	Amazon Linux 2	Arm64	8	16	128GB	Amazon EC2

이름	운영 체제	아키텍처	vCPU	메모리 (GiB)	디스크 공간	지원되는 컴퓨팅 유형
Linux.x86-64.Large	Amazon Linux 2	x86-64	2	4	64GB	Amazon EC2
					10GB	Lambda
Linux.x86-64.XLarge	Amazon Linux 2	x86-64	4	8	128GB	Amazon EC2
					10GB	Lambda
Linux.x86-64.2XLarge	Amazon Linux 2	x86-64	8	16	128GB	Amazon EC2

Note

온디맨드 플릿의 사양은 청구 계층에 따라 달라집니다. 자세한 내용은 [요금](#)을 참조하세요.

플릿을 선택하지 않으면 CodeCatalyst는 Linux.x86-64.Large를 사용합니다.

프로비저닝된 플릿 속성

프로비저닝된 플릿에는 다음 속성이 포함됩니다.

운영 체제

운영 체제입니다. 사용할 수 있는 운영 체제는 다음과 같습니다.

- Amazon Linux 2
- Windows Server 2022

Note

Windows 플릿은 빌드 작업에서만 지원됩니다. 다른 작업은 현재 Windows를 지원하지 않습니다.

아키텍처

프로세서 아키텍처. 사용할 수 있는 아키텍처는 다음과 같습니다.

- x86_64
- Arm64

시스템 유형

각 인스턴스의 머신 유형. 사용할 수 있는 유형의 로그는 다음과 같습니다.

vCPU	메모리(GiB)	디스크 공간	운영 체제
2	4	64GB	Amazon Linux 2
4	8	128GB	Amazon Linux 2
			Windows Server 2022
8	16	128GB	Amazon Linux 2
			Windows Server 2022

Capacity

플릿에 할당된 초기 머신 수로, 병렬로 실행할 수 있는 작업 수를 정의합니다.

규모 조정 모드

작업 수가 플릿 용량을 초과할 때의 동작을 정의합니다.

온디맨드 방식으로 추가 용량 프로비저닝

온디맨드 방식으로 추가 머신이 설치되어 새 작업이 실행되면 자동으로 스케일 업이 실시되고 작업이 완료되면 기본 용량으로 스케일 다운이 실시됩니다. 실행 중인 각 머신에 대해 분 단위로 비용을 지불해야 하므로 추가 비용이 발생할 수 있습니다.

추가 플릿 용량을 사용할 수 있을 때까지 대기

머신을 사용할 수 있을 때까지 작업 실행이 대기열에 배치됩니다. 이렇게 하면 추가 머신이 할당되지 않으므로 추가 비용이 제한됩니다.

프로비저닝된 플릿 생성

다음 지침에 따라 프로비저닝된 플릿을 생성합니다.

Note

프로비저닝된 플릿은 2주 동안 사용하지 않으면 비활성화됩니다. 다시 사용하면 자동으로 다시 활성화되지만, 다시 활성화하면 지연 시간이 발생할 수 있습니다.

프로비저닝된 플릿을 생성하려면

1. 탐색 창에서 CI/CD를 선택한 후, 컴퓨팅을 선택합니다.
2. 프로비저닝된 플릿 생성을 선택합니다.
3. 프로비저닝된 플릿 이름 텍스트 필드에 플릿의 이름을 입력합니다.
4. 운영 체제 드롭다운 메뉴에서 운영 체제를 선택합니다.
5. 머신 유형 드롭다운 메뉴에서 해당 머신 유형을 선택합니다.
6. 용량 텍스트 필드에 플릿의 최대 머신 수를 입력합니다.
7. 규모 조정 모드 드롭다운 메뉴에서 원하는 오버플로 동작을 선택합니다. 필드에 대한 자세한 내용은 [프로비저닝된 플릿 속성](#) 섹션을 참조하세요.
8. 생성(Create)을 선택합니다.

프로비저닝된 플릿을 생성하고 해당 플릿을 작업에 할당할 준비가 되었습니다. 자세한 내용은 [작업에 플릿 또는 컴퓨팅 할당](#) 단원을 참조하십시오.

프로비저닝된 플릿 편집

다음 지침에 따라 프로비저닝된 플릿을 편집합니다.

Note

프로비저닝된 플릿은 2주 동안 사용하지 않으면 비활성화됩니다. 다시 사용하면 자동으로 다시 활성화되지만, 다시 활성화하면 지연 시간이 발생할 수 있습니다.

프로비저닝된 플릿을 편집하려면

1. 탐색 창에서 CI/CD를 선택한 후, 컴퓨팅을 선택합니다.
2. 프로비저닝된 플릿 목록에서 편집할 플릿을 선택합니다.
3. 편집을 선택합니다.
4. 용량 텍스트 필드에 플릿의 최대 머신 수를 입력합니다.
5. 규모 조정 모드 드롭다운 메뉴에서 원하는 오버플로 동작을 선택합니다. 필드에 대한 자세한 내용은 [프로비저닝된 플릿 속성](#) 섹션을 참조하세요.
6. 저장을 선택합니다.

프로비저닝된 플릿 삭제

다음 지침에 따라 프로비저닝된 플릿을 삭제합니다.

프로비저닝된 플릿을 삭제하려면

Warning

프로비저닝된 플릿을 삭제하기 전에 작업의 YAML 코드에서 Fleet 속성을 삭제하여 모든 작업에서 제거합니다. 프로비저닝된 플릿을 삭제한 후에도 계속 참조하는 작업은 다음 번에 작업이 실행될 때 실패합니다.

1. 탐색 창에서 CI/CD를 선택한 후, 컴퓨팅을 선택합니다.
2. 프로비저닝된 플릿 목록에서 삭제할 플릿을 선택합니다.
3. 삭제를 선택합니다.
4. **delete**를 입력하여 삭제를 확인합니다.

5. 삭제 선택합니다.

작업에 플릿 또는 컴퓨팅 할당

기본적으로 워크플로 작업은 Amazon EC2 컴퓨팅 유형의 Linux.x86-64.Large 온디맨드 플릿을 사용합니다. 대신 프로비저닝된 플릿을 사용하거나 Linux.x86-64.2XLarge와 같은 다른 온디맨드 플릿을 사용하려면 다음 지침을 사용합니다.

Visual

시작하기 전 준비 사항

- 프로비저닝된 플릿을 할당하려면 먼저 프로비저닝된 플릿을 생성해야 합니다. 자세한 내용은 [프로비저닝된 플릿 생성](#) 섹션을 참조하세요.

프로비저닝된 플릿 또는 다른 플릿 유형 작업 할당

- <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
- 프로젝트를 선택합니다.
- 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
- 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
- 편집을 선택합니다.
- 비주얼을 선택합니다.
- 워크플로 다이어그램에서 프로비저닝된 플릿 또는 새 플릿 유형을 할당할 작업을 선택합니다.
- 구성 탭을 선택합니다.
- 컴퓨팅 플릿에서 다음을 수행합니다.

워크플로 또는 워크플로 작업을 실행할 시스템 또는 플릿을 지정합니다. 온디맨드 플릿의 경우 작업이 시작되면 워크플로가 필요한 리소스를 프로비저닝하고 작업이 완료되면 시스템이 파괴됩니다. 온디맨드 플릿의 예시: Linux.x86-64.Large, Linux.x86-64.XLarge. 온디맨드 플릿에 대한 자세한 내용은 [온디맨드 플릿 속성](#) 섹션을 참조하세요.

프로비저닝된 플릿을 사용하면 워크플로 작업을 실행하도록 전용 시스템 세트를 구성할 수 있습니다. 이러한 시스템은 유휴 상태로 유지되므로 작업을 즉시 처리할 수 있습니다. 프로비저닝된 플릿에 대한 자세한 내용은 [프로비저닝된 플릿 속성](#) 섹션을 참조하세요.

Fleet 생략 시 기본값은 `Linux.x86-64.Large`입니다.

10. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
11. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

시작하기 전 준비 사항

- 프로비저닝된 플릿을 할당하려면 먼저 프로비저닝된 플릿을 생성해야 합니다. 자세한 내용은 [프로비저닝된 플릿 생성](#) 섹션을 참조하세요.

프로비저닝된 플릿 또는 다른 플릿 유형 작업 할당

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 프로비저닝된 플릿 또는 새 플릿 유형을 할당할 작업을 찾습니다.
8. 작업에서 Compute 속성을 추가하고 플릿 이름 또는 온디맨드 플릿 유형으로 Fleet을 설정합니다. 자세한 내용은 작업에 대한 [빌드 및 테스트 작업 YAML](#)의 Fleet 속성 설명을 참조하세요.
9. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
10. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

작업 간에 컴퓨팅 공유

기본적으로 워크플로의 작업은 [플릿](#)의 개별 인스턴스에서 실행됩니다. 이 동작은 입력 상태에 대한 격리 및 예측 가능성을 갖춘 작업을 제공합니다. 기본 동작에는 작업 간에 파일 및 변수와 같은 컨텍스트를 공유하기 위한 명시적 구성이 필요합니다.

컴퓨팅 공유는 동일한 인스턴스의 워크플로에서 모든 작업을 실행할 수 있는 기능입니다. 컴퓨팅 공유를 사용하면 인스턴스 프로비저닝에 소요되는 시간이 줄어들기 때문에 워크플로 런타임이 빨라질 수 있습니다. 추가 워크플로 구성 없이 작업 간에 파일(아티팩트)을 공유할 수도 있습니다.

컴퓨팅 공유를 사용하여 워크플로를 실행할 때 기본 또는 지정된 플릿의 인스턴스는 해당 워크플로의 모든 작업 기간 동안 예약됩니다. 워크플로 실행이 완료되면 인스턴스 예약이 해제됩니다.

주제

- [공유 컴퓨팅에서 여러 작업 실행](#)
- [컴퓨팅 공유 고려 사항](#)
- [컴퓨팅 공유 켜기](#)
- [예시](#)

공유 컴퓨팅에서 여러 작업 실행

워크플로 수준에서 정의 YAML의 Compute 속성을 사용하여 작업의 플릿 및 컴퓨팅 공유 속성을 모두 지정할 수 있습니다. CodeCatalyst의 시각적 편집기를 사용하여 컴퓨팅 속성을 구성할 수도 있습니다. 플릿을 지정하려면 기존 플릿의 이름을 설정하고 컴퓨팅 유형을 EC2로 설정하고 컴퓨팅 공유를 켭니다.

Note

컴퓨팅 공유는 컴퓨팅 유형이 EC2로 설정되어 있고 Windows Server 2022 운영 체제에서는 지원되지 않는 경우에만 지원됩니다. 컴퓨팅 플릿, 컴퓨팅 유형 및 속성에 대한 자세한 내용은 섹션을 참조하세요 [컴퓨팅 및 런타임 이미지 구성](#).

Note

프리 티어에 있고 워크플로 정의 YAML에서 Linux.x86-64.XLarge 또는 Linux.x86-64.2XLarge 플릿을 수동으로 지정하는 경우에도 작업은 기본 플릿 (Linux.x86-64.Large)에서 계속 실행됩니다. 컴퓨팅 가용성 및 요금에 대한 자세한 내용은 [계층 옵션의 표](#)를 참조하세요.

컴퓨팅 공유가 켜져 있으면 워크플로 소스가 포함된 폴더가 작업 간에 자동으로 복사됩니다. 워크플로 정의(YAML 파일) 전체에서 출력 아티팩트를 구성하고 입력 아티팩트로 참조할 필요가 없습니다. 워크

폴로 작성자는 컴퓨팅 공유를 사용하지 않는 것처럼 입력과 출력을 사용하여 환경 변수를 연결해야 합니다. 워크폴로 소스 외부의 작업 간에 폴더를 공유하려면 파일 캐싱을 고려하세요. 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 및 [워크폴로 실행 간 파일 캐싱](#) 섹션을 참조하세요.

워크폴로 정의 파일이 있는 소스 리포지토리는 WorkflowSource 레이블로 식별됩니다. 컴퓨팅 공유를 사용하는 동안 워크폴로 소스는 이를 참조하는 첫 번째 작업에서 다운로드되고 워크폴로 실행에서 사용할 후속 작업에 자동으로 사용할 수 있습니다. 파일 추가, 수정 또는 제거와 같은 작업에 의해 워크폴로 소스가 포함된 폴더에 대한 모든 변경 사항은 워크폴로의 후속 작업에도 표시됩니다. 컴퓨팅 공유를 사용하지 않고도 워크폴로 작업에서 워크폴로 소스 폴더에 있는 파일을 최대한 참조할 수 있습니다. 자세한 내용은 [소스 리포지토리 파일 참조](#) 섹션을 참조하세요.

Note

컴퓨팅 공유 워크폴로는 엄격한 작업 시퀀스를 지정해야 하므로 병렬 작업을 설정할 수 없습니다. 출력 아티팩트는 시퀀스의 모든 작업에서 구성할 수 있지만 입력 아티팩트는 지원되지 않습니다.

컴퓨팅 공유 고려 사항

컴퓨팅 공유를 사용하여 워크폴로를 실행하여 워크폴로 실행을 가속화하고 동일한 인스턴스를 사용하는 워크폴로의 작업 간에 컨텍스트를 공유할 수 있습니다. 컴퓨팅 공유 사용이 시나리오에 적합한지 확인하려면 다음을 고려하세요.

	컴퓨팅 공유	컴퓨팅 공유 없음
컴퓨팅 유형	Amazon EC2	Amazon EC2, AWS Lambda
DB 인스턴스 프로비저닝	동일한 인스턴스에서 실행되는 작업	별도의 인스턴스에서 실행되는 작업
운영 체제	Amazon Linux 2	Amazon Linux 2, Windows Server 2022(빌드 작업만 해당)
파일 참조	\$CATALYST_SOURCE_DIR_WorkflowSource , /sources/WorkflowSource/	\$CATALYST_SOURCE_DIR_WorkflowSource , /sources/WorkflowSource/

	컴퓨팅 공유	컴퓨팅 공유 없음
워크플루 구조	작업은 순차적으로만 실행할 수 있습니다.	작업을 병렬로 실행할 수 있습니다.
워크플로 작업 전반의 데이터 액세스	캐시된 워크플로 소스 액세스 (WorkflowSource)	공유 아티팩트의 출력에 액세스(추가 구성 필요)

컴퓨팅 공유 켜기

다음 지침에 따라 워크플로에 대한 컴퓨팅 공유를 켭니다.

Visual

시각적 편집기를 사용하여 컴퓨팅 공유를 켜려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 워크플로 속성을 선택합니다.
8. 컴퓨팅 유형 드롭다운 메뉴에서 EC2를 선택합니다.
9. (선택 사항) 컴퓨팅 플릿 - 선택 사항 드롭다운 메뉴에서 워크플로 작업을 실행하는 데 사용할 플릿을 선택합니다. 온디맨드 플릿을 선택하거나 프로비저닝된 플릿을 생성하고 선택할 수 있습니다. 자세한 내용은 [프로비저닝된 플릿 생성](#) 및 [작업에 플릿 또는 컴퓨팅 할당](#) 섹션을 참조하세요.
10. 토글을 전환하여 컴퓨팅 공유를 켜고 워크플로에서 동일한 플릿에서 작업을 실행합니다.
11. (선택 사항) 워크플로의 실행 모드를 선택합니다. 자세한 내용은 [실행의 대기열 동작 구성](#) 섹션을 참조하세요.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 컴퓨팅 공유를 켜려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 컴퓨팅 공유를 켜서 SharedInstance 필드를 TRUE로 설정하고 Type을 EC2로 설정합니다. 워크플로 작업을 실행하는 데 사용할 컴퓨팅 플릿을 Fleet으로 설정합니다. 온디맨드 플릿을 선택하거나 프로비저닝된 플릿을 생성하고 선택할 수 있습니다. 자세한 내용은 [프로비저닝된 플릿 생성](#) 및 [작업에 플릿 또는 컴퓨팅 할당](#) 섹션을 참조하세요.

워크플로 YAML에서 다음과 유사한 코드를 추가합니다.

```
Name: MyWorkflow
SchemaVersion: "1.0"
Compute: # Define compute configuration.
  Type: EC2
  Fleet: MyFleet # Optionally, choose an on-demand or provisioned fleet.
  SharedInstance: true # Turn on compute sharing. Default is False.
Actions:
  BuildFirst:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: ...
        ...
```

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

예시

주제

- [예시: Amazon S3 게시](#)

예시: Amazon S3 게시

다음 워크플로 예시는 Amazon S3 게시 작업을 두 가지 방식으로 수행하는 방법을 보여줍니다. 먼저 입력 아티팩트를 사용한 다음 컴퓨팅 공유를 사용합니다. 컴퓨팅 공유를 사용하면 캐시된 WorkflowSource에 액세스할 수 있으므로 입력 아티팩트가 필요하지 않습니다. 또한 빌드 작업의 출력 아티팩트는 더 이상 필요하지 않습니다. S3 게시 작업은 명시적 DependsOn 속성을 사용하여 순차적 작업을 유지하도록 구성됩니다. 빌드 작업은 S3 게시 작업이 실행되려면 성공적으로 실행되어야 합니다.

- 컴퓨팅 공유가 없으면 입력 아티팩트를 사용하고 출력을 후속 작업과 공유해야 합니다.

```
Name: S3PublishUsingInputArtifact
SchemaVersion: "1.0"
Actions:
  Build:
    Identifier: aws/build@v1
    Outputs:
      Artifacts:
        - Name: ArtifactToPublish
          Files: [output.zip]
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: ./build.sh # Build script that generates output.zip
  PublishToS3:
    Identifier: aws/s3-publish@v1
    Inputs:
      Artifacts:
        - ArtifactToPublish
    Environment:
      Connections:
        - Role: codecatalyst-deployment-role
          Name: dev-deployment-role
```

```

Name: dev-connection
Configuration:
  SourcePath: output.zip
  DestinationBucketName: amzn-s3-demo-bucket

```

- SharedInstance를 TRUE로 설정하여 컴퓨팅 공유를 사용하는 경우 동일한 인스턴스에서 여러 작업을 실행하고 단일 워크플로 소스를 지정하여 아티팩트를 공유할 수 있습니다. 입력 아티팩트는 필요하지 않으며 지정할 수 없습니다.

```

Name: S3PublishUsingComputeSharing
SchemaVersion: "1.0"
Compute:
  Type: EC2
  Fleet: dev-fleet
  SharedInstance: TRUE
Actions:
  Build:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: ./build.sh # Build script that generates output.zip
  PublishToS3:
    Identifier: aws/s3-publish@v1
    DependsOn:
      - Build
    Environment:
      Connections:
        - Role: codecatalyst-deployment-role
          Name: dev-deployment-role
      Name: dev-connection
    Configuration:
      SourcePath: output.zip
      DestinationBucketName: amzn-s3-demo-bucket

```

런타임 환경 이미지 지정

런타임 환경 이미지는 CodeCatalyst가 워크플로 작업을 실행하는 Docker 컨테이너입니다. Docker 컨테이너는 선택한 컴퓨팅 플랫폼을 기반으로 실행되며, 운영 체제와 AWS CLI, Node.js 및 .tar와 같은 워크플로 작업에 필요할 수 있는 추가 도구를 포함합니다.

기본적으로 워크플로 작업은 CodeCatalyst에서 제공하고 유지 관리하는 [활성 이미지](#) 중 하나에서 실행됩니다. 빌드 및 테스트 작업만 사용자 지정 이미지를 지원합니다. 자세한 내용은 [작업에 사용자 지정 런타임 환경 Docker 이미지 할당](#) 단원을 참조하십시오.

주제

- [활성 이미지](#)
- [활성 이미지에 필요한 도구가 포함되지 않은 경우 어떻게 해야 하나요?](#)
- [작업에 사용자 지정 런타임 환경 Docker 이미지 할당](#)
- [예제](#)

활성 이미지

활성 이미지는 CodeCatalyst에서 완전히 지원되며 사전 설치된 도구를 포함하는 런타임 환경 이미지입니다. 현재 두 가지 활성 이미지 세트가 있습니다. 하나는 2024년 3월에, 다른 하나는 2022년 11월에 릴리스되었습니다.

작업이 2024년 3월 또는 2022년 11월 이미지 중 사용할 이미지는 작업에 따라 달라집니다.

- 2024년 3월 26일 또는 그 이후에 워크플로에 추가된 빌드 및 테스트 작업은 [2024년 3월 이미지](#)를 명시적으로 지정하는 Container 섹션을 YAML 정의에 포함합니다. 선택적으로 Container 섹션을 제거하여 [2022년 11월 이미지](#)로 되돌릴 수 있습니다.
- 2024년 3월 26일 이전에 워크플로에 추가된 빌드 및 테스트 작업은 YAML 정의에 Container 섹션을 포함하지 않으므로 [2022년 11월 이미지](#)를 사용합니다. 2022년 11월 이미지를 유지하거나 업그레이드할 수 있습니다. 이미지를 업그레이드하려면 시각적 편집기에서 작업을 열고 구성 탭을 선택한 다음 런타임 환경 Docker 이미지 드롭다운 목록에서 2024년 3월 이미지를 선택합니다. 이 항목을 선택하면 작업의 YAML 정의에 Container 섹션이 추가되며, 이 섹션은 적절한 2024년 3월 이미지로 채워집니다.
- 다른 모든 작업은 [2022년 11월 이미지](#) 또는 [2024년 3월 이미지](#)를 사용합니다. 자세한 내용은 작업 문서를 참조하세요.

주제

- [2024년 3월 이미지](#)
- [2022년 11월 이미지](#)

2024년 3월 이미지

2024년 3월 이미지는 CodeCatalyst에서 제공하는 최신 이미지입니다. 컴퓨팅 유형/플랫폼 조합당 2024년 3월 이미지가 하나 있습니다.

다음 표는 각 2024년 3월 이미지에 설치된 도구를 보여줍니다.

2024년 3월 이미지 도구

도구	Linux x86_64용 CodeCatalyst Amazon EC2 - CodeCatalystLinux_x86_64:2024_03	Linux x86_64용 CodeCatalyst Lambda - CodeCatalystLinuxLambda_x86_64:2024_03	Linux Arm64용 CodeCatalyst Amazon EC2 - CodeCatalystLinux_Arm64:2024_03	Linux Arm64용 CodeCatalyst Lambda - CodeCatalystLinuxLambda_Arm64:2024_03
AWS CLI	2.15.17	2.15.17	2.15.17	2.15.17
AWS Copilot CLI	1.32.1	1.32.1	1.32.1	1.32.1
Docker	24.0.9	해당 사항 없음	24.0.9	해당 사항 없음
Docker Compose	2.23.3	해당 사항 없음	2.23.3	해당 사항 없음
Git	2.43.0	2.43.0	2.43.0	2.43.0
Go	1.21.5	1.21.5	1.21.5	1.21.5
Gradle	8.5	8.5	8.5	8.5
Java	Corretto17	Corretto17	Corretto17	Corretto17
Maven	3.9.6	3.9.6	3.9.6	3.9.6
Node.js	18.19.0	18.19.0	18.19.0	18.19.0
npm	10.2.3	10.2.3	10.2.3	10.2.3

도구	Linux x86_64 용 CodeCatalyst Amazon EC2 - CodeCatalystLinux_ x86_64:2024_03	Linux x86_64 용 CodeCatalyst Lambda - CodeCatalystLinuxL ambda_x86_64:2024_03	Linux Arm64용 CodeCatalyst Amazon EC2 - CodeCatalystLinux_ Arm64:2024_03	Linux Arm64용 CodeCatalyst Lambda - CodeCatalystLinux_ ambda_Ar 64:2024_03
Python	3.9.18	3.9.18	3.9.18	3.9.18
Python3	3.11.6	3.11.6	3.11.6	3.11.6
pip	22.3.1	22.3.1	22.3.1	22.3.1
.NET	8.0.100	8.0.100	8.0.100	8.0.100

2022년 11월 이미지

컴퓨팅 유형/플랫폼 조합당 2022년 11월 이미지가 하나 있습니다. [프로비저닝된 플릿](#)을 구성한 경우 빌드 작업과 함께 사용할 수 있는 2022년 11월 Windows 이미지도 있습니다.

다음 표는 각 2022년 11월 이미지에 설치된 도구를 보여줍니다.

2022년 11월 이미지 도구

도구	Linux x86_64 용 CodeCatalyst Amazon EC2 - CodeCatalystLinux_ x86_64:2022_11	Linux x86_64 용 CodeCatalyst Lambda - CodeCatalystLinuxL ambda_x86_64:2022_11	Linux Arm64용 CodeCatalyst Amazon EC2 - CodeCatalystLinux_ Arm64:2022_11	Linux Arm64용 CodeCatalyst Lambda - CodeCatalystLinux_ ambda_Ar 64:2022_11
AWS CLI	2.15.17	2.15.17	2.15.17	2.15.17
AWS Copilot CLI	0.6.0	0.6.0	해당 사항 없음	해당 사항 없음
Docker	23.01	해당 사항 없음	23.0.1	해당 사항 없음

도구	Linux x86_64용 CodeCatalyst Amazon EC2 - CodeCatalystLinux_x86_64:2022_11	Linux x86_64용 CodeCatalyst Lambda - CodeCatalystLinuxLambda_x86_64:2022_11	Linux Arm64용 CodeCatalyst Amazon EC2 - CodeCatalystLinux_Arm64:2022_11	Linux Arm64용 CodeCatalyst Lambda - CodeCatalystLinuxLambda_Arm64:2022_11
Docker Compose	2.16.0	해당 사항 없음	2.16.0	해당 사항 없음
Git	2.40.0	2.40.0	2.39.2	2.39.2
Go	1.20.2	1.20.2	1.20.1	1.20.1
Gradle	8.0.2	8.0.2	8.0.1	8.0.1
Java	Corretto17	Corretto17	Corretto17	Corretto17
Maven	3.9.4	3.9.4	3.9.0	3.9.0
Node.js	16.20.2	16.20.2	16.19.1	16.14.2
npm	8.19.4	8.19.4	8.19.3	8.5.0
Python	3.9.15	2.7.18	3.11.2	2.7.18
Python3	해당 사항 없음	3.9.15	해당 사항 없음	3.11.2
pip	22.2.2	22.2.2	23.0.1	23.0.1
.NET	6.0.407	6.0.407	6.0.406	6.0.406

활성 이미지에 필요한 도구가 포함되지 않은 경우 어떻게 해야 하나요?

CodeCatalyst에서 제공하는 [활성 이미지](#)에 필요한 도구가 포함되어 있지 않은 경우 몇 가지 옵션이 있습니다.

- 필요한 도구가 포함된 사용자 지정 런타임 환경 Docker 이미지를 제공할 수 있습니다. 자세한 내용은 [작업에 사용자 지정 런타임 환경 Docker 이미지 할당](#) 단원을 참조하십시오.

Note

사용자 지정 런타임 환경 Docker 이미지를 제공하려면 사용자 지정 이미지에 Git이 설치되어 있는지 확인합니다.

- 워크플로의 빌드 또는 테스트 작업을 통해 필요한 도구를 설치할 수 있습니다.

예를 들어 빌드 또는 테스트 작업의 YAML 코드 Steps 섹션에 다음 지침을 포함할 수 있습니다.

Configuration:

Steps:

- Run: `./setup-script`

그런 다음 `setup-script` 명령은 다음 스크립트를 실행하여 노드 패키지 관리자(npm)를 설치합니다.

```
#!/usr/bin/env bash
echo "Setting up environment"

touch ~/.bashrc
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash
source ~/.bashrc
nvm install v16.1.0
source ~/.bashrc
```

빌드 작업 YAML에 대한 자세한 내용은 [빌드 및 테스트 작업 YAML](#) 섹션을 참조하세요.

작업에 사용자 지정 런타임 환경 Docker 이미지 할당

CodeCatalyst에서 제공하는 [활성 이미지](#)를 사용하지 않으려면 사용자 지정 런타임 환경 Docker 이미지를 제공할 수 있습니다. 사용자 지정 이미지를 제공하려는 경우 Git이 설치되어 있는지 확인합니다. 이미지는 Docker Hub, Amazon Elastic Container Registry 또는 모든 퍼블릭 리포지토리에 저장될 수 있습니다.

사용자 지정 Docker 이미지를 생성하는 방법을 알아보려면 Docker 설명서의 [애플리케이션 컨테이너화](#)를 참조하세요.

다음 지침을 사용하여 사용자 지정 런타임 환경 Docker 이미지를 작업에 할당합니다. 이미지를 지정하면 작업이 시작될 때 CodeCatalyst가 해당 이미지를 컴퓨팅 플랫폼에 배포합니다.

Note

CloudFormation 스택 배포, ECS에 배포, GitHub 작업 등의 작업은 사용자 지정 런타임 환경 Docker 이미지를 지원하지 않습니다. 사용자 지정 런타임 환경 Docker 이미지도 Lambda 컴퓨팅 유형을 지원하지 않습니다.

Visual

시각적 편집기를 사용하여 사용자 지정 런타임 환경 Docker 이미지를 할당하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
3. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
4. 편집을 선택합니다.
5. 비주얼을 선택합니다.
6. 워크플로 다이어그램에서 사용자 지정 런타임 환경 Docker 이미지를 사용할 작업을 선택합니다.
7. 구성 탭을 선택합니다.
8. 하단 근처에 다음 필드를 입력합니다.

런타임 환경 Docker 이미지 - 선택 사항

이미지가 저장되는 레지스트리를 지정합니다. 유효한 값으로는 다음이 포함됩니다.

- CODECATALYST (YAML 편집기)

이미지는 CodeCatalyst 레지스트리에 저장됩니다.

- Docker Hub(시각 편집기) 또는 DockerHub(YAML 편집기)

이미지는 Docker Hub 이미지 레지스트리에 저장됩니다.

- 기타 레지스트리(시각 편집기) 또는 Other(YAML 편집기)

이미지는 사용자 지정 이미지 레지스트리에 저장됩니다. 공개적으로 사용 가능한 레지스트리를 사용할 수 있습니다.

- Amazon Elastic Container Registry(시각 편집기) 또는 ECR(YAML 편집기)

이미지는 Amazon Elastic Container Registry 이미지 리포지토리에 저장됩니다. Amazon ECR 리포지토리에서 이미지를 사용하려면 이 작업에서 Amazon ECR에 대한 액세스 권한이 필요합니다. 이 액세스를 활성화하려면 다음 권한과 사용자 지정 신뢰 정책을 포함하는 [IAM 역할](#)을 생성해야 합니다. (원하는 경우 권한 및 정책을 포함하도록 기존 역할을 수정할 수 있습니다.)

IAM 역할에는 역할 정책의 다음 권한이 포함되어야 합니다.

- `ecr:BatchCheckLayerAvailability`
- `ecr:BatchGetImage`
- `ecr:GetAuthorizationToken`
- `ecr:GetDownloadUrlForLayer`

IAM 역할에는 다음과 같은 사용자 지정 신뢰 정책이 포함되어야 합니다.

IAM 역할 생성에 대한 자세한 정보는 IAM 사용자 설명서의 [사용자 지정 신뢰 정책을 사용하여 역할 생성\(콘솔\)](#)을 참조하세요.

역할을 생성한 후에는 환경을 통해 작업에 할당해야 합니다. 자세한 내용은 [작업과 환경 연결](#) 단원을 참조하십시오.

ECR 이미지 URL, Docker Hub 이미지 또는 이미지 URL

다음 중 하나를 지정하세요.

- CODECATALYST 레지스트리를 사용하는 경우 이미지를 다음 [활성 이미지](#) 중 하나로 설정합니다.
- `CodeCatalystLinux_x86_64:2024_03`
- `CodeCatalystLinux_x86_64:2022_11`
- `CodeCatalystLinux_Arm64:2024_03`
- `CodeCatalystLinux_Arm64:2022_11`
- `CodeCatalystLinuxLambda_x86_64:2024_03`
- `CodeCatalystLinuxLambda_x86_64:2022_11`
- `CodeCatalystLinuxLambda_Arm64:2024_03`
- `CodeCatalystLinuxLambda_Arm64:2022_11`
- `CodeCatalystWindows_x86_64:2022_11`

- Docker Hub 레지스트리를 사용하는 경우 이미지를 Docker Hub 이미지 이름과 선택적 태그로 설정합니다.

예시: postgres:latest

- Amazon ECR 레지스트리를 사용하는 경우 이미지를 Amazon ECR 레지스트리 URI로 설정합니다.

예시: 111122223333.dkr.ecr.us-west-2.amazonaws.com/codecatalyst-ecs-image-repo

- 사용자 지정 레지스트리를 사용하는 경우 이미지를 사용자 지정 레지스트리에서 예상되는 값으로 설정합니다.

9. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
10. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 사용자 지정 런타임 환경 Docker 이미지를 할당하려면

1. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
2. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
3. 편집을 선택합니다.
4. YAML을 선택합니다.
5. 런타임 환경 Docker 이미지를 할당하려는 작업을 찾습니다.
6. 작업에 Container 섹션과 기본 Registry 및 Image 속성을 추가합니다. 자세한 내용은 작업에 대한 [작업](#)의 Container, Registry, Image 속성에 대한 설명을 참조하세요.
7. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
8. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

예제

다음 예시에서는 워크플로 정의 파일의 작업에 사용자 지정 런타임 환경 Docker 이미지를 할당하는 방법을 보여줍니다.

주제

- [예시: 사용자 지정 런타임 환경 Docker 이미지를 사용하여 Amazon ECR에서 Node.js 18에 대한 지원 추가](#)
- [예시: 사용자 지정 런타임 환경 Docker 이미지를 사용하여 Docker Hub에서 Node.js 18에 대한 지원 추가](#)

예시: 사용자 지정 런타임 환경 Docker 이미지를 사용하여 Amazon ECR에서 Node.js 18에 대한 지원 추가

다음 예시에서는 사용자 지정 런타임 환경 Docker 이미지를 사용하여 [Amazon ECR](#)에서 Node.js 18에 대한 지원을 추가하는 방법을 보여줍니다.

Configuration:

Container:

Registry: ECR

Image: public.ecr.aws/amazonlinux/amazonlinux:2023

예시: 사용자 지정 런타임 환경 Docker 이미지를 사용하여 Docker Hub에서 Node.js 18에 대한 지원 추가

다음 예시에서는 사용자 지정 런타임 환경 Docker 이미지를 사용하여 [Docker Hub](#)에서 Node.js 18에 대한 지원을 추가하는 방법을 보여줍니다.

Configuration:

Container:

Registry: DockerHub

Image: node:18.18.2

워크플로에 소스 리포지토리 연결

입력 소스라고도 하는 소스는 [워크플로 작업](#)이 작업을 수행하는 데 필요한 파일을 얻기 위해 연결되는 소스 리포지토리입니다. 예를 들어 워크플로 작업은 애플리케이션을 빌드하기 위해 애플리케이션 소스 파일을 얻기 위해 소스 리포지토리에 연결할 수 있습니다.

CodeCatalyst 워크플로는 다음 소스를 지원합니다.

- CodeCatalyst 소스 리포지토리 - 자세한 내용은 [CodeCatalyst의 소스 리포지토리로 코드 저장 및 협업](#) 섹션을 참조하세요.
- GitHub 리포지토리, Bitbucket 리포지토리 및 GitLab 프로젝트 리포지토리 - 자세한 내용은 [CodeCatalyst에서 확장 프로그램이 있는 프로젝트에 기능 추가](#) 섹션을 참조하세요.

주제

- [워크플로 파일의 소스 리포지토리 지정](#)
- [워크플로 작업의 소스 리포지토리 지정](#)
- [소스 리포지토리 파일 참조](#)
- ['BranchName' 및 'CommitId' 변수](#)

워크플로 파일의 소스 리포지토리 지정

다음 지침에 따라 워크플로 정의 파일을 저장할 CodeCatalyst 소스 리포지토리를 지정합니다. GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 지정하려면 대신 [CodeCatalyst에서 확장 프로그램이 있는 프로젝트에 기능 추가](#)를 참조하세요.

워크플로 정의 파일이 있는 소스 리포지토리는 WorkflowSource 레이블로 식별됩니다.

Note

워크플로 정의 파일을 처음 커밋할 때 워크플로 정의 파일이 있는 소스 리포지토리를 지정합니다. 이 커밋 후에는 리포지토리와 워크플로 정의 파일이 영구적으로 서로 연결됩니다. 초기 커밋 후 리포지토리를 변경하는 유일한 방법은 다른 리포지토리에서 워크플로를 다시 만드는 것입니다.

워크플로 정의 파일을 저장할 소스 리포지토리 지정

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로 생성을 선택하여 워크플로를 생성합니다. 자세한 내용은 [워크플로 생성](#) 섹션을 참조하세요.

워크플로 생성 과정에서 워크플로 정의 파일을 저장할 CodeCatalyst 리포지토리, 브랜치 및 폴더를 지정할 수 있습니다.

워크플로 작업의 소스 리포지토리 지정

다음 지침에 따라 워크플로 작업에 사용할 소스 리포지토리를 지정합니다. 시작 시 작업은 구성된 소스 리포지토리에 있는 파일을 아티팩트로 번들링하고, 작업이 실행 중인 [컨타임 환경 Docker 이미지](#)로 아티팩트를 다운로드한 다음 다운로드한 파일을 사용하여 처리를 완료합니다.

Note

현재 워크플로 작업 내에서 소스 리포지토리는 워크플로 정의 파일이 있는 소스 리포지토리 (.codecatalyst/workflows/ 디렉터리 또는 그 하위 디렉터리 중 하나)를 하나만 지정할 수 있습니다. 이 소스 리포지토리는 WorkflowSource 레이블로 표시됩니다.

Visual

작업이 사용할 소스 리포지토리 지정(시각적 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 워크플로 다이어그램에서 소스를 지정하려는 작업을 선택합니다.
8. 입력을 선택합니다.
9. 소스 - 선택 사항으로 다음을 수행합니다.

작업에 필요한 소스 리포지토리를 나타내는 레이블을 지정합니다. 현재 지원되는 유일한 레이블은 워크플로 정의 파일이 저장되는 소스 리포지토리를 나타내는 WorkflowSource입니다.

소스를 생략하는 경우 **action-name**/Inputs/Artifacts 아래에 하나 이상의 입력 아티팩트를 지정해야 합니다.

소스에 대한 자세한 내용은 [워크플로에 소스 리포지토리 연결](#) 섹션을 참조하세요.

10. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
11. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

작업이 사용할 소스 리포지토리 지정(YAML 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 작업에서 다음과 유사한 코드를 추가합니다.

```
action-name:
  Inputs:
    Sources:
      - WorkflowSource
```

자세한 내용은 작업에 대한 [워크플로 YAML 정의](#)의 Sources 속성 설명을 참조하세요.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

소스 리포지토리 파일 참조

소스 리포지토리에 있는 파일이 있고 워크플로 작업 중 하나에서 이러한 파일을 참조해야 하는 경우 다음 절차를 완료하세요.

Note

또한 [아티팩트의 파일 참조](#) 섹션도 참조하세요.

소스 리포지토리에 저장된 파일 참조

- 파일을 참조하려는 작업에서 다음과 유사한 코드를 추가합니다.

```
Actions:
```



```

My-action:
  Inputs:
    Sources:
      - WorkflowSource
  Configuration:
    Steps:
      - run: cd my-app && cat file1.jar

```

이전 코드에서 작업은 WorkflowSource 소스 리포지토리 루트의 my-app 디렉터리에서 file1.jar 파일을 찾아 표시합니다.

'BranchName' 및 'CommitId' 변수

CodeCatalyst 소스는 워크플로가 실행될 때 BranchName 및 CommitId 변수를 생성하고 설정합니다. 이를 사전 정의된 변수라고 합니다. 이러한 변수에 대한 자세한 내용은 다음 표를 참조하세요.

워크플로에서 이러한 변수를 참조하는 방법에 대한 자세한 내용은 [사전 정의된 변수 사용](#) 섹션을 참조하세요.

Key(키)	값
CommitId	워크플로 실행이 시작된 시점의 리포지토리 상태를 나타내는 커밋 ID입니다. 예시: example3819261db00a3ab59468c8b 또한 예시: 'CommitId' 사전 정의된 변수 참조 섹션도 참조하세요.
BranchName	워크플로 실행이 시작된 브랜치의 이름입니다. 예: main, feature/branch , test-LiJuan 또한 예시: 'BranchName' 사전 정의된 변수 참조 섹션도 참조하세요.

워크플로에 패키지 리포지토리 연결

패키지는 소프트웨어와 모든 종속성을 해결하고 소프트웨어를 설치하는 데 필요한 메타데이터를 포함한 번들입니다. CodeCatalyst는 npm 패키지 형식을 지원합니다.

패키지는 다음으로 구성됩니다.

- 이름(예: webpack는 널리 사용되는 npm 패키지의 이름)
- 선택적 [네임스페이스](#)(예: @types/node의 @types)
- [버전](#) 세트(예: 1.0.0, 1.0.1, 1.0.2)
- 패키지 수준 메타데이터(예: npm dist 태그)

CodeCatalyst에서는 워크플로에서 CodeCatalyst 패키지 리포지토리에 패키지를 게시하고 패키지를 사용할 수 있습니다. CodeCatalyst 패키지 리포지토리로 빌드 또는 테스트 작업을 구성하여 지정된 리포지토리에서 패키지를 푸시 및 풀링하도록 작업의 npm 클라이언트를 자동으로 구성할 수 있습니다.

패키지에 대한 자세한 내용은 [CodeCatalyst에서 소프트웨어 패키지 게시 및 공유](#) 섹션을 참조하세요.

Note

현재 빌드 및 테스트 작업은 CodeCatalyst 패키지 리포지토리를 지원합니다.

주제

- [자습서: 패키지 리포지토리에서 가져오기](#)
- [워크플로에서 CodeCatalyst 패키지 리포지토리 지정](#)
- [워크플로 작업에서 권한 부여 토큰 사용](#)
- [예시: 워크플로의 패키지 리포지토리](#)

자습서: 패키지 리포지토리에서 가져오기

이 자습서에서는 [CodeCatalyst 패키지 리포지토리](#)에서 종속성을 가져오는 애플리케이션을 실행하는 워크플로를 생성하는 방법을 알아봅니다. 이 애플리케이션은 CodeCatalyst 로그에 'Hello World' 메시지를 출력하는 간단한 Node.js 앱입니다. 이 애플리케이션에는 단일 종속성으로 [lodash](#) npm 패키지가 있습니다. lodash 패키지는 hello-world 문자열을 Hello World로 변환하는 데 사용됩니다. 이 패키지의 버전 4.17.20을 사용합니다.

애플리케이션 및 워크플로를 설정한 후 CodeCatalyst가 퍼블릭 외부 레지스트리(npmjs.com)에서 CodeCatalyst 패키지 리포지토리로 lodash의 추가 버전을 가져오는 것을 차단하도록 구성합니다. 그런 다음 lodash의 추가 버전이 성공적으로 차단되었는지 테스트합니다.

이 자습서를 마치면 워크플로가 CodeCatalyst 내부 및 외부의 패키지 리포지토리와 상호 작용하는 방식을 잘 이해하여 패키지를 검색할 수 있습니다. 또한 npm, 패키지 리포지토리, 워크플로 및 애플리케이션의 package.json 파일 간에 발생하는 백그라운드 상호 작용도 이해해야 합니다.

주제

- [사전 조건](#)
- [1단계: 소스 리포지토리 생성](#)
- [2단계: CodeCatalyst 및 게이트웨이 패키지 리포지토리 생성](#)
- [3단계: 'Hello World' 애플리케이션 생성](#)
- [4단계: 'Hello World'를 실행하는 워크플로 생성](#)
- [5단계: 워크플로 확인](#)
- [6단계: npmjs.com 가져오기 차단](#)
- [7단계: 차단 기능 테스트](#)
- [정리](#)

사전 조건

시작하기 전:

- CodeCatalyst 스페이스가 필요합니다. 자세한 내용은 [스페이스 생성](#) 섹션을 참조하세요.
- CodeCatalyst 스페이스에는 다음과 같은 빈 프로젝트가 필요합니다.

```
codecatalyst-package-project
```

처음부터 시작 옵션을 사용하여 이 프로젝트를 생성합니다.

자세한 내용은 [Amazon CodeCatalyst에서 빈 프로젝트 생성](#) 섹션을 참조하세요.

1단계: 소스 리포지토리 생성

이 단계에서는 CodeCatalyst에 소스 리포지토리를 생성합니다. 이 리포지토리에는 index.js 및 package.json 파일과 같은 자습서의 소스 파일이 저장됩니다.

소스 리포지토리에 대한 자세한 정보는 [소스 리포지토리 생성](#) 섹션을 참조하세요.

소스 리포지토리를 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. codecatalyst-package-project 프로젝트로 이동합니다.
3. 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
4. 리포지토리 추가를 선택하고 리포지토리 생성을 선택합니다.
5. 리포지토리 이름에 다음과 같이 입력합니다.

hello-world-app

6. 생성(Create)을 선택합니다.

2단계: CodeCatalyst 및 게이트웨이 패키지 리포지토리 생성

이 단계에서는 CodeCatalyst 프로젝트에서 패키지 리포지토리를 생성하고 이를 게이트웨이 리포지토리와 CodeCatalyst 프로젝트에 연결합니다. 나중에 자습서의 종속성인 lodash를 npmjs.com에서 두 리포지토리로 가져옵니다.

게이트웨이 리포지토리는 CodeCatalyst의 패키지 리포지토리를 퍼블릭 npmjs.com 연결하는 '접착제' 역할을 합니다.

패키지 리포지토리에 대한 자세한 내용은 [CodeCatalyst에서 소프트웨어 패키지 게시 및 공유](#) 섹션을 참조하세요.

Note

이 자습서에서는 CodeCatalyst 패키지 리포지토리 및 게이트웨이 리포지토리라는 용어를 사용하여 다음 절차에서 CodeCatalyst에서 생성한 두 리포지토리를 참조합니다.

CodeCatalyst 패키지 및 게이트웨이 리포지토리를 생성하려면

1. 탐색 창에서 패키지를 선택합니다.
2. 리포지토리 생성을 선택합니다.
3. 리포지토리 이름에 다음과 같이 입력합니다.

codecatalyst-package-repository

4. + 업스트림 리포지토리 선택을 선택합니다.
5. 게이트웨이 리포지토리를 선택합니다.
6. npm-public-registry-gateway 상자에서 생성을 선택합니다.
7. 선택을 선택하세요.
8. 생성(Create)을 선택합니다.

CodeCatalyst는 게이트웨이 리포지토리에 연결된 codecatalyst-package-repository라는 패키지 리포지토리를 생성합니다. 게이트웨이 리포지토리는 npmjs.com 레지스트리에 연결됩니다.

3단계: 'Hello World' 애플리케이션 생성

이 단계에서는 'Hello World' Node.js 애플리케이션을 생성하고 해당 종속성(lodash)을 게이트웨이 및 CodeCatalyst 패키지 리포지토리로 가져옵니다.

애플리케이션을 생성하려면 Node.js와 연결된 npm 클라이언트가 설치된 개발 시스템이 필요합니다.

이 자습서에서는 CodeCatalyst 개발 환경을 개발 시스템으로 사용할 것이라고 가정합니다.

CodeCatalyst 개발 환경을 사용할 필요는 없지만, 깨끗한 작업 환경을 제공하고 Node.js 및 npm가 사전 설치되어 있으며 자습서를 완료하고나서 쉽게 삭제할 수 있으므로 사용하는 것이 좋습니다.

CodeCatalyst 개발 환경에 대한 자세한 내용은 [개발 환경 생성](#) 섹션을 참조하세요.

다음 지침을 사용하여 CodeCatalyst 개발 환경을 시작하고 이를 사용하여 'Hello World' 애플리케이션을 생성합니다.

CodeCatalyst 개발 환경을 시작하려면

1. 탐색 창에서 코드를 선택한 다음 개발 환경을 선택합니다.
2. 상단 근처에서 개발 환경 생성을 선택한 다음 AWS Cloud9(브라우저에서)를 선택합니다.
3. 리포지토리가 hello-world-app로 설정되어 있고 기존 브랜치가 main로 설정되어 있는지 확인합니다. 생성(Create)을 선택합니다.

개발 환경이 새 브라우저 탭에서 시작되고 리포지토리(hello-world-app)가 복제됩니다.

4. 두 CodeCatalyst 브라우저 탭을 모두 열어 두고 다음 절차로 이동합니다.

'Hello World' Node.js 애플리케이션을 생성하려면

1. 개발 환경으로 이동합니다.
2. 터미널 프롬프트에서 hello-world-app 소스 리포지토리 루트 디렉터리로 변경합니다.

```
cd hello-world-app
```

3. Node.js 프로젝트 초기화:

```
npm init -y
```

초기화는 hello-world-app의 루트 디렉터리에 package.json 파일을 생성합니다.

4. 개발 환경의 npm 클라이언트를 CodeCatalyst 패키지 리포지토리에 연결합니다.

1. CodeCatalyst 콘솔로 전환합니다.
2. 탐색 창에서 패키지를 선택합니다.
3. codecatalyst-package-repository를 선택합니다.
4. 리포지토리에 연결을 선택합니다.
5. 토큰 생성을 선택합니다. 개인 액세스 토큰(PAT)이 생성됩니다.
6. 복사를 선택하여 해당 명령을 복사합니다.
7. 개발 환경으로 전환합니다.
8. hello-world-app 디렉터리에 있는지 확인합니다.
9. 해당 명령을 붙여넣습니다. 다음처럼 보일 것입니다.

```
npm set registry=https://packages.us-west-2.codecatalyst.aws/npm/ExampleCompany/
codecatalyst-package-project/codecatalyst-package-repository/ --location project
npm set //packages.us-west-2.codecatalyst.aws/npm/ExampleCompany/codecatalyst-
package-project/hello-world-app/:_authToken=username:token-secret
```

5. lodash 버전 4.17.20 가져오기:

```
npm install lodash@v4.17.20 --save --save-exact
```

npm은 다음 위치에서 lodash 버전 4.17.20을 다음 순서로 찾습니다.

- 개발 환경. 여기에서는 찾을 수 없습니다.
- CodeCatalyst 패키지 리포지토리. 여기에서는 찾을 수 없습니다.

- 게이트웨이 리포지토리. 여기에서는 찾을 수 없습니다.
- npmjs.com. 여기에서 발견합니다.

npm은 lodash를 게이트웨이 리포지토리, CodeCatalyst 패키지 리포지토리 및 개발 환경으로 가져옵니다.

Note

4단계에서 npm 클라이언트를 CodeCatalyst 패키지 리포지토리에 연결하지 않았다면, npm은 lodash를 npmjs.com에서 직접 가져오고 두 리포지토리로는 해당 패키지를 가져오지 않았을 것입니다.

또한 npm은 lodash 종속성을 사용하여 package.json 파일을 업데이트하고 lodash 및 모든 종속성을 포함하는 node_modules 디렉터리를 생성합니다.

6. 개발 환경으로 lodash를 성공적으로 가져왔는지 테스트입니다. 입력:

```
npm list
```

가져오기가 성공했음을 나타내는 다음 메시지가 나타납니다.

```
`-- lodash@4.17.20
```

7. (선택 사항) hello-world-app/package.json 를 열고 ##### ## 표시된 선이 추가되었는지 확인합니다.

```
{
  "name": "hello-world-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  dependencies": {
```

```
"lodash": "4.17.20"
}
}
```

8. /hello-world-app에서 다음 콘텐츠로 index.js 파일을 생성합니다.

 Tip

개발 환경에서 측면 탐색을 사용하여 이 파일을 생성할 수 있습니다.

```
// Importing lodash library
const _ = require('lodash');

// Input string
const inputString = 'hello-world';

// Transforming the string using lodash
const transformedString = _.startCase(inputString.replace('-', ' '));

// Outputting the transformed string to the console
console.log(transformedString);
```

게이트웨이 및 CodeCatalyst 패키지 리포지토리로 'lodash'를 가져왔는지 테스트하려면

1. CodeCatalyst 콘솔로 전환합니다.
2. 탐색 창에서 패키지를 선택합니다.
3. npm-public-registry-gateway를 선택합니다.
4. lodash가 표시되는지 확인합니다. 최신 버전 열은 4.17.20을 나타냅니다.
5. codecatalyst-package-repository에 대해 이 절차를 반복합니다. 가져온 패키지를 보려면 브라우저 창을 새로 고쳐야 할 수 있습니다.

개발 환경에서 'Hello World'를 테스트하려면

1. 개발 환경으로 전환합니다.
2. hello-world-app 디렉터리에 아직 있는지 확인한 다음 애플리케이션을 실행합니다.


```
node index.js
```

Hello World 메시지가 나타납니다. Node.js가 이전 단계에서 개발 환경에 다운로드한 lodash 패키지를 사용하여 애플리케이션을 실행했습니다.

'node_modules' 디렉터리를 무시하고 'Hello World'를 커밋하려면

1. node_modules 디렉터리를 무시합니다. 입력:

```
echo "node_modules/" >> .gitignore
```

이 디렉터리를 커밋하지 않는 것이 가장 좋습니다. 또한 이 디렉터리를 커밋하면 이 자습서의 이후 단계에 방해가 됩니다.

2. 추가, 커밋 및 푸시:

```
git add .  
git commit -m "add the Hello World application"  
git push
```

'Hello World' 애플리케이션 및 프로젝트 파일이 소스 리포지토리에 추가됩니다.

4단계: 'Hello World'를 실행하는 워크플로 생성

이 단계에서는 lodash 종속성을 사용하여 'Hello World' 애플리케이션을 실행하는 워크플로를 생성합니다. 워크플로에는 단일 작업 또는 RunHelloWorldApp라는 작업이 포함됩니다. 이 RunHelloWorldApp 작업에는 다음과 같은 주목할 만한 명령 및 섹션이 포함됩니다.

- **Packages**

이 섹션에서는 npm install를 실행할 때 작업이 연결해야 하는 CodeCatalyst 패키지 리포지토리의 이름을 나타냅니다.

- **- Run: npm install**

이 명령은 package.json 파일에 지정된 종속성을 설치하도록 npm에 지시합니다. package.json 파일에 지정된 유일한 종속성은 lodash입니다. npm은 다음 위치에서 lodash를 찾습니다.

- 작업을 실행하는 Docker 이미지. 여기에서는 찾을 수 없습니다.
- CodeCatalyst 패키지 리포지토리. 여기에서 발견합니다.

npm이 lodash를 찾으면 작업을 실행하는 Docker 이미지로 가져옵니다.

- - **Run: npm list**

이 명령은 해당 작업을 실행하는 Docker 이미지에 다운로드된 lodash 버전을 출력합니다.

- - **Run: node index.js**

이 명령은 package.json 파일에 지정된 종속성을 사용하여 'Hello World' 애플리케이션을 실행합니다.

RunHelloWorldApp 작업은 워크플로 상단 근처의 aws/build@v1 식별자로 표시되는 빌드 작업입니다. 빌드 작업에 대한 자세한 내용은 [워크플로로 빌드하기](#) 섹션을 참조하세요.

다음 지침을 사용하여 CodeCatalyst 패키지 리포지토리에서 lodash 종속성을 가져온 다음 'Hello World' 애플리케이션을 실행하는 워크플로를 생성합니다.

워크플로 생성 방법

1. CodeCatalyst 콘솔로 전환합니다.
2. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
3. 워크플로 생성을 선택합니다.
4. 소스 리포지토리에서 hello-world-app을 선택합니다.
5. 브랜치에서 main을 선택합니다.

워크플로 정의 파일은 선택한 소스 리포지토리 및 브랜치에 생성됩니다.

6. 생성(Create)을 선택합니다.
7. 상단 근처에서 YAML을 선택합니다.
8. YAML 샘플 코드를 삭제합니다.
9. 다음 YAML 코드를 추가합니다.

```
Name: codecatalyst-package-workflow
SchemaVersion: "1.0"

# Required - Define action configurations.
Actions:
```

```

RunHelloWorldApp:
  # Identifies the action. Do not modify this value.
  Identifier: aws/build@v1
  Compute:
    Type: Lambda
  Inputs:
    Sources:
      - WorkflowSource # This specifies your source repository.
  Configuration:
    Steps:
      - Run: npm install
      - Run: npm list
      - Run: node index.js
    Container: # This specifies the Docker image that runs the action.
      Registry: CODECATALYST
      Image: CodeCatalystLinuxLambda_x86_64:2024_03
  Packages:
    NpmConfiguration:
      PackageRegistries:
        - PackagesRepository: codecatalyst-package-repository

```

앞의 코드에서 *codecatalyst-package-repository*를 [2단계: CodeCatalyst 및 게이트웨이 패키지 리포지토리 생성](#)에서 생성한 CodeCatalyst 패키지 리포지토리의 이름으로 바꿉니다.

이러한 파일의 속성에 대한 자세한 내용은 [빌드 및 테스트 작업 YAML](#) 섹션을 참조하세요.

10. (선택 사항) 커밋하기 전에 YAML 코드가 유효한지 확인하려면 검증을 선택합니다.
11. 커밋을 선택합니다.
12. 워크플로 커밋 대화 상자에서 다음을 입력합니다.
 - a. 워크플로 파일 이름의 경우 기본값인 `codecatalyst-package-workflow`를 유지합니다.
 - b. 커밋 메시지에 다음을 입력합니다.

add initial workflow file

- c. 리포지토리에서 `hello-world-app`를 선택합니다.
- d. 브랜치 이름에서 기본을 선택합니다.
- e. 커밋을 선택합니다.

이제 워크플로를 생성했습니다.

해당 워크플로를 실행하려면

1. 방금 생성한 워크플로(`codecatalyst-package-workflow`) 옆에 작업을 선택한 다음 실행을 선택합니다.

워크플로 실행이 시작됩니다.

2. 상단의 녹색 알림에서 오른쪽의 실행에 대한 링크를 선택합니다. 링크는 View Run-1234와 같습니다.

실행을 시작한 사람과 RunHelloWorldApp 작업을 보여주는 워크플로 다이어그램이 나타납니다.

3. RunHelloWorldApp 작업 상자를 선택하여 작업의 진행 상황을 확인합니다.
4. 실행이 완료되면 [5단계: 워크플로 확인](#)로 이동합니다.

5단계: 워크플로 확인

이 단계에서는 워크플로가 `lodash` 종속성으로 'Hello World' 애플리케이션을 성공적으로 실행했는지 확인합니다.

'Hello World' 애플리케이션이 종속성을 사용하여 실행되었는지 확인하려면

1. 워크플로 다이어그램에서 RunHelloWorldApp 상자를 선택합니다.

로그 메시지 목록이 나타납니다.

2. `node index.js` 로그 메시지를 확장합니다.

다음 메시지가 나타납니다.

```
[Container] 2024/04/24 21:15:41.545650 Running command node index.js
Hello World
```

Hello Word(hello-world 대신)이 나타나면 `lodash` 종속성이 성공적으로 사용되었음을 나타냅니다.

3. `npm list` 로그를 확장합니다.

다음과 비슷한 메시지가 나타납니다.

```
### lodash@4.17.20
```

이 메시지는 lodash 버전 4.17.20이 워크플로 작업을 실행하는 Docker 이미지에 다운로드되었음을 나타냅니다.

6단계: npmjs.com 가져오기 차단

이제 게이트웨이 및 CodeCatalyst 패키지 리포지토리에 lodash 버전 4.17.20이 존재하므로 다른 버전 가져오기를 차단할 수 있습니다. 차단하면 악성 코드가 포함될 수 있는 lodash의 이후(또는 이전) 버전을 실수로 가져오는 일을 방지합니다. 자세한 내용은 [패키지 원본 제어 편집](#) 및 [종속성 대체 공격](#)(을)를 참조하세요.

다음 지침에 따라 게이트웨이 리포지토리로 lodash 가져오기를 차단합니다. 게이트웨이에서 패키지를 차단하면 다운스트림 위치에서도 차단됩니다.

게이트웨이 리포지토리로의 가져오기를 차단하려면

1. 탐색 창에서 패키지를 선택합니다.
2. npm-publish-registry-gateway를 선택합니다.
3. lodash를 선택합니다.
4. 상단 근처에서 원본 제어를 선택합니다.
5. 업스트림에서 차단을 선택합니다.
6. 저장을 선택합니다.

이제 npmjs.com에서 게이트웨이 리포지토리(및 다운스트림 리포지토리 및 컴퓨터)로의 가져오기를 차단했습니다.

7단계: 차단 기능 테스트

이 섹션에서는 [6단계: npmjs.com 가져오기 차단](#)에서 설정한 차단이 작동하는지 확인합니다. 게이트웨이 리포지토리에서 사용할 수 있는 버전인 4.17.20 lodash 대신 버전 4.17.21을 요청하도록 'Hello World'를 구성하는 것으로 시작합니다. 그런 다음 애플리케이션이 npmjs.com 버전 4.17.21을 가져올 수 없는지 확인하며, 이는 차단이 성공했음을 나타냅니다. 최종 테스트에서는 게이트웨이 리포지토리로의 가져오기 차단을 해제하고, 애플리케이션이 lodash의 버전 4.17.21을 성공적으로 가져올 수 있는지 확인합니다.

다음 절차 세트를 사용하여 차단 기능을 테스트합니다.

시작하기 전 준비 사항

1. 개발 환경으로 전환합니다.
2. CodeCatalyst 콘솔을 사용하여 이전에 생성한 `codecatalyst-package-workflow.yaml` 파일을 가져옵니다.

```
git pull
```

'lodash' 버전 4.17.21을 요청하도록 'Hello World'를 구성하려면

1. `/hello-world-app/package.json`를 엽니다.
2. ##### ## 표시된 내용과 같이 lodash 버전을 4.17.21로 변경합니다.

```
{
  "name": "hello-world-app",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "lodash": "4.17.21"
  }
}
```

이제 `package.json` 파일의 버전(4.17.21)과 게이트웨이 및 CodeCatalyst 패키지 리포지토리의 버전(4.17.20)이 일치하지 않습니다.

3. 추가, 커밋 및 푸시:

```
git add .
git commit -m "update package.json to use lodash 4.17.21"
git push
```

'Hello World'가 'lodash' 버전 4.17.21을 가져올 수 있는지 테스트하려면

1. 버전이 불일치하는 워크플로를 실행합니다.

1. CodeCatalyst 콘솔로 전환합니다.
2. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
3. codecatalyst-package-workflow 옆에 작업을 선택한 후 실행을 선택합니다.

npm은 package.json에서 종속성을 찾고 'Hello World'에서 lodash의 버전 4.17.21이 필요함을 확인합니다. npm은 다음 위치에서 다음 순서로 종속성을 찾습니다.

- 작업을 실행하는 Docker 이미지. 여기에서는 찾을 수 없습니다.
- CodeCatalyst 패키지 리포지토리. 여기에서는 찾을 수 없습니다.
- 게이트웨이 리포지토리. 여기에서는 찾을 수 없습니다.
- npmjs.com. 여기에서 발견합니다.

npm이 npmjs.com에서 버전 4.17.21을 찾은 후 게이트웨이 리포지토리로 가져오려고 시도하지만 lodash의 가져오기를 차단하도록 게이트웨이를 설정했으므로 가져오기가 발생하지 않습니다.

가져오기가 발생하지 않으므로 워크플로가 실패합니다.

2. 워크플로가 실패했는지 확인합니다.

1. 상단의 녹색 알림에서 오른쪽의 실행에 대한 링크를 선택합니다. 링크는 View Run-2345와 같습니다.
2. 워크플로 다이어그램에서 RunHelloWorldApp 상자를 선택합니다.
3. npm install 로그 메시지를 확장합니다.

다음 메시지가 나타납니다.

```
[Container] 2024/04/25 17:20:34.995591 Running command npm install
npm ERR! code ETARGET
npm ERR! notarget No matching version found for lodash@4.17.21.
npm ERR! notarget In most cases you or one of your dependencies are requesting
npm ERR! notarget a package version that doesn't exist.

npm ERR! A complete log of this run can be found in: /tmp/.npm/
_logs/2024-05-08T22_03_26_493Z-debug-0.log
```

오류는 버전 4.17.21을 찾을 수 없음을 나타냅니다. 이는 차단했기 때문에 나타나는 것으로 보입니다.

npmjs.com에서 가져오기 차단을 해제하려면

1. 탐색 창에서 패키지를 선택합니다.
2. npm-publish-registry-gateway를 선택합니다.
3. lodash를 선택합니다.
4. 상단 근처에서 원본 제어를 선택합니다.
5. 업스트림에서 허용을 선택합니다.
6. 저장을 선택합니다.

이제 lodash의 가져오기가 차단 해제됩니다.

이제 워크플로에서 lodash의 버전 4.17.21을 가져올 수 있습니다.

npmjs.com에서 가져오기가 차단 해제되었는지 테스트하려면

1. 워크플로를 다시 실행합니다. 이제 4.17.21의 가져오기가 작동하므로 이번에는 워크플로가 성공해야 합니다. 해당 워크플로를 실행하려면:
 1. CI/CD를 선택한 다음 워크플로를 선택합니다.
 2. codecatalyst-package-workflow 옆의 작업을 선택하고 실행을 선택합니다.
 3. 상단의 녹색 알림에서 오른쪽의 실행에 대한 링크를 선택합니다. 링크는 View Run-3456와 같습니다.

실행을 시작한 사람과 RunHelloWorldApp 작업을 보여주는 워크플로 다이어그램이 나타납니다.

4. RunHelloWorldApp 작업 상자를 선택하여 작업의 진행 상황을 확인합니다.
5. npm list 로그 메시지를 확장하고 다음과 유사한 메시지가 나타나는지 확인합니다.

```
### lodash@4.17.21
```

이 메시지는 lodash 버전 4.17.21이 다운로드되었음을 나타냅니다.

2. 버전 4.17.21을 CodeCatalyst 및 게이트웨이 리포지토리로 가져왔는지 확인합니다.

1. 탐색 창에서 패키지를 선택합니다.
2. npm-public-registry-gateway를 선택합니다.
3. lodash를 찾아 버전이 4.17.21인지 확인합니다.

 Note

버전 4.17.20은 이 페이지에 나열되어 있지 않지만 lodash를 선택한 다음 상단에서 버전을 선택하여 찾을 수 있습니다.

4. 이 단계를 반복하여 버전 4.17.21을 codecatalyst-package-repository로 가져왔는지 확인합니다.

정리

요금이 부과되지 않도록 이 자습서에서 사용하는 파일과 서비스를 정리합니다.

패키지 자습서를 정리하려면

1. codecatalyst-package-project를 삭제합니다.
 - a. CodeCatalyst 콘솔에서 아직 프로젝트로 이동하지 않은 경우, codecatalyst-package-project 프로젝트로 이동합니다.
 - b. 탐색 창에서 프로젝트 설정을 선택합니다.
 - c. 프로젝트 삭제를 선택하고 **delete**를 입력한 다음 프로젝트 삭제를 선택합니다.

CodeCatalyst는 소스, 게이트웨이 및 CodeCatalyst 패키지 리포지토리를 포함한 모든 프로젝트 리소스를 삭제합니다. 개발 환경도 삭제됩니다.

2. PAT 토큰을 삭제합니다.
 - a. 오른쪽에서 사용자 이름을 선택한 다음 내 설정을 선택합니다.
 - b. 개인 액세스 토큰에서 이 자습서에서 생성한 토큰을 선택하고 삭제를 선택합니다.

이 자습서에서는 CodeCatalyst 패키지 리포지토리에서 종속성을 가져오는 애플리케이션을 실행하는 워크플로를 생성하는 방법을 배웠습니다. 또한 패키지가 게이트웨이 및 CodeCatalyst 패키지 리포지토리로 들어오는 것을 차단 및 차단 해제하는 방법을 배웠습니다.

워크플로에서 CodeCatalyst 패키지 리포지토리 지정

CodeCatalyst에서는 워크플로에서 빌드 및 테스트 작업에 CodeCatalyst 패키지 리포지토리를 추가할 수 있습니다. 패키지 리포지토리는 npm과 같은 패키지 형식으로 구성해야 합니다. 선택한 패키지 리포지토리에 대해 일련의 범위를 포함하도록 선택할 수도 있습니다.

다음 지침에 따라 워크플로 작업에 사용할 패키지 구성을 지정합니다.

Visual

작업에서 사용할 패키지 구성 지정(시각적 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 워크플로 다이어그램에서 패키지 리포지토리로 구성할 빌드 또는 테스트 작업을 선택합니다.
8. 패키지를 선택합니다.
9. 구성 추가 드롭다운 메뉴에서 워크플로 작업에 사용할 패키지 구성을 선택합니다.
10. 패키지 리포지토리 추가를 선택합니다.
11. 패키지 리포지토리 드롭다운 메뉴에서 작업을 사용할 CodeCatalyst 패키지 리포지토리의 이름을 지정합니다.

패키지 리포지토리에 대한 자세한 내용은 [패키지 리포지토리](#) 섹션을 참조하세요.

12. (선택 사항) 범위 - 선택 사항에서 패키지 레지스트리에 정의할 범위 순서를 지정합니다.

범위를 정의할 때 지정된 패키지 리포지토리는 나열된 모든 범위에 대한 레지스트리로 구성됩니다. 해당 범위의 패키지가 npm 클라이언트를 통해 요청되면 기본값 대신 해당 리포지토리를 사용합니다. 각 범위 이름 앞에는 '@' 접두사를 붙여야 합니다.

Scopes 생략 시 지정된 패키지 리포지토리가 작업에 사용되는 모든 패키지의 기본 레지스트리로 구성됩니다.

범위에 대한 자세한 내용은 [패키지 네임스페이스](#) 및 [범위 지정 패키지](#) 섹션을 참조하세요.

13. 추가를 선택합니다.
14. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
15. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

작업이 사용할 패키지 구성 지정(YAML 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 빌드 또는 테스트 작업에서 다음과 유사한 코드를 추가합니다.

```
action-name:
  Configuration:
    Packages:
      NpmConfiguration:
        PackageRegistries:
          - PackagesRepository: package-repository
        Scopes:
          - "@scope"
```

자세한 내용은 작업에 대한 [빌드 및 테스트 작업 YAML](#)의 Packages 속성 설명을 참조하세요.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

워크플로 작업에서 권한 부여 토큰 사용

워크플로 작업에서 제공하는 토큰을 사용하여 패키지 관리자가 CodeCatalyst 패키지 리포지토리로 인증하도록 수동으로 구성할 수 있습니다. CodeCatalyst는 이 토큰을 작업에서 참조할 수 있도록 환경 변수로 사용할 수 있도록 합니다.

환경 변수	값
CATALYST_MACHINE_RESOURCE_NAME	권한 부여 토큰의 사용자 ID입니다.
CATALYST_PACKAGES_AUTHORIZATION_TOKEN	권한 부여 토큰의 값입니다.

Note

이러한 환경 변수는 권한 부여 토큰을 내보내도록 작업을 구성한 경우에만 채워집니다.

다음 지침에 따라 워크플로 작업과 함께 권한 부여 토큰을 사용합니다.

Visual

작업에서 내보낸 권한 부여 토큰 사용(시각적 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 워크플로 다이어그램에서 패키지 리포지토리로 구성할 빌드 또는 테스트 작업을 선택합니다.
8. 패키지를 선택합니다.
9. 권한 부여 토큰 내보내기를 엽니다.

YAML

작업에서 내보낸 권한 부여 토큰 사용(YAML 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.

3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 빌드 또는 테스트 작업에서 다음과 유사한 코드를 추가합니다.

```
Actions:
  action-name:
    Packages:
      ExportAuthorizationToken: true
```

YAML의 Steps 섹션에서 \$CATALYST_MACHINE_RESOURCE_NAME 및 \$CATALYST_PACKAGES_AUTHORIZATION_TOKEN 환경 변수를 참조할 수 있습니다. 자세한 정보는 [예시: CodeCatalyst로 인증하도록 수동으로 pip 구성](#) 섹션을 참조하세요.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

예시: 워크플로의 패키지 리포지토리

다음 예시에서는 워크플로 정의 파일의 패키지를 참조하는 방법을 보여줍니다.

주제

- [예시: NpmConfiguration을 사용하여 패키지 정의](#)
- [예시: 기본 레지스트리 재정의](#)
- [예시: 패키지 레지스트리의 범위 재정의](#)
- [예시: CodeCatalyst로 인증하도록 수동으로 pip 구성](#)

예시: NpmConfiguration을 사용하여 패키지 정의

다음 예시는 워크플로 정의 파일에서 NpmConfiguration를 사용하여 패키지를 정의하는 방법을 보여줍니다.

```
Actions:
  Build:
    Identifier: aws/build-beta@v1
```

```
Configuration:
  Packages:
    NpmConfiguration:
      PackageRegistries:
        - PackagesRepository: main-repo
        - PackagesRepository: scoped-repo
      Scopes:
        - "@scope1"
```

이 예시에서는 npm 클라이언트를 다음과 같이 구성합니다.

```
default: main-repo
@scope1: scoped-repo
```

이 예시에서는 두 개의 리포지토리가 정의되어 있습니다. 기본 레지스트리는 범위 없이 정의되므로 기본 레지스트리는 main-repo로 설정됩니다. 범위 @scope1은 scoped-repo에 대해 PackageRegistries에 구성됩니다.

예시: 기본 레지스트리 재정의

다음 예시에서는 기본 레지스트리를 재정의하는 방법을 보여줍니다.

```
NpmConfiguration:
  PackageRegistries:
    - PackagesRepository: my-repo-1
    - PackagesRepository: my-repo-2
    - PackagesRepository: my-repo-3
```

이 예시에서는 npm 클라이언트를 다음과 같이 구성합니다.

```
default: my-repo-3
```

여러 기본 리포지토리를 지정하는 경우 마지막 리포지토리가 우선합니다. 이 예시에서 나열된 마지막 리포지토리는 my-repo-3이며, 이는 npm이 my-repo-3에 연결된다는 것을 의미합니다. 이렇게 하면 리포지토리 my-repo-1 및 my-repo-2를 재정의합니다.

예시: 패키지 레지스트리의 범위 재정의

다음 예시에서는 패키지 레지스트리의 범위를 재정의하는 방법을 보여줍니다.

```
NpmConfiguration:
```

```
PackageRegistries:
  - PackagesRepository: my-default-repo
  - PackagesRepository: my-repo-1
  Scopes:
    - "@scope1"
    - "@scope2"
  - PackagesRepository: my-repo-2
  Scopes:
    - "@scope2"
```

이 예시에서는 npm 클라이언트를 다음과 같이 구성합니다.

```
default: my-default-repo
@scope1: my-repo-1
@scope2: my-repo-2
```

범위 재정의의 포함하면 마지막 리포지토리가 우선합니다. 이 예시에서 @scope2 범위가 my-repo-2에 대해 마지막으로 구성된 것은 PackageRegistries입니다. 이렇게 하면 my-repo-1에 대해 구성된 범위 @scope2가 재정의됩니다.

예시: CodeCatalyst로 인증하도록 수동으로 **pip** 구성

다음 예시에서는 빌드 작업에서 CodeCatalyst 권한 부여 환경 변수를 참조하는 방법을 보여줍니다.

```
Actions:
  Build:
    Identifier: aws/build@v1.0.0
    Configuration:
      Steps:
        - Run: pip config set global.index-url https://$CATALYST_MACHINE_RESOURCE_NAME:
$CATALYST_PACKAGES_AUTHORIZATION_TOKEN@codecatalyst.aws/pypi/my-space/my-project/my-repo/simple/
      Packages:
        ExportAuthorizationToken: true
```

워크플로를 사용하여 Lambda 함수 호출

이 섹션에서는 CodeCatalyst 워크플로를 사용하여 AWS Lambda 함수를 호출하는 방법을 설명합니다. 이렇게 하려면 워크플로에 AWS Lambda 간접 호출 작업을 추가해야 합니다. AWS Lambda 간접 호출 작업은 지정한 Lambda 함수를 호출합니다.

함수를 간접적으로 호출하는 것 외에도 AWS Lambda 간접 호출 작업은 Lambda 함수에서 수신한 응답 페이로드의 각 최상위 키를 [워크플로 출력 변수](#)로 변환합니다. 그런 다음 후속 워크플로 작업에서 이러한 변수를 참조할 수 있습니다. 모든 최상위 키를 변수로 변환하지 않으려면 필터를 사용하여 정확한 키를 지정할 수 있습니다. 자세한 내용은 ['AWS Lambda 간접 호출' 작업 YAML](#)의 [ResponseFilters](#) 속성 설명을 참조하세요.

주제

- [이 작업을 사용해야 하는 경우](#)
- ['AWS Lambda 호출' 작업에서 사용하는 런타임 이미지](#)
- [예시: Lambda 함수를 간접적으로 호출합니다.](#)
- ['AWS Lambda 호출' 작업 추가](#)
- ['AWS Lambda 간접 호출' 변수](#)
- ['AWS Lambda 간접 호출' 작업 YAML](#)

이 작업을 사용해야 하는 경우

Lambda 함수에 캡슐화되고 Lambda 함수에서 수행하는 워크플로에 기능을 추가하려면, 이 작업을 사용합니다.

예를 들어, 애플리케이션 빌드를 시작하기 전에 워크플로가 Slack 채널에 Build started 알림을 보내도록 할 수 있습니다. 이 경우, 워크플로에는 Lambda를 호출하여 Slack 알림을 보내는 AWS Lambda 간접 호출 작업과 애플리케이션을 빌드하는 [빌드 작업](#)이 포함됩니다.

또 다른 예로 워크플로가 배포되기 전에 워크플로에서 애플리케이션의 취약성 스캔을 수행하게 할 수 있습니다. 이 경우, 빌드 작업을 사용하여 애플리케이션을 빌드하고, AWS Lambda 간접 호출 작업을 사용하여 Lambda를 호출해 취약성을 스캔하고, 배포 작업을 사용하여 스캔한 애플리케이션을 배포합니다.

'AWS Lambda 호출' 작업에서 사용하는 런타임 이미지

AWS Lambda 간접 호출 작업은 [2022년 11월 이미지](#)에서 실행됩니다. 자세한 내용은 [활성 이미지](#) 단원을 참조하십시오.

예시: Lambda 함수를 간접적으로 호출합니다.

다음 워크플로 예시에는 AWS Lambda 간접 호출 작업과 배포 작업이 포함됩니다. 워크플로는 배포가 시작되었음을 나타내는 Slack 알림을 전송한 다음 CloudFormation 템플릿을 AWS 사용하여 애플리케이션을 배포합니다. 워크플로는 순차적으로 실행되는 다음 구성 요소로 구성됩니다.

- 트리거 - 이 트리거는 소스 리포지토리에 변경 사항을 푸시할 때 워크플로 실행을 자동으로 시작합니다. 트리거에 대한 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 주제를 참조하세요.
- AWS Lambda 호출 작업(LambdaNotify) - 트리거 시 이 작업은 지정된 AWS 계정 및 리전(my-aws-account 및 us-west-2)에서 Notify-Start Lambda 함수를 간접적으로 호출합니다. 간접 호출 시 Lambda 함수는 배포가 시작되었음을 나타내는 Slack 알림을 보냅니다.
- CloudFormation 스택 배포 작업(Deploy) - AWS Lambda 호출 작업이 완료되면 스택 배포 CloudFormation 작업은 템플릿(cfn-template.yml)을 실행하여 애플리케이션 스택을 배포합니다. CloudFormation 스택 배포 작업에 대한 자세한 내용은 [섹션을 참조하세요](#) [CloudFormation 스택 배포](#).

Note

다음 워크플로 예시는 설명을 돕기 위한 참고용이며 추가 구성 없이는 작동하지 않습니다.

Note

다음 YAML 코드에서 원하는 경우 Connections: 섹션을 생략할 수 있습니다. 이러한 섹션을 생략하는 경우 환경의 기본 IAM 역할 필드에 지정된 역할에 AWS Lambda 스택 호출 및 배포 CloudFormation 작업에 필요한 권한 및 신뢰 정책이 포함되어 있는지 확인해야 합니다. 기본 IAM 역할이 있는 환경 설정에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요. 스택 AWS Lambda 호출 및 배포 작업에 필요한 권한 및 신뢰 정책에 대한 자세한 내용은 '[AWS Lambda 간접 호출' 작업 YAML](#) 및의 Role 속성 설명을 참조하세요 '[스 CloudFormation 스택 배포' 작업 YAML](#). CloudFormation

Name: codecatalyst-lambda-invoke-workflow

SchemaVersion: 1.0

Triggers:

- Type: PUSH

Branches:

- main

Actions:

LambdaNotify:

Identifier: aws/lambda-invoke@v1

Environment:

```

    Name: my-production-environment
    Connections:
      - Name: my-aws-account
        Role: codecatalyst-lambda-invoke-role
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Function: Notify-Start
      AWSRegion: us-west-2

Deploy:
  Identifier: aws/cfn-deploy@v1
  Environment:
    Name: my-production-environment
    Connections:
      - Name: my-aws-account
        Role: codecatalyst-deploy-role
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      name: my-application-stack
      region: us-west-2
      role-arn: arn:aws:iam::111122223333:role/StackRole
      template: ./cfn-template.yml
      capabilities: CAPABILITY_IAM,CAPABILITY_AUTO_EXPAND

```

'AWS Lambda 호출' 작업 추가

다음 지침에 따라 워크플로에 AWS Lambda 간접 호출 작업을 추가합니다.

사전 조건

시작하기 전에 AWS Lambda 함수 및 연결된 Lambda 실행 역할이에서 준비되고 사용 가능한지 확인합니다 AWS. 자세한 내용은 AWS Lambda 개발자 안내서의 [Lambda 실행 역할](#) 주제를 참조하세요.

Visual

시각적 편집기를 사용하여 'AWS Lambda 호출' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.

3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
 4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
 5. 편집을 선택합니다.
 6. 비주얼을 선택합니다.
 7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.
 8. 드롭다운 목록에서 Amazon CodeCatalyst를 선택합니다.
 9. AWS Lambda 간접 호출 작업을 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
- 또는
- AWS Lambda 간접 호출을 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 소스 보기를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
10. 입력, 구성 및 출력 탭에서 필요에 따라 필드를 작성합니다. 각 필드의 설명은 '[AWS Lambda 간접 호출 작업 YAML](#)' 섹션을 참조하세요. 이 참조는 YAML 및 시각적 편집기 모두에 나타나는 각 필드(및 해당 YAML 속성 값)에 대한 자세한 정보를 제공합니다.
 11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
 12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 'AWS Lambda 호출' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.

8. 드롭다운 목록에서 Amazon CodeCatalyst를 선택합니다.
9. AWS Lambda 간접 호출 작업을 검색하고 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
 - 또는
 - AWS Lambda 간접 호출을 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 소스 보기를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
10. 필요에 따라 YAML 코드의 속성을 수정합니다. 사용 가능한 각 속성에 대한 설명은 ['AWS Lambda 간접 호출' 작업 YAML](#)에서 볼 수 있습니다.
11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

'AWS Lambda 간접 호출' 변수

기본적으로 AWS Lambda 간접 호출 작업은 Lambda 응답 페이로드의 최상위 키당 하나의 변수를 생성합니다.

예를 들어, 응답 페이로드가 다음과 같은 경우:

```
responsePayload = {
  "name": "Saanvi",
  "location": "Seattle",
  "department": {
    "company": "Amazon",
    "team": "AWS"
  }
}
```

...그런 다음 해당 작업은 다음과 같은 변수를 생성합니다.

Key(키)	값
이름	Saanvi
location	시애틀

Key(키)	값
department	{"company": "Amazon", "team": "AWS"}

Note

ResponseFilters YAML 속성을 사용하여 생성되는 변수를 변경할 수 있습니다. 자세한 내용은 ['AWS Lambda 간접 호출' 작업 YAML](#)의 [ResponseFilters](#)를 참조하세요.

실행 시간에 'AWS Lambda 호출' 작업에 의해 생성되고 설정된 변수를 사전 정의된 변수라고 합니다.

워크플로에서 이러한 변수를 참조하는 방법에 대한 자세한 내용은 [사전 정의된 변수 사용](#) 섹션을 참조하세요.

'AWS Lambda 간접 호출' 작업 YAML

다음은 AWS Lambda 간접 호출 작업의 YAML 정의입니다. 이러한 작업 사용 방법을 배우려면 [워크플로를 사용하여 Lambda 함수 호출](#) 섹션을 참조하세요.

이 작업 정의는 더 광범위한 워크플로 정의 파일 내의 섹션으로 존재합니다. 이 파일에 대한 자세한 내용은 [워크플로 YAML 정의](#)를 참조합니다.

Note

이어지는 대부분의 YAML 속성에는 시각적 편집기에 해당 UI 요소가 있습니다. UI 요소를 찾으려면 Ctrl+F를 사용합니다. 요소가 연결된 YAML 속성과 함께 나열됩니다.

```
# The workflow definition starts here.
# See ### ## for details.
```

```
Name: MyWorkflow
SchemaVersion: 1.0
Actions:
```

```
# The action definition starts here.
LambdaInvoke\_nn:
```

```

Identifier: aws/lambda-invoke@v1
DependsOn:
  - dependent-action
Compute:
  Type: EC2 | Lambda
  Fleet: fleet-name
  Timeout: timeout-minutes
Inputs:
  # Specify a source or an artifact, but not both.
  Sources:
    - source-name-1
  Artifacts:
    - request-payload
  Variables:
    - Name: variable-name-1
      Value: variable-value-1
    - Name: variable-name-2
      Value: variable-value-2
Environment:
  Name: environment-name
Connections:
  - Name: account-connection-name
    Role: iam-role-name
Configuration:
  Function: my-function/function-arn
  AWSRegion: us-west-2
  # Specify RequestPayload or RequestPayloadFile, but not both.
  RequestPayload: '{"firstname": "Li", lastname: "Jean", "company": "ExampleCo",
"team": "Development"}'
  RequestPayloadFile: my/request-payload.json
  ContinueOnError: true/false
  LogType: Tail/None
  ResponseFilters: '{"name": ".name", "company": ".department.company"}'
Outputs:
  Artifacts:
    - Name: lambda_artifacts
  Files:
    - "lambda-response.json"

```

LambdaInvoke

(필수)

작업 이름을 지정합니다. 워크플로 내의 모든 작업 이름은 고유해야 합니다. 작업 이름은 영숫자 문자 (a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 작업 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

기본값: Lambda_Invoke_Action_Workflow_nn.

해당 UI: 구성 탭/작업 이름

Identifier

(*LambdaInvoke*/Identifier)

(필수)

작업을 식별합니다. 버전을 변경하려는 경우가 아니면 이 속성을 변경하지 마세요. 자세한 내용은 [사용할 작업 버전 지정](#) 섹션을 참조하세요.

기본값: aws/lambda-invoke@v1.

해당 UI: 워크플로 다이어그램/LambdaInvoke_nn/aws/lambda-invoke@v1 레이블

DependsOn

(*LambdaInvoke*/DependsOn)

(선택 사항)

이 작업을 실행하기 위해 성공적으로 실행해야 하는 작업, 작업 그룹 또는 게이트를 지정합니다.

'depends on' 함수에 대한 자세한 내용은 [작업 순서 지정](#) 섹션을 참조하세요.

해당 UI: 입력 탭/의존 - 선택 사항

Compute

(*LambdaInvoke*/Compute)

(선택 사항)

워크플로 작업을 실행하는 데 사용되는 컴퓨팅 엔진입니다. 워크플로 수준 또는 작업 수준에서 컴퓨팅을 지정할 수 있지만 둘 다 지정할 수는 없습니다. 워크플로 수준에서 지정하면 컴퓨팅 구성이 워크플로

로에 정의된 모든 작업에 적용됩니다. 워크플로 수준에서는 동일한 인스턴스에서 여러 작업을 실행할 수도 있습니다. 자세한 내용은 [작업 간에 컴퓨팅 공유](#) 섹션을 참조하세요.

해당 UI: 없음

Type

(*LambdaInvoke*/Compute/Type)

([Compute](#) 포함 시 필수)

컴퓨팅 엔진의 유형입니다. 다음 값 중 하나를 사용할 수 있습니다.

- EC2(시각 편집기) 또는 EC2(YAML 편집기)
작업 실행 중 유연성을 위해 최적화되었습니다.
- Lambda(시각 편집기) 또는 Lambda(YAML 편집기)
작업 시작 속도를 최적화했습니다.

컴퓨팅 유형에 대한 자세한 정보는 [컴퓨팅 유형](#)을 참고하세요.

해당 UI: 구성 탭/컴퓨팅 유형

Fleet

(*LambdaInvoke*/Compute/Fleet)

(선택 사항)

워크플로 또는 워크플로 작업을 실행할 시스템 또는 플릿을 지정합니다. 온디맨드 플릿의 경우 작업이 시작되면 워크플로가 필요한 리소스를 프로비저닝하고 작업이 완료되면 시스템이 파괴됩니다. 온디맨드 플릿의 예시: Linux.x86-64.Large, Linux.x86-64.XLarge. 온디맨드 플릿에 대한 자세한 내용은 [온디맨드 플릿 속성](#) 섹션을 참조하세요.

프로비저닝된 플릿을 사용하면 워크플로 작업을 실행하도록 전용 시스템 세트를 구성할 수 있습니다. 이러한 시스템은 유휴 상태로 유지되므로 작업을 즉시 처리할 수 있습니다. 프로비저닝된 플릿에 대한 자세한 내용은 [프로비저닝된 플릿 속성](#) 섹션을 참조하세요.

Fleet 생략 시 기본값은 Linux.x86-64.Large입니다.

해당 UI: 구성 탭/컴퓨팅 플릿

Timeout

(*LambdaInvoke*/Timeout)

(필수)

CodeCatalyst가 작업을 종료하기 전에 작업을 실행할 수 있는 시간을 분(YAML 편집기) 또는 시간 및 분(시각적 편집기) 단위로 지정합니다. 최소값은 5분이고 최대값은 [CodeCatalyst의 워크플로 할당량](#)에 설명되어 있습니다. 기본 제한 시간은 최대 제한 시간과 동일합니다.

해당 UI: 구성 탭/제한 시간 - 선택 사항

Inputs

(*LambdaInvoke*/Inputs)

(필수)

이 Inputs 섹션에서는 워크플로 실행 중에 AWS Lambda 간접 호출 작업에 필요한 데이터를 정의합니다.

Note

AWS Lambda 간접 호출 작업당 하나의 입력(소스 또는 아티팩트)만 허용됩니다. 변수는 이 합계에 포함되지 않습니다.

해당 UI: 입력 탭

Sources

(*LambdaInvoke*/Inputs/Sources)

([RequestPayloadFile](#)이 제공된 경우 필수)

요청 페이로드 JSON 파일을 AWS Lambda 간접 호출 작업에 전달하고자 하며 이 페이로드 파일이 소스 리포지토리에 저장되는 경우, 해당 소스 리포지토리의 레이블을 지정합니다. 현재, WorkflowSource 레이블만 지원됩니다.

요청 페이로드 파일이 소스 리포지토리에 포함되어 있지 않은 경우, 다른 작업에서 생성된 아티팩트에 있어야 합니다.

페이로드 파일에 대한 자세한 내용은 [RequestPayloadFile](#)을 참조하세요.

Note

페이로드 파일을 지정하는 대신 `RequestPayload` 속성을 사용하여 페이로드의 JSON 코드를 작업에 직접 추가할 수 있습니다. 자세한 내용은 [RequestPayload](#) 단원을 참조하십시오.

소스에 대한 자세한 내용은 [워크플로에 소스 리포지토리 연결](#) 섹션을 참조하세요.

해당 UI: 입력 탭/소스 - 선택 사항

Artifacts - input

(*LambdaInvoke*/Inputs/Artifacts)

([RequestPayloadFile](#)이 제공된 경우 필수)

요청 페이로드 JSON 파일을 AWS Lambda 간접 호출 작업에 전달하고자 하며 이 페이로드 파일이 이전 작업의 [출력 아티팩트](#)에 포함된 경우, 여기에 해당 아티팩트를 지정합니다.

페이로드 파일에 대한 자세한 내용은 [RequestPayloadFile](#)을 참조하세요.

Note

페이로드 파일을 지정하는 대신 `RequestPayload` 속성을 사용하여 페이로드의 JSON 코드를 작업에 직접 추가할 수 있습니다. 자세한 내용은 [RequestPayload](#) 단원을 참조하십시오.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

해당 UI: 구성 탭/아티팩트 - 선택 사항

Variables - input

(*LambdaInvoke*/Inputs/Variables)

(선택 사항)

작업에 사용할 수 있도록 하려는 입력 변수를 정의하는 이름/값 페어의 시퀀스를 지정합니다. 변수 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 변수 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

예시를 비롯한 변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

해당 UI: 입력 탭/변수 - 선택 사항

Environment

(*LambdaInvoke*/Environment)

(필수)

작업에 사용할 CodeCatalyst 환경을 지정합니다. 작업은 선택한 환경에 지정된 AWS 계정 및 선택적 Amazon VPC에 연결됩니다. 작업은 환경에 지정된 기본 IAM 역할을 사용하여 연결하고 [Amazon VPC 연결](#)에 지정된 IAM 역할을 AWS 계정으로 사용하여 Amazon VPC에 연결합니다.

Note

기본 IAM 역할에 작업에 필요한 권한이 없는 경우 다른 역할을 사용하도록 작업을 구성할 수 있습니다. 자세한 내용은 [작업의 IAM 역할 변경](#) 섹션을 참조하세요.

환경에 대한 자세한 내용은 [AWS 계정 및 VPCs에 배포](#) 및 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 구성 탭/환경

Name

(*LambdaInvoke*/Environment/Name)

([Environment](#) 포함 시 필수)

작업과 연결하려는 기존 환경의 이름을 지정합니다.

해당 UI: 구성 탭/환경

Connections

(*LambdaInvoke*/Environment/Connections)

(최신 버전의 작업에서는 선택 사항, 이전 버전에서는 필수)

작업과 연결할 계정 연결을 지정합니다. Environment에서 계정 연결을 최대 1개까지 지정할 수 있습니다.

계정 연결을 지정하지 않는 경우:

- 작업은 CodeCatalyst 콘솔의 환경에 지정된 AWS 계정 연결 및 기본 IAM 역할을 사용합니다. 환경에 계정 연결 및 기본 IAM 역할을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.
- 기본 IAM 역할에는 작업에 필요한 정책 및 권한이 포함되어야 합니다. 이러한 정책 및 권한이 무엇인지 확인하려면 작업의 YAML 정의 설명서에서 역할 속성에 대한 설명을 참조하세요.

계정 연결에 대한 자세한 정보는 [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 섹션을 참조하세요. 환경에 계정 연결을 추가하는 방법에 대한 자세한 내용은 [환경 생성](#) 섹션을 참조하세요.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/AWS 계정 연결

Name

(*LambdaInvoke*/Environment/Connections/Name)

([Connections](#) 포함 시 필수)

계정 연결의 이름을 지정합니다.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/AWS 계정 연결

Role

(*LambdaInvoke*/Environment/Connections/Role)

([Connections](#) 포함 시 필수)

AWS Lambda 간접 호출 작업이 Lambda 함수에 액세스 AWS 하고 간접 호출하는 데 사용하는 IAM 역할의 이름을 지정합니다. [CodeCatalyst 스페이스에 역할을 추가](#)했고 역할에 다음 정책이 포함되어 있는지 확인합니다.

IAM 역할을 지정하지 않으면 작업은 CodeCatalyst 콘솔의 [환경](#)에 나열된 기본 IAM 역할을 사용합니다. 환경에서 기본 역할을 사용하는 경우 다음 정책이 있는지 확인합니다.

- 다음 권한 정책:

 Warning

다음 정책에 표시된 대로 권한을 제한합니다. 더 광범위한 권한을 가진 역할을 사용하면 보안 위험이 발생할 수 있습니다.

- 다음 사용자 지정 신뢰 정책:

 Note

원하는 경우 이 작업에서 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 사용할 수 있습니다. 이에 대한 자세한 내용은 [계정 및 스페이스의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 생성](#) 섹션을 참조하세요. CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할에 보안 위험을 초래할 수 있는 전체 액세스 권한이 있음을 이해합니다. 보안에 대한 우려가 적은 자습서 및 시나리오에서만 이 역할을 사용하는 것이 좋습니다.

해당 UI: 작업 버전에 따라 다음 중 하나:

- (최신 버전) 구성 탭/환경/**# ##**의 내용/점 3개 메뉴/역할 전환
- (이전 버전) 구성 탭/환경/계정/역할/역할

Configuration

(*LambdaInvoke*/Configuration)

(필수)

작업의 구성 속성을 정의할 수 있는 섹션입니다.

해당 UI: 구성 탭

Function

(*LambdaInvoke*/Configuration/Function)

(필수)

이 작업이 호출할 AWS Lambda 함수를 지정합니다. 이 함수의 이름 또는 Amazon 리소스 이름(ARN)을 지정합니다. Lambda 콘솔에서 해당 이름 또는 ARN을 찾을 수 있습니다.

 Note

Lambda 함수가 있는 AWS 계정은 아래에 지정된 계정과 다를 수 있습니다Connections:.

해당 UI: 구성 탭/기능

AWSRegion

(*LambdaInvoke*/Configuration/AWSRegion)

(필수)

AWS Lambda 함수가 있는 AWS 리전을 지정합니다. 리전 코드 목록은 AWS 일반 참조의 [리전 엔드포인트](#)를 참조하세요.

해당 UI: 구성 탭/대상 버킷 - 선택 사항

RequestPayload

(*LambdaInvoke*/Configuration/RequestPayload)

(선택 사항)

AWS Lambda 간접 호출 작업에 요청 페이로드를 전달하려면, JSON 형식으로 여기에 요청 페이로드를 지정합니다.

요청 페이로드 예시:

```
{ "key": "value" }
```

Lambda 함수에 요청 페이로드를 전달하지 않으려면 이 속성을 생략합니다.

 Note

RequestPayload 또는 RequestPayloadFile을 지정할 수 있지만 둘 다 함께 지정할 수는 없습니다.

요청 페이로드에 대한 자세한 내용은 AWS Lambda API 참조의 [Invoke](#) 주제를 참조하세요.

해당 UI: 구성 탭/요청 페이로드 - 선택 사항

RequestPayloadFile

(*LambdaInvoke*/Configuration/RequestPayloadFile)

(선택 사항)

AWS Lambda 간접 호출 작업에 요청 페이로드를 전달하려면, 여기에 이 요청 페이로드 파일의 경로를 지정합니다. 파일은 JSON 형식이어야 합니다.

요청 페이로드 파일은 소스 리포지토리 또는 이전 작업의 아티팩트에 상주할 수 있습니다. 파일 경로는 소스 리포지토리 또는 아티팩트 루트를 기준으로 합니다.

Lambda 함수에 요청 페이로드를 전달하지 않으려면 이 속성을 생략합니다.

Note

RequestPayload 또는 RequestPayloadFile을 지정할 수 있지만 둘 다 함께 지정할 수는 없습니다.

요청 페이로드 파일에 대한 자세한 내용은 AWS Lambda API 참조의 [Invoke](#) 주제를 참조하세요.

해당 UI: 구성 탭/요청 페이로드 파일 - 선택 사항

ContinueOnError

(*LambdaInvoke*/Configuration/RequestPayloadFile)

(선택 사항)

간접적으로 호출된 AWS Lambda 함수가 실패하더라도 AWS Lambda 간접 호출 작업을 성공으로 표시할지 여부를 지정합니다. Lambda 실패에도 불구하고 워크플로의 후속 작업이 시작되도록 하려면 이 속성을 true로 설정하는 것을 고려하세요.

기본적으로 Lambda 함수가 실패하면(시각 편집기에서 'off' 또는 YAML 편집기에서 false) 작업에 실패합니다.

해당 UI: 구성 탭/오류 발생 시 계속 진행

LogType

(*LambdaInvoke*/Configuration/LogType)

(선택 사항)

간접 호출된 후 Lambda 함수의 응답에 오류 로그를 포함할지 여부를 지정합니다. 이러한 로그는 CodeCatalyst 콘솔의 Lambda 간접 호출 작업의 로그 탭에서 볼 수 있습니다. 가능한 값은 다음과 같습니다.

- Tail - 로그를 반환함
- None - 로그를 반환하지 않음

기본값은 Tail입니다.

로그 유형에 대한 자세한 내용은 AWS Lambda API 참조의 [Invoke](#) 주제를 참조하세요.

로그 보기에 대한 자세한 내용은 [워크플로 실행 상태 및 세부 정보 보기](#) 섹션을 참조하세요.

해당 UI: 구성 탭/로그 유형

ResponseFilters

(*LambdaInvoke*/Configuration/ResponseFilters)

(선택 사항)

Lambda 응답 페이로드에서 출력 변수로 변환할 키를 지정합니다. 그런 다음 워크플로의 후속 작업에서 출력 변수를 참조할 수 있습니다. CodeCatalyst의 변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

예를 들어, 응답 페이로드가 다음과 같은 경우:

```
responsePayload = {
  "name": "Saanvi",
  "location": "Seattle",
  "department": {
    "company": "Amazon",
    "team": "AWS"
  }
}
```


...응답 필터는 다음과 같습니다.

Configuration:

...

ResponseFilters: '{"name": ".name", "company": ".department.company"}'

...그런 다음 해당 작업은 다음과 같은 출력 변수를 생성합니다.

Key(키)	값
이름	Saanvi
company	Amazon

그런 다음 후속 작업에서 name 및 company 변수를 참조할 수 있습니다.

ResponseFilters에서 키를 지정하지 않으면, 해당 작업은 Lambda 응답의 각 최상위 키를 출력 변수로 변환합니다. 자세한 내용은 ['AWS Lambda 간접 호출' 변수](#) 단원을 참조하십시오.

응답 필터를 사용하여 생성된 출력 변수를 실제로 사용하려는 변수로 제한하는 것이 좋습니다.

해당 UI: 구성 탭/응답 필터 - 선택 사항

Outputs

(*LambdaInvoke*/Outputs)

(선택 사항)

워크플로 실행 중에 작업에 의해 출력되는 데이터를 정의합니다.

해당 UI: 출력 탭

Artifacts

(*LambdaInvoke*/Outputs/Artifacts)

(선택 사항)

작업에서 생성된 아티팩트를 지정합니다. 이러한 아티팩트를 다른 작업의 입력으로 참조할 수 있습니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

해당 UI: 출력 탭/아티팩트/빌드 아티팩트 이름

Name

(*LambdaInvoke*/Outputs/Artifacts/Name)

(선택 사항)

Lambda 함수에서 반환되는 Lambda 응답 페이로드를 포함할 아티팩트의 이름을 지정합니다. 기본값은 `lambda_artifacts`입니다. 아티팩트를 지정하지 않으면, 작업의 로그에서 Lambda 응답 페이로드를 볼 수 있습니다. 이 로그는 CodeCatalyst 콘솔의 작업에 대한 로그 탭에서 사용할 수 있습니다. 로그 보기에 대한 자세한 내용은 [워크플로 실행 상태 및 세부 정보 보기](#) 섹션을 참조하세요.

해당 UI: 출력 탭/아티팩트/빌드 아티팩트 이름

Files

(*LambdaInvoke*/Outputs/Artifacts/Files)

(선택 사항)

아티팩트에 포함할 파일을 지정합니다. Lambda 응답 페이로드 파일이 포함 `lambda-response.json` 되도록 을 지정해야 합니다.

해당 UI: 출력 탭/아티팩트/빌드에서 생성된 파일

Amazon ECS 작업 정의 수정

이 섹션에서는 CodeCatalyst 워크플로를 사용하여 Amazon Elastic Container Service(Amazon ECS) [작업 정의 파일](#)의 `image` 필드를 업데이트하는 방법을 설명합니다. 이렇게 하려면, 워크플로에 Amazon ECS 작업 정의 렌더링 작업을 추가해야 합니다. 이 작업은 런타임 시 워크플로에서 제공하는 Docker 이미지 이름으로 작업 정의 파일의 이미지 필드를 업데이트합니다.

Note

이 작업을 사용하여 작업 정의의 `environment` 필드를 환경 변수로 업데이트할 수도 있습니다.

주제

- [이 작업을 사용해야 하는 경우](#)
- ['Amazon ECS 작업 정의 렌더링' 작업 작동 방식](#)
- ['Amazon ECS 작업 정의 렌더링' 작업에 사용되는 런타임 이미지](#)
- [예시: Amazon ECS 작업 정의 수정](#)
- ['Amazon ECS 작업 정의 렌더링' 작업 추가](#)
- [업데이트된 작업 정의 파일 보기](#)
- ['Amazon ECS 작업 정의 렌더링' 변수](#)
- ['Amazon ECS 작업 정의 렌더링' 작업 YAML](#)

이 작업을 사용해야 하는 경우

커밋 ID 또는 타임스탬프와 같은 동적 콘텐츠가 포함된 Docker 이미지를 빌드하고 태그를 지정하는 워크플로가 있는 경우 이 옵션을 사용합니다.

작업 정의 파일에 항상 동일하게 유지되는 이미지 값이 포함된 경우 이 작업을 사용하지 마세요. 이 경우 작업 정의 파일에 이미지 이름을 수동으로 입력할 수 있습니다.

'Amazon ECS 작업 정의 렌더링' 작업 작동 방식

워크플로의 빌드 및 Amazon ECS에 배포 작업과 함께 Amazon ECS 작업 정의 렌더링 작업을 사용해야 합니다. 이 작업들을 함께 사용하면 다음과 같이 작동합니다.

1. 빌드 작업은 Docker 이미지를 빌드하고 이름, 커밋 ID, 타임스탬프 또는 기타 동적 콘텐츠로 태그를 지정합니다. 예를 들어 빌드 작업은 다음과 같습니다.

```
MyECSWorkflow
Actions:
  BuildAction:
    Identifier: aws/build@v1
    ...
  Configuration:
    Steps:
      # Build, tag, and push the Docker image...
      - Run: docker build -t MyDockerImage:${WorkflowSource.CommitId} .
      ...
```

앞의 코드에서 `docker build -t` 지시문은 Docker 이미지를 빌드하고 작업 런타임 시 커밋 ID로 태그를 지정하도록 나타냅니다. 생성된 이미지 이름은 다음과 같을 수 있습니다.

`MyDockerImage:a37bd7e`

2. 렌더링 Amazon ECS 작업 정의 작업은 다음과 같이 동적으로 생성된 이미지 이름 `MyDockerImage:a37bd7e`를 작업 정의 파일에 추가합니다.

```
{
  "executionRoleArn": "arn:aws:iam::account_ID:role/codecatalyst-ecs-task-
execution-role",
  "containerDefinitions": [
    {
      "name": "codecatalyst-ecs-container",
      "image": MyDockerImage:a37bd7e,
      "essential": true,
      ...
      "portMappings": [
        {
          "hostPort": 80,
          "protocol": "tcp",
          "containerPort": 80
        }
      ]
    }
  ],
  ...
}
```

필요에 따라 렌더링 Amazon ECS 작업 정의 작업에서 다음과 같은 환경 변수를 작업 정의에 추가하도록 할 수도 있습니다.

```
{
  "executionRoleArn": "arn:aws:iam::account_ID:role/codecatalyst-ecs-task-execution-
role",
  "containerDefinitions": [
    {
      "name": "codecatalyst-ecs-container",
      "image": MyDockerImage:a37bd7e,
      ...
      "environment": [
        {
```

```

        name": "ECS_LOGLEVEL",
        value": "info"
    }
  ]
}
],
...
}

```

자세한 내용을 알아보려면 Amazon Elastic Container Service 개발자 안내서의 [환경 변수 지정](#)을 참조하세요.

3. Amazon ECS에 배포 작업은 업데이트된 작업 정의 파일을 Amazon ECS에 등록합니다. 업데이트된 작업 정의 파일을 등록하면 새 이미지 MyDockerImage:a37bd7e가 Amazon ECS에 배포됩니다.

'Amazon ECS 작업 정의 렌더링' 작업에 사용되는 런타임 이미지

Amazon ECS 작업 정의 렌더링 작업은 [2022년 11월 이미지](#)에서 실행됩니다. 자세한 내용은 [활성 이미지](#) 단원을 참조하십시오.

예시: Amazon ECS 작업 정의 수정

다음은 빌드 및 배포 작업과 함께 렌더링 Amazon ECS 작업 정의 작업을 포함하는 전체 워크플로의 예입니다. 워크플로의 목적은 Amazon ECS 클러스터에 Docker 이미지를 빌드하고 배포하는 것입니다. 워크플로는 순차적으로 실행되는 다음 구성 요소로 구성됩니다.

- 트리거 - 이 트리거는 소스 리포지토리에 변경 사항을 푸시할 때 워크플로 실행을 자동으로 시작합니다. 트리거에 대한 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 주제를 참조하세요.
- 빌드 작업(BuildDocker) - 트리거 시 작업은 Dockerfile을 사용하여 Docker 이미지를 빌드하고, 커밋 ID로 태그를 지정하고, 이미지를 Amazon ECR로 푸시합니다. 빌드 작업에 대한 자세한 내용은 [워크플로로 빌드하기](#) 섹션을 참조하세요.
- 렌더링 Amazon ECS 작업 정의 작업(RenderTaskDef) - 빌드 작업이 완료되면 이 작업은 소스 리포지토리의 루트에 있는 기존 taskdef.json를 올바른 커밋 ID가 포함된 image 필드 값으로 업데이트합니다. 업데이트된 파일을 새 파일 이름(task-definition-random-string.json)으로 저장한 다음 이 파일이 포함된 출력 아티팩트를 생성합니다. 렌더링 작업은 task-definition라는 변수 역시 생성하며, 이를 새 작업 정의 파일의 이름으로 설정합니다. 아티팩트와 변수는 배포 작업, 즉 다음 작업을 사용합니다.
- Amazon ECS에 배포 작업(DeployToECS) - Amazon ECS에 배포 작업 정의 작업이 완료되면, Amazon ECS에 배포 작업은 렌더링 작업(TaskDefArtifact)에서 생성된 출력 아티팩트를 찾고

그 안의 `task-definition-random-string.json` 파일을 발견해 Amazon ECS 서비스에 등록합니다. 그런 다음 Amazon ECS 서비스는 `task-definition-random-string.json` 파일의 지침에 따라 Amazon ECS 클러스터 내에서 Amazon ECS 작업 및 연결된 Docker 이미지 컨테이너를 실행합니다.

```
Name: codecatalyst-ecs-workflow
```

```
SchemaVersion: 1.0
```

```
Triggers:
```

```
- Type: PUSH
```

```
  Branches:
```

```
    - main
```

```
Actions:
```

```
  BuildDocker:
```

```
    Identifier: aws/build@v1
```

```
    Environment:
```

```
      Name: codecatalyst-ecs-environment
```

```
    Connections:
```

```
      - Name: codecatalyst-account-connection
```

```
        Role: codecatalyst-ecs-build-role
```

```
    Inputs:
```

```
      Variables:
```

```
        - Name: REPOSITORY_URI
```

```
          Value: 111122223333.dkr.ecr.us-east-2.amazonaws.com/codecatalyst-ecs-image-repo
```

```
        - Name: IMAGE_TAG
```

```
          Value: ${WorkflowSource.CommitId}
```

```
    Configuration:
```

```
      Steps:
```

```
        #pre_build:
```

```
          - Run: echo Logging in to Amazon ECR...
```

```
          - Run: aws --version
```

```
          - Run: aws ecr get-login-password --region us-east-2 | docker login --username
```

```
AWS --password-stdin 111122223333.dkr.ecr.us-east-2.amazonaws.com
```

```
        #build:
```

```
          - Run: echo Build started on `date`
```

```
          - Run: echo Building the Docker image...
```

```
          - Run: docker build -t $REPOSITORY_URI:latest .
```

```
          - Run: docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
```

```
        #post_build:
```

```
          - Run: echo Build completed on `date`
```

```
          - Run: echo Pushing the Docker images...
```

- Run: docker push \$REPOSITORY_URI:latest
- Run: docker push \$REPOSITORY_URI:\$IMAGE_TAG

RenderTaskDef:**DependsOn:**

- BuildDocker

Identifier: aws/ecs-render-task-definition@v1

Inputs:**Variables:**

- Name: REPOSITORY_URI
Value: 111122223333.dkr.ecr.us-east-2.amazonaws.com/codecatalyst-ecs-image-repo
- Name: IMAGE_TAG
Value: \${WorkflowSource.CommitId}

Configuration:

task-definition: taskdef.json

container-definition-name: codecatalyst-ecs-container

image: \$REPOSITORY_URI:\$IMAGE_TAG

The output artifact contains the updated task definition file.

The new file is prefixed with 'task-definition'.

The output variable is set to the name of the updated task definition file.

Outputs:**Artifacts:**

- Name: TaskDefArtifact
Files:
- "task-definition"

Variables:

- task-definition

DeployToECS:

Identifier: aws/ecs-deploy@v1

Environment:

Name: codecatalyst-ecs-environment

Connections:

- Name: codecatalyst-account-connection
Role: codecatalyst-ecs-deploy-role

#Input artifact contains the updated task definition file.

Inputs:

Sources: []

Artifacts:

- TaskDefArtifact

Configuration:

region: us-east-2

cluster: codecatalyst-ecs-cluster

```
service: codecatalyst-ecs-service
task-definition: ${RenderTaskDef.task-definition}
```

'Amazon ECS 작업 정의 렌더링' 작업 추가

다음 지침을 사용하여 워크플로에 Amazon ECS 작업 정의 렌더링 작업을 추가합니다.

사전 조건

시작하기 전에, Docker 이미지를 동적으로 생성하는 빌드 작업이 포함된 워크플로가 있는지 확인합니다. 자세한 내용은 앞의 [워크플로 예시](#)를 참조하세요.

Visual

시각적 편집기를 사용하여 'Amazon ECS 작업 정의 렌더링' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.
8. 드롭다운 목록에서 Amazon CodeCatalyst를 선택합니다.
9. Amazon ECS 작업 정의 렌더링 작업을 검색하고, 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
 - 또는
 - Amazon ECS 작업 정의 렌더링을 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 소스 보기를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
10. 입력 및 구성 탭에서 필요에 따라 필드를 작성합니다. 각 필드의 설명은 '[Amazon ECS 작업 정의 렌더링' 작업 YAML](#) 섹션을 참조하세요. 이 참조는 YAML 및 시각적 편집기 모두에 나타나는 각 필드(및 해당 YAML 속성 값)에 대한 자세한 정보를 제공합니다.

11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

YAML 편집기를 사용하여 'Amazon ECS 작업 정의 렌더링' 작업을 추가하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
 2. 프로젝트를 선택합니다.
 3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
 4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
 5. 편집을 선택합니다.
 6. YAML을 선택합니다.
 7. 왼쪽 상단에서 + 작업을 선택하여 작업 카탈로그를 엽니다.
 8. 드롭다운 목록에서 Amazon CodeCatalyst를 선택합니다.
 9. Amazon ECS 작업 정의 렌더링 작업을 검색하고, 다음 중 하나를 수행합니다.
 - 더하기 기호(+)를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
- 또는
- Amazon ECS 작업 정의 렌더링을 선택합니다. 작업 세부 정보 대화 상자가 나타납니다. 이 대화 상자에서 다음을 수행합니다.
 - (선택 사항) 소스 보기를 선택하여 [작업의 소스 코드를 봅니다](#).
 - 워크플로에 추가를 선택하여 워크플로 다이어그램에 작업을 추가하고 구성 창을 엽니다.
10. 필요에 따라 YAML 코드의 속성을 수정합니다. 사용 가능한 각 속성에 대한 설명은 '[Amazon ECS 작업 정의 렌더링' 작업 YAML](#)에서 볼 수 있습니다.
 11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
 12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

다음 단계

렌더링 작업을 추가한 후, [워크플로를 사용하여 Amazon ECS에 배포](#)의 지침에 따라 Amazon ECS에 배포 작업을 워크플로에 추가합니다. 배포 작업을 추가하는 동안 다음을 수행합니다.

1. 배포 작업의 입력 탭의 아티팩트 - 선택 사항에서 렌더링 작업으로 생성된 아티팩트를 선택합니다. 여기에는 업데이트된 작업 정의 파일이 포함되어 있습니다.

아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

2. 배포 작업의 구성 탭의 작업 정의 필드에서 RenderTaskDef와 같이 **## ##**이 렌더링 작업의 이름인 `${action-name}.task-definition` 작업 변수를 지정합니다. 렌더링 작업은 이 변수를 태스크 정의 파일의 새 이름으로 설정합니다.

변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

배포 작업을 구성하는 방법에 대한 자세한 내용은 앞의 [워크플로 예시](#)를 참조하세요.

업데이트된 작업 정의 파일 보기

업데이트된 작업 정의 파일의 이름과 내용을 볼 수 있습니다.

Amazon ECS 작업 정의 렌더링 작업이 파일을 처리한 후에 업데이트된 작업 정의 파일의 이름을 볼 수 있습니다.

1. 완료된 렌더링 작업이 포함된 실행을 찾습니다.
 - a. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
 - b. 프로젝트를 선택합니다.
 - c. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
 - d. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
 - e. 완료된 렌더링 작업이 포함된 실행을 선택합니다.
2. 워크플로 다이어그램에서 렌더링 작업을 선택합니다.
3. 출력을 선택합니다.
4. 변수를 선택합니다.
5. 작업 정의 파일 이름이 표시됩니다. 이 이름은 task-definition--259-0a2r7gx1TF5X-.json와 유사합니다.

업데이트된 작업 정의 파일의 내용을 보려면

1. 완료된 렌더링 작업이 포함된 실행을 찾습니다.

- a. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
 - b. 프로젝트를 선택합니다.
 - c. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
 - d. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
 - e. 완료된 렌더링 작업이 포함된 실행을 선택합니다.
2. 워크플로 실행의 상단에서 비주얼 및 YAML 옆에 있는 워크플로 출력을 선택합니다.
 3. 아티팩트 섹션에서 업데이트된 작업 정의 파일이 포함된 아티팩트 옆에 있는 다운로드를 선택합니다. 이 아티팩트에는 렌더링 작업의 이름으로 설정된 생성 기준 열이 있습니다.
 4. .zip 파일을 열어 작업 정의 .json 파일을 봅니다.

'Amazon ECS 작업 정의 렌더링' 변수

Amazon ECS 태스크 정의 렌더링 작업은 런타임에 다음 변수를 생성하고 설정합니다. 이를 사전 정의된 변수라고 합니다.

워크플로에서 이러한 변수를 참조하는 방법에 대한 자세한 내용은 [사전 정의된 변수 사용](#) 섹션을 참조하세요.

Key(키)	값
작업 정의	렌더링 Amazon ECS 작업 정의 작업에 의해 업데이트된 작업 정의 파일에 지정된 이름입니다. 이름은 <code>task-definition- random-string .json</code> 형식입니다. 예시: <code>task-definition--259-0a2r7gx1TF5Xr.json</code>

'Amazon ECS 작업 정의 렌더링' 작업 YAML

다음은 렌더링 Amazon ECS 작업 정의 작업의 YAML 정의입니다. 이러한 작업 사용 방법을 배우려면 [Amazon ECS 작업 정의 수정](#) 섹션을 참조하세요.

이 작업 정의는 더 광범위한 워크플로 정의 파일 내의 섹션으로 존재합니다. 이 파일에 대한 자세한 내용은 [워크플로 YAML 정의](#)를 참조합니다.

Note

이어지는 대부분의 YAML 속성에는 시각적 편집기에 해당 UI 요소가 있습니다. UI 요소를 찾으려면 Ctrl+F를 사용합니다. 요소가 연결된 YAML 속성과 함께 나열됩니다.

```
# The workflow definition starts here.
# See ### ## for details.

Name: MyWorkflow
SchemaVersion: 1.0
Actions:

# The action definition starts here.
ECSRenderTaskDefinition\_nn:
  Identifier: aws/ecs-render-task-definition@v1
  DependsOn:
    - build-action
  Compute:
    Type: EC2 | Lambda
    Fleet: fleet-name
    Timeout: timeout-minutes
  Inputs:
    # Specify a source or an artifact, but not both.
    Sources:
      - source-name-1
    Artifacts:
      - task-definition-artifact
    Variables:
      - Name: variable-name-1
        Value: variable-value-1
      - Name: variable-name-2
        Value: variable-value-2
  Configuration
    task-definition: task-definition-path
    container-definition-name: container-definition-name
    image: docker-image-name
    environment-variables:
      - variable-name-1=variable-value-1
```

```

- variable-name-2=variable-value-2
Outputs:
  Artifacts:
    - Name: TaskDefArtifact
      Files: "task-definition*"
  Variables:
    - task-definition

```

ECSRenderTaskDefinition

(필수)

작업 이름을 지정합니다. 워크플로 내의 모든 작업 이름은 고유해야 합니다. 작업 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 작업 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

기본값: ECSRenderTaskDefinition_nn.

해당 UI: 구성 탭/작업 이름

Identifier

(*ECSRenderTaskDefinition*/Identifier)

(필수)

작업을 식별합니다. 버전을 변경하려는 경우가 아니면 이 속성을 변경하지 마세요. 자세한 내용은 [사용할 작업 버전 지정](#) 섹션을 참조하세요.

기본값: aws/ecs-render-task-definition@v1.

해당 UI: 워크플로 다이어그램/ECSRenderTaskDefinition_nn/aws/ecs-render-task-definition@v1 레이블

DependsOn

(*ECSRenderTaskDefinition*/DependsOn)

(선택 사항)

이 작업을 실행하기 위해 성공적으로 실행해야 하는 작업, 작업 그룹 또는 게이트를 지정합니다.

'depends on' 함수에 대한 자세한 내용은 [작업 순서 지정](#) 섹션을 참조하세요.

해당 UI: 입력 탭/의존 - 선택 사항

Compute

(*ECSRenderTaskDefinition*/Compute)

(선택 사항)

워크플로 작업을 실행하는 데 사용되는 컴퓨팅 엔진입니다. 워크플로 수준 또는 작업 수준에서 컴퓨팅을 지정할 수 있지만 둘 다 지정할 수는 없습니다. 워크플로 수준에서 지정하면 컴퓨팅 구성이 워크플로에 정의된 모든 작업에 적용됩니다. 워크플로 수준에서는 동일한 인스턴스에서 여러 작업을 실행할 수도 있습니다. 자세한 내용은 [작업 간에 컴퓨팅 공유](#) 섹션을 참조하세요.

해당 UI: 없음

Type

(*ECSRenderTaskDefinition*/Compute/Type)

([Compute](#) 포함 시 필수)

컴퓨팅 엔진의 유형입니다. 다음 값 중 하나를 사용할 수 있습니다.

- EC2(시각 편집기) 또는 EC2(YAML 편집기)

작업 실행 중 유연성을 위해 최적화되었습니다.

- Lambda(시각 편집기) 또는 Lambda(YAML 편집기)

작업 시작 속도를 최적화했습니다.

컴퓨팅 유형에 대한 자세한 정보는 [컴퓨팅 유형](#)을 참고하세요.

해당 UI: 구성 탭/컴퓨팅 유형

Fleet

(*ECSRenderTaskDefinition*/Compute/Fleet)

(선택 사항)

워크플로 또는 워크플로 작업을 실행할 시스템 또는 플릿을 지정합니다. 온디맨드 플릿의 경우 작업이 시작되면 워크플로가 필요한 리소스를 프로비저닝하고 작업이 완료되면 시스템이 파괴됩니다. 온디맨드 플릿의 예시: `Linux.x86-64.Large`, `Linux.x86-64.XLarge`. 온디맨드 플릿에 대한 자세한 내용은 [온디맨드 플릿 속성](#) 섹션을 참조하세요.

프로비저닝된 플릿을 사용하면 워크플로 작업을 실행하도록 전용 시스템 세트를 구성할 수 있습니다. 이러한 시스템은 유휴 상태로 유지되므로 작업을 즉시 처리할 수 있습니다. 프로비저닝된 플릿에 대한 자세한 내용은 [프로비저닝된 플릿 속성](#) 섹션을 참조하세요.

Fleet 생략 시 기본값은 `Linux.x86-64.Large`입니다.

해당 UI: 구성 탭/컴퓨팅 플릿

Timeout

(*ECSRenderTaskDefinition*/Timeout)

(선택 사항)

CodeCatalyst가 작업을 종료하기 전에 작업을 실행할 수 있는 시간을 분(YAML 편집기) 또는 시간 및 분(시각적 편집기) 단위로 지정합니다. 최소값은 5분이고 최대값은 [CodeCatalyst의 워크플로 할당량](#)에 설명되어 있습니다. 기본 제한 시간은 최대 제한 시간과 동일합니다.

해당 UI: 구성 탭/제한 시간 - 선택 사항

Inputs

(*ECSRenderTaskDefinition*/Inputs)

(선택 사항)

이 Inputs 섹션에서는 워크플로 실행 중에 *ECSRenderTaskDefinition*에 필요한 데이터를 정의합니다.

Note

렌더링 Amazon ECS 작업 정의 작업당 하나의 입력(소스 또는 아티팩트)만 허용됩니다. 변수는 이 합계에 포함되지 않습니다.

해당 UI: 입력 탭

Sources

(*ECSRenderTaskDefinition*/Inputs/Sources)

(작업 정의 파일이 소스 리포지토리에 저장된 경우 필수)

태스크 정의 파일이 소스 리포지토리에 저장되어 있는 경우 해당 소스 리포지토리의 레이블을 지정합니다. 현재, WorkflowSource 레이블만 지원됩니다.

태스크 정의 파일이 소스 리포지토리에 포함되어 있지 않은 경우, 다른 작업에서 생성된 아티팩트에 있어야 합니다.

소스에 대한 자세한 내용은 [워크플로에 소스 리포지토리 연결](#) 섹션을 참조하세요.

해당 UI: 입력 탭/소스 - 선택 사항

Artifacts - input

(*ECSRenderTaskDefinition*/Inputs/Artifacts)

(작업 정의 파일이 이전 작업의 [출력 아티팩트](#)에 저장된 경우 필수)

배포하려는 작업 정의 파일이 이전 작업에서 생성된 아티팩트에 포함된 경우 여기에 해당 아티팩트를 지정합니다. 태스크 정의 파일이 아티팩트 내에 포함되어 있지 않은 경우 소스 리포지토리에 있어야 합니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

해당 UI: 구성 탭/아티팩트 - 선택 사항

Variables - input

(*ECSRenderTaskDefinition*/Inputs/Variables)

(필수)

작업에 사용할 수 있도록 하려는 입력 변수를 정의하는 이름/값 페어의 시퀀스를 지정합니다. 변수 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 변수 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

예시를 비롯한 변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

해당 UI: 입력 탭/변수 - 선택 사항

Configuration

(*ECSRenderTaskDefinition*/Configuration)

(필수)

작업의 구성 속성을 정의할 수 있는 섹션입니다.

해당 UI: 구성 탭

task-definition

(*ECSRenderTaskDefinition*/Configuration/task-definition)

(필수)

기존 작업 정의 파일의 경로를 지정합니다. 파일이 소스 리포지토리에 있는 경우 경로는 소스 리포지토리 루트 폴더를 기준으로 합니다. 파일이 이전 워크플로 작업의 아티팩트에 있는 경우 경로는 아티팩트 루트 폴더를 기준으로 합니다. 자세한 내용을 알아보려면 Amazon Elastic Container Service 개발자 안내서의 [작업 정의](#)를 참조하세요.

해당 UI: 구성 탭/작업 정의

container-definition-name

(*ECSRenderTaskDefinition*/Configuration/container-definition-name)

(필수)

Docker 이미지가 실행될 컨테이너의 이름을 지정합니다. 이 이름은 작업 정의 파일의 `containerDefinitions`, `name` 필드에서 찾을 수 있습니다. 자세한 내용은 Amazon Elastic Container Service 개발자 안내서의 [이름](#) 섹션을 참조하세요.

해당 UI: 구성 탭/컨테이너 이름

image

(*ECSRenderTaskDefinition*/Configuration/image)

(필수)

Render Amazon ECS 작업 정의 작업을 작업 정의 파일에 추가할 Docker 이미지의 이름을 지정합니다. 이 작업은 작업 정의 파일의 `containerDefinitions`, `image` 필드에 이 이름을 추가합니다. `image` 필드에 값이 이미 있는 경우, 이 작업이 값을 덮어씁니다. 이미지 이름에 변수를 포함할 수 있습니다.

예시:

`MyDockerImage:${WorkflowSource.CommitId}` 를 지정하면 `MyDockerImage:commit-id` 작업이 작업 정의 파일에 추가되며, 여기서 `commit-id`는 워크플로에서 런타임에 생성된 커밋 ID입니다.

`my-ecr-repo/image-repo:$(date +%m-%d-%y-%H-%m-%s)`를 지정하면 작업이 태스크 정의 파일에 `my-ecr-repo /image-repo:date +%m-%d-%y-%H-%m-%s`를 추가합니다. 여기서 `my-ecr-repo`는 Amazon Elastic Container Registry(ECR)의 URI이며, `date +%m-%d-%y-%H-%m-%s`는 워크플로에서 런타임에 생성된 month-day-year-hour-minute-second 형식의 타임스탬프입니다.

`image` 필드에 대한 자세한 내용을 알아보려면, Amazon Elastic Container Service 개발자 안내서의 [이미지](#)를 참조하세요. 변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

해당 UI: 구성 탭/이미지 이름

environment-variables

(*ECSRenderTaskDefinition*/Configuration/environment-variables)

(필수)

Amazon ECS 작업 정의 렌더링 작업을 작업 정의 파일에 추가할 환경 변수를 지정합니다. 작업은 작업 정의 파일의 `containerDefinitions`, `environment` 필드에 변수를 추가합니다. 파일에 변수가 이미 있는 경우, 작업은 기존 변수의 값을 덮어쓰고 새 변수를 추가합니다. 자세한 내용을 알아보려면 Amazon Elastic Container Service 개발자 안내서의 [환경 변수 지정](#)을 참조하세요.

해당 UI: 구성 탭/환경 변수 - 선택 사항

Outputs

(*ECSRenderTaskDefinition*/Outputs)

(필수)

워크플로 실행 중에 작업에 의해 출력되는 데이터를 정의합니다.

해당 UI: 출력 탭

Artifacts

(*ECSRenderTaskDefinition*/Outputs/Artifacts)

(필수)

작업에서 생성된 아티팩트를 지정합니다. 이러한 아티팩트를 다른 작업의 입력으로 참조할 수 있습니다.

예시를 포함해 아티팩트에 대한 자세한 내용은 [작업 간 아티팩트 및 파일 공유](#) 섹션을 참조하세요.

해당 UI: 출력 탭/아티팩트

Name

(*ECSRenderTaskDefinition*/Outputs/Artifacts/Name)

(필수)

업데이트된 작업 정의 파일을 포함할 아티팩트의 이름을 지정합니다. 기본값은 MyTaskDefinitionArtifact입니다. 그런 다음 이 아티팩트를 Amazon ECS에 배포 작업에 대한 입력으로 지정해야 합니다. 이 아티팩트를 Amazon ECS에 배포 작업에 입력으로 추가하는 방법을 알아보려면 [예시: Amazon ECS 작업 정의 수정](#) 섹션을 참조하세요.

해당 UI: 출력 탭/아티팩트/이름

Files

(*ECSRenderTaskDefinition*/Outputs/Artifacts/Files)

(필수)

아티팩트에 포함할 파일을 지정합니다. task-definition-으로 시작하는 업데이트된 작업 정의 파일이 포함되도록 task-definition-*을 지정해야 합니다.

해당 UI: 출력 탭/아티팩트/파일

Variables

(*ECSRenderTaskDefinition*/Outputs/Variables)

(필수)

렌더링 작업에서 설정할 변수의 이름을 지정합니다. 렌더링 작업은 이 변수의 값을 업데이트된 작업 정의 파일 이름(예: `task-definition-random-string.json`)으로 설정합니다. 그런 다음 Amazon ECS 작업에 배포 작업의 작업 정의(시각 편집기) 또는 `task-definition`(yaml 편집기) 속성에서 이 변수를 지정해야 합니다. 이 변수를 Amazon ECS에 배포 작업에 추가하는 방법을 알아보려면 [예시: Amazon ECS 작업 정의 수정](#) 섹션을 참조하세요.

기본값: `task-definition`

해당 UI: 출력 탭/변수/이름 필드

워크플로에서 변수 사용

변수는 Amazon CodeCatalyst 워크플로에서 참조할 수 있는 정보를 포함하는 키-값 페어입니다. 변수의 값 부분은 워크플로가 실행될 때 실제 값으로 대체됩니다.

워크플로에서 사용할 수 있는 변수 유형에는 다음 두 가지가 있습니다.

- 사용자 정의 변수 - 사용자가 정의한 키-값 페어입니다.
- 사전 정의된 변수 - 워크플로에서 자동으로 내보내는 키-값 페어입니다. 정의할 필요가 없습니다.

워크플로에 대한 자세한 내용은 [워크플로를 사용하여 빌드, 테스트 및 배포](#) 섹션을 참조하세요.

Note

CodeCatalyst는 변수처럼 작동하고 다른 작업에서 참조할 수 있는 [GitHub 출력 파라미터](#)도 지원합니다. 자세한 내용은 [GitHub 출력 파라미터 내보내기](#) 및 [GitHub 출력 파라미터 참조](#) 섹션을 참조하세요.

주제

- [사용자 정의된 변수 사용](#)
- [사전 정의된 변수 사용](#)

사용자 정의된 변수 사용

사용자 정의 변수는 사용자가 정의하는 키-값 페어입니다. 두 가지의 유형이 있습니다.

- 일반 텍스트 변수 또는 간단히 변수 - 워크플로 정의 파일 내에서 일반 텍스트로 정의하는 키-값 페어입니다.
- 보안 암호 - Amazon CodeCatalyst 콘솔의 별도의 보안 암호 페이지에서 정의하는 키-값 페어입니다. 키(이름)는 퍼블릭 레이블이며, 값은 비공개로 유지하려는 정보를 포함합니다. 워크플로 정의 파일에서 키만 지정하면 됩니다. 워크플로 정의 파일에 암호 및 기타 민감한 정보 대신 보안 암호를 사용합니다.

Note

간결성을 위해 이 가이드에서는 변수라는 용어를 일반 텍스트 변수를 의미하는 용어로 사용합니다.

변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

주제

- [변수의 예](#)
- [변수 정의](#)
- [보안 암호 정의](#)
- [다른 작업에서 사용할 수 있도록 변수 내보내기](#)
- [변수를 정의하는 작업에서 변수 참조](#)
- [다른 작업에 의한 변수 출력 참조](#)
- [보안 암호 참조](#)

변수의 예

다음 예시에서는 워크플로 정의 파일에서 변수를 정의하고 참조하는 방법을 보여줍니다.

변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

예제

- [예시: Inputs 속성을 사용하여 변수 정의](#)
- [예시: Steps 속성을 사용하여 변수 정의](#)
- [예시: Outputs 속성을 사용하여 변수 내보내기](#)

- [예시: 동일한 작업에 정의된 변수 참조](#)
- [예시: 다른 작업에 정의된 변수 참조](#)
- [예시: 보안 암호 참조](#)

예시: Inputs 속성을 사용하여 변수 정의

다음 예시에서는 Inputs 섹션에서 두 개의 변수인 VAR1 및 VAR2를 정의하는 방법을 보여줍니다.

```
Actions:
  Build:
    Identifier: aws/build@v1
    Inputs:
      Variables:
        - Name: VAR1
          Value: "My variable 1"
        - Name: VAR2
          Value: "My variable 2"
```

예시: Steps 속성을 사용하여 변수 정의

다음 예시에서는 Steps 섹션에서 DATE 변수를 명시적으로 정의하는 방법을 보여줍니다.

```
Actions:
  Build:
    Identifier: aws/build@v1
    Configuration:
      Steps:
        - Run: DATE=$(date +%m-%d-%y)
```

예시: Outputs 속성을 사용하여 변수 내보내기

다음 예시에서는 두 개의 변수인 REPOSITORY-URI 및 TIMESTAMP를 정의하고 Outputs 섹션을 사용하여 내보내는 방법을 보여줍니다.

```
Actions:
  Build:
    Identifier: aws/build@v1
    Inputs:
      Variables:
        - Name: REPOSITORY-URI
```

```

    Value: 111122223333.dkr.ecr.us-east-2.amazonaws.com/codecatalyst-ecs-image-
repo
  Configuration:
    Steps:
      - Run: TIMESTAMP=$(date +%m-%d-%y-%H-%m-%s)
    Outputs:
      Variables:
        - REPOSITORY-URI
        - TIMESTAMP

```

예시: 동일한 작업에 정의된 변수 참조

다음 예시에서는 MyBuildAction에서 VAR1 변수를 지정한 다음 \$VAR1를 사용하여 동일한 작업에서 변수를 참조하는 방법을 보여줍니다.

```

Actions:
  MyBuildAction:
    Identifier: aws/build@v1
    Inputs:
      Variables:
        - Name: VAR1
          Value: my-value
    Configuration:
      Steps:
        - Run: $VAR1

```

예시: 다른 작업에 정의된 변수 참조

다음 예시에서는 BuildActionA에서 TIMESTAMP 변수를 지정하고 Outputs 속성을 사용하여 내보낸 다음 \${BuildActionA.TIMESTAMP}를 사용하여 BuildActionB에서 참조하는 방법을 보여줍니다.

```

Actions:
  BuildActionA:
    Identifier: aws/build@v1
    Configuration:
      Steps:
        - Run: TIMESTAMP=$(date +%m-%d-%y-%H-%m-%s)
    Outputs:
      Variables:
        - TIMESTAMP
  BuildActionB:

```

```

Identifier: aws/build@v1
Configuration:
  Steps:
    - Run: docker build -t my-ecr-repo/image-repo:latest .
    - Run: docker tag my-ecr-repo/image-repo:${BuildActionA.TIMESTAMP}

# Specifying just '$TIMESTAMP' here will not work
# because TIMESTAMP is not a variable
# in the BuildActionB action.

```

예시: 보안 암호 참조

다음 예시에서는 my-password 보안 암호를 참조하는 방법을 보여줍니다. my-password는 보안 암호의 키입니다. 이 보안 암호의 키와 해당 암호 값은 워크플로 정의 파일에서 사용하기 전에 CodeCatalyst 콘솔의 보안 암호 페이지에서 지정해야 합니다. 자세한 내용은 [보안 암호를 사용하여 데이터 마스킹](#) 섹션을 참조하세요.

```

Actions:
  BuildActionA:
    Identifier: aws/build@v1
    Configuration:
      Steps:
        - Run: curl -u LiJuan:${Secrets.my-password} https://example.com

```

변수 정의

두 가지 방법으로 변수를 정의할 수 있습니다.

- 워크플로 작업 Inputs 섹션의 - ['입력' 섹션의 변수 정의](#)를 참조하세요.
- 워크플로 작업 Steps 섹션의 - ['단계' 섹션의 변수 정의](#)를 참조하세요.

Note

이 Steps 메서드는 CodeCatalyst 빌드, 테스트 및 GitHub Actions 작업에서만 작동합니다. Steps 섹션이 포함된 유일한 작업이기 때문입니다.

예시는 [변수의 예](#) 섹션을 참조하세요.

변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

Visual

'입력' 섹션에서 변수 정의(시각적 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 워크플로 다이어그램에서 변수를 설정하려는 작업을 선택합니다.
8. 입력을 선택합니다.
9. 변수 - 선택 사항에서 변수 추가를 선택한 후 다음을 수행합니다.

작업에 사용할 수 있도록 하려는 입력 변수를 정의하는 이름/값 페어의 시퀀스를 지정합니다. 변수 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 변수 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

예시를 비롯한 변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

10. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
11. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

'입력' 섹션에서 변수 정의(YAML 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.

- 워크플로 작업에서 다음과 유사한 코드를 추가합니다.

```

action-name:
  Inputs:
  Variables:
    - Name: variable-name
      Value: variable-value

```

더 많은 예시는 [변수의 예](#)를 참조합니다. 자세한 내용은 작업에 해당하는 [워크플로 YAML 정의](#) 섹션을 참조하세요.

- (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
- 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

Visual

'단계' 섹션에서 변수 정의(시각 편집기)

- <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
- 프로젝트를 선택합니다.
- 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
- 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
- 편집을 선택합니다.
- 비주얼을 선택합니다.
- 워크플로 다이어그램에서 변수를 설정하려는 작업을 선택합니다.
- 구성을 선택합니다.
- 사용 가능한 셸 명령 또는 GitHub Actions YAML 중 하나에서 명시적으로 또는 암시적으로 작업 Steps의 변수를 정의합니다.
 - 변수를 명시적으로 정의하려면 Steps 섹션으로 직접 bash 명령에 포함시킵니다.
 - 변수를 암시적으로 정의하려면 작업의 Steps 섹션에서 참조하는 파일에 변수를 지정합니다.

예시는 [변수의 예](#) 섹션을 참조하세요. 자세한 내용은 작업에 해당하는 [워크플로 YAML 정의](#) 섹션을 참조하세요.

- (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.

11. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

'단계' 섹션에서 변수 정의(YAML 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 워크플로 작업에서 작업의 Steps 섹션에서 변수를 명시적으로 또는 암시적으로 정의합니다.
 - 변수를 명시적으로 정의하려면 Steps 섹션으로 직접 bash 명령에 포함시킵니다.
 - 변수를 암시적으로 정의하려면 작업의 Steps 섹션에서 참조하는 파일에 변수를 지정합니다.

예시는 [변수의 예](#) 섹션을 참조하세요. 자세한 내용은 작업에 해당하는 [워크플로 YAML 정의](#) 섹션을 참조하세요.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

보안 암호 정의

CodeCatalyst 콘솔의 보안 암호 페이지에서 보안 암호를 정의합니다. 자세한 내용은 [보안 암호를 사용하여 데이터 마스킹](#) 섹션을 참조하세요.

예를 들어 보안 암호는 다음과 같이 정의할 수 있습니다.

- 이름(키): **my-password**
- 값: **^*H3#!b9**

보안 암호가 정의된 후 워크플로 정의 파일에 보안 암호 키(**my-password**)를 지정할 수 있습니다. 이렇게 하는 방법의 예는 [예시: 보안 암호 참조](#) 섹션을 참조하세요.

다른 작업에서 사용할 수 있도록 변수 내보내기

다음 지침에 따라 작업에서 변수를 내보내 다른 작업에서 참조할 수 있도록 합니다.

변수를 내보내기 전에 다음 사항에 유의하세요.

- 변수가 정의된 작업 내에서 변수를 참조하기만 하면 되는 경우에는 내보낼 필요가 없습니다.
- 모든 작업이 변수 내보내기를 지원하는 것은 아닙니다. 작업이 이 특성을 지원하는지 확인하려면 다음 시각적 편집기 지침을 실행하고 작업에 출력 탭의 변수 버튼이 포함되어 있는지 확인합니다. 그렇다면 변수 내보내기가 지원됩니다.
- GitHub 작업에서 변수를 내보내려면 [GitHub 출력 파라미터 내보내기](#) 섹션을 참조하세요.

변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

사전 조건

내보낼 변수를 정의했는지 확인합니다. 자세한 내용은 [변수 정의](#) 섹션을 참조하세요.

Visual

변수 내보내기(시각적 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 워크플로 다이어그램에서 변수를 내보낼 작업을 선택합니다.
8. 출력을 선택합니다.
9. 변수 - 선택 사항에서 변수 추가를 선택한 후 다음을 수행합니다.

작업을 내보낼 변수의 이름을 지정합니다. 이 변수는 동일한 작업의 Inputs 또는 Steps 섹션에 이미 정의되어 있어야 합니다.

10. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.

11. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

YAML

변수 내보내기(YAML 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 변수를 내보내려는 작업에서 다음과 유사한 코드를 추가합니다.

```
action-name:
  Outputs:
    Variables:
      - Name: variable-name
```

더 많은 예시는 [변수의 예](#)를 참조합니다.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

변수를 정의하는 작업에서 변수 참조

다음 지침에 따라 변수를 정의하는 작업에서 변수를 참조합니다.

Note

GitHub 작업에서 생성된 변수를 참조하려면 [GitHub 출력 파라미터 참조](#) 섹션을 참조하세요.

변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

사전 조건

참조하려는 변수를 정의했는지 확인합니다. 자세한 내용은 [변수 정의](#) 섹션을 참조하세요.

Visual

사용할 수 없습니다. YAML을 선택하여 YAML 지침을 봅니다.

YAML

변수를 정의하는 작업에서 변수 참조

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. 참조하려는 변수를 정의하는 CodeCatalyst 액션에서 다음 bash 구문을 사용하여 변수를 추가합니다.

```
$variable-name
```

예제:

```
MyAction:
  Configuration:
    Steps:
      - Run: $variable-name
```

더 많은 예시는 [변수의 예](#)를 참조합니다. 자세한 내용은 [워크플로 YAML 정의](#)에서 작업에 대한 참조 정보를 참조하세요.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

다른 작업에 의한 변수 출력 참조

다음 지침에 따라 다른 작업의 변수를 참조합니다.

Note

GitHub 작업의 변수 출력을 참조하려면 [GitHub 출력 파라미터 참조](#) 섹션을 참조하세요.

변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

사전 조건

참조하려는 변수를 내보냈는지 확인합니다. 자세한 내용은 [다른 작업에서 사용할 수 있도록 변수 내보내기](#) 섹션을 참조하세요.

Visual

사용할 수 없습니다. YAML을 선택하여 YAML 지침을 봅니다.

YAML

다른 작업의 변수 출력 참조(YAML 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. CodeCatalyst 작업에서 다음 구문을 사용하여 변수에 대한 참조를 추가합니다.

```
${action-group-name.action-name.variable-name}
```

다음과 같이 바꿉니다.

- 변수를 출력하는 작업이 포함된 작업 그룹의 이름이 포함된 *action-group-name*입니다.

Note

작업 그룹이 없거나 변수가 동일한 작업 그룹의 작업에서 생성되는 경우 *action-group-name*을 생략할 수 있습니다.

- **action-name**을 변수를 출력하는 작업의 이름으로 바꿉니다.
- **variable-name**을 변수 이름으로 바꿉니다.

예제:

```
MySecondAction:
  Configuration:
    Steps:
      - Run: ${MyFirstAction.TIMESTAMP}
```

더 많은 예시는 [변수의 예](#)를 참조합니다. 자세한 내용은 작업에 해당하는 [워크플로 YAML 정의](#) 섹션을 참조하세요.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

보안 암호 참조

워크플로 정의 파일에서 보안 암호 참조에 대한 지침은 [보안 암호 사용](#) 섹션을 참조하세요.

관련 예시는 [예시: 보안 암호 참조](#) 섹션을 참조하세요

사전 정의된 변수 사용

사전 정의된 변수는 워크플로에서 자동으로 내보내고 워크플로 작업에 사용할 수 있는 키-값 페어입니다.

변수에 대한 자세한 내용은 [워크플로에서 변수 사용](#) 섹션을 참조하세요.

주제

- [사전 정의된 변수 참조의 예](#)
- [사전 정의된 변수 참조](#)
- [워크플로에서 내보내는 사전 정의된 변수 확인](#)
- [사전 정의된 변수 목록](#)

사전 정의된 변수 참조의 예

다음 예시에서는 워크플로 정의 파일에서 사전 정의된 변수를 참조하는 방법을 보여줍니다.

사전 정의된 변수에 대한 자세한 내용은 [사전 정의된 변수 사용](#) 섹션을 참조하세요.

예시

- [예시: 'CommitId' 사전 정의된 변수 참조](#)
- [예시: 'BranchName' 사전 정의된 변수 참조](#)

예시: 'CommitId' 사전 정의된 변수 참조

다음 예시에서는 MyBuildAction 작업에서 사전 정의된 CommitId 변수를 참조하는 방법을 보여줍니다. CommitId 변수는 CodeCatalyst에서 자동으로 출력됩니다. 자세한 내용은 [사전 정의된 변수 목록](#) 섹션을 참조하세요.

예시에서는 빌드 작업에서 사용되는 변수를 보여주지만 모든 작업에서 CommitId를 사용할 수 있습니다.

```
MyBuildAction:
  Identifier: aws/build@v1
  Inputs:
    Sources:
      - WorkflowSource
  Configuration:
    Steps:
      #Build Docker image and tag it with a commit ID
      - Run: docker build -t image-repo/my-docker-image:latest .
      - Run: docker tag image-repo/my-docker-image:${WorkflowSource.CommitId}
```

예시: 'BranchName' 사전 정의된 변수 참조

다음 예시에서는 CDKDeploy 작업에서 사전 정의된 BranchName 변수를 참조하는 방법을 보여줍니다. BranchName 변수는 CodeCatalyst에서 자동으로 출력됩니다. 자세한 내용은 [사전 정의된 변수 목록](#) 섹션을 참조하세요.

예시에서는 AWS CDK 배포 작업에 사용되는 변수를 보여 주지만, 모든 작업에서 BranchName을 사용할 수 있습니다.

```
CDKDeploy:
  Identifier: aws/cdk-deploy@v2
  Inputs:
    Sources:
```

```
- WorkflowSource
Configuration:
  StackName: app-stack-${WorkflowSource.BranchName}
```

사전 정의된 변수 참조

Amazon CodeCatalyst 워크플로 내의 모든 작업에서 사전 정의된 변수를 참조할 수 있습니다.

다음 지침에 따라 워크플로에서 사전 정의된 변수를 참조합니다.

사전 정의된 변수에 대한 자세한 내용은 [사전 정의된 변수 사용](#) 섹션을 참조하세요.

전제 조건

CommitId와 같이 참조하려는 사전 정의된 변수의 이름을 결정합니다. 자세한 내용은 [워크플로에서 내보내는 사전 정의된 변수 확인](#) 섹션을 참조하세요.

Visual

사용할 수 없습니다. YAML을 선택하여 YAML 지침을 봅니다.

YAML

사전 정의된 변수 참조(YAML 편집기)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. CodeCatalyst 작업에서 다음 구문을 사용하여 사전 정의된 변수 참조를 추가합니다.

```
${action-group-name.action-name-or-WorkflowSource.variable-name}
```

다음과 같이 바꿉니다.

- **action-group-name**: 작업 이름의 그룹.

Note

작업 그룹이 없거나 변수가 동일한 작업 그룹의 작업에서 생성되는 경우 *action-group-name*을 생략할 수 있습니다.

- *action-name-or-WorkflowSource*:

변수를 출력하는 작업의 이름.

or

WorkflowSource, 변수가 BranchName 또는 CommitId 변수인 경우.

- *variable-name*을 변수 이름으로 바꿉니다.

예:

```
MySecondAction:
  Configuration:
    Steps:
      - Run: echo ${MyFirstECSAction.cluster}
```

또 다른 예시:

```
MySecondAction:
  Configuration:
    Steps:
      - Run: echo ${WorkflowSource.CommitId}
```

더 많은 예시는 [사전 정의된 변수 참조의 예](#)를 참조합니다. 자세한 내용은 작업에 해당하는 [워크플로 YAML 정의](#) 섹션을 참조하세요.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

워크플로에서 내보내는 사전 정의된 변수 확인

다음 절차를 사용하여 워크플로가 실행될 때 내보내는 사전 정의된 변수를 결정합니다. 그런 다음 동일한 워크플로 내에서 이러한 변수를 참조할 수 있습니다.

사전 정의된 변수에 대한 자세한 내용은 [사전 정의된 변수 사용](#) 섹션을 참조하세요.

워크플로에서 내보내는 사전 정의된 변수 확인

- 다음 중 하나를 수행합니다.
 - 워크플로를 한 번 실행합니다. 실행이 완료되면 워크플로에서 내보내는 변수가 실행 세부 정보 페이지의 변수 탭에 표시됩니다. 자세한 내용은 [워크플로 실행 상태 및 세부 정보 보기](#) 섹션을 참조하세요.
 - [사전 정의된 변수 목록](#) 섹션을 참조하세요. 이 참조에는 사전 정의된 각 변수의 변수 이름(키)과 값이 나열됩니다.

Note

워크플로 변수의 최대 총 크기는 [CodeCatalyst의 워크플로 할당량](#)에 나열되어 있습니다. 총 크기가 최대값을 초과하면 최대값에 도달한 후 발생하는 작업이 실패할 수 있습니다.

사전 정의된 변수 목록

다음 섹션을 참조하여 워크플로 실행의 일부로 CodeCatalyst 작업에 의해 자동으로 생성되는 사전 정의된 변수를 확인하세요.

사전 정의된 변수에 대한 자세한 내용은 [사전 정의된 변수 사용](#) 섹션을 참조하세요.

Note

이 목록에는 CodeCatalyst 소스 및 [CodeCatalyst 작업](#)에서 내보내는 사전 정의된 변수만 포함됩니다. GitHub Actions 또는 CodeCatalyst Labs 작업과 같은 다른 유형의 작업을 사용하는 경우 대신 [워크플로에서 내보내는 사전 정의된 변수 확인](#)을 참조하세요.

나열

Note

모든 CodeCatalyst 작업이 사전 정의된 변수를 생성하는 것은 아닙니다. 작업이 목록에 없으면 변수가 생성되지 않습니다.

- ['BranchName' 및 'CommitId' 변수](#)
- ['스 CloudFormation 템플릿 배포' 변수](#)
- ['Amazon ECS에 배포' 변수](#)
- ['Kubernetes 클러스터에 배포' 변수](#)
- ['AWS CDK 배포' 변수](#)
- ['AWS CDK 부트스트랩' 변수](#)
- ['AWS Lambda 간접 호출' 변수](#)
- ['Amazon ECS 작업 정의 렌더링' 변수](#)

보안 암호를 사용하여 데이터 마스킹

워크플로에서 인증 자격 증명과 같은 민감한 데이터를 사용해야 하는 경우가 있을 수 있습니다. 보안 암호가 포함된 리포지토리에 액세스할 수 있는 사람은 누구나 볼 수 있으므로 이러한 값을 리포지토리 어디에도 일반 텍스트로 저장하는 것은 피해야 합니다. 마찬가지로 이러한 값은 리포지토리에 파일로 표시되므로 워크플로 정의에 직접 사용해서는 안 됩니다. CodeCatalyst를 사용하면 프로젝트에 보안 암호를 추가한 다음 워크플로 정의 파일에서 암호를 참조하여 이러한 값을 보호할 수 있습니다. 작업당 최대 5개의 보안 암호를 사용할 수 있습니다.

Note

보안 암호는 워크플로 정의 파일의 암호 및 민감한 정보를 대체하는 데만 사용할 수 있습니다.

주제

- [보안 암호 생성](#)
- [보안 암호 편집](#)
- [보안 암호 사용](#)
- [보안 암호 삭제](#)

보안 암호 생성

다음 절차에 따라 보안 암호를 생성합니다. 보안 암호에는 보이지 않게 숨기고 싶은 민감한 정보가 포함되어 있습니다.

Note

보안 암호는 작업에 표시되며 파일에 기록될 때 가려지지 않습니다.

보안 암호 생성

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 CI/CD를 선택한 후, 보안 암호를 선택합니다.
3. 보안 암호 생성을 선택합니다.
4. 다음 정보를 입력합니다.

이름

보안 암호의 이름을 입력합니다.

값

보안 암호 값을 입력합니다. 표시되지 않도록 숨기고 싶은 민감한 정보입니다. 기본적으로 값은 표시되지 않습니다. 값을 표시하려면 값 표시를 선택합니다.

설명

(선택 사항) 보안 암호에 대한 설명을 입력합니다.

5. 생성(Create)을 선택합니다.

보안 암호 편집

보안 암호를 편집하려면 다음 절차에 따르세요.

보안 암호 편집

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 CI/CD를 선택한 후, 보안 암호를 선택합니다.
3. 보안 암호 목록에서 편집할 보안 암호를 선택합니다.
4. 편집을 선택합니다.
5. 다음 속성을 참조하세요.

값

보안 암호 값을 입력합니다. 이 값은 표시되지 않도록 숨기려는 값입니다. 기본적으로 값은 표시되지 않습니다.

설명

(선택 사항) 보안 암호에 대한 설명을 입력합니다.

6. 저장을 선택합니다.

보안 암호 사용

워크플로 작업에서 보안 암호를 사용하려면 암호의 참조 식별자를 얻은 다음 워크플로 작업에서 해당 식별자를 사용해야 합니다.

주제

- [보안 암호의 식별자 가져오기](#)
- [워크플로에서 보안 암호 참조](#)

보안 암호의 식별자 가져오기

다음 절차에 따라 보안 암호의 참조 식별자를 가져옵니다. 워크플로에 이 식별자를 추가합니다.

보안 암호의 참조 식별자 가져오기

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 CI/CD를 선택한 후, 보안 암호를 선택합니다.
3. 보안 암호 목록에서 사용하려는 보안 암호를 찾습니다.
4. 참조 ID 옆에서 보안 암호의 식별자를 복사합니다. 다음은 참조 ID의 구문입니다.

```
${Secrets.<name>}
```

워크플로에서 보안 암호 참조

다음 절차에 따라 워크플로에서 보안 암호를 참조합니다.

보안 암호 참조

1. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
2. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
3. 편집을 선택합니다.
4. YAML을 선택합니다.
5. 보안 암호의 식별자를 사용하도록 YAML을 수정합니다. 예를 들어 보안 암호로 저장된 사용자 이름과 암호를 curl 명령에 사용하려면 다음과 유사한 Run 명령을 사용합니다.

```
- Run: curl -u <username-secret-identifier>:<password-secret-identifier> https://example.com
```

6. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
7. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

보안 암호 삭제

다음 절차에 따라 보안 암호와 보안 암호 참조 식별자를 삭제합니다.

Note

보안 암호를 삭제하기 전에 모든 워크플로 작업에서 해당 보안 암호의 참조 식별자를 제거하는 것이 좋습니다. 참조 식별자를 삭제하지 않고 보안 암호를 삭제하면 다음에 실행할 때 작업이 실패합니다.

워크플로에서 보안 암호의 참조 식별자 삭제

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
3. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
4. 편집을 선택합니다.
5. YAML을 선택합니다.
6. 워크플로에서 다음 문자열을 검색합니다.


```
${Secrets.
```

이렇게 하면 모든 보안 암호의 모든 참조 식별자가 검색됩니다.

7. 선택한 보안 암호의 참조 식별자를 삭제하거나 일반 텍스트 값으로 바꿉니다.
8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.

보안 암호를 삭제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 CI/CD를 선택한 후, 보안 암호를 선택합니다.
3. 보안 암호 목록에서 삭제할 보안 암호를 선택합니다.
4. 삭제를 선택합니다.
5. **delete**를 입력하여 삭제를 확인합니다.
6. 삭제를 선택합니다.

워크플로의 상태 보기

워크플로 상태를 확인하여 해결해야 할 워크플로 구성 문제가 있는지 확인하거나 시작에 실패하는 실행 문제를 해결하려고 할 수 있습니다. CodeCatalyst는 워크플로의 기본 워크플로 [정의 파일](#)을 생성하거나 업데이트할 때마다 워크플로 상태를 평가합니다.

Note

워크플로 상태와 다른 워크플로의 실행 상태를 볼 수도 있습니다. 자세한 내용은 [워크플로 실행 상태 및 세부 정보 보기](#) 섹션을 참조하세요.

가능한 워크플로 상태 목록은 [CodeCatalyst의 워크플로 상태](#) 섹션을 참조하세요.

워크플로 상태 보기

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.

3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.

상태는 목록에 워크플로와 함께 표시됩니다.

5. (선택 사항) 워크플로의 이름을 선택하고 워크플로 정의 필드를 찾습니다. 워크플로 상태를 보여줍니다.

CodeCatalyst의 워크플로 할당량

다음 테이블에서는 Amazon CodeCatalyst의 워크플로에 대한 쿼터 및 제한에 대해 설명합니다.

Amazon CodeCatalyst의 할당량에 대한 자세한 내용은 [CodeCatalyst 할당량](#) 섹션을 참조하세요.

스페이스당 최대 워크플로 수	800
최대 워크플로 정의 파일 크기	256KB
단일 소스 이벤트에서 처리된 최대 워크플로 파일 수	50
단일 소스 이벤트에서 처리된 최대 파일 수	4,000
스페이스당 최대 활성 플릿 수	10
플릿당 최대 활성 컴퓨팅 인스턴스 수	20
작업당 최대 입력 아티팩트 수	10
작업당 최대 출력 아티팩트 수	10
단일 작업의 출력 변수의 최대 총 크기	120KB
출력 변수 값의 최대 길이	값을 내보내는 작업에 따라 500자 이상입니다.

 Note

값이 작업의 한도를 초과하면 값이 잘릴 수 있습니다.

워크플로 실행 중 생성된 아티팩트를 보관할 수 있는 최대 일수	30
작업당 최대 보고서 수	50
테스트 보고서당 최대 테스트 사례 수	20,000건
코드 적용 범위 보고서당 최대 파일 수	20,000건
보고서당 최대 소프트웨어 구성 분석 조사 결과 수	20,000건
정적 분석 보고서당 최대 파일 수	20,000건
페이스당 최대 동시 워크플로 실행 수	100
워크플로당 최대 작업 수	50
워크플로당 동시에 실행 중인 최대 작업 수	50
스페이스당 동시에 실행 중인 최대 작업 수	200
작업이 실행될 수 있는 최대 시간	빌드 및 테스트 작업의 경우 제한 시간은 8시간입니다. 다른 모든 작업의 경우 제한 시간은 1시간입니다.
스페이스당 AWS 계정 과 연결된 최대 환경 수	5,000
작업당 최대 보안 암호 수	5
스페이스당 최대 보안 암호 수	500,000

워크플로 실행 상태

워크플로 실행은 다음 상태 중 하나일 수 있습니다.

- 성공 - 워크플로 실행이 성공적으로 처리되었습니다.
- 실패 - 워크플로 실행에서 하나 이상의 작업이 실패했습니다.
- 진행 중 - 워크플로 실행이 현재 처리 중입니다.
- 중지됨 - 워크플로가 진행 중인 동안 사용자가 워크플로 실행을 중지했습니다.
- 중지 중 - 워크플로 실행이 현재 중지 중입니다.
- 취소됨 - 실행이 진행되는 동안 연결된 워크플로가 삭제되거나 업데이트되어 CodeCatalyst에서 워크플로 실행이 취소되었습니다.
- 대체됨 - [대체된 실행 모드](#)를 구성한 경우에만 발생합니다. 이후 워크플로 실행이 워크플로 실행을 대체했기 때문에 CodeCatalyst에서 워크플로 실행을 취소했습니다.

CodeCatalyst의 워크플로 상태

워크플로는 다음 중 하나의 상태를 가질 수 있습니다.

- 유효 - 워크플로를 실행할 수 있으며 [트리거](#)로 활성화할 수 있습니다.

워크플로를 유효로 표시하려면 다음 두 조건이 모두 true여야 합니다.

- 워크플로 정의 파일이 유효 상태여야 합니다.
- 워크플로에는 트리거, 푸시 트리거 또는 현재 브랜치의 파일을 사용하여 실행되는 푸시 트리거가 없어야 합니다. 자세한 내용은 [트리거 및 브랜치 사용 지침](#) 섹션을 참조하세요.
- 유효하지 않음 - 워크플로의 정의 파일이 유효하지 않습니다. 워크플로는 수동으로 실행하거나 트리거를 통해 자동으로 실행할 수 없습니다. 유효하지 않은 워크플로는 CodeCatalyst 콘솔에서 Workflow definition has **n** errors(또는 이와 유사한 메시지)와 함께 표시됩니다.

워크플로를 유효하지 않음으로 표시하려면 다음 조건이 true여야 합니다.

- 워크플로 정의 파일은 잘못 구성된 상태여야 합니다.

잘못 구성된 워크플로 정의 파일을 수정하려면 ['Workflow definition has n errors' 오류를 해결하려면 어떻게 해야 하나요?](#) 섹션을 참조하세요.

- 비활성 - 워크플로 정의가 유효하지만 수동으로 실행하거나 트리거를 통해 자동으로 실행할 수 없습니다.

워크플로를 비활성으로 표시하려면 다음 두 조건이 모두 true여야 합니다.

- 워크플로 정의 파일이 유효 상태여야 합니다.
- 워크플로 정의 파일에는 워크플로 정의 파일이 현재 있는 브랜치와 다른 브랜치를 지정하는 푸시 트리거가 포함되어야 합니다. 자세한 내용은 [트리거 및 브랜치 사용 지침](#) 섹션을 참조하세요.

워크플로를 비활성에서 활성으로 전환하려면 ['Workflow is inactive' 메시지를 해결하려면 어떻게 해야 하나요?](#) 섹션을 참조하세요.

Note

비활성 상태의 워크플로가 많으면 보기에서 필터링할 수 있습니다. 비활성 워크플로를 필터링하려면 워크플로 페이지 상단의 워크플로 필터링 필드를 선택하고 상태를 선택한 다음 Status != Does not equal을 선택한 다음 비활성을 선택합니다.

Note

워크플로에 나중에 제거할 리소스(예시: 패키지 리포지토리)가 지정되어 있는 경우 CodeCatalyst는 이 변경 사항을 감지하지 않고 워크플로를 계속 유효한 것으로 표시합니다. 이러한 유형의 문제는 워크플로가 실행될 때 발견됩니다.

워크플로 YAML 정의

다음은 워크플로 정의 파일의 참조 설명서입니다.

워크플로 정의 파일은 워크플로를 설명하는 YAML 파일입니다. 기본적으로 파일은 [소스 리포지토리](#)의 루트에 있는 `~/.codecatalyst/workflows/` 폴더에 저장됩니다. 파일은 확장자가 `.yml` 또는 `.yaml` 일 수 있으며 확장자는 소문자여야 합니다.

워크플로 정의 파일을 생성하고 편집하기 위해 vim과 같은 편집기를 사용하거나, CodeCatalyst 콘솔의 시각적 편집기 또는 YAML 편집기를 사용할 수 있습니다. 자세한 내용은 [CodeCatalyst 콘솔의 시각적 및 YAML 편집기 사용](#) 섹션을 참조하세요.

Note

이어지는 대부분의 YAML 속성에는 시각적 편집기에 해당 UI 요소가 있습니다. UI 요소를 찾으려면 Ctrl+F를 사용합니다. 요소가 연결된 YAML 속성과 함께 나열됩니다.

주제

- [워크플로 정의 파일의 예시](#)
- [구문 지침 및 규칙](#)
- [최상위 속성](#)

워크플로 정의 파일의 예시

다음은 간단한 워크플로 정의 파일의 예시입니다. 여기에는 몇 가지 최상위 속성, Triggers 섹션, Build 및 Test의 두 가지 작업이 있는 Actions 섹션이 포함됩니다. 자세한 내용은 [워크플로 정의 파일 정보](#) 섹션을 참조하세요.

```
Name: MyWorkflow
SchemaVersion: 1.0
RunMode: QUEUED
Triggers:
  - Type: PUSH
    Branches:
      - main
Actions:
  Build:
    Identifier: aws/build@v1
    Inputs:
      Sources:
        - WorkflowSource
    Configuration:
      Steps:
        - Run: docker build -t MyApp:latest .
  Test:
    Identifier: aws/managed-test@v1
    DependsOn:
      - Build
    Inputs:
      Sources:
        - WorkflowSource
```

Configuration:**Steps:**

- Run: npm install
- Run: npm run test

구문 지침 및 규칙

이 섹션에서는 워크플로 정의 파일의 구문 규칙과 이 참조 설명서에 사용된 이름 지정 규칙에 대해 설명합니다.

YAML 구문 지침

워크플로 정의 파일은 YAML로 작성되고 [YAML 1.1 사양](#)을 따르므로 해당 사양에서 허용되는 모든 항목도 워크플로 YAML에서 허용됩니다. YAML을 처음 사용하는 경우 유효한 YAML 코드를 제공하기 위한 몇 가지 간단한 지침이 있습니다.

- 대/소문자 구분: 워크플로 정의 파일은 대/소문자를 구분하므로 이 설명서에 표시된 케이스를 사용해야 합니다.
- 특수 문자: {, }, [,], &, *, #, ?, |, -, <, >, =, !, %, @, :, `,, 등의 특수 문자가 포함된 속성 값을 따옴표 또는 큰따옴표로 묶어 사용하는 것이 좋습니다.

따옴표를 포함하지 않으면 이전에 나열된 특수 문자가 예상치 못한 방식으로 해석될 수 있습니다.

- 속성 이름: 속성 names(속성 values가 아님)은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 따옴표나 큰따옴표를 사용하여 속성 이름에 특수 문자와 공백을 사용할 수 없습니다.

허용되지 않음:

```
'My#Build@action'
```

```
My#Build@action
```

```
My Build Action
```

허용:

```
My-Build-Action_1
```

- 이스케이프 코드: 속성 값에 이스케이프 코드(예: \n 또는 \t)가 포함된 경우 다음 지침을 따릅니다.
 - 작은따옴표를 사용하여 이스케이프 코드를 문자열로 반환합니다. 예를 들어, 'my string \n my string', my string \n my string 문자열을 반환합니다.

- 큰따옴표를 사용하여 이스케이프 코드를 구문 분석합니다. 예를 들어, "my string \n my new line"는 다음을 반환합니다.

```
my string
my new line
```

- 설명: #를 사용한 서문 설명입니다.

예제:

```
Name: MyWorkflow
# This is a comment.
SchemaVersion: 1.0
```

- 트리플 대시(---): YAML 코드에 ---를 사용하지 마세요. CodeCatalyst는 --- 이후의 모든 항목을 무시합니다.

이름 지정 규칙

이 가이드에서는 속성 및 섹션이라는 용어를 사용하여 워크플로 정의 파일의 주요 항목을 참조합니다.

- 속성은 콜론(:)을 포함하는 모든 항목입니다. 예를 들어, 다음 코드 조각에서 Name, SchemaVersion, RunMode, Type, Triggers 및 Branches 속성은 모두입니다.
- 섹션은 하위 속성이 있는 모든 속성입니다. 다음 코드 조각에는 Triggers 섹션이 하나 있습니다.

Note

이 가이드에서는 컨텍스트에 따라 '섹션'을 '속성'이라고 하며 그 반대로 볼 수 있습니다.

```
Name: MyWorkflow
SchemaVersion: 1.0
RunMode: QUEUED
Triggers:
  - Type: PUSH
    Branches:
      - main
```


최상위 속성

다음은 워크플로 정의 파일의 최상위 속성에 대한 참조 설명서입니다.

```
# Name
Name: workflow-name

# Schema version
SchemaVersion: 1.0

# Run mode
RunMode: QUEUED|SUPERSEDED|PARALLEL

# Compute
Compute:
...

# Triggers
Triggers:
...

# Actions
Actions:
...
```

Name

(필수)

워크플로의 이름입니다. 워크플로 이름은 워크플로 목록에 표시되고 알림 및 로그에 언급됩니다. 워크플로 이름과 워크플로 정의 파일 이름이 일치하거나 이름을 다르게 지정할 수 있습니다. 워크플로 이름은 고유하지 않아도 됩니다. 워크플로 이름은 영숫자 문자(a-z, A-Z, 0-9), 하이픈(-) 및 밑줄(_)로 제한됩니다. 스페이스는 허용되지 않습니다. 워크플로 이름에서 특수 문자와 공백을 활성화하는 데 따옴표를 사용할 수 없습니다.

해당 UI: 시각적 편집기/워크플로 속성/워크플로 이름

SchemaVersion

(필수)

워크플로 정의의 스키마 버전입니다. 현재 유일한 유효 값은 1.0입니다.

해당 UI: 없음

RunMode

(선택 사항)

CodeCatalyst가 여러 실행을 처리하는 방법. 다음 값 중 하나를 사용할 수 있습니다.

- QUEUED - 여러 실행이 대기 중이고 하나씩 실행됩니다. 대기열에서 최대 50회까지 실행할 수 있습니다.
- SUPERSEDED - 여러 실행이 대기 중이고 하나씩 실행됩니다. 대기열은 한 번만 실행할 수 있으므로 두 개의 실행이 같은 대기열에서 동시에 종료되면, 나중에 실행된 것이 먼저 실행된 것을 대체(인계)하고, 먼저 실행된 것은 취소됩니다.
- PARALLEL - 여러 번 동시에 실행됩니다.

생략 시 기본값은 QUEUED입니다.

자세한 내용은 [실행의 대기열 동작 구성](#) 섹션을 참조하세요.

해당 UI: 시각적 편집기/워크플로 속성/고급/실행 모드

Compute

(선택 사항)

워크플로 작업을 실행하는 데 사용되는 컴퓨팅 엔진입니다. 워크플로 수준 또는 작업 수준에서 컴퓨팅을 지정할 수 있지만 둘 다 지정할 수는 없습니다. 워크플로 수준에서 지정하면 컴퓨팅 구성이 워크플로에 정의된 모든 작업에 적용됩니다. 워크플로 수준에서는 동일한 인스턴스에서 여러 작업을 실행할 수도 있습니다. 자세한 내용은 [작업 간에 컴퓨팅 공유](#) 섹션을 참조하세요.

컴퓨팅에 대한 자세한 내용은 [컴퓨팅 및 런타임 이미지 구성](#) 섹션을 참조하세요.

해당 UI: 없음

```
Name: MyWorkflow
SchemaVersion: 1.0
...
Compute:
  Type: EC2 | Lambda
  Fleet: fleet-name
  SharedInstance: true | false
```

Type

(Compute/Type)

(Compute로 설정된 경우 필수)

컴퓨팅 엔진의 유형입니다. 다음 값 중 하나를 사용할 수 있습니다.

- EC2(시각 편집기) 또는 EC2(YAML 편집기)
작업 실행 중 유연성을 위해 최적화되었습니다.
- Lambda(시각 편집기) 또는 Lambda(YAML 편집기)
작업 시작 속도를 최적화했습니다.

컴퓨팅 유형에 대한 자세한 정보는 [컴퓨팅 유형](#)을 참고하세요.

해당 UI: 시각적 편집기/워크플로 속성/고급/컴퓨팅 유형

Fleet

(Compute/Fleet)

(선택 사항)

워크플로 또는 워크플로 작업을 실행할 시스템 또는 플릿을 지정합니다. 온디맨드 플릿의 경우 작업이 시작되면 워크플로가 필요한 리소스를 프로비저닝하고 작업이 완료되면 시스템이 파괴됩니다. 온디맨드 플릿의 예시: Linux.x86-64.Large, Linux.x86-64.XLarge. 온디맨드 플릿에 대한 자세한 내용은 [온디맨드 플릿 속성](#) 섹션을 참조하세요.

프로비저닝된 플릿을 사용하면 워크플로 작업을 실행하도록 전용 시스템 세트를 구성할 수 있습니다. 이러한 시스템은 유휴 상태로 유지되므로 작업을 즉시 처리할 수 있습니다. 프로비저닝된 플릿에 대한 자세한 내용은 [프로비저닝된 플릿 속성](#) 섹션을 참조하세요.

Fleet 생략 시 기본값은 Linux.x86-64.Large입니다.

컴퓨팅 플릿에 대한 자세한 내용은 [컴퓨팅 플릿](#) 섹션을 참조하세요.

해당 UI: 시각적 편집기/워크플로 속성/고급/컴퓨팅 플릿

SharedInstance

(Compute/SharedInstance)

(선택 사항)

작업에 대한 컴퓨팅 공유 기능을 지정합니다. 컴퓨팅 공유를 사용하면 워크플로의 작업이 동일한 인스턴스(런타임 환경 이미지)에서 실행됩니다. 다음 값 중 하나를 사용할 수 있습니다.

- TRUE는 런타임 환경 이미지가 워크플로 작업 간에 공유됨을 의미합니다.
- FALSE는 워크플로의 각 작업에 대해 별도의 런타임 환경 이미지가 시작되고 사용되므로 추가 구성 없이 아티팩트 및 변수와 같은 리소스를 공유할 수 없음을 의미합니다.

컴퓨팅 공유에 대한 자세한 내용은 [작업 간에 컴퓨팅 공유](#) 섹션을 참조하세요.

해당 UI: 없음

Triggers

(선택 사항)

이 워크플로를 위한 하나 이상의 트리거의 순서입니다. 트리거가 지정되지 않은 경우, 수동으로 워크플로를 시작해야 합니다.

트리거에 대한 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 주제를 참조하세요.

해당 UI: 시각적 편집기/워크플로 다이어그램/트리거

```
Name: MyWorkflow
SchemaVersion: 1.0
...
Triggers:
  - Type: PUSH
    Branches:
      - branch-name
    FilesChanged:
      - folder1/file
      - folder2/

  - Type: PULLREQUEST
    Events:
      - OPEN
      - CLOSED
      - REVISION
    Branches:
      - branch-name
    FilesChanged:
```

```

- file1.txt

- Type: SCHEDULE
  # Run the workflow at 10:15 am (UTC+0) every Saturday
  Expression: "15 10 ? * 7 *"
  Branches:
    - branch-name

```

Type

(Triggers/Type)

(Triggers로 설정된 경우 필수)

트리거 유형을 지정합니다. 다음 값 중 하나를 사용할 수 있습니다.

- 푸시(시각적 편집기) 또는 PUSH(YAML 편집기)

푸시 트리거는 변경 사항이 소스 리포지토리로 푸시될 때 워크플로 실행을 시작합니다. 워크플로 실행은 푸시 대상 브랜치(즉, 대상 브랜치)의 파일을 사용합니다.

- 풀 요청(시각적 편집기) 또는 PULLREQUEST(YAML 편집기)

풀 요청 트리거는 소스 리포지토리에서 풀 요청이 열리거나 업데이트되거나 닫힐 때 워크플로 실행을 시작합니다. 워크플로 실행은 가져오려는 브랜치(즉, 소스 브랜치)의 파일을 사용합니다.

- 일정(시각적 편집기) 또는 SCHEDULE(YAML 편집기)

일정 트리거는 지정한 cron 표현식으로 정의된 일정에 따라 워크플로를 실행합니다. 브랜치 파일을 사용하여 소스 리포지토리의 각 브랜치에 대해 별도의 워크플로 실행이 시작됩니다. (트리거가 활성화되는 브랜치를 제한하려면 브랜치 필드(시각적 편집기) 또는 Branches 속성(YAML 편집기)을 사용합니다.)

일정 트리거를 구성할 때는 다음 지침을 따르세요.

- 워크플로당 하나의 일정 트리거만 사용합니다.
- CodeCatalyst 스페이스에 여러 워크플로를 정의한 경우 동시에 시작하도록 10개 이하로 예약하는 것이 좋습니다.
- 실행 사이에 충분한 시간을 두고 트리거의 cron 표현식을 구성해야 합니다. 자세한 내용은 [Expression](#) 섹션을 참조하세요.

예시는 [예시: 워크플로의 트리거](#) 섹션을 참조하세요.

해당 UI: 시각적 편집기/워크플로 다이어그램/트리거/트리거 유형

Events

(Triggers/Events)

(Type 트리거가 PULLREQUEST로 설정된 경우 필수)

워크플로 실행을 시작할 풀 요청 이벤트의 유형을 지정합니다. 유효한 값은 다음과 같습니다.

- 풀 요청이 생성됨(시각적 편집기) 또는 OPEN(YAML 편집기)

풀 요청이 생성되면 워크플로 실행이 시작됩니다.

- 풀 요청이 닫힘(시각적 편집기) 또는 CLOSED(YAML 편집기)

풀 요청이 종료되면 워크플로 실행이 시작됩니다. CLOSED 이벤트의 동작은 까다롭고 예시를 통해 가장 잘 이해됩니다. 자세한 정보는 [예시: 풀, 브랜치 및 'CLOSED' 이벤트가 있는 트리거](#)을 참조하세요.

- 풀 요청에 대한 새 개정(시각적 편집기) 또는 REVISION(YAML 편집기)

워크플로 실행은 풀 요청에 대한 개정이 생성될 때 시작됩니다. 풀 요청이 생성되면 첫 번째 개정이 생성됩니다. 그런 다음 풀 요청에 지정된 소스 브랜치에 새 커밋을 푸시할 때마다 새 개정이 생성됩니다. 풀 요청 트리거에 REVISION 이벤트를 포함하는 경우 REVISION은 OPEN의 슈퍼 세트이므로 OPEN 이벤트를 생략할 수 있습니다.

동일한 풀 요청 트리거에서 여러 이벤트를 지정할 수 있습니다.

예시는 [예시: 워크플로의 트리거](#) 섹션을 참조하세요.

해당 UI: 시각적 편집기/워크플로 다이어그램/트리거/풀 요청을 위한 이벤트

Branches

(Triggers/Branches)

(선택 사항)

워크플로 실행을 시작할 시기를 알기 위해 트리거가 모니터링하는 소스 리포지토리의 브랜치를 지정합니다. 정규식 패턴을 사용하여 브랜치 이름을 정의할 수 있습니다. 예를 들어 main.*을 사용하여 main으로 시작하는 모든 브랜치를 일치시킵니다.

지정할 브랜치는 트리거 유형에 따라 다릅니다.

- 푸시 트리거의 경우 푸시 대상 브랜치, 즉 대상 브랜치를 지정합니다. 일치하는 브랜치의 파일을 사용하여 일치하는 브랜치당 하나의 워크플로 실행이 시작됩니다.

예시: `main.*, mainline`

- 풀 요청의 경우 푸시할 브랜치, 즉 대상 브랜치를 지정합니다. 워크플로 정의 파일과 소스 브랜치(일치하는 브랜치 아님)의 소스 파일을 사용하여 일치하는 브랜치당 하나의 워크플로 실행이 시작됩니다.

예시: `main.*, mainline, v1\-.*(v1-로 시작하는 브랜치와 일치)`

- 일정 트리거의 경우 예약된 실행에서 사용할 파일이 포함된 브랜치를 지정합니다. 워크플로 정의 파일과 일치하는 브랜치의 소스 파일을 사용하여 일치하는 브랜치당 하나의 워크플로 실행이 시작됩니다.

예시: `main.*, version\-1\.0`

Note

브랜치를 지정하지 않으면 트리거는 소스 리포지토리의 모든 브랜치를 모니터링하고 다음에서 워크플로 정의 파일 및 소스 파일을 사용하여 워크플로 실행을 시작합니다.

- 푸시 대상 브랜치(푸시 트리거용). 자세한 내용은 [예시: 간단한 코드 푸시 트리거](#) 섹션을 참조하세요.
- 풀 원본 브랜치(풀 요청 트리거용). 자세한 내용은 [예시: 간단한 풀 요청 트리거](#) 섹션을 참조하세요.
- 모든 브랜치(일정 트리거용). 소스 리포지토리의 브랜치당 워크플로 실행이 한 번 시작됩니다. 자세한 내용은 [예시: 간단한 일정 트리거](#) 섹션을 참조하세요.

브랜치 및 트리거에 대한 자세한 내용은 [트리거 및 브랜치 사용 지침](#) 섹션을 참조하세요.

더 많은 예시는 [예시: 워크플로의 트리거](#)를 참조합니다.

해당 UI: 시각 편집기/워크플로 다이어그램/트리거/브랜치

FilesChanged

(Triggers/FilesChanged)

(Type 트리거가 PUSH 또는 PULLREQUEST로 설정된 경우 선택 사항. Type 트리거가 SCHEDULE로 설정된 경우에는 지원되지 않음)

워크플로 실행을 시작해야 하는 시기를 알 수 있도록 트리거가 모니터링하는 소스 리포지토리의 파일 또는 폴더를 지정합니다. 정규식을 사용하여 파일 이름 또는 경로와 일치시킬 수 있습니다.

예시는 [예시: 워크플로의 트리거](#) 섹션을 참조하세요.

해당 UI: 시각적 편집기/워크플로 다이어그램/트리거/파일 변경됨

Expression

(Triggers/Expression)

(Type 트리거가 SCHEDULE로 설정된 경우 필수)

예약된 워크플로 실행 시기를 설명하는 cron 표현식을 지정합니다.

CodeCatalyst의 Cron 표현식은 다음과 같은 6개 필드 구문을 사용합니다. 여기서 각 필드는 공백으로 구분됩니다.

minutes hours days-of-month month days-of-week year

cron 표현식의 예

분	시간	일	월	요일	연도	의미
0	0	?	*	월-금	*	매주 월요일부터 금요일까지 자정(UTC +0)에 워크플로를 실행합니다.
0	2	*	*	?	*	매일 오전 2시(UTC+0)에 워크플로를 실행합니다.

분	시간	일	월	요일	연도	의미
15	22	*	*	?	*	매일 오후 10:15(UTC +0)에 워크플로를 실행합니다.
0/30	22-2	?	*	토-일	*	시작일 오후 10시부터 다음 날 오전 2시 (UTC+0) 사이에 토요일부터 일요일까지 30분마다 워크플로를 실행합니다.
45	13	L	*	?	2023-2027	2023년부터 2027년까지 매월 마지막 날 오후 1시 45분(UTC +0)에 워크플로를 실행합니다.

CodeCatalyst에서 cron 표현식을 지정할 때는 다음 지침을 따라야 합니다.

- SCHEDULE 트리거당 단일 cron 표현식을 지정합니다.
- YAML 편집기에서 cron 표현식을 큰따옴표(")로 묶습니다.
- 시간은 협정 세계시(UTC)로 지정합니다. 다른 시간대는 지원되지 않습니다.

- 실행 간격은 최소 30분으로 구성합니다. 더 빠른 주기는 지원되지 않습니다.
- *days-of-month* 또는 *days-of-week* 필드를 지정하되 둘 다 지정하지는 않습니다. 필드 중 하나에 값 또는 별표(*)를 지정하는 경우 다른 필드에는 물음표(?)를 사용해야 합니다. 별표는 '모두'를 의미하고 물음표는 '모두'를 의미합니다.

cron 표현식의 더 많은 예시와 ?, *, L과 같은 와일드카드에 대한 정보는 Amazon EventBridge 사용 설명서의 [cron 표현식 참조](#)를 참조하세요. EventBridge 및 CodeCatalyst의 Cron 표현식은 정확히 동일한 방식으로 작동합니다.

일정 트리거의 예시는 [예시: 워크플로의 트리거](#) 섹션을 참조하세요.

해당 UI: 시각적 편집기/워크플로 다이어그램/트리거/일정

작업

이 워크플로에 대한 하나 이상의 작업 시퀀스입니다. CodeCatalyst는 다양한 유형의 기능을 제공하는 빌드 및 테스트 작업과 같은 여러 작업 유형을 지원합니다. 각 작업 유형에는 다음이 포함됩니다.

- 작업의 고유 하드 코딩 ID를 나타내는 Identifier 속성입니다. 예를 들어 aws/build@v1은 빌드 작업을 식별합니다.
- 작업과 관련된 속성이 포함된 Configuration 섹션입니다.

각 작업 유형에 대한 자세한 내용은 [작업 유형](#) 항목을 참조하세요. [작업 유형](#) 주제에는 각 작업에 대한 설명서 링크가 있습니다.

다음은 워크플로 정의 파일의 작업 및 작업 그룹에 대한 YAML 참조입니다.

```
Name: MyWorkflow
SchemaVersion: 1.0
...
Actions:
  action-or-gate-name:
    Identifier: identifier
    Configuration:
      ...
  #Action groups
  action-group-name:
    Actions:
      ...
```

action-or-gate-name

(Actions/*action-or-gate-name*)

(필수)

*action-name*을 작업을 수행할 이름으로 바꿉니다. 작업 이름은 워크플로 내에서 고유해야 하며 영숫자, 하이픈 및 밑줄만 포함해야 합니다. 구문 규칙에 대한 자세한 내용은 [YAML 구문 지침](#) 항목을 참조하세요.

제한을 포함한 작업의 이름 지정 방법에 대한 자세한 내용은 [action-or-gate-name](#) 섹션을 참조하세요.

해당 UI: 시각적 편집기/*action-name*/구성 탭/작업 이름 또는 작업 표시 이름

action-group-name

(Actions/*action-group-name*)

(선택 사항)

작업 그룹에는 하나 이상의 작업이 포함됩니다. 작업을 작업 그룹으로 그룹화하면 워크플로를 체계적으로 관리할 수 있고, 서로 다른 그룹 간의 종속성을 구성할 수도 있습니다.

*action-group-name*을 작업 그룹에 부여하려는 이름으로 바꿉니다. 작업 그룹 이름은 워크플로 내에서 고유해야 하며 영숫자, 하이픈 및 밑줄만 포함해야 합니다. 구문 규칙에 대한 자세한 내용은 [YAML 구문 지침](#) 항목을 참조하세요.

작업 그룹에 대한 자세한 내용은 [작업을 작업 그룹으로 그룹화](#) 섹션을 참조하세요.

해당 UI: 없음

CodeCatalyst의 문제 처리 추적 및 구성

CodeCatalyst에서 특성, 버그 및 프로젝트와 관련된 기타 작업을 모니터링할 수 있습니다. 각 작업은 문제라는 고유한 레코드에 보관되어 있습니다. 작업 체크리스트를 추가하여 문제를 더 작은 목표로 나눌 수 있습니다. 각 문제에는 검색, 그룹화 및 필터링할 수 있는 설명, 담당자, 상태 및 기타 속성이 있을 수 있습니다. 기본 보기를 사용하여 문제를 보거나 사용자 지정 필터링, 정렬 또는 그룹화로 고유한 보기를 생성할 수 있습니다. 문제와 관련된 개념에 대한 자세한 내용은 [문제 개념](#) 섹션을 참조하세요. 첫 번째 문제를 생성하는 방법을 알아보려면 [CodeCatalyst에서 문제 생성](#) 섹션을 참조하세요.

다음은 문제를 사용하는 팀이 사용할 수 있는 워크플로 중 하나입니다.

Jorge Souza는 프로젝트에서 작업하는 개발자입니다. 그와 동료 프로젝트 멤버인 Li Juan, Mateo Jackson, Wang Xiulan은 협업을 통해 어떤 작업을 수행해야 하는지 결정합니다. 매일 그와 동료 개발자는 Wang Xiulan이 이끄는 업무 조율 회의를 개최합니다. 그들은 보드에 대한 팀 보기 중 하나로 이동하여 보드를 가져옵니다. 사용자와 팀은 보기를 생성하여 필터, 그룹화 및 문제 정렬을 저장하여 지정된 기준을 충족하는 문제를 쉽게 볼 수 있습니다. 이들의 보기에는 각 개발자에 대한 가장 중요한 문제와 문제의 상태를 표시하기 위해 담당자별로 그룹화되고 우선순위별로 정렬된 문제가 포함되어 있습니다. Jorge는 완료해야 할 작업이 할당됨에 따라 각 작업에 대한 문제를 생성하여 작업을 계획합니다. 문제를 생성할 때 Jorge는 적절한 상태, 우선순위 및 작업 예측 노력을 선택할 수 있습니다. 큰 문제의 경우, Jorge는 문제에 작업을 추가하여 작업을 작은 목표로 나눕니다. Jorge는 즉시 시작할 계획이 없으므로 백로그와 같은 초안 상태의 문제를 생성합니다. 초안 상태의 문제는 초안 보기에 나타나며, 여기서 계획하고 우선순위를 정해야 합니다. Jorge는 작업을 시작할 준비가 되면, 상태를 다른 범주(시작되지 않음, 시작됨 또는 완료됨)의 상태로 업데이트하여 해당 문제를 보드로 이동합니다. 각 작업이 진행 중일 때, 팀은 제목, 상태, 담당자, 레이블, 우선순위 및 추정을 기준으로 필터링하여 지정된 파라미터와 일치하는 특정 문제 또는 유사한 문제를 찾을 수 있습니다. 보드를 사용하여 Jorge와 그의 팀은 각 문제에 대해 완료된 작업 수를 확인하고 작업이 완료될 때까지 각 문제를 한 상태에서 다음 상태로 끌어올려 일상적인 진행 상황을 추적할 수 있습니다. 프로젝트가 진행됨에 따라 완료된 문제가 완료된 상태로 누적됩니다. Wang Xiulan은 개발자가 현재 및 향후 작업과 관련된 문제에 집중할 수 있도록 빠른 보관 버튼을 사용하여 아카이브하여 보기에서 제거하기로 결정합니다.

작업을 계획할 때 프로젝트에서 작업하는 개발자는 정렬 기준 및 그룹화 기준을 선택하여 백로그에서 보드로 이동하려는 문제를 찾습니다. 우선순위가 가장 높은 고객 요청에 따라 보드에 문제를 추가하도록 선택할 수도 있으므로 고객 요청 레이블을 기준으로 보드를 그룹화하고 우선순위별로 정렬합니다. 또한 추정치를 기준으로 정렬하여 달성할 수 있는 작업량을 확보하고 있는지 확인할 수 있습니다. 프로젝트 관리자인 Saanvi Sarkar는 백로그를 정기적으로 검토하고 정리하여 우선순위가 프로젝트의 성공에 대한 각 문제의 중요성을 정확하게 반영하도록 합니다.

주제

- [문제 개념](#)
- [문제 관련 작업 추적](#)
- [백로그, 레이블 및 보드로 작업 구성](#)
- [CodeCatalyst 문제에 대한 할당량](#)

문제 개념

문제를 생성하는 것은 프로젝트 내에서 수행되는 작업을 추적하는 빠르고 효율적인 방법입니다. 문제를 사용하여 일일 업무 조율 회의에서 작업에 대해 논의하고 작업의 우선 순위를 지정하는 등 다양한 작업을 수행할 수 있습니다.

이 페이지에는 CodeCatalyst에서 문제를 효과적으로 사용하는 데 도움이 되는 개념 목록이 포함되어 있습니다.

활성 문제

활성 문제는 초안 상태가 아니거나 보관되지 않은 문제입니다. 즉, 활성 문제는 상태가 시작되지 않음, 시작됨 및 완료됨과 같은 상태 범주에 속하는 문제입니다. 상태 및 상태 범주에 대한 자세한 내용은 [상태 및 상태 범주](#) 섹션을 참조하세요.

기본 활성 문제 보기에서 프로젝트의 모든 활성 문제를 볼 수 있습니다.

보관된 문제

보관된 문제는 더 이상 프로젝트와 관련이 없는 문제입니다. 예를 들어, 문제가 완료되어 더 이상 완료 열에서 볼 필요가 없거나 오류로 생성된 경우 [문제를 보관할](#) 수 있습니다. 필요한 경우 보관된 문제를 보관 해제할 수 있습니다.

담당자

담당자는 문제가 할당된 사람입니다. 검색 시 목록에 검색한 사람이 표시되지 않으면, 그 사람은 프로젝트에 추가되지 않은 것입니다. 담당자를 추가하려면 [프로젝트에 사용자 초대](#) 섹션을 참조하세요. 문제에 대해 여러 담당자를 활성화하려면 [다수의 담당자 활성화 또는 비활성화](#) 섹션을 참조하세요. 여러 담당자가 할당된 문제는 서로 다른 색상의 아바타와 함께 보드에 표시되며, 각 아바타는 담당자 중 하나를 나타냅니다.

사용자 지정 필드

사용자 지정 필드를 사용하면 프로젝트 내에서 문제를 추적하고 유지하기 위한 필요에 따라 문제의 다양한 속성을 사용자 지정할 수 있습니다. 예를 들어 로드맵 작성을 위한 필드, 특정 기한 또는 요청자 필드를 추가할 수 있습니다.

추정

애자일 개발에서는 추정을 스토리 포인트라고 합니다. 문제의 모호성과 복잡성 외에도 문제에 대한 추정치를 사용하여 필요한 작업량을 나타낼 수 있습니다. 더 위험도가 높거나 어렵거나 잘 알 수 없는 문제에는 더 높은 추정치를 사용하는 것이 좋습니다.

추정 유형 및 구성 방법에 대한 자세한 내용은 [문제에 대한 노력 추정 구성](#) 섹션을 참조하세요.

문제

문제는 프로젝트와 관련된 작업을 추적하는 레코드입니다. 기능, 작업, 버그 또는 프로젝트와 관련된 기타 작업에 대한 문제를 생성할 수 있습니다. 애자일 개발을 사용하는 경우 문제가 에픽 또는 사용자 스토리를 설명할 수도 있습니다.

Label

레이블은 문제를 그룹화, 정렬 및 필터링하는 데 사용됩니다. 새 레이블 이름을 입력하거나 채워진 목록에서 레이블 중 하나를 선택할 수 있습니다. 이 목록은 프로젝트에서 최근에 사용된 레이블로 구성됩니다. 문제에는 레이블이 여러 개 있을 수 있으며, 레이블을 문제에서 제거할 수 있습니다. 레이블을 사용자 지정하려면 [레이블로 작업 범주화](#) 섹션을 참조하세요.

우선순위

우선 순위는 문제의 중요도를 나타냅니다. 낮음, 중간, 높음, 우선순위 없음의 네 가지 옵션이 있습니다.

상태 및 상태 범주

상태는 문제의 현재 상태이며 시작부터 완료까지 수명 주기 동안 문제의 진행 상황을 빠르게 확인하는 데 사용됩니다. 모든 문제에는 상태가 있어야 하며 각 상태는 상태 범주에 속합니다. 상태 범주는 상태를 구성하고 기본 문제 보기를 채우는 데 사용됩니다.

CodeCatalyst에는 5개의 기본 상태와 4개의 상태 범주가 있습니다. 기본 상태 이외에 다른 상태를 생성할 수는 있지만 기본 이외의 다른 상태 범주는 생성할 수 없습니다. 다음 목록에는 괄호 안에 기본 상태 및 상태 범주가 포함되어 있습니다. 백로그(초안), 할 일(시작되지 않음), 진행 중(시작됨), 검토 중(시작됨) 및 완료(완료됨).

상태에 대한 작업의 자세한 내용은 [사용자 지정 상태로 작업 추적](#) 섹션을 참조하세요.

작업

문제에 작업을 추가하여 해당 문제의 작업을 더 세분화하고 구성할 수 있습니다. 생성 시 문제에 작업을 추가하거나 기존 문제에 작업을 추가할 수 있습니다. 문제를 볼 때 작업을 재정렬, 제거 또는 완료됨으로 표시할 수 있습니다.

보기

CodeCatalyst 프로젝트의 문제는 보기에 표시됩니다. 보기는 목록 형식의 문제를 표시하는 그리드 보기 또는 문제 상태별로 구성된 열의 타일로 문제를 표시하는 보드 보기일 수 있습니다. 네 가지 기본 보기가 있으며 사용자 [지정 그룹화, 필터링 및 정렬을 사용하여 자체 보기를 생성할 수 있습니다](#). 다음 목록에는 네 가지 기본 보기에 대한 세부 정보가 포함되어 있습니다.

- 초안 보기는 현재 작업 중이 아닌 문제를 보여주는 그리드 보기입니다. 초안 상태 범주의 상태로 생성된 모든 문제가 이 보기에 표시됩니다. 팀은 이 보기를 사용하여 아직 정의 중이거나 할당 및 작업을 기다리고 있는 문제를 확인할 수 있습니다.
- 활성 문제 보기는 현재 작업 중인 모든 문제에 대한 보드 보기입니다. 시작되지 않음, 시작됨 또는 완료됨 상태 범주의 상태 문제가 이 보기에 표시됩니다.
- 모든 문제 보기는 프로젝트의 모든 문제, 초안 및 활성 문제를 모두 보여주는 그리드 보기입니다.
- 보관된 보기에는 보관된 모든 문제가 표시됩니다.

문제 관련 작업 추적

문제를 사용하여 프로젝트 작업을 계획하고 추적할 수 있습니다. 각 문제는 고유한 레코드에 보관된 작업의 일부입니다. 문제를 작업으로 세분화하여 해당 문제의 작업을 추가로 구성하고 추적할 수 있습니다. 또한 문제 간에 링크를 생성하여 관련 작업을 추적하고, 레이블을 추가하여 작업의 구성 및 분류, 문제 그룹화, 작업 우선순위 할당, 작업의 차단 여부 표시를 할 수 있습니다.

문제 또는 일련의 문제를 해결할 준비가 되면, 작업을 추정하고, 사용자에게 할당하고, 다른 사람이 작업 및 진행 상황을 이해하는 데 도움을 주기 위해 설명을 추가할 수 있습니다. 문제를 내보내 다른 형식으로 포함된 정보를 가져올 수도 있습니다.

CodeCatalyst에서 문제 생성

개발 팀은 작업을 추적하고 관리하는 데 도움이 되는 문제를 생성합니다. 필요에 따라 프로젝트 내에서 문제를 생성할 수 있습니다. 예를 들어 코드를 통해 변수 업데이트를 추적하는 문제를 생성할 수 있

습니다. 프로젝트의 다른 사용자에게 문제를 할당하고, 레이블을 사용하여 작업 추적에 도움을 받는 등 다양한 작업을 수행할 수 있습니다.

다음 지침에 따라 CodeCatalyst에서 문제를 생성합니다.

문제를 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 문제를 생성하려는 프로젝트로 이동합니다.
3. 프로젝트 홈 페이지에서 문제 생성을 선택합니다. 또는 탐색 창에서 문제를 선택합니다.
4. 문제 생성을 선택합니다.

 Note

그리드 보기를 사용할 때 문제를 인라인으로 추가할 수도 있습니다.

5. 문제의 제목을 입력합니다.
6. (선택 사항) 설명을 입력합니다. 마크다운을 사용하여 서식 지정을 추가할 수 있습니다.
7. (선택 사항) 문제에 대한 상태, 우선 순위 및 추정을 선택합니다.

 Note

프로젝트의 추정 설정이 추정 숨기기로 설정된 경우 추정 필드는 없습니다.

8. (선택 사항) 문제에 작업을 추가합니다. 작업을 사용하여 문제의 작업을 더 작은 목표로 세분화할 수 있습니다. 작업을 추가하려면 + 작업을 추가를 선택합니다. 그런 다음 텍스트 필드에 작업 이름을 입력하고 Enter 키를 누릅니다. 작업을 추가한 후 확인란을 선택하여 완료로 표시하거나, 확인란의 왼쪽에서 작업을 선택하고 드래그하여, 순서를 변경할 수 있습니다.
9. (선택 사항) 기존 레이블을 추가하거나 새 레이블을 생성하고 + 레이블 추가를 선택하여 추가합니다.
 - a. 기존 레이블을 추가하려면 목록에서 레이블을 선택합니다. 필드에 검색어를 입력하여 프로젝트에 해당 용어가 포함된 모든 레이블을 검색할 수 있습니다.
 - b. 새 레이블을 생성하고 추가하려면 검색 필드에 생성하려는 레이블의 이름을 입력하고 Enter 키를 누릅니다.
10. (선택 사항) + 담당자 추가를 선택하여 담당자를 추가합니다. + 나를 추가를 선택하여 자신을 담당자로 빠르게 추가할 수 있습니다.

Tip

Amazon Q가 문제를 해결하도록 Amazon Q에 문제를 할당할 수 있습니다. 자세한 내용은 [자습서: CodeCatalyst 생성형 AI 기능을 사용하여 개발 작업 속도 향상](#) 단원을 참조하십시오. 이 기능은 미국 서부(오리건) 리전에서만 사용할 수 있습니다.

이 기능을 사용하려면 스페이스에 생성형 AI 기능을 활성화해야 합니다. 자세한 내용은 [생성형 AI 기능 관리](#)를 참조하세요.

11. (선택 사항) 기존 사용자 지정 필드를 추가하거나 새 사용자 지정 필드를 생성합니다. 문제에는 여러 사용자 지정 필드가 있을 수 있습니다.
 - a. 기존 사용자 지정 필드를 추가하려면 목록에서 사용자 지정 필드를 선택합니다. 필드에 검색어를 입력하여 프로젝트에 해당 용어가 포함된 모든 사용자 지정 필드를 검색할 수 있습니다.
 - b. 새 사용자 지정 필드를 생성하고 추가하려면, 검색 필드에 생성하려는 사용자 지정 필드의 이름을 입력하고 Enter 키를 누릅니다. 그런 다음 생성하려는 사용자 지정 필드 유형을 선택하고 값을 설정합니다.
12. 문제 생성을 선택합니다. 오른쪽 하단 모서리에 알림이 나타납니다. 문제가 성공적으로 생성된 경우 문제가 성공적으로 생성되었다는 확인 메시지가 나타납니다. 문제가 성공적으로 생성되지 않은 경우 실패 이유와 함께 오류 메시지가 나타납니다. 이런 경우 재시도를 선택하여 문제를 편집하고 다시 생성하거나 취소를 선택하여 문제를 폐기할 수 있습니다. 두 옵션 모두 알림을 무시합니다.

Note

풀 요청을 생성할 때는 풀 요청을 문제에 연결할 수 없습니다. 그러나 문제를 생성한 후에는 풀 요청에 연결하는 링크를 추가하기 위해 문제를 [편집](#)할 수 있습니다.

Amazon Q에 할당된 문제를 생성하고 작업할 때의 모범 사례

문제를 생성할 때 일부 문제는 오랜 시간 그대로 방치되기도 합니다. 이에 대한 원인은 복잡하며 많은 변수가 있을 수 있습니다. 누가 작업해야 하는지 명확하지 않기 때문에 발생하는 것일 수 있습니다. 또는 이 문제로 인해 코드 베이스의 특정 부분에 대한 조사 또는 전문 지식이 필요한 상황에서 작업에 가장 적합한 후보가 다른 문제로 바뀔 수 있습니다. 주로 다른 긴급한 작업을 먼저 진행해야 하는 경우가 많습니다. 이러한 원인 중 일부 또는 전체로 인해 작업이 진행되지 않은 문제가 생길 수 있습니다. CodeCatalyst에는 제목과 설명을 기반으로 문제를 분석할 수 있는 Amazon Q라는 생성형 AI 어시스턴트와의 통합 기능이 포함됩니다. Amazon Q에 문제를 할당하면 평가할 초안 솔루션을 생성하

려고 시도합니다. 이렇게 하면 사용자와 팀이 주의가 필요한 문제에 집중하고 작업을 최적화하는 데 도움이 되며, Amazon Q는 즉시 해결할 리소스가 없는 문제에 대한 솔루션을 제공합니다.

Note

Note

Amazon Bedrock:implements 자동 침해 탐지로 구동됩니다. AWS <https://docs.aws.amazon.com//bedrock/latest/userguide/abuse-detection.html> 소프트웨어 개발용 Amazon Q Developer Agent의 설명 쓰기, 콘텐츠 요약 생성, 권장 작업, Amazon Q를 사용하여 프로젝트에 기능을 생성하거나 추가, Amazon Q에 문제 할당 기능은 Amazon Bedrock에 구축되므로, 사용자는 Amazon Bedrock에서 구현된 제어 기능을 최대한 활용하여 안전, 보안 및 인공 지능(AI)을 책임 있게 사용할 수 있습니다.

Amazon Q는 간단하고 단순한 문제를 다룰 때 최상의 성능을 발휘합니다. 최상의 결과를 얻으려면 일반적인 언어를 사용하여 수행하려는 작업을 명확하게 설명하세요. 다음은 Amazon Q가 작업할 수 있도록 최적화된 문제를 생성하는 데 도움이 되는 몇 가지 모범 사례입니다.

Important

생성형 AI 기능은 미국 서부(오리건) 리전에서만 사용할 수 있습니다.

- 간단해야 합니다. Amazon Q는 문제의 제목 및 설명에서 설명할 수 있는 간단한 코드 변경 및 수정 사항에 사용하는 것이 가장 좋습니다. 문제를 할당할 때 모호한 제목이나 지나치게 유동적이거나 모순되는 설명을 쓰지 마세요.
- 구체적이어야 합니다. 문제를 해결하는 데 필요한 정확한 변경 사항에 대해 더 많은 정보를 제공할수록 Amazon Q가 문제를 해결하는 솔루션을 생성할 가능성이 높아집니다. 가능한 경우 변경하려는 API 이름, 업데이트하려는 방법, 변경해야 하는 테스트, 생각나는 기타 세부 정보와 같은 특정 세부 정보를 포함합니다.
- Amazon Q에 할당하기 전에 문제의 제목과 설명에 모든 세부 정보가 포함되어 있는지 확인합니다. Amazon Q에 문제를 할당한 후에는 문제의 제목이나 설명을 변경할 수 없으므로, Amazon Q에 할당하기 전에 문제에 필요한 모든 정보가 있는지 확인하세요.
- 단일 소스 리포지토리에서 코드 변경이 필요한 문제만 할당합니다. Amazon Q는 CodeCatalyst의 단일 소스 리포지토리에 있는 코드에서만 작업할 수 있습니다. 연결된 리포지토리는 지원되지 않습니다.

다. Amazon Q에 문제를 할당하기 전에 단일 소스 리포지토리에서만 변경해야 하는 문제인지 확인하세요.

- 각 단계를 승인하려면 Amazon Q에서 제안한 기본값을 사용합니다. 기본적으로 Amazon Q는 수행하는 각 단계에 대해 승인을 요구합니다. 이렇게 하면 문제에 대한 설명뿐만 아니라 Amazon Q가 생성한 풀 요청에 대한 설명에서도 Amazon Q와 상호 작용할 수 있습니다. 이렇게 하면 Amazon Q를 통해 보다 대화형 경험을 제공하여, Amazon Q가 접근 방식을 조정하고 문제를 해결하기 위해 생성하는 코드를 구체화하도록 할 수 있습니다.

Note

Amazon Q는 문제 또는 풀 요청의 개별 의견에는 응답하지 않지만 접근 방식을 재고하거나 수정을 생성하라는 메시지가 표시되면 이를 검토합니다.

- Amazon Q에서 제안한 접근 방식을 항상 주의 깊게 검토합니다. 한 번 접근 방식을 승인하면 Amazon Q는 해당 접근 방식을 기반으로 코드를 생성하는 작업을 시작합니다. Amazon Q에 계속 진행하라고 알리기 전에 접근 방식이 올바르고 예상하는 모든 세부 정보가 포함되어 있는지 확인하세요.
- 검토하기 전에 배포할 수 있는 기존 워크플로가 없는 경우에만 Amazon Q가 워크플로에서 작업하도록 허용해야 합니다. 프로젝트에 풀 요청 이벤트에 대한 실행을 시작하도록 구성된 워크플로가 있을 수 있습니다. 그렇다면 Amazon Q가 생성했으며 워크플로 YAML 생성 또는 업데이트를 포함하고 있는 풀 요청은 해당 풀 요청에 포함된 워크플로의 실행을 시작할 수 있습니다. 프로젝트에서 생성한 풀 요청을 검토하고 승인하기 전에 이러한 워크플로를 자동으로 실행하는 워크플로가 없다는 사실이 확인되지 않았다면 Amazon Q가 워크플로 파일에서 작업하도록 허용하지 않는 것이 좋습니다.

자세한 내용은 [자습서: CodeCatalyst 생성형 AI 기능을 사용하여 개발 작업 속도 향상](#) 섹션 및 [생성형 AI 기능 관리](#)를 참조하세요.

문제 추정

애자일 개발에서는 추정을 스토리 포인트라고 합니다. 문제의 모호성과 복잡성 외에도 문제에 대한 추정치를 사용하여 필요한 작업량을 나타낼 수 있습니다. 더 위험도가 높거나 어렵거나 잘 알 수 없는 문제에는 더 높은 추정치를 사용하는 것이 좋습니다.

문제 추정을 시작하려면 먼저 프로젝트에 사용할 추정 유형을 선택해야 합니다. 기본의 다음 두 가지 유형 중에서 선택할 수 있습니다. 티셔츠 크기 조정 또는 Fibonacci 시퀀스를 효과적으로 사용하려면

팀이 각 크기가 나타내는 것을 서로 맞춰야 합니다. 각 추정치가 나에게 무엇을 나타내는지 함께 결정한 다음 각 문제에 해당 추정치를 적용하기 시작합니다. 정기적인 검토를 하는 것이 좋습니다.

CodeCatalyst의 문제에 대한 노력 추정의 설정을 구성하려면 다음 단계를 따르세요.

문제에 대한 노력 추정을 구성하려면

1. 탐색 창에서 문제를 선택합니다.
2. 활성 문제를 선택하여 문제 보기 전환기 드롭다운 메뉴를 열고 설정을 선택합니다.
3. 기본 설정 섹션의 추정에서 추정 값이 표시되는 방법을 선택합니다. 사용 가능한 추정 유형은 티셔츠 크기 조정, Fibonacci 시퀀싱 또는 추정 숨기기입니다. 프로젝트의 추정 설정이 추정 숨기기로 설정된 경우 프로젝트의 문제에 추정 필드는 없습니다.

추정 유형이 업데이트되면, 데이터가 손실되지 않고 모든 문제의 추정 값이 자동으로 변환됩니다. 변환 매핑은 다음 표에 나와 있습니다.

티셔츠 사이즈	Fibonacci 시퀀스
XS	1
XS	2
S	3
M	5
L	8
XL	13

문제에 대한 추정을 추가하거나 변경하려면 [문제를 편집할 수 있습니다](#).

CodeCatalyst의 문제 편집 및 공동 작업

목차

- [문제 편집](#)
- [첨부 파일 작업](#)
 - [첨부 파일 보기 및 관리](#)

- [문제에 대한 작업 관리](#)
- [문제를 다른 문제에 연결](#)
- [문제를 차단됨 또는 차단 해제됨으로 표시](#)
- [설명의 추가, 편집 또는 삭제](#)
 - [설명에서 언급 기능 사용](#)

문제 편집

다음 단계에 따라 문제의 제목, 설명, 상태, 담당자, 우선순위, 추정 또는 레이블을 편집합니다.

문제를 편집하려면

1. 편집하려는 문제를 선택하여 문제의 세부 정보를 봅니다. 문제를 찾는 방법에 대한 도움말은 [문제 찾기 및 보기](#) 섹션을 참조하세요.
2. 그래프의 제목을 편집하려면, 해당 제목을 선택하고 새로운 제목을 입력한 다음, Enter 키를 누릅니다.
3. 설명을 편집하려면, 설명을 선택하고 새 설명을 입력한 다음 Enter 키를 누릅니다. 마크다운을 사용하여 서식 지정을 추가할 수 있습니다.
4. 작업에서 문제에 대한 작업을 보고 관리할 수 있습니다. 작업이 없는 경우, Amazon Q가 문제를 분석하고 문제에서 업무를 사용자에게 할당할 수 있는 별도의 항목으로 나눌 수 있는 작업을 추천하도록 할 수 있습니다. 자세한 내용은 [문제에 대한 작업 관리](#) 단원을 참조하십시오.
5. 상태, 추정 또는 우선순위를 편집하려면 해당 드롭다운 메뉴에서 옵션을 선택합니다.
6. 레이블에서 기존 레이블을 추가하거나, 새 레이블을 생성하거나, 레이블을 제거할 수 있습니다.
 - a. 기존 레이블을 추가하려면 + 레이블 추가를 선택하고 목록에서 레이블을 선택합니다. 필드에 검색어를 입력하여 프로젝트에 해당 용어가 포함된 모든 레이블을 검색할 수 있습니다.
 - b. 새 레이블을 생성하고 추가하려면, + 레이블 추가를 선택하고 검색 필드에 생성하려는 레이블의 이름을 입력한 다음 Enter 키를 누릅니다.
 - c. 레이블을 제거하려면, 제거하려는 레이블 옆에 있는 X 아이콘을 선택합니다. 모든 문제에서 레이블을 제거하면, 레이블이 문제 설정의 레이블 섹션에서 미사용 레이블 섹션에 표시됩니다. 미사용 레이블은 필터를 사용하거나 문제에 레이블을 추가할 때 레이블 목록 마지막에 나타납니다. 문제 설정에서 모든 레이블(사용 및 미사용) 및 해당 레이블이 있는 문제에 대한 개요를 확인할 수 있습니다.
7. 문제를 할당하려면, 담당자 섹션에서 + 담당자 추가를 선택한 다음 목록에서 담당자를 검색하고 선택합니다. + 나를 추가를 선택하여 자신을 담당자로 빠르게 추가할 수 있습니다.

8. 첨부 파일에서 첨부 파일을 추가, 다운로드 또는 제거할 수 있습니다. 자세한 내용은 [첨부 파일 작업](#) 단원을 참조하십시오.
9. 풀 요청을 연결하려면, 풀 요청 연결을 선택한 다음, 목록에서 풀 요청을 선택하거나 풀 요청의 URL 또는 ID를 입력합니다. 풀 요청을 연결 해제하려면, 연결 해제 아이콘을 선택합니다.

 Tip

문제에 풀 요청에 대한 링크를 추가한 후, 연결된 풀 요청 목록에서 해당 ID를 선택하여 해당 풀 요청을 빠르게 탐색할 수 있습니다. 풀 요청의 URL을 사용하여 문제 보드와 다른 프로젝트에 있는 풀 요청을 연결할 수 있지만, 해당 프로젝트의 멤버인 사용자만 해당 풀 요청을 보거나 탐색할 수 있습니다.

10. (선택 사항) 기존 사용자 지정 필드를 추가 및 설정하거나, 새 사용자 지정 필드를 생성하거나, 사용자 지정 필드를 제거합니다. 문제에는 여러 사용자 지정 필드가 있을 수 있습니다.
 - a. 기존 사용자 지정 필드를 추가하려면 목록에서 사용자 지정 필드를 선택합니다. 필드에 검색어를 입력하여 프로젝트에 해당 용어가 포함된 모든 사용자 지정 필드를 검색할 수 있습니다.
 - b. 새 사용자 지정 필드를 생성하고 추가하려면, 검색 필드에 생성하려는 사용자 지정 필드의 이름을 입력하고 Enter 키를 누릅니다. 그런 다음 생성하려는 사용자 지정 필드 유형을 선택하고 값을 설정합니다.
 - c. 사용자 지정 필드를 제거하려면, 제거하려는 사용자 지정 필드 옆에 있는 X 아이콘을 선택합니다. 모든 문제에서 사용자 지정 필드를 제거하면, 사용자 지정 필드가 삭제되고 필터링 시 더 이상 표시되지 않습니다.

첨부 파일 작업

CodeCatalyst에서는 문제에 첨부 파일을 추가하여 관련 파일에 쉽게 액세스할 수 있습니다. 다음 절차에 따라 문제의 첨부 파일을 관리할 수 있습니다.

문제에 추가한 첨부 파일의 크기는 스페이스의 스토리지 할당량에 포함됩니다. 프로젝트의 첨부 파일을 보고 관리하는 방법에 대한 자세한 내용은 [첨부 파일 보기 및 관리](#) 섹션을 참조하세요.

 Important

문제의 첨부 파일은 Amazon CodeCatalyst에서 스캔하거나 분석하지 않습니다. 모든 사용자는 악성 코드 또는 콘텐츠가 포함된 문제에 첨부 파일을 추가했을 수 있습니다. 사용자가 첨부 파

일을 관리하고 악성 코드, 콘텐츠 또는 바이러스로부터 보호하는 모범 사례를 알고 있는지 확인합니다.

첨부 파일을 추가, 다운로드 또는 제거하려면

1. 첨부 파일을 관리할 문제를 선택합니다. 문제를 찾는 방법에 대한 도움말은 [문제 찾기 및 보기](#) 섹션을 참조하세요.
2. 첨부 파일을 추가하려면 파일 업로드를 선택합니다. 운영 체제의 파일 탐색기에서 파일로 이동하여 선택합니다. 열기를 선택하여 첨부 파일로 추가합니다. 최대 첨부 파일 크기와 같은 할당량 정보는 [CodeCatalyst 문제에 대한 할당량](#) 섹션을 참조하세요.

첨부 파일 이름 및 콘텐츠 유형에 대한 다음 제한 사항에 유의하세요.

- 파일 이름에는 다음 문자가 허용되지 않습니다.
 - 제어 문자: 0x00-0x1f 및 0x80-0x9f
 - 예약 문자: /, ?, <, >, \, :, *, | 및 "
 - Unix 예약 파일 이름: . 및 ..
 - 생략 부호 및 공백
 - Windows 예약 파일 이름: CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9
- 첨부 파일의 콘텐츠 유형은 다음 미디어 유형 패턴을 준수해야 합니다.

```
media-type = type "/" [tree "."] subtype ["+" suffix]* [";" parameter];
```

예시: text/html; charset=UTF-8.

3. 첨부 파일을 다운로드하려면 다운로드하려는 첨부 파일 옆의 줄임표 메뉴를 선택하고 다운로드를 선택합니다.
4. 첨부 파일의 URL을 복사하려면, URL을 복사할 첨부 파일 옆의 줄임표 메뉴를 선택하고 URL 복사를 선택합니다.
5. 첨부 파일을 제거하려면, 제거하려는 첨부 파일 옆의 줄임표 메뉴를 선택하고 삭제를 선택합니다.

첨부 파일 보기 및 관리

문제 설정에서 프로젝트의 문제에 추가된 모든 첨부 파일이 담긴 테이블을 볼 수 있습니다. 이 테이블에는 콘텐츠 유형, 첨부된 시간, 첨부된 문제 및 문제 상태, 파일 크기와 같은 정보를 포함하여 각 첨부 파일의 세부 정보가 포함되어 있습니다.

이 테이블을 사용하면 완료되거나 보관된 문제의 큰 첨부 파일을 쉽게 식별하여 스페이스의 스토리지를 확보할 수 있습니다.

Important

문제의 첨부 파일은 Amazon CodeCatalyst에서 스캔하거나 분석하지 않습니다. 모든 사용자는 악성 코드 또는 콘텐츠가 포함된 문제에 첨부 파일을 추가했을 수 있습니다. 사용자가 첨부 파일을 관리하고 악성 코드, 콘텐츠 또는 바이러스로부터 보호하는 모범 사례를 알고 있는지 확인합니다.

프로젝트의 모든 문제 첨부 파일을 보고 관리하려면

1. 탐색 창에서 문제를 선택합니다.
2. 줄임표 아이콘을 선택하고 설정을 선택합니다.
3. 첨부 파일 탭을 선택합니다.

문제에 대한 작업 관리

문제에 작업을 추가하여 해당 문제의 작업을 더 세분화, 구성 및 추적할 수 있습니다. 작업을 직접 생성하거나 Amazon Q를 사용하여 문제 분석 및 복잡성에 따라 작업을 추천할 수 있습니다.

Amazon Q Developer는 AWS 애플리케이션을 이해, 구축, 확장 및 운영하는 데 도움이 되는 생성형 AI 기반 대화형 어시스턴트입니다. 빌드 속도를 높이기 위해 Amazon Q AWS를 지원하는 모델을 고품질 AWS 콘텐츠로 보강하여 더 완전하고 실행 가능하며 참조된 답변을 생성합니다. 자세한 내용은 Amazon Q Developer 사용 설명서의 [Amazon Q Developer란 무엇인가요?](#)를 참조하세요.

문제에 대한 작업 관리하려면

1. 작업을 관리하려는 문제를 선택합니다. 문제를 찾는 방법에 대한 도움말은 [문제 찾기 및 보기](#) 섹션을 참조하세요.
2. 작업에서 문제에 대한 작업을 보고 관리할 수 있습니다.

1. 작업을 추가하려면 텍스트 필드에 작업 이름을 입력하고 Enter 키를 누릅니다.
2. 문제에 대한 작업이 없는 경우 Amazon Q가 문제를 분석하고 문제 제목, 설명, 문제의 복잡성 분석 및 리포지토리 코드를 기반으로 작업을 생성하도록 선택하고 작업 추천을 선택합니다. 문제의 코드가 포함된 소스 리포지토리를 지정해야 합니다. 작업 추천 시작을 선택하여 작업 분석을 시작합니다. 대화 상자가 닫힙니다. 추천이 완료되면, 추천 작업 보기를 선택하여 작업을 검토하고, 작업 생성을 선택하기 전에 목록에 작업을 삭제 또는 추가하거나 추천 작업의 순서를 변경하는 등 필요한 조치를 취합니다.

작업을 생성한 후에는 사용자에게 할당할 수 있으며, 수동으로 생성한 작업과 동일한 방식으로 작업할 수 있습니다.

3. 작업을 완료로 표시하려면 작업의 확인란을 선택합니다.
4. 작업의 세부 정보를 보거나 업데이트하려면 목록에서 선택합니다.
5. 작업을 재정렬하려면, 확인란 왼쪽에서 작업을 선택하여 끌어옵니다.
6. 작업을 제거하려면, 작업의 줄임표 메뉴를 선택하고 제거를 선택합니다.

문제를 다른 문제에 연결

문제가 다른 문제의 작업과 관련된 경우, 연결할 수 있습니다. 이는 작업 항목 간의 종속성을 이해하고 추적하는 데 도움이 되는 빠른 방법입니다.

문제를 연결할 때 선택할 수 있는 네 가지 상태가 있습니다.

- 관련 항목: 연결된 문제와 관련된 작업을 나타내려면 이 상태를 사용합니다.
- 차단 기준: 연결된 문제로 인해 문제가 차단되었음을 나타내려면 이 상태를 사용합니다.
- 차단 대상: 해당 문제가 연결된 문제의 진행을 차단하고 있음을 나타내려면 이 상태를 사용합니다.
- 중복: 해당 문제가 다른 문제에 캡처된 작업을 복제했다는 점을 나타내려면 이 상태를 사용합니다.

링크의 상태는 생성한 후에 변경할 수 있습니다. 링크와 링크 상태는 링크를 생성하는 문제와 연결된 문제 모두에서 연결된 문제에 나타납니다. 링크가 생성된 후에도 링크를 제거할 수 있습니다.

문제를 다른 문제 또는 작업에 연결하려면

1. 다른 문제에 연결하려는 문제를 엽니다. 문제를 찾는 방법에 대한 도움말은 [문제 찾기 및 보기](#) 섹션을 참조하세요.
2. 문제 연결을 선택합니다.

3. 다음으로 표시에서 링크의 상태를 선택합니다.

문제 번호에 문제의 번호를 입력하거나 드롭다운 목록에서 해당 문제를 선택합니다.

4. 다른 링크를 추가하려면 연결된 문제 추가를 선택합니다.

5. 완료되면 업데이트를 선택하여 연결된 관계와 관련된 문제 및 모든 연결된 문제를 업데이트합니다.

문제를 차단됨 또는 차단 해제됨으로 표시

문제를 해결하지 못하도록 막는 요소가 있다면, 차단됨으로 표시할 수 있습니다. 예를 들어, 아직 병합되지 않은 코드 베이스의 다른 부분에 대한 변경에 의존하는 경우 문제가 차단될 수 있습니다.

문제를 차단됨으로 표시하면, CodeCatalyst는 문제에 빨간색 차단됨 레이블을 추가하여, 백로그 또는 아카이브 또는 보드에서 잘 보이도록 합니다.

외부 상황이 해결되면, 문제를 차단 해제할 수 있습니다.

문제를 차단됨으로 표시하려면

1. 차단됨으로 표시할 문제를 엽니다. 문제를 찾는 방법에 대한 도움말은 [문제 찾기 및 보기](#) 섹션을 참조하세요.
2. 작업을 선택한 다음, 차단됨으로 표시를 선택합니다.

문제를 차단 해제하려면

1. 차단 해제하려는 문제를 엽니다. 문제를 찾는 방법에 대한 도움말은 [문제 찾기 및 보기](#) 섹션을 참조하세요.
2. 작업을 선택한 다음, 차단 해제됨으로 표시를 선택합니다.

설명의 추가, 편집 또는 삭제

문제에 대한 설명을 남길 수 있습니다. 설명에서 다른 스페이스 멤버, 스페이스의 다른 프로젝트, 관련 문제 및 코드에 태그를 지정할 수 있습니다.

문제에 설명을 추가하려면

1. 프로젝트로 이동합니다.
2. 탐색 바에서 문제를 선택합니다.

3. 설명을 추가할 문제를 선택합니다. 문제를 찾는 방법에 대한 도움말은 [문제 찾기 및 보기](#) 섹션을 참조하세요.
4. 설명 필드에 설명을 입력합니다. 마크다운을 사용하여 서식 지정을 추가할 수 있습니다.
5. 전송을 선택합니다.

설명을 편집하려면

문제에 대한 설명은 편집할 수 있습니다. 직접 작성한 설명만 편집할 수 있습니다.

1. 프로젝트로 이동합니다.
2. 탐색 바에서 문제를 선택합니다.
3. 설명을 편집할 문제를 선택합니다. 문제를 찾는 방법에 대한 도움말은 [문제 찾기 및 보기](#) 섹션을 참조하세요.
4. 설명을 편집하려면, 편집할 설명을 찾습니다.

Tip

오래된 순 또는 최신 순으로 설명을 정렬할 수 있습니다. 설명은 한 번에 10개씩 로드됩니다.

5. 줄임표 아이콘을 선택한 다음 편집을 선택합니다.
6. 설명을 편집합니다. 마크다운을 사용하여 서식 지정을 추가할 수 있습니다.
7. 저장을 선택합니다. 이제 설명이 업데이트됩니다.

설명을 삭제하려면

문제에 대한 설명은 삭제할 수 있습니다. 직접 작성한 설명만 삭제할 수 있습니다. 설명이 삭제되면, 사용자 이름이 표시되지만 원본 설명 텍스트 대신 이 설명은 삭제되었습니다라는 문구가 나타납니다.

1. 프로젝트로 이동합니다.
2. 탐색 바에서 문제를 선택합니다.
3. 설명을 삭제할 문제를 선택합니다. 문제를 찾는 방법에 대한 도움말은 [문제 찾기 및 보기](#) 섹션을 참조하세요.
4. 줄임표 아이콘을 선택하고 삭제를 선택한 다음 확인을 선택합니다.

설명에서 언급 기능 사용

스페이스 멤버, 스페이스의 다른 프로젝트, 관련 문제 및 코드를 설명에서 언급할 수 있습니다. 이렇게 하면 사용자가 언급한 사용자 또는 리소스에 대한 빠른 링크가 생성됩니다.

설명에서 @mention을 사용하려면

1. 프로젝트로 이동합니다.
2. 탐색 바에서 문제를 선택합니다.
3. 편집하려는 문제를 선택하여 문제의 세부 정보를 봅니다. 문제를 찾는 방법에 대한 도움말은 [문제 찾기 및 보기](#) 섹션을 참조하세요.
4. 설명 추가 텍스트 상자를 선택합니다.
5. 다른 사용자를 언급하려면 `@user_name`을 입력합니다.
6. 프로젝트를 언급하려면 `@project_name`을 입력합니다.
7. 다른 문제를 언급하려면 `@issue_name` 또는 `@issue_number`를 입력합니다.
8. 소스 리포지토리에서 특정 파일 또는 코드를 언급하려면 `@file_name`을 입력합니다.

Note

입력 시 @mention과 일치하는 용어가 포함된 상위 5개 항목(사용자, 소스 리포지토리, 프로젝트 등)의 목록에서 작성됩니다.

9. 언급하려는 항목을 선택합니다. 항목이 있는 위치를 보여주는 경로가 설명 텍스트 상자에 작성됩니다.
10. 설명을 완료하고 전송을 선택합니다.

문제 찾기 및 보기

다음 섹션에서는 CodeCatalyst 프로젝트 내에서 문제를 효과적으로 검색하고 보는 방법을 설명합니다.

문제 검색

특정 파라미터를 검색하여 문제를 찾을 수 있습니다. 보다 구체적인 검색에 대한 자세한 내용은 [CodeCatalyst에서 코드, 문제, 프로젝트 및 사용자 검색](#) 섹션을 참조하세요.

문제를 검색하려면

1. 프로젝트로 이동합니다.
2. 검색 창을 사용하여 문제 또는 문제와 관련된 정보를 검색합니다. 쿼리 파라미터를 사용하여 검색을 구체화할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 코드, 문제, 프로젝트 및 사용자 검색 단원을 참조하십시오](#).

문제 정렬

기본적으로 CodeCatalyst의 문제는 수동 순서로 정렬됩니다. 수동 순서는 사용자가 이동시킨 순서대로 문제를 표시합니다. 수동 순서로 정렬할 때는 문제를 끌어다 놓아 순서를 변경할 수 있습니다. 이 정렬 옵션은 문제의 백로그를 정리하고 문제의 우선순위를 지정할 때 유용합니다.

다음 표는 그리드 보기와 보드 보기 모두에서 문제를 정렬하는 방법을 보여줍니다.

그리드 보기 정렬 옵션	보드 보기 정렬 옵션
수동 순서	수동 순서
최종 업데이트 날짜	최종 업데이트 날짜
우선순위	우선순위
추정	추정
제목	제목
ID	
Status	
차단됨	
사용자 지정 필드	

다음 절차에 따라 문제가 정렬되는 방법을 변경할 수 있습니다.

문제를 정렬하려면

1. 프로젝트로 이동합니다.

2. 탐색 창에서 문제를 선택합니다. 기본 보기는 보드입니다.
3. (선택 사항) 활성 문제를 선택하여 문제 보기 전환기 드롭다운 메뉴를 열어 다른 문제 보기로 이동합니다.
4. 그리드 보기를 정렬하려면 두 가지 옵션이 있습니다.
 - a. 정렬할 필드의 헤더를 선택합니다. 헤더를 선택하면 오름차순과 내림차순으로 순환됩니다.
 - b. 정렬 기준 드롭다운 메뉴를 선택하고 정렬 기준 파라미터를 선택합니다. 문제는 오름차순으로 정렬됩니다.
5. 보드 보기를 정렬하려면 정렬 기준 드롭다운 메뉴를 선택하고 정렬 기준 파라미터를 선택합니다. 문제는 오름차순으로 정렬됩니다.

문제 그룹화

그룹화는 담당자, 레이블 및 우선순위와 같은 여러 파라미터별로 보드의 문제를 구성하는 데 사용됩니다.

문제를 그룹화하려면

1. 프로젝트로 이동합니다.
2. 탐색 창에서 문제를 선택합니다. 기본 보기는 보드입니다.
3. (선택 사항) 활성 문제를 선택하여 문제 보기 전환기 드롭다운 메뉴를 열어 다른 문제 보기로 이동합니다.
4. 그룹을 선택합니다.
5. 그룹화에서 다음과 같이 그룹화 기준 파라미터를 선택합니다.
 - 담당자 또는 우선순위를 선택한 경우, 그룹 순서를 선택합니다.
 - 레이블을 선택한 경우, 레이블을 선택한 다음 그룹 순서를 선택합니다.
6. (선택 사항) 빈 그룹 표시 토글을 선택하여 현재 할당된 문제가 없는 그룹을 표시하거나 숨깁니다.
7. 선택에 따라 보기가 업데이트됩니다. 구성된 파라미터와 일치하는 그룹에만 문제가 나타납니다.

문제 필터링

필터링을 사용하여 지정된 이름, 우선순위, 레이블, 사용자 지정 필드 또는 담당자가 포함된 문제를 찾습니다.

문제를 필터링하려면

1. 프로젝트로 이동합니다.
2. 탐색 창에서 문제를 선택합니다.
3. (선택 사항) 활성 문제를 선택하여 문제 보기 전환기 드롭다운 메뉴를 열어 다른 문제 보기로 이동합니다.

Note

문제 이름 또는 설명의 문자열을 기준으로 필터링하려면, 문제 검색창에 문자열을 입력합니다.

4. 필터를 선택한 다음 +필터 추가를 선택합니다.
5. 필터링할 파라미터를 선택합니다. 여러 필터와 파라미터를 선택할 수 있습니다. 맞이나 또는을 선택하여 모든 필터 또는 개별 필터와 일치하는 문제를 표시하도록 필터를 구성할 수 있습니다. 보기가 업데이트되어 필터와 일치하는 문제가 표시됩니다.

문제 진행

모든 문제에는 수명 주기가 있습니다. CodeCatalyst에서 문제는 일반적으로 백로그에서 초안으로 시작됩니다. 해당 문제에 대한 작업을 시작해야 할 경우, 문제를 다른 상태 범주로 이동하고 완료될 때까지 다양한 상태로 이동한 다음 보관됩니다. 다음과 같은 방법으로 수명 주기 동안 문제를 이동하거나 진행할 수 있습니다.

- 백로그와 보드 간에 문제를 이동할 수 있습니다.
- 진행 중인 문제를 다양한 완료 단계로 이동할 수 있습니다.
- 완료된 문제를 보관할 수 있습니다.

백로그와 보드 사이에서 문제 이동

문제를 작업하기 시작하면, 백로그에서 보드로 문제를 이동할 수 있습니다. 작업이 연기된 경우, 문제를 백로그로 다시 이동할 수도 있습니다.

백로그와 보드 간에 문제를 이동하려면

1. 프로젝트로 이동합니다.
2. 탐색 창에서 문제를 선택합니다. 기본 보기는 보드입니다.

3. 보드에서 백로그로 문제를 이동하려면:

- a. 이동할 문제를 선택합니다. 문제를 찾는 방법에 대한 도움말은 [문제 찾기 및 보기](#) 섹션을 참조하세요.
- b. 드롭다운 상태 메뉴에서 백로그를 선택합니다.

4. 보드에서 백로그로 문제를 이동하려면:

- a. 백로그로 이동하려면, 보드를 선택하고 백로그를 선택합니다.
- b. 이동할 문제를 선택합니다. 문제를 찾는 방법에 대한 도움말은 [문제 찾기 및 보기](#) 섹션을 참조하세요.
- c. 보드에 추가를 선택하거나 백로그 이외의 상태를 선택합니다.

보드의 라이프 사이클 단계를 통한 문제의 진행

완료 시까지 문제는 보드 내의 다양한 상태로 이동할 수 있습니다.

보드 내에서 문제를 이동시키려면

1. 탐색 창에서 문제를 선택합니다. 기본 보기는 보드입니다.
2. 다음 중 하나를 수행하세요.
 - 문제를 다른 상태로 끌어다 놓습니다.
 - 문제를 선택한 다음 상태 드롭다운 메뉴에서 상태를 선택합니다.
 - 문제를 선택한 다음 다음으로 이동: **## ##**를 선택합니다.

문제 보관에 대한 자세한 내용은 [문제 보관](#) 섹션을 참조하세요.

그룹 간 문제 이동

모든 문제 및 보드 보기에서 다양한 파라미터별로 [문제를 그룹화](#)할 수 있습니다. 문제가 그룹화되면, 한 그룹에서 다른 그룹으로 문제를 이동할 수 있습니다. 한 그룹에서 다른 그룹으로 문제를 이동하면 대상 그룹과 일치하도록 문제가 그룹화된 필드가 자동으로 편집됩니다.

예시 시나리오로, CodeCatalyst를 사용하는 회사에서 Wang Xiulan과 Saanvi Sarkar라는 두 사람에게 문제가 할당되었다고 가정해 보겠습니다. 보드는 Assignee별로 그룹화되며 각 담당자마다 하나씩, 총 두 개의 그룹이 있습니다. Wang Xiulan의 그룹에서 Saanvi Sarkar의 그룹으로 문제를 이동하면, 문제의 담당자가 Saanvi Sarkar로 업데이트됩니다.

문제 보관

Note

문제는 프로젝트 내에서 삭제되지 않고 보관됩니다. 문제를 삭제하려면 프로젝트를 삭제해야 합니다.

프로젝트에서 문제가 더 이상 필요하지 않을 때 해당 문제를 보관할 수 있습니다. 문제를 보관하면 CodeCatalyst는 보관된 문제를 필터링하는 모든 보기에서 문제를 제거합니다. 보관된 문제는 보관된 문제 기본 보기에서 볼 수 있으며, 필요한 경우 이 보기에서 보관을 취소할 수 있습니다.

다음과 같은 경우 문제를 보관합니다.

- 문제를 완료했으며 완료 열에 더 이상 문제를 둘 필요가 없습니다.
- 이 작업을 수행할 계획이 없습니다.
- 실수로 문제를 생성했습니다.
- 최대 활성 문제 수에 도달했습니다.

문제를 보관하려면

1. 보관하려는 문제를 엽니다. 문제를 찾는 방법에 대한 도움말은 [문제 찾기 및 보기](#) 섹션을 참조하세요.
2. 작업을 선택한 후 이동하여 보관을 선택합니다.
3. (선택 사항) 완료됨 상태의 여러 문제를 빠르게 보관하려면 보드의 완료됨 상태 상단에 있는 세로 줄임표를 선택하고 문제 보관을 선택합니다.

문제를 보관 해제하려면

1. 보관 해제하려는 문제를 엽니다. 문제 보기 전환기 드롭다운 메뉴에서 보관된 문제 보기를 열어 보관된 문제 목록을 볼 수 있습니다. 문제를 찾는 방법에 대한 도움말은 [문제 찾기 및 보기](#) 섹션을 참조하세요.
2. 보관 해제를 선택합니다.

문제 내보내기

현재 보기의 문제를 .xlsx 파일로 내보낼 수 있습니다. 문제를 내보내려면, 다음 단계를 수행합니다.

문제를 내보내려면

1. 프로젝트로 이동합니다.
2. 탐색 바에서 문제를 선택합니다.
3. 활성 문제를 선택하여 문제 보기 전환기 드롭다운 메뉴를 열고 내보내려는 문제가 포함된 보기로 이동합니다. 보기에 표시된 문제만 내보내집니다.
4. 줄임표 메뉴를 선택하고 Excel로 내보내기를 선택합니다.
5. .xlsx 파일이 다운로드됩니다. 기본적으로 프로젝트의 이름과 내보내기가 완료된 날짜로 제목이 지정됩니다.

백로그, 레이블 및 보드로 작업 구성

모든 팀이 동일한 방식으로 작업하는 것은 아닙니다. 문제가 나타나는 방식을 구성하고 Amazon CodeCatalyst에서 할당하여 진행 중인 작업과 해당 작업의 상태를 정확하게 이해할 수 있습니다. 사용자가 모두 동일한 추정 방법을 사용하도록 문제를 허용할 추정 방법을 선택할 수 있습니다. 사용자 지정 레이블 및 상태를 생성하여 작업에 대한 보기를 필터링할 때 사용할 수도 있습니다. 팀의 작업 방식에 따라 문제에 여러 담당자를 허용할지 아니면 단일 사용자에게만 문제를 할당할지 여부를 구성할 수 있습니다. 문제에 대한 사용자 지정 보기를 생성하여 사용자 또는 팀과 가장 관련성이 높은 정보를 표시하는 방식으로 작업이 표시되도록 할 수도 있습니다.

레이블로 작업 범주화

문제에 대한 레이블을 사용자 지정할 수 있습니다. 여기에는 레이블 편집 및 색상 변경이 포함됩니다. 레이블은 작업을 분류하고 구성하는 데 도움이 될 수 있습니다. 예를 들어, 소프트웨어의 특정 측면 또는 다른 그룹 또는 팀에 대한 레이블을 생성할 수 있습니다.

주제

- [레이블 생성](#)
- [레이블 편집](#)
- [레이블 삭제](#)

레이블 생성

CodeCatalyst에서는 새 문제를 생성하거나 기존 문제를 편집할 때 레이블을 추가하여 레이블을 생성합니다. 자세한 내용은 [CodeCatalyst에서 문제 생성](#) 및 [CodeCatalyst의 문제 편집 및 공동 작업](#) 섹션을 참조하세요.

레이블 편집

다음 절차에 따라 기존 레이블의 이름 또는 색상을 변경합니다.

레이블을 편집하려면

1. 탐색 창에서 문제를 선택합니다.
2. 활성 문제를 선택하여 문제 보기 전환기 드롭다운 메뉴를 열고 설정을 선택합니다.
3. 레이블 타일에는 프로젝트에 사용되는 레이블 목록이 있습니다. 편집하려는 레이블 옆의 편집 아이콘을 선택합니다. 다음 중 한 개 이상을 수행할 수 있습니다.
 - a. 레이블의 이름을 편집합니다.
 - b. 색상을 변경하려면 색상을 선택합니다. 선택기를 사용하여 새 색상을 선택합니다.
4. 레이블에 대한 변경 사항을 저장하려면, 체크 표시 아이콘을 선택합니다.
5. 이제 변경된 레이블이 사용 가능한 레이블 목록에 표시됩니다. 또한 해당 레이블을 사용하는 문제의 수를 확인할 수 있습니다.

Note

각 레이블 옆에 표시되는 번호를 선택해 모든 문제 페이지로 이동하고 해당 레이블이 포함된 모든 문제를 볼 수 있습니다.

레이블 삭제

현재 CodeCatalyst에서 문제 레이블을 삭제할 수 없습니다. 모든 문제에서 레이블을 제거하면, 레이블이 문제 설정의 레이블 섹션에서 미사용 레이블 섹션에 표시됩니다. 미사용 레이블은 필터를 사용하거나 문제에 레이블을 추가할 때 레이블 목록 마지막에 나타납니다. 문제 설정에서 모든 레이블(사용 및 미사용) 및 해당 레이블이 있는 문제에 대한 개요를 확인할 수 있습니다.

사용자 지정 필드를 사용하여 작업 구성

사용자 지정 필드를 생성하여, 프로젝트의 작업을 구성하고 볼 때 도움을 받을 수 있습니다. 사용자 지정 필드는 필터의 사용 가능한 필터 목록에 추가되므로 사용자 지정 필드별로 문제를 필터링할 수 있습니다. 사용자 지정 필드는 이름과 값 페어입니다. 사용자 지정 필드의 이름을 기준으로 필터링한 다음 해당 사용자 지정 필드의 값을 기준으로 필터링합니다.

문제에는 여러 사용자 지정 필드가 있을 수 있습니다.

사용자 지정 필드 생성

CodeCatalyst에서 문제를 생성하거나 기존 문제를 편집할 때 사용자 지정 필드를 추가하여 사용자 지정 필드를 생성합니다. 자세한 내용은 [CodeCatalyst에서 문제 생성](#) 및 [CodeCatalyst의 문제 편집 및 공동 작업](#) 섹션을 참조하세요.

사용자 지정 필드 삭제

사용자 지정 필드를 삭제하려면 사용자 지정 필드를 추가한 각 문제에서 사용자 지정 필드를 제거해야 합니다. 사용자 지정 필드가 삭제되면 필터에 사용자 지정 필드가 더 이상 표시되지 않습니다. 필터를 사용하여 사용자 지정 필드가 있는 모든 문제를 보고, 문제를 편집하여 사용자 지정 필드를 제거할 수 있습니다. 자세한 내용은 [문제 찾기 및 보기](#) 및 [문제 편집](#) 섹션을 참조하세요.

사용자 지정 상태로 작업 추적

보드에서 사용자 지정 상태를 추가할 수 있습니다. 각 사용자 지정 상태는 초안, 시작되지 않음, 시작됨 또는 완료됨 범주 중 하나에 속해야 합니다. 상태 범주는 상태를 구성하고 기본 보기를 채우는 데 사용됩니다. 상태 및 상태 범주에 대한 자세한 내용은 [상태 및 상태 범주](#) 섹션을 참조하고 보기에 대한 자세한 내용은 [문제 찾기 및 보기](#) 섹션을 참조하세요.

상태 생성

1. 탐색 창에서 문제를 선택합니다.
2. 활성 문제를 선택하여 문제 보기 전환기 드롭다운 메뉴를 열고 설정을 선택합니다.
3. 상태에서 상태를 지정할 범주 옆에 있는 더하기 아이콘을 선택합니다.
4. 상태 이름을 지정한 다음 체크 표시 아이콘을 선택합니다.

Note

X 아이콘을 선택하여 상태 추가를 취소합니다.

이제 사용자 지정 상태가 보드에 표시되며 문제를 생성할 때 옵션으로 표시됩니다.

상태를 편집하려면

1. 탐색 창에서 문제를 선택합니다.
2. 활성 문제를 선택하여 문제 보기 전환기 드롭다운 메뉴를 열고 설정을 선택합니다.
3. 상태에서 편집하거나 변경할 상태 옆에 있는 편집 아이콘을 선택합니다.
4. 상태를 편집한 다음, 체크 표시 아이콘을 선택합니다.

이제 편집된 상태가 보드에 표시됩니다.

상태를 이동하려면

1. 탐색 창에서 문제를 선택합니다.
2. 줄임표 아이콘을 선택하고 설정을 선택합니다.
3. 상태에서 이동할 상태를 선택합니다.
4. 선택한 상태를 원하는 곳에 끌어다 놓습니다.

Note

지정된 범주 내에서만 상태를 이동할 수 있습니다.

이제 보드에서 상태가 재정렬됩니다.

상태를 비활성화하려면

1. 탐색 창에서 문제를 선택합니다.
2. 활성 문제를 선택하여 문제 보기 전환기 드롭다운 메뉴를 열고 설정을 선택합니다.
3. 상태에서 비활성화할 상태를 선택합니다.
4. 비활성화하려는 상태에서 상태 토글을 선택합니다. 이제 해당 상태가 회색으로 표시됩니다.

Note

비활성화된 상태는 모든 문제가 보드 밖으로 이동할 때까지 보드에 표시됩니다. 비활성화된 상태에는 문제를 추가할 수 없습니다.

- 비활성화된 상태를 다시 활성화하려면 상태 토글을 선택합니다. 해당 상태는 더 이상 회색으로 표시되지 않습니다.

Note

각 범주에는 활성 상태가 하나 이상 있어야 합니다. 범주에 상태가 하나만 있는 경우, 해당 상태는 비활성화할 수 없습니다.

문제에 대한 노력 추정 구성

CodeCatalyst의 문제에 대한 노력 추정의 설정을 구성하려면 다음 단계를 따르세요.

문제에 대한 노력 추정을 구성하려면

- 탐색 창에서 문제를 선택합니다.
- 활성 문제를 선택하여 문제 보기 전환기 드롭다운 메뉴를 열고 설정을 선택합니다.
- 기본 설정 섹션의 추정에서 추정 값이 표시되는 방법을 선택합니다. 사용 가능한 추정 유형은 티셔츠 크기 조정, Fibonacci 시퀀싱 또는 추정 숨기기입니다. 추정 유형이 업데이트되면, 데이터가 손실되지 않고 모든 문제의 추정 값이 자동으로 변환됩니다. 변환 매핑은 다음 표에 나와 있습니다.

티셔츠 사이즈	Fibonacci 시퀀스
XS	1
XS	2
S	3
M	5
L	8

티셔츠 사이즈	Fibonacci 시퀀스
XL	13

다수의 담당자 활성화 또는 비활성화

CodeCatalyst의 문제에 다수의 담당자에 대한 설정을 구성하려면 다음 단계를 따르세요.

다수의 담당자를 활성화 또는 비활성화하려면

1. 탐색 창에서 문제를 선택합니다.
2. 활성 문제를 선택하여 문제 보기 전환기 드롭다운 메뉴를 열고 설정을 선택합니다.
3. 기본 설정 섹션의 담당자에서 표시기를 전환하여 다수의 담당자가 동일한 문제에 할당될 수 있도록 합니다. 한 문제에는 최대 10명의 담당자가 있을 수 있습니다. 이 옵션을 활성화하지 않으면, 한 문제에 한 명의 담당자만 할당할 수 있습니다.

문제 보기 생성

[보기](#)를 생성하여 특정 필터 집합과 일치하는 문제를 빠르게 볼 수 있습니다. 이렇게 하면 시간을 절약하고 이전에 필터링, 그룹화 또는 정렬한 문제를 빠르게 확인할 수 있습니다.

문제 보기를 생성하려면

1. 탐색 창에서 문제를 선택합니다.
2. (선택 사항) 사용 사례에 따라 기존 보기에서 보기를 생성할 수 있습니다. 다른 보기로 이동하려면 활성 문제를 선택하여 문제 보기 전환기 드롭다운 메뉴를 열고 보기를 선택합니다.
3. (선택 사항) 보기를 생성하기 전에 필터, 그룹화 및 정렬을 구성합니다. 보기를 생성할 때 이러한 항목을 추가할 수 있지만, 이전에 보기를 생성한 경우 보기를 생성하기 전에 보기에 표시된 내용을 미리 볼 수 있습니다.
4. 헤더 표시줄에서 문제 보기 전환기 드롭다운 메뉴를 엽니다. 상태에 따라 열에서 문제가 표시되는 보드 보기를 생성하려면 보드 열에서 +를 선택합니다. 목록에서 문제가 표시되는 그리드 보기를 생성하려면 그리드 열에서 +를 선택합니다. 생각이 바뀌었다면 보기가 생성되기 전에 보기 유형을 변경할 수 있습니다.
5. 보기 생성 대화 상자에서 보기 이름을 입력하고 생성을 선택합니다.
6. 필터, 문제 그룹화 기준 및 문제 정렬 기준 필드는 현재 보기 설정에 따라 채워집니다. 필요한 경우 업데이트를 하세요.

7. 보기 생성을 선택하여 보기를 생성하고 보기로 전환합니다.

CodeCatalyst 문제에 대한 할당량

다음 표에서는 Amazon CodeCatalyst 의 문제에 대한 할당량 및 한도에 대해 설명합니다. Amazon CodeCatalyst의 할당량에 대한 자세한 내용은 [CodeCatalyst 할당량](#) 섹션을 참조하세요.

리소스	기본 할당량
활성 문제	프로젝트당 최대 1,000개.
첨부 파일 크기	첨부 파일당 최대 500MB. 최대 총 첨부 파일 스토리지는 스페이스의 전체 스토리지 제한에 영향을 받습니다. 자세한 내용은 요금 을 참조하세요.
총 문제 수(활성 및 보관)	프로젝트당 최대 100,000개.
저장된 보기	프로젝트당 최대 50개의 저장된 문제 보기.
문제에 연결 가능한 풀 요청 수	문제당 풀 요청 최대 50개.
상태(프로젝트당)	프로젝트당 최대 50개.
상태(문제당)	문제당 최대 50개.
레이블(프로젝트당)	프로젝트당 최대 200개.
레이블(문제당)	문제당 최대 50개.
사용자 지정 필드(문제당)	문제당 최대 50개.
담당자	문제당 최대 10명.
설명	문제당 최대 1,000개.
작업	문제당 최대 100개.

CodeCatalyst에서 ID, 권한 및 액세스 구성

Amazon CodeCatalyst에 처음 로그인하면 AWS Builder ID를 생성합니다. AWS Builder IDs에 존재하지 않습니다 AWS Identity and Access Management. 처음 로그인하면서 선택한 사용자 이름은 ID의 고유한 사용자 ID가 됩니다.

CodeCatalyst에서는 처음 로그인할 때 다음 두 가지 방법 중 하나로 로그인할 수 있습니다.

- 스페이스를 생성하는 과정에서 로그인하는 방법
- CodeCatalyst의 프로젝트 또는 스페이스에 대한 초대를 수락하는 과정에서 로그인하는 방법

ID와 연결된 역할에 따라 CodeCatalyst에서 수행할 수 있는 작업이 결정됩니다. 프로젝트 관리자 및 기고자와 같은 프로젝트 역할은 프로젝트에만 해당되므로 한 프로젝트에서는 하나의 역할을, 다른 프로젝트에서는 다른 역할을 가질 수 있습니다. 스페이스를 생성하면 CodeCatalyst는 스페이스 관리자 역할을 자동으로 할당합니다. 사용자가 프로젝트 초대를 수락하면 CodeCatalyst는 해당 자격 증명을 스페이스에 추가하고 제한된 액세스 역할을 할당합니다. 사용자를 프로젝트에 초대할 때 사용자가 프로젝트에서 수행할 역할을 선택하면, 해당 사용자가 프로젝트 내에서 수행할 수 있는 작업과 수행할 수 없는 작업이 결정됩니다. 프로젝트를 수행하는 대부분의 사용자는 기고자 역할만 있으면 작업을 수행할 수 있습니다. 자세한 내용은 [사용자 역할로 액세스 권한 부여](#) 단원을 참조하십시오.

프로젝트 역할 외에도 프로젝트의 사용자는 Git 클라이언트 또는 통합 개발 환경(IDE)을 사용할 때 프로젝트의 소스 리포지토리에 액세스하려면 개인 액세스 토큰(PAT)이 필요합니다. 프로젝트 멤버는 이 PAT를 타사 애플리케이션과 함께 CodeCatalyst ID와 연결된 애플리케이션별 암호로 사용할 수 있습니다. 예를 들어 소스 리포지토리를 로컬 컴퓨터에 복제할 때는 PAT와 CodeCatalyst 사용자 이름을 제공해야 합니다.

워크플로에 작업을 배포할 때 [서비스 역할](#)을 사용하여 AWS CloudFormation 스택 및 AWS 리소스에 액세스하는 등의 작업을 수행하여 CodeCatalyst와 리소스 간의 액세스를 구성할 수 있습니다. 실행할 프로젝트 템플릿에 포함된 워크플로 작업에 대해 CodeCatalyst와 AWS 리소스 간의 액세스를 구성해야 합니다.

주제

- [사용자 역할로 액세스 권한 부여](#)
- [개인 액세스 토큰을 사용하여 사용자 리포지토리 액세스 권한 부여](#)
- [개인 연결을 사용하여 GitHub 리소스에 액세스](#)
- [다중 인증\(MFA\)으로 로그인하도록 AWS Builder ID 구성](#)

- [Amazon CodeCatalyst의 보안](#)
- [로그를 사용하여 이벤트 및 API 직접 호출 모니터링](#)
- [CodeCatalyst의 ID, 권한 및 액세스 할당량](#)
- [문제 해결](#)

사용자 역할로 액세스 권한 부여

Amazon CodeCatalyst에서는 프로젝트 수준과 스페이스 수준에서 모두 사용자에게 역할을 할당할 수 있습니다. 프로젝트에서 역할은 사용자가 해당 프로젝트의 리소스를 사용하여 프로젝트에서 수행할 수 있는 작업을 지정합니다. 사용자는 프로젝트에 참여할 때 스페이스에 대한 멤버십을 얻습니다. 사용자를 스페이스의 관리자로 추가하거나 제거할 수 있습니다. 스페이스 관리자 역할은 CodeCatalyst의 모든 역할 중에서 가장 광범위한 권한을 갖습니다. 가장 좋은 방법은 사용자에게 작업을 수행하는 데 필요한 가장 좁은 권한을 할당하는 것입니다.

스페이스의 사용자에게는 역할을 할당할 수 있습니다. 멤버인 프로젝트의 사용자에게 역할을 할당할 수도 있습니다. 각 사용자는 프로젝트 또는 스페이스에서 하나의 역할만 가질 수 있지만 각 프로젝트 및 스페이스에서는 서로 다른 역할을 가질 수 있습니다. 예를 들어, 사용자는 한 프로젝트에서 프로젝트 관리자 역할을, 다른 프로젝트에서 기고자 역할을 가질 수 있습니다.

주제

- [스페이스 및 프로젝트의 사용자 역할 이해](#)
- [각 역할이 사용할 수 있는 권한 보기](#)
- [사용자 역할 보기 및 변경](#)

스페이스 및 프로젝트의 사용자 역할 이해

스페이스에는 세 가지 역할이 있습니다.

- 스페이스 관리자
- 파워 유저
- 제한된 액세스

프로젝트 초대를 수락하는 사용자에게는 프로젝트가 포함된 스페이스에서 제한된 액세스 역할이 자동으로 할당됩니다.

프로젝트의 멤버는 네 가지 역할을 사용할 수 있습니다.

- 프로젝트 관리자
- 기고자
- 검토자
- 읽기 전용

프로젝트에 사용자를 추가하면 CodeCatalyst는 자동으로 사용자에게 제한된 액세스 역할을 부여합니다. 모든 프로젝트에서 사용자를 제거하면 CodeCatalyst는 해당 사용자로부터 제한된 액세스 역할을 자동으로 제거합니다.

스페이스 관리자 역할

스페이스 관리자 역할은 CodeCatalyst에서 가장 강력한 역할입니다. 이 역할에는 CodeCatalyst의 모든 권한이 있으므로 스페이스의 모든 측면을 관리해야 하는 사용자에게만 스페이스 관리자 역할을 할당합니다. 스페이스 관리자 역할을 지닌 사용자는 스페이스 관리자 역할에서 다른 사용자를 추가 또는 제거하고 스페이스를 삭제할 수 있는 유일한 사용자입니다.

스페이스를 만들면 CodeCatalyst는 스페이스 관리자 역할을 자동으로 할당합니다. 원래 스페이스 생성자가 활동할 수 없는 경우를 대비해 이 역할을 수행할 수 있는 한 명 이상의 다른 사용자에게 이 역할을 추가하는 것을 권장합니다.

파워 유저 역할

파워 유저 역할은 CodeCatalyst 스페이스에서 두 번째로 강력한 역할이지만, 스페이스의 프로젝트에 액세스할 수는 없습니다. 스페이스 내에 프로젝트를 생성하고 스페이스의 사용자와 리소스를 관리하는 데 도움이 필요한 사용자를 위해 설계되었습니다. 업무의 일환으로 프로젝트를 생성하고 스페이스에서 사용자를 관리하는 기능이 필요한 팀 리더 또는 관리자에게 파워 유저 역할을 할당합니다.

제한된 액세스 역할

제한된 액세스 역할은 대부분의 사용자가 CodeCatalyst 스페이스에서 갖게 될 역할입니다. 이 역할은 사용자가 스페이스의 프로젝트에 대한 초대를 수락하면 사용자에게 자동으로 할당되는 역할입니다. 해당 프로젝트가 포함된 스페이스 내에서 작업하는 데 필요한 제한된 권한을 제공합니다. 작업 시 스페이스의 일부를 관리해야 하는 경우가 아니라면 스페이스에 직접 초대하는 사용자에게 제한된 액세스 역할을 할당합니다.

프로젝트 관리자 역할

프로젝트 관리자 역할은 CodeCatalyst 프로젝트에서 가장 강력한 역할입니다. 프로젝트 설정 편집, 프로젝트 권한 관리, 프로젝트 삭제 등 프로젝트의 모든 측면을 관리해야 하는 사용자에게만 이 역할을 할당합니다.

프로젝트 역할에는 스페이스 수준의 권한이 없습니다. 따라서 프로젝트 관리자 역할을 지닌 사용자는 추가 프로젝트를 생성할 수 없습니다. 스페이스 관리자 또는 파워 유저 역할을 지닌 사용자만 프로젝트를 생성할 수 있습니다.

Note

스페이스 관리자 역할에는 CodeCatalyst의 모든 권한이 있습니다.

기고자 역할

기고자 역할은 CodeCatalyst 프로젝트의 대다수 멤버를 대상으로 합니다. 프로젝트에서 코드, 워크플로, 문제 및 작업을 수행할 수 있어야 하는 사용자에게 이 역할을 할당합니다.

검토자 역할

검토자 역할은 풀 요청 및 문제와 같은 프로젝트의 리소스와 상호 작용할 수 있어야 하지만, CodeCatalyst 프로젝트에서 코드를 생성 및 병합하거나 워크플로를 생성하거나 워크플로 실행을 시작 또는 중지할 수 없는 사용자를 위한 것입니다. 풀 요청을 승인하고 풀 요청에 대해 의견을 제시하며, 문제를 생성, 업데이트, 해결하고 문제에 대해 의견을 제시하며, 프로젝트의 코드 및 워크플로를 볼 수 있어야 하는 사용자에게 검토자 역할을 할당합니다.

읽기 전용 역할

읽기 전용 역할은 리소스 및 리소스 상태를 확인해야 하지만 리소스와 상호 작용하거나 프로젝트에 직접 기여하지 않는 사용자를 위한 것입니다. 이 역할을 가진 사용자는 CodeCatalyst에서 리소스를 생성할 수 없지만 리포지토리 복제 및 로컬 컴퓨터에 문제에 대한 첨부 파일 다운로드와 같이 리소스를 보고 복사할 수 있습니다. 리소스와 프로젝트 상태를 볼 필요가 있지만 직접 상호 작용하지 않는 사용자에게 읽기 전용 역할을 할당합니다.

각 역할이 사용할 수 있는 권한 보기

다음 표에는 각 CodeCatalyst 역할이 사용할 수 있는 권한이 나와 있습니다. 링크를 사용하여 적절한 권한 집합으로 이동합니다.

- [Space permissions](#)
- [Extensions permissions](#)
- [Project permissions](#)
- [Source repository permissions](#)
- [Dev Environment permissions](#)
- [Package repository and package permissions](#)
- [Workflow permissions](#)
- [Issues permissions](#)
- [Blueprint permissions](#)
- [Notifications permissions](#)
- [Search permissions](#)

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할

스페이스 권한

스페이스 생성							
스페이스 청구 세부 정보 편집							
Single Sign-On 설정 및 활성화							
Single Sign-On 제거							
스페이스에 대한							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
생성형 AI 기능 활성화							
스페이스에 대한 생성형 AI 기능 비활성화							
스페이스 삭제							
스페이스 관리자 역할에 다른 사용자 추가							
스페이스 관리자 역할에서 다른 사용자 제거							
팀 생성							
팀 삭제							
팀 업데이트							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
스페이스에 대한 시스템 리소스 비활성화							
스페이스에 대한 시스템 리소스 활성화							
프로젝트 생성							
AWS 계정 연결을 스페이스에 연결							
AWS 계정 연결 업데이트							
스페이스에서 AWS 계정 연결 해제							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
AWS 계정 연결을 삭제하고 스페이스에서 제거							
스페이스에서 프로젝트 제한 계정 연결 활성화 ¹							
스페이스에서 프로젝트 제한 계정 연결 비활성화 ²							
스페이스에 다른 사용자 초대							
VPC 연결 생성							
VPC 연결 편집							
VPC 연결 삭제							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
스페이스 내 활동 로그 보기							
AWS 계정 연결 보기							
CodeCatalyst에 대한 인시던트 보기							
스페이스 보기							
팀 보기							
VPC 연결 보기							

¹ 파워 유저 역할을 사용하면 계정에 대한 프로젝트 제한을 활성화할 수 있지만 멤버로 속한 프로젝트에 대해서만 액세스를 구성할 수 있습니다.

² 파워 유저 역할을 사용하면 계정에 대한 프로젝트 제한을 비활성화할 수 있지만 멤버로 속한 프로젝트에 대해서만 액세스를 구성할 수 있습니다.

확장 프로그램 권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
확장 프로그램 설치							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
확장 프로그램 업데이트							
확장 프로그램 삭제							
GitHub 계정 연결							
GitHub 계정 연결 해제							
Jira 사이트 연결							
Jira 사이트 연결 해제							
설치된 확장 프로그램에 대한 구성 세부 정보 보기							
확장 프로그램 보기							
프로젝트 권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
프로젝트 설정 편집							
프로젝트의 시스템 리소스 비활성화							
프로젝트의 시스템 리소스 활성화							
프로젝트 삭제							
프로젝트에 사용자 초대							
프로젝트 내 사용자의 역할 변경							
프로젝트에서 태그 제거							
프로젝트에 팀 추가							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
프로젝트에서 팀 제거							
팀의 프로젝트 역할 변경							
프로젝트 보기							
프로젝트 활동 보기							
프로젝트의 팀 보기							
블루프린트 보기							
소스 리포지토리 권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
리포지토리 생성							
리포지토리 연결							
리포지토리 연결 해제							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
리포지토리 삭제							
리포지토리 설정 편집							
리포지토리 보기							
리포지토리 설정 보기							
리포지토리 복제							
브랜치 생성							
브랜치 규칙 생성							
기본 브랜치 변경							
브랜치 삭제							
브랜치 병합							
브랜치 규칙 업데이트							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
브랜치 보기							
브랜치 규칙 보기							
폴더 생성							
폴더 삭제							
폴더 편집							
폴더 보기							
파일 생성							
파일 삭제							
파일 편집							
파일 보기							
커밋 생성 및 푸시							
커밋 보기							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
풀 요청 생성							
풀 요청에 대한 승인 규칙 생성							
풀 요청에 대한 병합 요구 사항 재정의							
풀 요청 업데이트							
풀 요청에 대한 승인 규칙 업데이트							
풀 요청 보기							
풀 요청에 대한 승인 규칙 보기							
풀 요청 닫기							
풀 요청 승인							
풀 요청에 대한 설명							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
풀 요청에 대한 설명에 대해 Amazon Q와 상호 작용							
Amazon Q에서 생성한 풀 요청에 대한 개정 생성							
풀 요청에 문제 연결							
풀 요청에서 문제 연결 해제							
개발 환경 권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
자체 개발 환경 생성							
자체 개발 환경 중지							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
다른 사용 자가 생성 한 개발 환경 중지							
자체 개발 환경 재개							
자체 개발 환경 보기							
다른 사용 자가 생성 한 개발 환경 보기							
자체 개발 환경 편집							
다른 사용 자가 생성 한 개발 환경 편집							
자체 개발 환경 삭제							
다른 사용 자가 생성 한 개발 환경 삭제							
개발 환경 용 devfile 생성							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
개발 환경용 devfile 편집							
개발 환경용 devfile 삭제							
개발 환경용 devfile 보기							
패키지 리포지토리 및 패키지 권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
패키지 리포지토리 생성							
패키지 리포지토리 보기							
패키지 리포지토리 편집							
패키지 리포지토리 삭제							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
게이트웨이 패키지 리포지토리 생성							
게이트웨이 패키지 리포지토리 보기							
게이트웨이 패키지 리포지토리 삭제							
업스트림 패키지 리포지토리 추가							
업스트림 패키지 리포지토리의 검색 순서 편집							
업스트림 패키지 리포지토리 제거							
패키지 리포지토리에 연결							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
패키지 리포지토리에서 패키지 읽기							
패키지 리포지토리에 패키지 게시							
업스트림 리포지토리에서 패키지 읽기 및 유지							
패키지 보기							
패키지 버전 보기							
패키지 버전 자산 보기							
패키지 버전 종속성 나열							
패키지 버전 상태 업데이트							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
패키지 오리지인 구성 업데이트							
패키지 버전 삭제							
워크플로 권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
워크플로 생성							
워크플로 업데이트							
워크플로 삭제							
워크플로 시작							
워크플로 중지							
워크플로 보안 암호 생성							
워크플로 보안 암호 업데이트							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
워크플로 보안 암호 삭제							
환경 생성							
환경 삭제							
플릿 생성							
플릿 업데이트							
플릿 삭제							
다른 계정의 워크플로 리소스 관리							
환경에 AWS 계정 연결							
기본 IAM 역할을 환경에 연결							
VPC 연결을 환경에 연결							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
환경에서 VPC 연결 해제							
VPC와 연결된 환경을 워크플로와 연결							
VPC와 연결된 환경을 워크플로와 연결 해제							
워크플로 실행 승인							
워크플로에서 커밋 추적							
환경 보기							
빌드 작업 로그 보기							
플릿 보기							
테스트 작업 로그 보기							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
워크플로 보기							
워크플로 실행 보기							
워크플로 실행 결과 보기							
워크플로 보안 암호 보기							
문제에 대한 권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
문제 생성							
문제 업데이트							
문제 보기							
작업 생성							
작업 업데이트							
작업 보기							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
문제 보관							
Amazon Q에 문제 할당							
문제에 대한 설명에 대해 Amazon Q와 상호 작용							
문제에서 Amazon Q 할당 해제							
Amazon Q로 문제에 대한 작업 권장							
Amazon Q에서 권장하는 작업 생성							
다른 사용자가 생성한 문제 업데이트							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
문제에 대한 설명 보기							
문제에 대한 설명 생성							
문제에 대한 설명 업데이트							
레이블 생성							
레이블 업데이트							
레이블 보기							
문제에 레이블 추가							
문제에서 레이블 제거							
문제에 대한 사용자 지정 상태 생성							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
사용자 지정 상태 업데이트							
사용자 지정 상태 보기							
사용자 지정 상태 이동							
사용자 지정 상태 비활성화							
문제에 첨부 파일 추가							
문제의 첨부 파일 보기							
문제에서 첨부 파일 제거							
문제를 다른 문제에 연결							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
다른 문제에서 문제와 연결 해제							
문제에 대한 링크 업데이트							
문제에 대한 링크 보기							
풀 요청을 문제에 연결							
문제에서 풀 요청과 연결 해제							
Jira 프로젝트 연결							
Jira 프로젝트 연결 해제							
블루프린트 권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
사용자 지정 블루프린트 프로젝트 생성							
미리 보기 사용자 지정 블루프린트 게시							
사용자 지정 블루프린트 게시							
스페이스 블루프린트 카탈로그에 사용자 지정 블루프린트 추가							
스페이스 블루프린트 카탈로그에서 사용자 지정 블루프린트 제거							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
사용자 지정 블루프린트에 대한 게시 권한 관리							
사용자 지정 블루프린트의 카탈로그 버전 관리							
사용자 지정 블루프린트 업데이트							
사용자 지정 블루프린트 버전 삭제							
사용자 지정 블루프린트 삭제							
소스 리포지토리를 사용자 지정 블루프린트로 변환							

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
프로젝트에 사용자 지정 블루프린트 추가							
프로젝트에서 사용자 지정 블루프린트 연결 해제							
적용된 사용자 지정 블루프린트의 버전 업데이트							
사용자 지정 블루프린트의 설정 편집							
게시된 사용자 지정 블루프린트 보기							
알림 권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할

권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
알림 채널 구성							
알림 채널 제거							
알림 설정 편집							
알림 설정 보기							
CodeCatalyst 인스턴트에 대한 알림 자동 수신							
연결된 이메일 계정에 대한 이메일 알림 구성							
검색 권한	스페이스 관리자 역할	파워 유저 역할	제한된 액세스 역할	프로젝트 관리자 역할	기고자 역할	검토자 역할	읽기 전용 역할
프로젝트 내에서 검색							
스페이스 전체 검색							

사용자 역할 보기 및 변경

사용자에게 할당된 역할은 확인이 가능합니다. 이는 프로젝트에서 수행할 수 있는 작업을 이해하는 데 도움이 됩니다. 추가 권한이 필요한 경우에도 사용자의 역할을 변경할 수 있습니다.

프로젝트에서 사용자의 역할을 보려면

1. 각 프로젝트 멤버와 연결된 역할을 보려는 프로젝트로 이동합니다.

Tip

상단 탐색 모음에서 보려는 프로젝트를 선택할 수 있습니다.

2. 탐색 창에서 프로젝트 설정을 선택합니다.
3. 멤버 탭에서 각 프로젝트 멤버의 역할이 역할에 표시됩니다.

프로젝트에서 사용자의 역할을 변경하려면

1. 프로젝트 멤버와 연결된 역할을 변경하려는 프로젝트로 이동합니다.

Tip

상단 탐색 모음에서 보려는 프로젝트를 선택할 수 있습니다.

2. 탐색 창에서 프로젝트 설정을 선택합니다.
3. 멤버 탭의 프로젝트 멤버에서 역할을 변경할 사용자를 선택합니다. 작업을 선택한 후 편집을 선택합니다.
4. 역할에서 프로젝트 역할을 선택한 다음 확인을 선택합니다.

스페이스에서 역할 보기 및 변경

CodeCatalyst에서 프로젝트에 대한 초대를 수락하는 모든 사용자는 프로젝트가 속한 스페이스의 멤버가 됩니다. 스페이스 멤버 목록은 확인이 가능합니다. 스페이스와 리소스를 더 잘 관리하기 위해 사용자의 역할을 제한된 액세스에서 스페이스 관리자로 변경할 수 있습니다. 스페이스 관리자 역할은 사용자가 CodeCatalyst에서 프로젝트를 생성할 수 있는 유일한 역할입니다.

⚠ Warning

스페이스 관리자 역할은 CodeCatalyst에서 가장 강력한 역할입니다. 이 역할을 지닌 사용자는 CodeCatalyst에서 스페이스 삭제를 포함한 모든 작업을 수행할 수 있습니다. 스페이스에 대해 이 정도 수준의 액세스가 필요한 사용자에게만 이 역할을 할당합니다. 자세한 내용은 [스페이스 관리자 역할](#) 단원을 참조하십시오.

스페이스에서 사용자의 역할을 변경하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 스페이스로 이동합니다.

i Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

3. 멤버 탭을 선택합니다.
4. 역할을 변경하려는 사용자를 선택한 다음 역할 변경을 선택합니다.
5. 역할 변경에서 할당할 역할을 선택한 다음 확인을 선택합니다.

개인 액세스 토큰을 사용하여 사용자 리포지토리 액세스 권한 부여

Git 클라이언트 또는 통합 개발 환경(IDE)이 있는 로컬 컴퓨터에서 소스 리포지토리와 같은 일부 CodeCatalyst 리소스에 액세스하려면, 애플리케이션용 암호를 입력해야 합니다. 이 용도로 사용할 개인용 액세스 토큰(PAT)을 만들 수 있습니다. 생성한 PAT는 CodeCatalyst의 모든 스페이스 및 프로젝트에서 사용자 ID와 연결됩니다. CodeCatalyst ID에 대해 둘 이상의 PAT를 생성할 수 있습니다.

생성한 PAT의 이름과 만료 날짜를 볼 수 있으며, 더 이상 필요하지 않은 PAT는 삭제할 수 있습니다. PAT 보안 암호는 생성할 때만 복사할 수 있습니다.

i Note

기본적으로 PAT는 1년 후에 만료됩니다.

PAT 생성

PAT는 CodeCatalyst의 사용자 ID와 연결됩니다. PAT 보안 암호는 생성할 때만 복사할 수 있습니다.

PAT 생성(콘솔)

콘솔을 사용하여 CodeCatalyst에서 PAT를 생성할 수 있습니다.

개인용 액세스 토큰(콘솔)을 만들려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 상단 메뉴 바에서 프로파일 배지를 선택한 다음 내 설정을 선택합니다. CodeCatalyst 내 설정 페이지가 열립니다.

Tip

프로젝트 또는 스페이스의 멤버 페이지로 이동하고 멤버 목록에서 이름을 선택하여 자신의 사용자 프로파일을 찾을 수도 있습니다.

3. 개인용 액세스 토큰에서 생성을 선택합니다.

PAT 생성 페이지가 표시됩니다.

4. PAT 이름에 PAT를 설명하는 이름을 입력합니다.
5. 만료 날짜에 기본 날짜를 유지하거나 달력 아이콘을 선택하여 사용자 지정 날짜를 선택합니다. 만료 날짜는 기본적으로 현재 날짜로부터 1년 후입니다.
6. 생성(Create)을 선택합니다.

Tip

소스 리포지토리의 복제 리포지토리를 선택할 때도 이 토큰을 생성할 수 있습니다.

7. PAT 보안 암호를 복사하려면 복사를 선택합니다. PAT 보안 암호를 검색할 수 있는 곳에 저장합니다.

Important

PAT 보안 암호는 한 번만 표시됩니다. 창을 닫은 후에는 검색할 수 없습니다. PAT 보안 암호를 안전한 위치에 저장하지 않은 경우 다른 보안 암호를 만들 수 있습니다.

PAT 생성(CLI)

CLI를 사용하여 CodeCatalyst에서 PAT를 생성할 수 있습니다.

개인용 액세스 토큰(AWS CLI)을 만들려면

1. 터미널 또는 명령줄에서 다음과 같이 `create-access-token` 명령을 실행합니다.

```
aws codecatalyst create-access-token
```

명령이 제대로 실행되면 다음 예시처럼 생성된 PAT에 대한 정보를 반환합니다.

```
{
  "secret": "value",
  "name": "marymajor-22222EXAMPLE",
  "expiresTime": "2024-02-04T01:56:04.402000+00:00"
}
```

- 2.

PAT 보안 암호를 PAT를 생성할 때 한 번만 볼 수 있습니다. PAT 보안 암호를 잘못 찾았거나 안전하게 저장되지 않은 것이 우려되는 경우 다른 보안 암호를 생성할 수 있습니다.

AWS CLI를 사용하여 사용자 계정과 연결된 PAT를 볼 수 있습니다. PAT 보안 암호 자체의 값이 아닌 PAT에 대한 정보만 볼 수 있습니다.

Note

CodeCatalyst를 사용하려면 최신 버전의 AWS CLI를 사용해야 합니다. 이전 버전에는 CodeCatalyst 명령이 포함되어 있지 않을 수 있습니다. CodeCatalyst와 함께 사용하려면 먼저 AWS CLI 프로파일을 구성해야 합니다. 자세한 내용은 [CodeCatalyst AWS CLI 에서를 사용하](#)
[도록 설정](#) 단원을 참조하십시오.

PAT 보기

CodeCatalyst에서 PAT볼 수 있습니다. 목록에는 사용자 ID와 연결된 모든 PAT가 표시됩니다. PAT는 CodeCatalyst의 모든 스페이스 및 프로젝트에서 해당 사용자 프로파일과 연결됩니다. 만료된 PAT 만료된 후 삭제되므로 표시되지 않습니다.

PAT 보기(콘솔)

콘솔을 사용하여 CodeCatalyst에서 사용자 ID와 연결된 PAT를 볼 수 있습니다.

개인 액세스 토큰을 보려면(콘솔)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 상단 메뉴 바에서 프로파일 배지를 선택한 다음 내 설정을 선택합니다. CodeCatalyst 내 설정 페이지가 열립니다.

Tip

프로젝트 또는 스페이스의 멤버 페이지로 이동하고 멤버 목록에서 이름을 선택하여 자신의 사용자 프로파일을 찾을 수도 있습니다.

3. 개인 액세스 토큰에서 현재 PAT의 이름과 만료 날짜를 확인합니다.

PAT 보기(CLI)

CLI를 사용하여 CodeCatalyst에서 사용자 ID와 연결된 PAT를 볼 수 있습니다.

개인 액세스 토큰을 보려면(AWS CLI)

- 터미널 또는 명령줄에서 다음과 같이 list-access-tokens 명령을 실행합니다.

```
aws codecatalyst list-access-tokens
```

성공하면 명령은 다음 예시와 같이 사용자 계정과 연결된 PAT에 대한 정보를 반환합니다.

```
{
  "items": [
    {
      "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEaaaa",
```

```

    "name": "marymajor-22222EXAMPLE",
    "expiresTime": "2024-02-04T01:56:04.402000+00:00"
  },
  {
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEebbbb",
    "name": "marymajor-11111EXAMPLE",
    "expiresTime": "2023-03-12T01:58:40.694000+00:00"
  }
]
}

```

PAT 삭제

CodeCatalyst에서 사용자 ID와 연결된 PAT를 삭제할 수 있습니다.

PAT 삭제(콘솔)

콘솔을 사용하여 CodeCatalyst에서 PAT를 삭제할 수 있습니다.

개인 액세스 토큰을 삭제하려면(콘솔)

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 상단 메뉴 바에서 프로파일 배지를 선택한 다음 내 설정을 선택합니다. CodeCatalyst 내 설정 페이지가 열립니다.

Tip

프로젝트 또는 스페이스의 멤버 페이지로 이동하고 멤버 목록에서 이름을 선택하여 자신의 사용자 프로파일을 찾을 수도 있습니다.

3. 개인 액세스 토큰에서 삭제하려는 PAT 옆의 선택기를 선택한 다음 삭제를 선택합니다.

PAT 삭제: <name>? 페이지에서 삭제를 확인하려면 텍스트 필드에 삭제를 입력합니다. 삭제를 선택합니다.

PAT 삭제(CLI)

AWS CLI를 사용하여 사용자 ID와 연결된 PAT를 삭제할 수 있습니다. 이렇게 하려면 delete-access-token 명령을 사용하여 볼 수 있는 PAT의 ID를 제공해야 합니다.

Note

CodeCatalyst를 사용하려면 최신 버전의 AWS CLI 를 사용해야 합니다. 이전 버전에는 CodeCatalyst 명령이 포함되어 있지 않을 수 있습니다. CodeCatalyst AWS CLI 에서를 사용하는 방법에 대한 자세한 내용은 섹션을 참조하세요 [CodeCatalyst AWS CLI 에서를 사용하도록 설정](#).

개인 액세스 토큰을 삭제하려면(AWS CLI)

- 터미널 또는 명령줄에서 delete-access-token 명령을 실행하여 삭제하려는 PAT의 ID를 제공합니다. 예를 들어 다음 명령을 실행하여 ID가 **123EXAMPLE**인 PAT를 삭제합니다.

```
aws codecatalyst delete-access-token --id a1b2c3d4-5678-90ab-cdef-EXAMPLEebbbb
```

성공하면 이 명령에서 빈 응답을 반환합니다.

개인 연결을 사용하여 GitHub 리소스에 액세스

개인 연결을 사용하여 타사 GitHub 리소스를 CodeCatalyst에 인증하고 연결할 수 있습니다. 예를 들어, 개인 연결을 사용하여, CodeCatalyst가 GitHub 계정에 액세스하고 프로젝트 또는 블루프린트의 소스로 리포지토리를 생성할 수 있도록 권한을 부여합니다. 연결은 CodeCatalyst ID에 매핑되며 하나 이상의 소스 리포지토리에 연결하는 데 사용할 수 있습니다. 생성한 연결은 CodeCatalyst의 모든 스페이스 및 프로젝트에서 사용자 ID와 연결됩니다.

Note

액세스할 수 있는 GitHub 조직에서 블루프린트를 사용하여 개인 연결을 관리할 수 있습니다.

공급자 유형별로 모든 스페이스에 걸쳐 하나의 사용자 ID(CodeCatalyst 별칭)에 대해 하나의 개인 연결을 생성할 수 있습니다.

CodeCatalyst에서 개인 연결을 사용하여 프로젝트의 GitHub 리포지토리를 생성하고, 블루프린트의 GitHub 소스 리포지토리를 선택하고, CodeCatalyst에서 GitHub 리포지토리의 풀 요청을 관리할 수 있습니다.

Note

블루프린트를 GitHub 리포지토리와 연결하기 위해 개인 연결을 사용하는 것은 CodeCatalyst에서 확장 프로그램을 사용하여 GitHub 리포지토리를 연결하는 것과는 다릅니다. 확장 프로그램에 대한 자세한 내용은 [CodeCatalyst에서 확장 프로그램이 있는 프로젝트에 기능 추가](#) 섹션을 참조하세요.

개인 연결 생성

콘솔을 사용하여 CodeCatalyst에서 사용자 ID와 연결된 개인 연결을 생성할 수 있습니다.

개인 연결을 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 상단 메뉴 바에서 프로필 배지를 선택한 다음 내 설정을 선택합니다. CodeCatalyst 내 설정 페이지가 열립니다.

Tip

프로젝트 또는 스페이스의 멤버 페이지로 이동하고 멤버 목록에서 이름을 선택하여 자신의 사용자 프로필을 찾을 수도 있습니다.

3. 개인 연결에서 생성을 선택합니다.

연결 생성 페이지가 표시됩니다.
4. 생성(Create)을 선택합니다. 연결 생성 페이지가 표시됩니다.
5. 연결 생성 페이지의 공급자에서 GitHub를 선택합니다. 연결 이름에 연결의 이름을 입력합니다. 생성(Create)을 선택합니다.
6. 메시지가 나타나면, GitHub 계정에 가입합니다.
7. 연결 확인 페이지에서 수락을 선택합니다.
8. 설치 확인 페이지에서 권한 부여 버튼을 선택하여 커넥터 애플리케이션을 설치할지 확인합니다.



Amazon CodeCatalyst by **Amazon-CodeCatalyst** would like permission to:

 **Verify your GitHub identity**

 **Know which resources you can access**

 **Act on your behalf**
[? Learn more](#)

[Learn more about Amazon CodeCatalyst](#)

Cancel

Authorize Amazon CodeCatalyst

Authorizing will redirect to

 **Not owned or operated by GitHub**

 Created **3 years ago**

 **More than 1K** GitHub users

개인 연결 삭제

CodeCatalyst에서 사용자 ID와 연결된 개인 연결을 삭제할 수 있습니다.

Note

CodeCatalyst에서 개인 연결을 삭제해도 GitHub 계정에서 애플리케이션이 제거되지 않습니다. 새 개인 연결을 생성하는 경우 해당 앱을 설치할 수 있습니다. GitHub에서 해당 애플리케이션을 제거하려면 애플리케이션을 취소하고 나중에 다시 설치할 수 있습니다.

CodeCatalyst에서 개인 연결을 삭제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 상단 메뉴 바에서 프로파일 배지를 선택한 다음 내 설정을 선택합니다. CodeCatalyst 내 설정 페이지가 열립니다.

Tip

프로젝트 또는 스페이스의 멤버 페이지로 이동하고 멤버 목록에서 이름을 선택하여 자신의 사용자 프로파일을 찾을 수도 있습니다.

3. 개인 연결에서 삭제하려는 연결 옆의 선택기를 선택한 다음 삭제를 선택합니다.

연결 삭제: <name>? 페이지에서 삭제를 확인하려면 텍스트 필드에 삭제를 입력합니다. 삭제를 선택합니다.

1. GitHub에 로그인하고 설치된 앱의 계정 설정으로 이동합니다. 프로파일 아이콘을 선택하고 설정을 선택한 다음, 애플리케이션을 선택합니다.
2. 승인된 GitHub 앱 탭의 승인된 애플리케이션 목록에서 CodeCatalyst에 설치된 앱을 확인합니다. 설치를 취소하려면 취소를 선택합니다.

다중 인증(MFA)으로 로그인하도록 AWS Builder ID 구성

개인용 또는 전문용 AWS Builder ID 프로필을 생성했든 관계없이 다중 인증(MFA)을 또 다른 보안 계층으로 구성하는 것이 좋습니다. 특히 스페이스의 멤버가 프로젝트에서 다른 사람과 협업하는 경우 MFA를 구성하는 것이 좋습니다. 한 명 이상의 사람이 프로젝트에 액세스할 수 있으므로 보안 위반이 발생할 가능성이 더 큼니다.

MFA를 활성화할 때는 이메일과 암호를 사용하여 Amazon CodeCatalyst에 로그인해야 합니다. 로그인에서 이 부분은 첫 번째 요소로 알고 있는 것을 사용합니다. 그런 다음 코드 또는 보안 키로 로그인합니다.

다. 이는 두 번째 요소로, 여러분이 가지고 있는 것입니다. 두 번째 요소는 모바일 디바이스에서 생성하거나 컴퓨터에 연결된 보안 키를 탭하거나 눌러 생성되는 인증 코드일 수 있습니다. 이러한 여러 요소를 종합하면 무단 액세스를 방지하여 보안을 강화할 수 있습니다.

다중 인증에 사용할 디바이스를 등록하는 방법

내 프로필 > 다중 인증에서 다음 절차를 사용하여 다중 인증(MFA)을 위해 새 디바이스를 등록합니다.

Note

이 절차의 단계를 시작하기 전에 먼저 적절한 인증 앱을 디바이스에 다운로드하는 것이 좋습니다. MFA 디바이스에서 사용할 수 있는 앱 목록은 [인증 애플리케이션](#)을 참조하세요.

MFA를 사용할 디바이스를 등록하는 방법

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 오른쪽 상단에서 첫 번째 이니셜이 있는 아이콘 옆의 화살표를 선택한 다음 사용자 프로필을 선택합니다. CodeCatalyst 프로필 페이지가 열립니다.
3. 프로필 관리 페이지에서 프로필 및 보안 관리를 선택합니다. AWS Builder ID 프로필 페이지가 열립니다.
4. 페이지 왼쪽에서 보안을 선택합니다.
5. 다중 인증(MFA) 페이지에서 디바이스 등록을 선택합니다.
6. MFA 디바이스 등록 페이지에서 다음 MFA 디바이스 유형 중 하나를 선택하고 지침을 따릅니다.

- 보안 키 또는 내장된 인증자

1. 사용자 보안 키 등록 페이지에서 브라우저 또는 플랫폼에 나와 있는 지침을 따릅니다.

Note

이 환경은 운영 체제 및 브라우저에 따라 달라질 수 있으므로 브라우저 또는 플랫폼에 표시되는 지침을 따르세요. 디바이스가 성공적으로 등록되면 새로 등록한 디바이스에 친숙한 표시 이름을 연결할 수 있는 옵션이 제공됩니다. 이를 변경하려면 이름 바꾸기를 선택하고 새 이름을 입력한 다음 저장을 선택합니다.

- 인증 앱

1. 인증 앱 설정 페이지에서 QR 코드 그래픽을 포함하여 새 MFA 디바이스의 구성 정보를 확인할 수 있습니다. 그래픽은 QR 코드를 지원하지 않는 디바이스 상에서 수동 입력할 수 있는 보안 암호 키를 표시한 것입니다.
2. 실물 MFA 디바이스를 사용하여 다음을 수행합니다.
 - a. 호환되는 MFA 인증 앱을 엽니다. MFA 디바이스에서 사용할 수 있는 테스트 완료된 앱 목록은 [테스트를 거친 인증 앱](#)을 참조하세요. MFA 앱이 다수의 디바이스를 지원하는 경우 새로운 MFA 디바이스를 생성하는 옵션을 선택합니다.
 - b. MFA 앱의 QR 코드 지원 여부를 결정한 후 인증 관리자 앱 설정 페이지에서 다음 중 한 가지를 실행합니다.
 - i. QR 코드 표시를 선택한 다음 해당 앱을 사용하여 QR 코드를 스캔합니다. 예를 들어, 카메라 모양의 아이콘을 선택하거나 코드 스캔과 비슷한 옵션을 선택합니다. 그런 다음 디바이스의 카메라를 사용하여 해당 코드를 스캔합니다.
 - ii. 비밀 키 표시를 선택한 다음 MFA 앱에 해당 보안 키를 입력합니다.

 Important

AWS Builder ID용 MFA 디바이스를 구성할 때는 QR 코드 또는 보안 암호 키 사본을 안전한 곳에 저장하는 것이 좋습니다. 휴대폰을 분실했거나 MFA 인증 앱을 다시 설치해야 하는 경우에 도움이 될 수 있습니다. 이러한 상황이 발생하면 동일한 MFA 구성을 사용하도록 앱을 빠르게 재구성할 수 있습니다.

3. 인증 앱 설정 페이지의 인증 코드에서 현재 실물 MFA 디바이스에 표시된 일회용 비밀번호를 입력합니다.

 Important

코드를 생성한 후 즉시 요청을 제출하세요. 코드를 생성한 다음 요청을 제출하기 위해 너무 오래 기다리면 MFA 디바이스가 AWS Builder ID 프로파일과 성공적으로 연결되지만 MFA 디바이스가 동기화되지 않습니다. 이는 시간 기반 일회용 암호(TOTP)가 잠시 후에 만료되기 때문입니다. 이 경우, 디바이스를 재동기화할 수 있습니다.

4. Assign MFA(MFA 할당)을 선택합니다. 이제 MFA 디바이스에서 일회용 비밀번호 생성을 시작할 수 있으며 사용할 준비가 되었습니다.

인증 애플리케이션

인증 앱은 일회용 암호(OTP) 기반 제3자 인증자입니다. 모바일 디바이스 또는 태블릿에 설치된 인증 애플리케이션을 승인된 MFA 디바이스로 사용할 수 있습니다. 제3자 인증 애플리케이션은 6자리 인증 코드를 생성할 수 있는 표준 기반 시간 기반 일회용 암호(TOTP) 알고리즘인 RFC 6238과 호환되어야 합니다.

MFA에 대한 메시지가 표시되면 사용자는 제공된 입력 상자에 인증 앱에서 보낸 유효한 코드를 입력해야 합니다. 사용자에게 할당된 각 MFA 디바이스는 고유해야 합니다. 특정 사용자에 대해 두 개의 인증 앱을 등록할 수 있습니다.

테스트를 거친 인증 앱

모든 TOTP 준수 애플리케이션은 IAM Identity Center MFA에서 작동하지만 다음 표에는 선택할 수 있는 잘 알려진 타사 인증자 앱이 나열되어 있습니다.

운영 체제	테스트를 거친 인증 앱
Android	Authy , Duo Mobile , LastPass Authenticator , Microsoft Authenticator , Google Authenticator
iOS	Authy , Duo Mobile , LastPass Authenticator , Microsoft Authenticator , Google Authenticator

MFA 디바이스 변경

MFA 디바이스를 등록한 후 이름을 변경하거나 삭제할 수 있습니다. 추가 보안 계층을 위해 항상 하나 이상의 MFA 디바이스를 활성화하는 것이 좋습니다. 최대 5개의 디바이스를 등록할 수 있습니다. 더 추가할 방법을 알아보려면 [다중 인증에 사용할 디바이스를 등록하는 방법](#) 섹션을 참조하세요.

MFA 디바이스 이름 변경

MFA 디바이스의 이름을 변경하기

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 오른쪽 상단에서 첫 번째 이니셜이 있는 아이콘 옆의 화살표를 선택한 다음 사용자 프로파일을 선택합니다. CodeCatalyst 프로필 페이지가 열립니다.

3. 프로파일 관리 페이지에서 프로파일 및 보안 관리를 선택합니다. AWS Builder ID 프로파일 페이지가 열립니다.
4. 페이지 왼쪽에서 다중 인증을 선택합니다. 이 페이지로 이동하면 이름 바꾸기가 회색으로 표시됩니다.
5. 변경할 MFA 디바이스를 선택합니다. 이름 바꾸기를 선택합니다. 그러면 모달이 팝업됩니다.
6. 표시되는 메시지에서 MFA 디바이스 이름에 새 이름을 입력하고 이름 바꾸기를 선택합니다. 이름이 변경된 디바이스는 다중 인증(MFA) 디바이스 아래에 표시됩니다.

MFA 디바이스 삭제

MFA 디바이스 삭제하기

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 오른쪽 상단에서 첫 번째 이니셜이 있는 아이콘 옆의 화살표를 선택한 다음 사용자 프로파일을 선택합니다. CodeCatalyst 프로파일 페이지가 열립니다.
3. 프로파일 관리 페이지에서 프로파일 및 보안 관리를 선택합니다. AWS Builder ID 프로파일 페이지가 열립니다.
4. 페이지 왼쪽에서 다중 인증을 선택합니다. 이 페이지로 이동하면 삭제가 회색으로 표시됩니다.
5. 변경할 MFA 디바이스를 선택합니다. 삭제를 선택합니다. MFA 디바이스를 삭제하시겠습니까?라는 모달이 나타납니다. 지침에 따라 디바이스를 삭제합니다.
6. 삭제를 선택합니다. 삭제된 디바이스는 더 이상 다중 인증(MFA) 디바이스 아래에 표시되지 않습니다.

Amazon CodeCatalyst의 보안

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 보안에 가장 민감한 공간의 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 는에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다 AWS 클라우드. AWS 또한는 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사자는 [AWS 규정 준수 프로그램](#) 일환으로 보안의 효과를 정기적으로 테스트하고 확인합니다. CodeCatalyst에 적용되는 규정

준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램 [AWS 제공 범위 내 서비스 규정 준수 프로그램](#).

- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 여러분은 데이터의 민감도, 회사 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 문서는 Amazon CodeCatalyst를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 보안 및 규정 준수 목표에 맞게 CodeCatalyst를 구성하는 방법을 보여줍니다. 또한 CodeCatalyst 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법을 알아봅니다.

내용

- [Amazon CodeCatalyst의 데이터 프라이버시](#)
- [Amazon CodeCatalyst의 데이터 보호](#)
- [Amazon CodeCatalyst의 ID 및 액세스 관리](#)
- [Amazon CodeCatalyst에 대한 규정 준수 검증](#)
- [Amazon CodeCatalyst의 복원력](#)
- [Amazon CodeCatalyst의 인프라 보안](#)
- [Amazon CodeCatalyst의 구성 및 취약성 분석](#)
- [Amazon CodeCatalyst의 데이터 및 개인 정보 보호](#)
- [Amazon CodeCatalyst의 워크플로 작업에 대한 모범 사례](#)
- [CodeCatalyst 신뢰 모델 이해](#)

Amazon CodeCatalyst의 데이터 프라이버시

CodeCatalyst는 IAM Identity Center에서 집계 정보를 수집합니다. CodeCatalyst는 이 집계 데이터를 사용하여 CodeCatalyst의 스페이스 및 프로젝트에 대한 액세스를 제공합니다. 다음과 같은 집계 정보가 수집됩니다.

- 사용자의 전체 이름
- 사용자의 이메일
- 사용자의 사용자 이름

CodeCatalyst 데이터는 저장 시 자동으로 암호화됩니다. 고객의 조치는 필요하지 않습니다. 사용자, Identity Center 또는 스페이스가 삭제되면 CodeCatalyst는 24시간 이내에 고객 데이터를 삭제합니다.

CodeCatalyst는 AWS 소유 AWS KMS 키를 사용합니다. CodeCatalyst는 IAM Identity Center에서 검색된 자격 증명 속성에 대해 고객 관리형 KMS 키를 사용한 저장 시 암호화를 지원하지 않습니다.

AWS KMS 키에 대한 자세한 내용은 사용자 설명서의 [AWS KMS 키](#)를 참조하세요.

Amazon CodeCatalyst의 데이터 보호

보안 및 규정 준수는 워크플로에 사용되는 AWS 리소스 사용에 공동 책임 AWS [모델공동 책임](#) 모델이 적용되는 것처럼 Amazon CodeCatalyst와 고객 간의 . 이 모델에서 설명하는 것처럼 CodeCatalyst는 모든 CodeCatalyst를 실행하는 글로벌 인프라를 보호할 책임이 있습니다. 사용자는 이 인프라에 호스팅되는 콘텐츠에 대한 통제 권한을 유지할 책임이 있습니다. 이 공동 책임 모델은 CodeCatalyst의 데이터 보호에 적용됩니다.

데이터 보호를 위해 계정 자격 증명을 보호하고 로그인할 때 다중 인증을 설정하는 것이 좋습니다. 자세한 내용은 [다중 인증\(MFA\)으로 로그인하도록 AWS Builder ID 구성](#) 섹션을 참조하세요.

태그 또는 이름 필드와 같은 자유 형식 필드에 고객의 이메일 주소와 같은 기밀 또는 민감한 정보를 입력하지 마세요. 여기에는 연결된 AWS 계정외에도 CodeCatalyst에 입력한 리소스 이름 및 기타 식별자가 포함됩니다. 예를 들어 스페이스, 프로젝트 또는 배포 플릿 이름의 일부로 기밀 또는 민감한 정보를 입력하지 마세요. 이름에 사용되는 태그 또는 자유 형식 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용되거나 URL 경로에 포함될 수 있습니다. 이는 콘솔, API AWS CLI, CodeCatalyst 작업 개발 키트 또는 모든 AWS SDKs 사용에 적용됩니다.

외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 보안 자격 증명 정보를 URL에 포함시켜서는 안 됩니다.

CodeCatalyst 소스 리포지토리는 상주 시 자동으로 암호화됩니다. 고객의 조치는 필요하지 않습니다. CodeCatalyst는 HTTPS 프로토콜을 사용하여 전송 중인 리포지토리 데이터도 암호화합니다.

CodeCatalyst는 IAM Identity Center에서 검색된 자격 증명 속성에 대해 고객 관리형 KMS 키를 사용한 저장 시 암호화를 지원하지 않습니다.

CodeCatalyst는 MFA를 지원합니다. 자세한 내용은 [다중 인증\(MFA\)으로 로그인하도록 AWS Builder ID 구성](#) 섹션을 참조하세요.

데이터 암호화

CodeCatalyst는 서비스 내에 데이터를 안전하게 저장하고 전송합니다. 모든 데이터는 전송 및 저장 시 암호화됩니다. 서비스에 대한 메타데이터를 포함하여 서비스에 의해 생성되거나 저장된 모든 데이터는 기본적으로 서비스에 저장되고 암호화됩니다.

Note

문제에 대한 정보는 서비스 내에 안전하게 저장되지만 미해결 문제에 대한 정보는 문제 보드, 백로그 및 개별 문제를 본 브라우저의 로컬 캐시에도 저장됩니다. 최적의 보안을 위해 브라우저 캐시를 지워 이 정보를 제거해야 합니다.

GitHub의 AWS 계정 또는 링크된 리포지토리에 대한 계정 연결과 같이 CodeCatalyst에 연결된 리소스를 사용하는 경우 CodeCatalyst에서 해당 연결된 리소스로 전송 중인 데이터는 암호화되지만 연결된 리소스의 데이터 처리는 해당 연결된 서비스에 의해 관리됩니다. 자세한 내용은 [링크된 서비스 및 Amazon CodeCatalyst의 워크플로 작업에 대한 모범 사례](#) 설명서를 참조하세요.

키 관리

CodeCatalyst는 AWS소유 KMS 키를 사용합니다. 즉, 키에서 관리 AWS 하며 고객은 키를 직접 생성하거나 관리할 필요가 없습니다. 이를 통해 AWS내의 일반적인 암호화 시나리오에 대한 키 관리가 간소화됩니다.

인터넷워크 트래픽 개인 정보 보호

CodeCatalyst에서 스페이스를 생성할 때 해당 스페이스에 대한 데이터 및 리소스가 저장될 AWS 리전을 선택합니다. 프로젝트 데이터 및 메타데이터는 해당 AWS 리전을 벗어나지 않습니다. 그러나 CodeCatalyst 내에서 탐색을 지원하기 위해 [파티션](#)의 모든 AWS 리전에 제한된 공간, 프로젝트 및 사용자 메타데이터 세트가 복제됩니다. 해당 파티션 AWS 리전 외부에는 복제되지 않습니다. 예를 들어 스페이스를 생성할 때 미국 서부(오리건)를 AWS 리전 로 선택하면 중국 리전 또는 AWS GovCloud (US)리전에 데이터가 복제되지 않습니다. 자세한 내용은 [관리 AWS 리전](#), [AWS 글로벌 인프라](#) 및 [AWS 서비스 엔드포인트](#)를 참조하세요.

파티션 AWS 리전 내에서 복제된 데이터는 다음과 같습니다.

- 스페이스 이름의 고유성을 보장하기 위해 스페이스 이름을 나타내는 암호화된 해시 값. 이 값은 사람이 읽을 수 없으며 실제 스페이스 이름을 노출하지 않습니다.
- 스페이스의 고유 ID
- 스페이스 간 탐색을 지원하는 스페이스의 메타데이터
- 스페이스가 AWS 리전 위치한 입니다.
- 스페이스 내 모든 프로젝트의 고유 ID
- 스페이스 또는 프로젝트에서 사용자의 역할을 나타내는 역할 ID
- CodeCatalyst 에 가입할 때 다음을 포함한 가입 프로세스에 대한 데이터 및 메타데이터:

- 의 고유 ID AWS Builder ID
- 에 있는 사용자의 표시 이름 AWS Builder ID
- 에 있는 사용자의 별칭 AWS Builder ID
- 사용자가 가입할 때 사용되는 이메일 주소 AWS Builder ID
- 가입 프로세스의 진행 상황
- 가입 프로세스의 일부로 스페이스를 생성하는 경우 스페이스의 결제 계정으로 사용되는 AWS 계정 ID

스페이스 이름은 CodeCatalyst에서 고유합니다. 스페이스 이름에 민감한 데이터를 포함하지 않도록 하세요.

AWS 계정 또는 GitHub 리포지토리에 대한 연결과 같은 연결된 리소스 및 연결된 계정으로 작업할 때는 각각이 지원하는 최고 수준의 보안으로 소스 및 대상 위치를 구성하는 것이 좋습니다. CodeCatalyst는 전송 계층 보안(TLS) 1.2를 사용하여 AWS 계정 AWS 리전 및 가용 영역 간의 연결을 보호합니다.

Amazon CodeCatalyst의 ID 및 액세스 관리

Amazon CodeCatalyst에서는 AWS Builder ID를 생성하고 사용하여 스페이스와 프로젝트에 로그인하고 액세스합니다. AWS Builder ID는 AWS Identity and Access Management (IAM)의 자격 증명이 아니며 존재하지 않습니다 AWS 계정. 그러나 CodeCatalyst는 결제 목적으로 공간을 확인할 때, 그리고 AWS 계정 에 연결되어 해당에서 리소스를 생성하고 사용할 때 IAM과 통합됩니다 AWS 계정.

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 되는 AWS 서비스입니다. IAM 관리자는 누가 리소스를 사용하도록 인증되고(로그인됨) 권한이 부여되는지(권한 있음)를 제어합니다. IAM은 추가 비용 없이 사용할 수 있는 AWS 서비스입니다.

Amazon CodeCatalyst에서 스페이스를 생성할 때 AWS 계정 를 청구 계정으로 연결해야 합니다. CodeCatalyst 스페이스를 확인하기 위한 AWS 계정 의 관리자 권한이나, 해당 권한이 있어야 합니다. 또한 CodeCatalyst가 연결된 AWS 계정에서 리소스를 생성하고 액세스하는 데 사용할 수 있는 스페이스에 IAM 역할을 추가할 수 있습니다. 이를 [서비스 역할](#)이라고 합니다. 둘 이상의 연결을 AWS 계정 생성하고 각 계정에서 CodeCatalyst에 대한 서비스 역할을 생성하도록 선택할 수 있습니다.

Note

CodeCatalyst에 대한 결제는 결제 계정으로 AWS 계정 지정된에서 이루어집니다. 그러나 해당 AWS 계정 또는 연결된 다른에서 CodeCatalyst 서비스 역할을 생성하는 경우 CodeCatalyst 서

비즈니스 역할에서 생성하고 사용하는 AWS 계정 리소스는 연결된에서 청구됩니다 AWS 계정. 자세한 내용은 Amazon CodeCatalyst 관리자 안내서의 [Managing billing](#)을 참조하세요.

주제

- [IAM의 ID 기반 정책](#)
- [IAM의 정책 작업](#)
- [IAM의 정책 리소스](#)
- [IAM의 정책 조건 키](#)
- [CodeCatalyst의 ID 기반 정책 예시](#)
- [태그를 사용하여 계정 연결 리소스에 대한 액세스 제어](#)
- [CodeCatalyst 권한 참조](#)
- [CodeCatalyst에 서비스 연계 역할 사용](#)
- [AWS Amazon CodeCatalyst에 대한 관리형 정책](#)
- [IAM 역할을 사용하여 프로젝트 AWS 리소스에 대한 액세스 권한 부여](#)

IAM의 ID 기반 정책

ID 기반 정책은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 해당 ID는 사용자, 사용자 그룹 또는 역할일 수 있습니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 [IAM 정책 생성](#)을 참조하세요.

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. ID 기반 정책에서는 위탁자가 연결된 사용자 또는 역할에 적용되므로 위탁자를 지정할 수 없습니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 [IAM JSON 정책 요소 참조](#)를 참조하세요.

CodeCatalyst의 ID 기반 정책 예시

CodeCatalyst ID 기반 정책 예시를 보려면 [CodeCatalyst의 ID 기반 정책 예시](#) 섹션을 참조하세요.

IAM의 정책 작업

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 Action 요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 정책 작업은 일반적으로 연결된 AWS API 작업과 이름이 동일합니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.

단일 문에서 여러 작업을 지정하려면 쉼표로 구분합니다.

```
"Action": [
    "prefix:action1",
    "prefix:action2"
]
```

IAM의 정책 리소스

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 지정할 수 있습니다.

Resource JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 문에는 Resource 또는 NotResource 요소가 반드시 추가되어야 합니다. 모범 사례에 따라 [Amazon 리소스 이름\(ARN\)](#)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"

```

IAM의 정책 조건 키

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 지정할 수 있습니다.

Condition 요소(또는 Condition 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. Condition 요소는 옵션입니다. 같거나 작음과 같은 [조건 연산자](#)를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다.

한 문에서 여러 Condition 요소를 지정하거나 단일 Condition 요소에서 여러 키를 지정하는 경우, AWS는 논리적 AND 작업을 사용하여 평가합니다. 단일 조건 키의 여러 값을 지정하는 경우, AWS는 논리적 OR 작업을 사용하여 조건을 평가합니다. 문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 자세한 내용은 IAM 사용 설명서의 [IAM 정책 요소: 변수 및 태그](#)를 참조하세요.

AWS 는 전역 조건 키와 서비스별 조건 키를 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 [AWS 전역 조건 컨텍스트 키](#)를 참조하세요.

CodeCatalyst의 ID 기반 정책 예시

CodeCatalyst에서 AWS 계정 는 스페이스에 대한 결제를 관리하고 프로젝트 워크플로의 리소스에 액세스하는 데 필요합니다. 계정 연결은 스페이스에 AWS 계정 의 추가를 승인하는 데 사용됩니다. ID 기반 정책은 연결된 AWS 계정에서 사용됩니다.

기본적으로 사용자 및 역할은 CodeCatalyst 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS Command Line Interface (AWS CLI) 또는 AWS API를 사용하여 작업을 수행할 수 없습니다. IAM 관리자는 리소스에서 작업을 수행할 수 있는 권한을 사용자와 역할에 게 부여하는 IAM 정책을 생성해야 합니다. 그런 다음 관리자는 해당 권한이 필요한 사용자에게 이러한 정책을 연결해야 합니다.

다음 예시 IAM 정책은 계정 연결과 관련된 작업에 대한 권한을 부여합니다. 이를 사용하여 계정을 CodeCatalyst에 연결하는 액세스를 제한합니다.

예제 1: 사용자가 단일에서 연결 요청을 수락하도록 허용 AWS 리전

다음 권한 정책은 사용자가 CodeCatalyst와 AWS 계정간의 연결 요청을 보고 수락하는 것만 할 수 있도록 허용합니다. 또한 정책은 조건을 사용하여 us-west-2 리전의 작업만 허용하고 다른 리전의 작업은 허용하지 않습니다 AWS 리전. 요청을 보고 승인하기 위해 사용자는 요청에 지정된 계정과 동일한 계정으로 AWS Management Console 에 로그인합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecatalyst:AcceptConnection",
        "codecatalyst:GetPendingConnection"
      ],
      "Resource": "*"
    }
  ]
}
```

```

    "Condition": {
      "StringEquals": {
        "aws:RequestedRegion": "us-west-2"
      }
    }
  ]
}

```

예제 2: 콘솔에서 단일 리전에 대한 연결 관리 허용 AWS 리전

다음 권한 정책은 사용자가 단일 리전 AWS 계정에서 CodeCatalyst와 간의 연결을 관리할 수 있도록 허용합니다. 이 정책은 조건을 사용하여 us-west-2 리전의 작업만 허용하고 다른 리전의 작업은 허용하지 않습니다. 연결을 생성한 후 AWS Management Console에서 옵션을 선택하여 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 생성할 수 있습니다. 예시 정책에서 iam:PassRole 작업의 조건에는 CodeCatalyst에 대한 서비스 위탁자가 포함됩니다. 해당 액세스 권한이 있는 역할만 AWS Management Console에 생성됩니다.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecatalyst:*"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": "us-west-2"
        }
      }
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:CreateRole",
        "iam:CreatePolicy",
        "iam:AttachRolePolicy",

```

```

        "iam:ListRoles"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "iam:PassRole"
    ],
    "Resource": "*",
    "Condition": {
      "StringEquals": {
        "iam:PassedToService": [
          "codecatalyst.amazonaws.com",
          "codecatalyst-runner.amazonaws.com"
        ]
      }
    }
  }
]
}

```

예시 3: 연결 관리 거부

다음 권한 정책은 사용자가 CodeCatalyst와 간의 연결을 관리하는 기능을 거부합니다 AWS 계정.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "codecatalyst:*"
      ],
      "Resource": "*"
    }
  ]
}

```

태그를 사용하여 계정 연결 리소스에 대한 액세스 제어

리소스에 태그가 연결되거나 태그 지정을 지원하는 서비스에 대한 요청에서 전달될 수 있습니다. 정책의 리소스에 태그가 있을 수 있고 정책의 일부 작업에 태그가 포함될 수 있습니다. 태깅 조건 키에는 `aws:RequestTag` 및 `aws:ResourceTag` 조건 키가 포함됩니다. IAM 정책을 생성하면 태그 조건 키를 사용하여 다음을 통제할 수 있습니다.

- 연결 리소스에 이미 있는 태그를 기반으로 해당 리소스에 대해 작업을 수행할 수 있는 사용자
- 작업의 요청에서 전달될 수 있는 태그
- 요청에서 특정 키를 사용할 수 있는지 여부를 통제합니다.

다음 예에서는 CodeCatalyst 계정 사용자에게 정책의 태그 조건을 지정하는 방법을 설명합니다. 조건 키에 대한 자세한 내용은 [IAM의 정책 조건 키](#) 섹션을 참조하세요.

예시 1: 요청의 태그 기반 작업 허용

다음 정책은 사용자에게 계정 연결을 허용할 수 있는 권한을 부여합니다.

이와 관련하여 정책은 요청이 ProjectA 값이 포함된 Project 태그를 지정하는 경우 `AcceptConnection` 및 `TagResource` 작업을 허용합니다. `aws:RequestTag` 조건 키는 IAM 요청에서 전달할 수 있는 태그를 제어하는 데 사용됩니다. `aws:TagKeys` 조건은 태그 키의 대/소문자를 구분합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecatalyst:AcceptConnection",
        "codecatalyst:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/Project": "ProjectA"
        }
      }
    }
  ]
}
```



```

        "ForAllValues:StringEquals": {
            "aws:TagKeys": ["Project"]
        }
    }
}
]
}

```

예 2: 리소스 태그 기반 작업 허용

다음 정책은 사용자에게 계정 연결 리소스에 대해 작업을 수행하고 관련 정보를 가져올 수 있는 권한을 부여합니다.

이와 관련하여 정책은 연결에 ProjectA 값이 포함된 Project 태그가 있으면 특정 작업을 허용합니다. `aws:ResourceTag` 조건 키는 IAM 요청에서 전달할 수 있는 태그를 제어하는 데 사용됩니다.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecatalyst:GetConnection",
        "codecatalyst:DeleteConnection",
        "codecatalyst:AssociateIamRoleToConnection",
        "codecatalyst:DisassociateIamRoleFromConnection",
        "codecatalyst:ListIamRolesForConnection",
        "codecatalyst:PutBillingAuthorization"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:ResourceTag/Project": "ProjectA"
        }
      }
    }
  ]
}

```

CodeCatalyst 권한 참조

이 섹션에서는 CodeCatalyst에 AWS 계정 연결된의 계정 연결 리소스에 사용되는 작업에 대한 권한 참조를 제공합니다. 다음 섹션에서는 계정 연결과 관련된 권한이 필요한 작업에 대해 설명합니다.

계정 연결에 필요한 권한

계정 연결을 사용하려면 다음 권한이 필요합니다.

계정 연결에 대한 CodeCatalyst 권한	필수 권한	리소스
AcceptConnection	이 계정을 CodeCatalyst 스페이스에 연결하기 위한 요청을 수락할 때 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	정책 Resource 요소에는 와일드카드(*)만 지원됩니다.
AssociateIamRoleToConnection	IAM 역할을 계정 연결에 연결하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/connections/ <i>connection_ID</i>
DeleteConnection	계정 연결을 삭제하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/connections/ <i>connection_ID</i>
DisassociateIamRoleFromConnection	계정 연결에서 IAM 역할을 연결 해제하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/connections/ <i>connection_ID</i>
GetBillingAuthorization	계정 연결에 대한 청구 권한을 설명하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/connections/ <i>connection_ID</i>

계정 연결에 대한 CodeCatalyst 권한	필수 권한	리소스
GetConnection	계정 연결을 얻는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/connections/ <i>connection_ID</i>
GetPendingConnection	이 계정을 CodeCatalyst 스페이스에 연결하기 위한 보류 상태의 요청을 수락할 때 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	정책 Resource 요소에는 와일드카드(*)만 지원됩니다.
ListConnections	보류 중이 아닌 계정 연결을 나열하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	정책 Resource 요소에는 와일드카드(*)만 지원됩니다.
ListIamRolesForConnection	계정 연결과 연결된 IAM 역할을 나열하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/connections/ <i>connection_ID</i>
ListTagsForResource	계정 연결과 연결된 태그를 나열하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/connections/ <i>connection_ID</i>
PutBillingAuthorization	계정 연결에 대한 청구 권한을 생성하거나 업데이트하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/connections/ <i>connection_ID</i>

계정 연결에 대한 CodeCatalyst 권한	필수 권한	리소스
RejectConnection	이 계정을 CodeCatalyst 스페이스에 연결하기 위한 요청을 거절할 때 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	정책 Resource 요소에는 와일드카드(*)만 지원됩니다.
TagResource	계정 연결과 연결된 태그를 생성하거나 편집하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/connections/ <i>connection_ID</i>
UntagResource	계정 연결과 연결된 태그를 제거하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/connections/ <i>connection_ID</i>

IAM Identity Center 애플리케이션에 필요한 권한

IAM Identity Center 애플리케이션을 사용하려면 다음 권한이 필요합니다.

IAM Identity Center 애플리케이션에 대한 CodeCatalyst 권한	필수 권한	리소스
AssociateIdentityCenterApplicationToSpace	IAM Identity Center 애플리케이션을 CodeCatalyst 스페이스와 연결하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/identity-center-applications/ <i>identity-center-application_ID</i>
AssociateIdentityToIdentityCenterApplication	CodeCatalyst 스페이스에 대한 IAM Identity Center 애플리케이션과 ID를 연결하는 데 필	arn:aws:codecatalyst:region: <i>account_ID</i> :/identity-

IAM Identity Center 애플리케이션에 대한 CodeCatalyst 권한	필수 권한	리소스
	요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	center-applications/ <i>identity-center-application_ID</i>
BatchAssociateIdentitiesToIdentityCenterApplication	CodeCatalyst 스페이스에 대한 IAM Identity Center 애플리케이션과 여러 ID를 연결하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/identity-center-applications/ <i>identity-center-application_ID</i>
BatchDisassociateIdentitiesFromIdentityCenterApplication	CodeCatalyst 스페이스에 대한 IAM Identity Center 애플리케이션에서 여러 ID의 연결 해제하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/identity-center-applications/ <i>identity-center-application_ID</i>
CreateIdentityCenterApplication	IAM Identity Center 애플리케이션을 생성하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/identity-center-applications/ <i>identity-center-application_ID</i>
CreateSpaceAdminRoleAssignment	주어진 CodeCatalyst 스페이스 및 IAM Identity Center 애플리케이션에 대한 관리자 역할 할당을 생성하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/identity-center-applications/ <i>identity-center-application_ID</i>

IAM Identity Center 애플리케이션에 대한 CodeCatalyst 권한	필수 권한	리소스
DeleteIdentityCenterApplication	IAM Identity Center 애플리케이션을 삭제하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/identity-center-applications/ <i>identity-center-application_ID</i>
DisassociateIdentityCenterApplicationFromSpace	CodeCatalyst 스페이스에서 IAM Identity Center 애플리케이션을 연결 해제하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/identity-center-applications/ <i>identity-center-application_ID</i>
DisassociateIdentityFromIdentityCenterApplication	CodeCatalyst 스페이스에 대한 IAM Identity Center 애플리케이션과 ID를 연결 해제하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/identity-center-applications/ <i>identity-center-application_ID</i>
GetIdentityCenterApplication	IAM Identity Center 애플리케이션에 대한 정보를 가져오는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/identity-center-applications/ <i>identity-center-application_ID</i>
ListIdentityCenterApplications	계정의 모든 IAM Identity Center 애플리케이션에 대한 목록을 보는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	정책 Resource 요소에는 와일드카드(*)만 지원됩니다.

IAM Identity Center 애플리케이션에 대한 CodeCatalyst 권한	필수 권한	리소스
ListIdentityCenterApplicationsForSpace	CodeCatalyst 스페이스별로 IAM Identity Center 애플리케이션의 목록을 보는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/identity-center-applications/ <i>identity-center-application_ID</i>
ListSpacesForIdentityCenterApplication	IAM Identity Center 애플리케이션별로 CodeCatalyst 스페이스의 목록을 보는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/identity-center-applications/ <i>identity-center-application_ID</i>
SynchronizeIdentityCenterApplication	IAM Identity Center 애플리케이션을 지원 Identity Store와 동기화하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/identity-center-applications/ <i>identity-center-application_ID</i>
UpdateIdentityCenterApplication	IAM Identity Center 애플리케이션을 업데이트하는 데 필요합니다. IAM 정책 권한일 뿐, API 작업이 아닙니다.	arn:aws:codecatalyst:region: <i>account_ID</i> :/identity-center-applications/ <i>identity-center-application_ID</i>

CodeCatalyst에 서비스 연결 역할 사용

Amazon CodeCatalyst는 AWS Identity and Access Management(IAM) [서비스 연결 역할](#)을 사용합니다. 서비스 연결 역할은 CodeCatalyst에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할

은 CodeCatalyst에 의해 사전 정의되며 서비스가 사용자를 대신하여 다른 AWS 서비스를 직접적으로 호출하는 데 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 CodeCatalyst를 더 쉽게 설정할 수 있습니다. CodeCatalyst에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한, CodeCatalyst만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔티티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제한 후에만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 CodeCatalyst 리소스에 대한 액세스 권한을 부주의로 삭제할 수 없기 때문에 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 [IAM으로 작업하는 AWS 서비스](#)를 참조하고 서비스 연결 역할(Service-linked roles) 열에 예(Yes)가 있는 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

CodeCatalyst에 대한 서비스 연결 역할 권한

CodeCatalyst는 `AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization`이라는 서비스 연결 역할을 사용하여 사용자를 대신하여 애플리케이션 인스턴스 프로파일 및 관련 디렉터리 사용자 및 그룹에 대한 Amazon CodeCatalyst 읽기 전용 액세스를 허용합니다.

`AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization` 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- `codecatalyst.amazonaws.com`

`AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronizationPolicy`이라는 역할 권한 정책은 지정된 리소스에서 CodeCatalyst가 다음 작업을 완료하도록 허용합니다.

- 작업: CodeCatalyst spaces that support identity federation and SSO users and groups에 대한 View application instance profiles and associated directory users and groups

사용자, 그룹 또는 역할이 서비스 연결 역할을 생성, 편집 또는 삭제할 수 있도록 사용 권한을 구성해야 합니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 권한](#)을 참조하세요.

CodeCatalyst에 대한 서비스 연결 역할 생성

서비스 링크 역할은 수동으로 생성할 필요가 없습니다. AWS Management Console, AWS CLI 또는 AWS API에서 스페이스를 생성하면 CodeCatalyst는 사용자를 위해 서비스 연결 역할을 생성합니다.

⚠ Important

이러한 서비스 연결 역할은 해당 역할이 지원하는 기능을 사용하는 다른 서비스에서 작업을 완료했을 경우 계정에 나타날 수 있습니다. 또한 CodeCatalyst 서비스가 서비스 연결 역할을 지원하기 시작한 2023년 11월 17일 이전에 이 서비스를 사용 중이었다면 CodeCatalyst에서 사용자 계정에 AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization 역할을 이미 생성했습니다. 자세한 내용은 [내 AWS 계정에 표시되는 새 역할](#)을 참조하세요.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 스페이스를 생성하면 CodeCatalyst가 서비스 연결 역할을 다시 생성합니다.

또한 IAM 콘솔을 사용하여 애플리케이션 인스턴스 프로파일 보기 및 연결된 디렉터리 사용자 및 그룹 사용 사례를 통해 서비스 연결 역할을 만들 수도 있습니다. AWS CLI 또는 AWS API에서 `codecatalyst.amazonaws.com` 서비스 이름의 서비스 연결 역할을 생성합니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 생성](#)을 참조하세요. 이 서비스 연결 역할을 삭제하면 동일한 프로세스를 사용하여 역할을 다시 생성할 수 있습니다.

CodeCatalyst에 대한 서비스 연결 역할 편집

CodeCatalyst에서는 AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization 서비스 연결 역할을 편집하도록 허용하지 않습니다. 서비스 링크 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 편집](#)을 참조하세요.

CodeCatalyst에 대한 서비스 연결 역할 삭제

AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization 역할은 수동으로 삭제할 필요가 없습니다. AWS Management Console, AWS CLI 또는 AWS API에서 스페이스를 삭제한 후에는 CodeCatalyst에서 사용자 대신 리소스를 정리하고 서비스 연결 역할을 삭제합니다.

또한 IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 서비스 연결 역할을 수동으로 삭제할 수 있습니다. 단, 서비스 연결 역할에 대한 리소스를 먼저 정리해야 수동으로 삭제할 수 있습니다.

ℹ Note

리소스를 삭제하려 할 때 CodeCatalyst 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization에서 사용하는 CodeCatalyst 리소스 삭제

- [스페이스를 삭제](#)합니다.

IAM을 사용하여 수동으로 서비스 연결 역할 삭제

IAM 콘솔, AWS CLI 또는 AWS API를 사용하여

AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 [서비스에 연결 역할 삭제](#)를 참조하세요.

CodeCatalyst 서비스 연결 역할에 대해 지원되는 리전

CodeCatalyst에서는 서비스를 사용할 수 있는 모든 리전에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용은 [AWS 리전 및 엔드포인트](#) 섹션을 참조하세요.

CodeCatalyst에서는 서비스가 제공되는 모든 리전에서 서비스 연결 역할을 사용하도록 지원하지 않습니다. 다음 리전에서 AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronization 역할을 사용할 수 있습니다.

지역명	리전 자격 증명	CodeCatalyst에서 지원
미국 동부(버지니아 북부)	us-east-1	아니요
미국 동부(오하이오)	us-east-2	아니요
미국 서부(캘리포니아 북부)	us-west-1	아니요
미국 서부(오리건)	us-west-2	예
아프리카(케이프타운)	af-south-1	아니요
아시아 태평양(홍콩)	ap-east-1	아니요
아시아 태평양(자카르타)	ap-southeast-3	아니요
아시아 태평양(뭄바이)	ap-south-1	아니요
아시아 태평양(오사카)	ap-northeast-3	아니요
아시아 태평양(서울)	ap-northeast-2	아니요

지역명	리전 자격 증명	CodeCatalyst에서 지원
아시아 태평양(싱가포르)	ap-southeast-1	아니요
아시아 태평양(시드니)	ap-southeast-2	아니요
아시아 태평양(도쿄)	ap-northeast-1	아니요
캐나다(중부)	ca-central-1	아니요
유럽(프랑크푸르트)	eu-central-1	아니요
유럽(아일랜드)	eu-west-1	예
유럽(런던)	eu-west-2	아니요
유럽(밀라노)	eu-south-1	아니요
유럽(파리)	eu-west-3	아니요
유럽(스톡홀름)	eu-north-1	아니요
중동(바레인)	me-south-1	아니요
중동(UAE)	me-central-1	아니요
남아메리카(상파울루)	sa-east-1	아니요
AWS GovCloud(미국 동부)	us-gov-east-1	아니요
AWS GovCloud(미국 서부)	us-gov-west-1	아니요

AWS Amazon CodeCatalyst에 대한 관리형 정책

AWS 관리형 정책은에서 생성하고 관리하는 독립 실행형 정책입니다 AWS. AWS 관리형 정책은 사용자, 그룹 및 역할에 권한 할당을 시작할 수 있도록 많은 일반적인 사용 사례에 대한 권한을 제공하도록 설계되었습니다.

AWS 관리형 정책은 모든 AWS 고객이 사용할 수 있으므로 특정 사용 사례에 대해 최소 권한 권한을 부여하지 않을 수 있습니다. 사용 사례에 고유한 [고객 관리형 정책](#)을 정의하여 권한을 줄이는 것이 좋습니다.

AWS 관리형 정책에 정의된 권한은 변경할 수 없습니다. 가 관리형 정책에 정의된 권한을 AWS 업데이트하는 AWS 경우 업데이트는 정책이 연결된 모든 보안 주체 자격 증명(사용자, 그룹 및 역할)에 영향을 미칩니다. AWS AWS 서비스는 새가 시작되거나 기존 서비스에 새 API 작업을 사용할 수 있게 되면 AWS 관리형 정책을 업데이트할 가능성이 높습니다.

자세한 내용은 IAM 사용자 가이드의 [AWS 관리형 정책](#)을 참조하세요.

AWS 관리형 정책: AmazonCodeCatalystSupportAccess

이는 모든 스페이스 관리자와 스페이스 멤버가 스페이스 청구 계정과 연결된 비즈니스 또는 엔터프라이즈 프리미엄 지원 플랜을 사용할 수 있는 권한을 부여하는 정책입니다. 이러한 권한을 통해 스페이스 관리자와 멤버는 CodeCatalyst 권한 정책 내에서 권한이 있는 리소스에 대한 프리미엄 지원 플랜을 활용할 수 있습니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- **support** - 사용자가 AWS 지원 사례를 검색, 생성 및 해결할 수 있는 권한을 부여합니다. 또한 커뮤니케이션, 심각도 수준, 첨부 파일 및 관련 지원 사례 세부 정보를 설명할 수 있는 권한을 부여합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
```

```

{
  "Effect": "Allow",
  "Action": [
    "support:DescribeAttachment",
    "support:DescribeCaseAttributes",
    "support:DescribeCases",
    "support:DescribeCommunications",
    "support:DescribeIssueTypes",
    "support:DescribeServices",
    "support:DescribeSeverityLevels",
    "support:DescribeSupportLevel",
    "support:SearchForCases",
    "support:AddAttachmentsToSet",
    "support:AddCommunicationToCase",
    "support:CreateCase",
    "support:InitiateCallForCase",
    "support:InitiateChatForCase",
    "support:PutCaseAttributes",
    "support:RateCaseCommunication",
    "support:ResolveCase"
  ],
  "Resource": "*"
}
]
}

```

AWS 관리형 정책: AmazonCodeCatalystFullAccess

이는 AWS Management Console의 Amazon CodeCatalyst 스페이스 페이지에서 CodeCatalyst 스페이스 및 연결된 계정을 관리할 수 있는 권한을 부여하는 정책입니다. 이 애플리케이션은 CodeCatalyst의 스페이스에 AWS 계정 연결된를 구성하는 데 사용됩니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `codecatalyst` – AWS Management Console의 Amazon CodeCatalyst 스페이스 페이지에 전체 권한을 부여합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CodeCatalystResourceAccess",
      "Effect": "Allow",
      "Action": [
        "codecatalyst:*",
        "iam:ListRoles"
      ],
      "Resource": "*"
    },
    {
      "Sid": "CodeCatalystAssociateIAMRole",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "iam:PassedToService": [
            "codecatalyst.amazonaws.com",
            "codecatalyst-runner.amazonaws.com"
          ]
        }
      }
    }
  ]
}
```

AWS 관리형 정책: AmazonCodeCatalystReadOnlyAccess

이는 AWS Management Console의 Amazon CodeCatalyst 스페이스 페이지에서 스페이스 및 연결된 계정에 대한 정보를 보고 나열할 수 있는 권한을 부여하는 정책입니다. 이 애플리케이션은 CodeCatalyst의 스페이스에 AWS 계정 연결된를 구성하는 데 사용됩니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `codecatalyst` – AWS Management Console의 Amazon CodeCatalyst 스페이스 페이지에 읽기 전용 권한을 부여합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "codecatalyst:Get*",
        "codecatalyst:List*"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS 관리형 정책:

`AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronizationPolicy`

`AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronizationPolicy`를 IAM 엔터티에 연결할 수 없습니다. 이 정책은 CodeCatalyst에서 사용자를 대신하여 작업을 수행할 수 있도록 서비스 연결 역할에 연결됩니다. 자세한 내용은 [CodeCatalyst에 서비스 연계 역할 사용](#) 단원을 참조하십시오.

이 정책을 통해 고객은 CodeCatalyst에서 스페이스를 관리할 때 애플리케이션 인스턴스 프로파일 및 관련 디렉터리 사용자 및 그룹을 볼 수 있습니다. 고객은 ID 페더레이션 및 SSO 사용자 및 그룹을 지원하는 스페이스를 관리할 때 이러한 리소스를 볼 수 있습니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- sso - 사용자가 CodeCatalyst의 관련 스페이스에 대해 IAM Identity Center에서 관리되는 애플리케이션 인스턴스 프로파일을 볼 수 있는 권한을 부여합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid":
      "AmazonCodeCatalystServiceRoleForIdentityCenterApplicationSynchronizationPolicy",
      "Effect": "Allow",
      "Action": [
        "sso:ListInstances",
        "sso:ListApplications",
        "sso:ListApplicationAssignments",
        "sso:DescribeInstance",
        "sso:DescribeApplication"
      ],
      "Resource": "*"
    }
  ]
}
```

AWS 관리형 정책에 대한 CodeCatalyst 업데이트

이 서비스가 이러한 변경 사항을 추적하기 시작한 이후부터 CodeCatalyst의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 이 페이지의 변경 사항에 대한 자동 알림을 받으려면 CodeCatalyst [문서 기록](#) 페이지에서 RSS 피드를 구독하세요.

변경	설명	Date
AmazonCodeCatalystServiceRoleForIdentityCenter	CodeCatalyst가 정책을 추가했습니다.	2023년 11월 17일

변경	설명	Date
rApplicationSynchronizationPolicy - 새 정책	CodeCatalyst 사용자가 애플리케이션 인스턴스 프로파일 및 연결된 디렉터리 사용자 및 그룹을 볼 수 있는 권한을 부여합니다.	
AmazonCodeCatalystSupportAccess - 새 정책	CodeCatalyst가 정책을 추가했습니다. CodeCatalyst 사용자가 지원 사례를 검색, 생성 및 해결하고 관련 커뮤니케이션 및 세부 정보를 볼 수 있는 권한을 부여합니다.	2023년 4월 20일
AmazonCodeCatalystFullAccess - 새 정책	CodeCatalyst가 정책을 추가했습니다. CodeCatalyst에 대한 전체 액세스 권한을 부여합니다.	2023년 4월 20일
AmazonCodeCatalystReadOnlyAccess - 새 정책	CodeCatalyst가 정책을 추가했습니다. CodeCatalyst에 대한 읽기 전용 액세스 권한을 부여합니다.	2023년 4월 20일
CodeCatalyst가 변경 내용 추적 시작	CodeCatalyst가 AWS 관리형 정책에 대한 변경 사항 추적을 시작했습니다.	2023년 4월 20일

IAM 역할을 사용하여 프로젝트 AWS 리소스에 대한 액세스 권한 부여

CodeCatalyst를 CodeCatalyst 스페이스에 연결하여 AWS 리소스 AWS 계정에 액세스할 수 있습니다. 그런 다음 계정을 연결할 때 다음 서비스 역할을 생성하고 연결할 수 있습니다.

JSON 정책에서 사용하는 요소에 대한 자세한 내용은 IAM 사용 설명서의 [IAM JSON 정책 요소 참조](#)를 참조하세요.

- CodeCatalyst 프로젝트 및 워크플로에 AWS 계정 대한의 리소스에 액세스하려면 먼저 CodeCatalyst가 사용자를 대신하여 해당 리소스에 액세스할 수 있는 권한을 부여해야 합니다. 이렇게 하려면 CodeCatalyst가 스페이스의 사용자 및 프로젝트를 대신하여 수임할 수 있는 AWS 계정 있는 연결된에서 서비스 역할을 생성해야 합니다. CodeCatalystWorkflowDevelopmentRole-**spaceName** 서비스 역할을 생성 및 사용하도록 선택하거나 사용자 지정 서비스 역할을 생성하고 이러한 IAM 정책 및 역할을 수동으로 구성할 수 있습니다. 가장 좋은 방법은 이러한 역할에 필요한 최소한의 권한을 할당하는 것입니다.

Note

사용자 지정 서비스 역할의 경우 CodeCatalyst 서비스 위탁자가 필요합니다. CodeCatalyst 서비스 위탁자 및 신뢰 모델에 대한 자세한 내용은 [CodeCatalyst 신뢰 모델 이해](#) 섹션을 참조하세요.

- 연결된를 통해 스페이스에 대한 지원을 관리하려면 CodeCatalyst 사용자가 지원에 액세스할 수 있도록 허용하는 AWSRoleForCodeCatalystSupport 서비스 역할을 생성하고 사용하도록 선택할 수 있습니다. CodeCatalyst 스페이스 지원에 대한 자세한 내용은 [Amazon CodeCatalyst의 지원](#) 섹션을 참조하세요.

CodeCatalystWorkflowDevelopmentRole-**spaceName** 서비스 역할 이해

CodeCatalyst가 연결된 AWS 계정에서 리소스를 생성하고 액세스하는 데 사용할 수 있는 스페이스의 IAM 역할을 추가할 수 있습니다. 이를 [서비스 역할](#)이라고 합니다. 서비스 역할을 생성하는 가장 간단한 방법은 스페이스를 생성할 때 하나를 추가하고 해당 역할에 대한 CodeCatalystWorkflowDevelopmentRole-**spaceName** 옵션을 선택하는 것입니다. 이렇게 하면 AdministratorAccess가 연결된 서비스 역할이 생성될 뿐만 아니라 CodeCatalyst가 스페이스 내 프로젝트의 사용자를 대신하여 역할을 맡을 수 있도록 허용하는 신뢰 정책도 생성됩니다. 서비스 역할은 개별 프로젝트가 아닌 스페이스에 따라 범위가 지정됩니다. 이 역할을 생성하려면 [계정 및 스페이스의 CodeCatalystWorkflowDevelopmentRole-**spaceName** 역할 생성](#) 섹션을 참조하세요. 각 계정의 각 스페이스에 대해 하나의 역할만 생성할 수 있습니다.

Note

이 역할은 개발 계정에서만 사용하는 것이 권장되며 AdministratorAccess AWS 관리형 정책을 사용하여 여기에 새 정책 및 리소스를 생성할 수 있는 전체 액세스 권한을 부여합니다 AWS 계정.

CodeCatalystWorkflowDevelopmentRole-**spaceName** 역할에 연결된 정책은 스페이스에 블루프린트로 생성된 프로젝트와 함께 작동하도록 설계되었습니다. 이를 통해 해당 프로젝트의 사용자는 연결된 AWS 계정의 리소스를 사용하여 코드를 개발, 빌드, 테스트 및 배포할 수 있습니다. 자세한 내용은 [AWS 서비스에 대한 역할 생성을 참조하세요](#).

CodeCatalystWorkflowDevelopmentRole-**spaceName** 역할에 연결된 정책은의 AdministratorAccess 관리형 정책입니다 AWS. 이는 모든 AWS 작업 및 리소스에 대한 전체 액세스 권한을 부여하는 정책입니다. IAM 콘솔에서 JSON 정책 문서를 보려면 [AdministratorAccess](#)를 참조하세요.

다음 신뢰 정책은 CodeCatalyst가 CodeCatalystWorkflowDevelopmentRole-**spaceName** 역할을 수입하도록 허용합니다. CodeCatalyst 신뢰 모델에 대한 자세한 내용은 [CodeCatalyst 신뢰 모델 이해](#) 섹션을 참조하세요.

```
"Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "codecatalyst-runner.amazonaws.com",
          "codecatalyst.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:codecatalyst:::space/spaceId/project/*"
        }
      }
    }
  ]
}
```

계정 및 스페이스의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할 생성

다음 단계에 따라 스페이스의 워크플로에 사용할

CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 생성합니다. 프로젝트에 사용할 IAM 역할을 보유하려는 각 계정의 스페이스에 개발자 역할과 같은 역할을 추가해야 합니다.

시작하기 전에에 대한 관리 권한이 AWS 계정 있거나 관리자와 협력할 수 있어야 합니다.

CodeCatalyst에서 AWS 계정 및 IAM 역할을 사용하는 방법에 대한 자세한 내용은 섹션을 참조하세요 [요연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#).

CodeCatalyst CodeCatalystWorkflowDevelopmentRole-*spaceName*을 생성하고 추가하려면

1. CodeCatalyst 콘솔에서 시작하기 전에를 열고 스페이스 AWS 계정 에 대해 동일한 로 로그인했는지 AWS Management Console확인합니다.
2. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
3. CodeCatalyst 스페이스로 이동합니다. 설정을 선택한 다음 AWS 계정을 선택합니다.
4. 역할을 생성하려는 AWS 계정 의 링크를 선택합니다. AWS 계정 세부 정보 페이지가 표시됩니다.
5. 역할 관리를 AWS Management Console 선택합니다.

AWS Management Console에서 Amazon CodeCatalyst 스페이스에 IAM 역할 추가 페이지가 열립니다. Amazon CodeCatalyst 스페이스 페이지입니다. 페이지에 액세스하려면 로그인해야 할 수 있습니다.

6. IAM에서 CodeCatalyst 개발 관리자 역할 생성을 선택합니다. 이 옵션은 개발 역할에 대한 권한 정책 및 신뢰 정책을 포함하는 서비스 역할을 생성합니다. 역할에는 CodeCatalystWorkflowDevelopmentRole-*spaceName* 이름이 있습니다. 역할 및 역할 정책에 대한 자세한 내용은 [CodeCatalystWorkflowDevelopmentRole-*spaceName* 서비스 역할 이해](#) 섹션을 참조하세요.

Note

이 역할은 개발자 계정에서만 사용하는 것이 권장되며 AdministratorAccess AWS 관리형 정책을 사용하여 여기에 새 정책 및 리소스를 생성할 수 있는 전체 액세스 권한을 부여합니다 AWS 계정.

7. 개발 역할 생성을 선택합니다.

8. 연결 페이지에 있는 CodeCatalyst에서 사용할 수 있는 IAM 역할에서 계정에 추가된 IAM 역할 목록의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 봅니다.
9. 스페이스로 돌아가려면 Amazon CodeCatalyst로 이동을 선택합니다.

AWSRoleForCodeCatalystSupport 서비스 역할 이해

스페이스의 CodeCatalyst 사용자가 지원 사례를 생성하고 액세스하는 데 사용할 수 있는 스페이스의 IAM 역할을 추가할 수 있습니다. 이를 지원을 위한 [서비스 역할](#)이라고 합니다. 지원을 위한 서비스 역할을 생성하는 가장 간단한 방법은 스페이스를 생성할 때 하나를 추가하고 해당 역할에 대한 AWSRoleForCodeCatalystSupport 옵션을 선택하는 것입니다. 이렇게 하면 정책 및 역할이 생성될 뿐만 아니라 CodeCatalyst가 스페이스의 프로젝트에서 사용자를 대신하여 역할을 맡을 수 있도록 허용하는 신뢰 정책도 생성됩니다. 서비스 역할은 개별 프로젝트가 아닌 스페이스에 따라 범위가 지정됩니다. 이 역할을 생성하려면 [계정 및 스페이스의 AWSRoleForCodeCatalystSupport 역할 생성](#) 섹션을 참조하세요.

AWSRoleForCodeCatalystSupport 역할에 연결된 정책은 지원 권한에 대한 액세스를 제공하는 관리형 정책입니다. 자세한 내용은 [AWS 관리형 정책: AmazonCodeCatalystSupportAccess](#) 섹션을 참조하세요.

정책에 대한 이 신뢰 역할은 CodeCatalyst가 역할을 맡도록 허용합니다.

JSON

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

계정 및 스페이스의 AWSRoleForCodeCatalystSupport 역할 생성

다음 단계에 따라 스페이스의 지원 사례에 사용할 AWSRoleForCodeCatalystSupport 역할을 생성합니다. 이 역할은 스페이스의 지정된 청구 계정에 추가해야 합니다.

시작하기 전에에 대한 관리 권한이 AWS 계정 있거나 관리자와 협력할 수 있어야 합니다.

CodeCatalyst에서 AWS 계정 및 IAM 역할을 사용하는 방법에 대한 자세한 내용은 섹션을 참조하세요 [요연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#).

CodeCatalyst AWSRoleForCodeCatalystSupport을 생성하고 추가하려면

1. CodeCatalyst 콘솔에서 시작하기 전에를 열고 스페이스 AWS 계정 에 대해 동일한 로 로그인했는지 AWS Management Console확인합니다.
2. CodeCatalyst 스페이스로 이동합니다. 설정을 선택한 다음 AWS 계정을 선택합니다.
3. 역할을 생성하려는 AWS 계정 의 링크를 선택합니다. AWS 계정 세부 정보 페이지가 표시됩니다.
4. 역할 관리를 AWS Management Console 선택합니다.

AWS Management Console에서 Amazon CodeCatalyst 스페이스에 IAM 역할 추가 페이지가 열립니다. Amazon CodeCatalyst 스페이스 페이지입니다. 페이지에 액세스하려면 로그인해야 할 수 있습니다.

5. CodeCatalyst 스페이스 세부 정보에서 CodeCatalyst 지원 역할 추가를 선택합니다. 이 옵션은 미리 보기 개발 역할에 대한 권한 정책 및 신뢰 정책을 포함하는 서비스 역할을 생성합니다. 역할에는 고유한 식별자가 추가된 AWSRoleForCodeCatalystSupport 이름이 있습니다. 역할 및 역할 정책에 대한 자세한 내용은 [AWSRoleForCodeCatalystSupport 서비스 역할 이해](#) 섹션을 참조하세요.
6. CodeCatalyst 지원에 역할 추가 페이지에서 기본값을 선택한 상태로 두고 역할 생성을 선택합니다.
7. CodeCatalyst에서 사용할 수 있는 IAM 역할에서 계정에 추가된 IAM 역할 목록의 CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할을 봅니다.
8. 스페이스로 돌아가려면 Amazon CodeCatalyst로 이동을 선택합니다.

CodeCatalyst의 워크플로 작업에 대한 IAM 역할 구성

이 섹션에서는 CodeCatalyst 계정과 함께 사용할 수 있도록 생성할 수 있는 IAM 역할 및 정책을 자세히 설명합니다. 예시 역할을 생성하는 방법에 대한 지침은 [워크플로 작업에 대한 역할 수동 생성](#)을 참조하세요. IAM 역할을 생성한 후 역할 ARN을 복사하여 계정 연결에 IAM 역할을 추가하고 프로젝트 환경과 연결합니다. 자세한 내용은 [계정 연결에 IAM 역할 추가](#)을 참조하세요.

Amazon S3 액세스를 위한 CodeCatalyst 빌드 역할

CodeCatalyst 워크플로 빌드 작업의 경우 기본 CodeCatalystWorkflowDevelopmentRole-**spaceName** 서비스 역할을 사용하거나 CodeCatalystBuildRoleforS3Access라는 IAM 역할을 생성할 수 있습니다. 이 역할은 CodeCatalyst가의 CloudFormation 리소스에서 작업을 실행하는 데 필요한 범위가 지정된 권한이 있는 정책을 사용합니다 AWS 계정.

이 역할은 다음에 대한 권한을 부여합니다.

- Amazon S3 버킷에 작성합니다.
- 를 사용한 리소스 구축을 지원합니다 CloudFormation. 이렇게 하려면 Amazon S3 액세스 권한이 필요합니다.

이 역할은 다음과 같은 정책을 사용합니다.

Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

CloudFormation에 대한 CodeCatalyst 빌드 역할

CodeCatalyst 워크플로 빌드 작업의 경우 기본 CodeCatalystWorkflowDevelopmentRole-**spaceName** 서비스 역할을 사용하거나 필요한 권한으로 IAM 역할을 생성할 수 있습니다. 이 역할은 CodeCatalyst가의 CloudFormation 리소스에서 작업을 실행하는 데 필요한 범위가 지정된 권한이 있는 정책을 사용합니다 AWS 계정.

이 역할은 다음에 대한 권한을 부여합니다.

- 를 사용한 리소스 구축을 지원합니다 CloudFormation. 이는 Amazon S3 액세스에 대한 CodeCatalyst 빌드 역할 및 CloudFormation에 대한 CodeCatalyst 배포 역할과 함께 필요합니다.

다음 AWS 관리형 정책들이 이 역할에 연결해야 합니다.

- AWSCloudFormationFullAccess

- IAMFullAccess
- AmazonS3FullAccess
- AmazonAPIGatewayAdministrator
- AWSLambdaFullAccess

CDK에 대한 CodeCatalyst 빌드 역할

최신 3계층 웹 애플리케이션과 같은 CDK 빌드 작업을 실행하는 CodeCatalyst 워크플로의 경우 기본 CodeCatalystWorkflowDevelopmentRole-**spaceName** 서비스 역할을 사용하거나 필요한 권한으로 IAM 역할을 생성할 수 있습니다. 이 역할은 CodeCatalyst가의 CloudFormation 리소스에 대해 CDK 빌드 명령을 부트스트랩하고 실행하는 데 필요한 범위가 지정된 권한이 있는 정책을 사용합니다 AWS 계정.

이 역할은 다음에 대한 권한을 부여합니다.

- Amazon S3 버킷에 작성합니다.
- CDK 구문 및 CloudFormation 리소스 스택 구축을 지원합니다. 이를 위해서는 아티팩트 스토리지용 Amazon S3, 이미지 리포지토리 지원용 Amazon ECR, 가상 인스턴스에 대한 시스템 거버넌스 및 모니터링을 위한 SSM에 대한 액세스 권한이 필요합니다.

이 역할은 다음과 같은 정책을 사용합니다.

Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

CloudFormation에 대한 CodeCatalyst 배포 역할

를 사용하는 CodeCatalyst 워크플로 배포 작업의 경우 기본

CodeCatalystWorkflowDevelopmentRole-**spaceName** 서비스 역할을 CloudFormation사용하거나 CodeCatalyst가의 CloudFormation 리소스에서 작업을 실행하는 데 필요한 범위가 지정된 권한이 있는 정책을 사용할 수 있습니다 AWS 계정.

이 역할은 다음에 대한 권한을 부여합니다.

- CodeCatalyst가 CloudFormation을 통해 블루/그린 배포를 수행하기 위해 $\wedge$ 함수를 간접 호출하도록 허용합니다.
- CodeCatalyst가에서 스택 및 변경 세트를 생성하고 업데이트하도록 허용합니다 CloudFormation.

이 역할은 다음과 같은 정책을 사용합니다.

```
{
  "Action": [
    "cloudformation:CreateStack",
    "cloudformation:DeleteStack",
    "cloudformation:Describe*",
    "cloudformation:UpdateStack",
    "cloudformation:CreateChangeSet",
    "cloudformation:DeleteChangeSet",
    "cloudformation:ExecuteChangeSet",
    "cloudformation:SetStackPolicy",
    "cloudformation:ValidateTemplate",
    "cloudformation:List*",
    "iam:PassRole"
  ],
  "Resource": "resource_ARN",
  "Effect": "Allow"
}
```

Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

Amazon EC2에 대한 CodeCatalyst 배포 역할

CodeCatalyst 워크플로 배포 작업은 필요한 권한이 있는 IAM 역할을 사용합니다. 이 역할은 CodeCatalyst가의 Amazon EC2 리소스에서 작업을 실행하는 데 필요한 범위가 지정된 권한이 있는 정책을 사용합니다 AWS 계정. CodeCatalystWorkflowDevelopmentRole-*spaceName* 역할에 대한 기본 정책에는 Amazon EC2 또는 Amazon EC2 Auto Scaling에 대한 권한이 포함되지 않습니다.

이 역할은 다음에 대한 권한을 부여합니다.

- Amazon EC2 배포를 생성합니다.
- Auto Scaling 그룹 이름으로 인스턴스의 태그를 읽거나 Amazon EC2 인스턴스를 식별합니다.
- Amazon EC2 Auto Scaling 그룹, 수명 주기 후크 및 규모 조정 정책을 읽고, 생성하고, 업데이트하고, 삭제합니다.
- Amazon SNS 주제로 정보를 게시합니다.
- CloudWatch 경보에 관한 정보를 검색합니다.
- Elastic Load Balancing을 읽고 업데이트합니다.

이 역할은 다음과 같은 정책을 사용합니다.

Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

Amazon ECS에 대한 CodeCatalyst 배포 역할

CodeCatalyst 워크플로 작업의 경우 필요한 권한을 가진 IAM 역할을 생성할 수 있습니다. 기본 CodeCatalystWorkflowDevelopmentRole-*spaceName* 서비스 역할을 사용하거나 Lambda 배포에 사용할 CodeCatalyst 배포 작업에 대한 IAM 역할을 생성할 수 있습니다. 이 역할은 CodeCatalyst가 Amazon ECS 리소스에서 작업을 실행하는 데 필요한 범위가 지정된 권한이 있는 정책을 사용합니다 AWS 계정.

이 역할은 다음에 대한 권한을 부여합니다.

- CodeCatalyst 연결에 지정된 계정에서 CodeCatalyst 사용자를 대신하여 롤링 Amazon ECS 배포를 시작합니다.
- Amazon ECS 작업 세트를 읽고, 업데이트하고, 삭제합니다.
- Elastic Load Balancing 대상그룹, 리스너 및 규칙을 업데이트합니다.
- Lambda 함수 간접 호출
- Amazon S3 버킷에 있는 개정 파일에 액세스합니다.

- CloudWatch 경보에 관한 정보를 검색합니다.
- Amazon SNS 주제로 정보를 게시합니다.

이 역할은 다음과 같은 정책을 사용합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Action": [
      "ecs:DescribeServices",
      "ecs:CreateTaskSet",
      "ecs>DeleteTaskSet",
      "ecs:ListClusters",
      "ecs:RegisterTaskDefinition",
      "ecs:UpdateServicePrimaryTaskSet",
      "ecs:UpdateService",
      "elasticloadbalancing:DescribeTargetGroups",
      "elasticloadbalancing:DescribeListeners",
      "elasticloadbalancing:ModifyListener",
      "elasticloadbalancing:DescribeRules",
      "elasticloadbalancing:ModifyRule",
      "lambda:InvokeFunction",
      "lambda:ListFunctions",
      "cloudwatch:DescribeAlarms",
      "sns:Publish",
      "sns:ListTopics",
      "s3:GetObject",
      "s3:GetObjectVersion",
      "codedeploy:CreateApplication",
      "codedeploy:CreateDeployment",
      "codedeploy:CreateDeploymentGroup",
      "codedeploy:GetApplication",
      "codedeploy:GetDeployment",
      "codedeploy:GetDeploymentGroup",
      "codedeploy:ListApplications",
      "codedeploy:ListDeploymentGroups",
      "codedeploy:ListDeployments",
      "codedeploy:StopDeployment",
      "codedeploy:GetDeploymentTarget",
      "codedeploy:ListDeploymentTargets",
```

```

        "codedeploy:GetDeploymentConfig",
        "codedeploy:GetApplicationRevision",
        "codedeploy:RegisterApplicationRevision",
        "codedeploy:BatchGetApplicationRevisions",
        "codedeploy:BatchGetDeploymentGroups",
        "codedeploy:BatchGetDeployments",
        "codedeploy:BatchGetApplications",
        "codedeploy:ListApplicationRevisions",
        "codedeploy:ListDeploymentConfigs",
        "codedeploy:ContinueDeployment"
    ],
    "Resource": "*",
    "Effect": "Allow"
  }, { "Action": [
        "iam:PassRole"
    ],
    "Effect": "Allow",
    "Resource": "*",
    "Condition": { "StringLike": { "iam:PassedToService": [
        "ecs-tasks.amazonaws.com",
        "codedeploy.amazonaws.com"
    ] } }
  }
}
}]
}

```

Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"
```

Lambda에 대한 CodeCatalyst 배포 역할

CodeCatalyst 워크플로 작업의 경우 필요한 권한을 가진 IAM 역할을 생성할 수 있습니다. 기본 CodeCatalystWorkflowDevelopmentRole-**spaceName** 서비스 역할을 사용하거나 Lambda 배포에 사용할 CodeCatalyst 배포 작업에 대한 IAM 역할을 생성할 수 있습니다. 이 역할은 CodeCatalyst가

Lambda 리소스에서 작업을 실행하는 데 필요한 범위가 지정된 권한이 있는 정책을 사용합니다 AWS 계정.

이 역할은 다음에 대한 권한을 부여합니다.

- Lambda 함수 및 별칭을 읽고 업데이트하고 간접 호출합니다.
- Amazon S3 버킷에 있는 개정 파일에 액세스합니다.
- CloudWatch 이벤트 경보에 관한 정보를 검색합니다.
- Amazon SNS 주제로 정보를 게시합니다.

이 역할은 다음과 같은 정책을 사용합니다.

Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

Lambda에 대한 CodeCatalyst 배포 역할

CodeCatalyst 워크플로 작업의 경우 기본 CodeCatalystWorkflowDevelopmentRole-*spaceName* 서비스 역할을 사용하거나 필요한 권한으로 IAM 역할을 생성할 수 있습니다. 이 역할은 CodeCatalyst가의 Lambda 리소스에서 작업을 실행하는 데 필요한 범위가 지정된 권한이 있는 정책을 사용합니다 AWS 계정.

이 역할은 다음에 대한 권한을 부여합니다.

- Lambda 함수 및 별칭을 읽고 업데이트하고 간접 호출합니다.
- Amazon S3 버킷에 있는 개정 파일에 액세스합니다.
- CloudWatch 경보에 관한 정보를 검색합니다.
- Amazon SNS 주제로 정보를 게시합니다.

이 역할은 다음과 같은 정책을 사용합니다.

Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

에 대한 CodeCatalyst 배포 역할 AWS SAM

CodeCatalyst 워크플로 작업의 경우 기본 CodeCatalystWorkflowDevelopmentRole-*spaceName* 서비스 역할을 사용하거나 필요한 권한으로 IAM 역할을 생성할 수 있습니다. 이 역할은 CodeCatalyst가 AWS SAM 및 CloudFormation 리소스에서 작업을 실행하는 데 필요한 범위가 지정된 권한이 있는 정책을 사용합니다 AWS 계정.

이 역할은 다음에 대한 권한을 부여합니다.

- CodeCatalyst가 서버리스 및 AWS SAM CLI 애플리케이션의 배포를 수행하기 위해 Lambda 함수를 간접 호출하도록 허용합니다.
- CodeCatalyst가에서 스택 및 변경 세트를 생성하고 업데이트하도록 허용합니다 CloudFormation.

이 역할은 다음과 같은 정책을 사용합니다.

Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

Amazon EC2에 대한 CodeCatalyst 읽기 전용 역할

CodeCatalyst 워크플로 작업의 경우 필요한 권한을 가진 IAM 역할을 생성할 수 있습니다. 이 역할은 CodeCatalyst가 Amazon EC2 리소스에서 작업을 실행하는 데 필요한 범위가 지정된 권한이 있는 정책을 사용합니다 AWS 계정. CodeCatalystWorkflowDevelopmentRole-*spaceName* 서비스 역할에는 Amazon EC2에 대한 권한 또는 Amazon CloudWatch에 대해 설명된 작업이 포함되지 않습니다.

이 역할은 다음에 대한 권한을 부여합니다.

- Amazon EC2 인스턴스 상태 가져오기
- Amazon EC2 인스턴스에 대한 CloudWatch 지표를 가져옵니다.

이 역할은 다음과 같은 정책을 사용합니다.

 Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

Amazon ECS에 대한 CodeCatalyst 읽기 전용 역할

CodeCatalyst 워크플로 작업의 경우 필요한 권한을 가진 IAM 역할을 생성할 수 있습니다. 이 역할은 CodeCatalyst가 Amazon ECS 리소스에서 작업을 실행하는 데 필요한 범위가 지정된 권한이 있는 정책을 사용합니다 AWS 계정.

이 역할은 다음에 대한 권한을 부여합니다.

- Amazon ECS 작업 태스크 세트를 읽습니다.
- CloudWatch 경보에 관한 정보를 검색합니다.

이 역할은 다음과 같은 정책을 사용합니다.

 Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

Lambda에 대한 CodeCatalyst 읽기 전용 역할

CodeCatalyst 워크플로 작업의 경우 필요한 권한을 가진 IAM 역할을 생성할 수 있습니다. 이 역할은 CodeCatalyst가 Lambda 리소스에서 작업을 실행하는 데 필요한 범위가 지정된 권한이 있는 정책을 사용합니다 AWS 계정.

이 역할은 다음에 대한 권한을 부여합니다.

- Lambda 함수 및 별칭을 읽습니다.
- Amazon S3 버킷에 있는 개정 파일에 액세스합니다.
- CloudWatch 경보에 관한 정보를 검색합니다.

이 역할은 다음과 같은 정책을 사용합니다.

Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

워크플로 작업에 대한 역할 수동 생성

CodeCatalyst 워크플로 작업은 빌드 역할, 배포 역할 및 스택 역할이라고 하는 사용자가 생성한 IAM 역할을 사용합니다.

IAM에서 역할을 생성하려면 이 단계를 따릅니다.

배포 역할을 생성하려면

1. 역할에 대한 정책을 다음과 같이 생성합니다.
 - a. 에 로그인합니다 AWS.
 - b. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
 - c. 탐색 창에서 Policies를 선택합니다.
 - d. 정책 생성을 선택합니다.
 - e. JSON 탭을 선택합니다.

- f. 기존 코드를 삭제합니다.
- g. 다음 코드를 붙여넣습니다.

 Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

- h. 다음: 태그를 선택합니다.
- i. 다음: 검토를 선택합니다.
- j. 이름에 다음과 같이 입력합니다.

```
codecatalyst-deploy-policy

```

- k. 정책 생성을 선택합니다.

이제 권한 정책을 생성했습니다.

2. 다음과 같이 배포 역할을 생성합니다.

- a. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
- b. 사용자 지정 신뢰 정책을 선택합니다.
- c. 기존 사용자 지정 신뢰 정책을 삭제합니다.
- d. 다음 사용자 지정 신뢰 정책을 추가합니다.
- e. 다음을 선택합니다.
- f. 권한 정책에서 codecatalyst-deploy-policy를 검색하고 해당 확인란을 선택합니다.
- g. 다음을 선택합니다.
- h. 역할 이름에 다음과 같이 입력합니다.

```
codecatalyst-deploy-role

```

- i. 역할 설명에 다음과 같이 입력합니다.

```
CodeCatalyst deploy role

```

j. 역할 생성을 선택합니다.

이제 신뢰 정책 및 권한 정책으로 배포 역할을 생성했습니다.

3. 다음과 같이 배포 역할 ARN을 가져옵니다.

- a. 탐색 창에서 Roles를 선택합니다.
- b. 검색 상자에 방금 생성한 역할의 이름을 입력합니다(codecatalyst-deploy-role).
- c. 목록에서 역할을 선택합니다.

역할의 요약 페이지가 나타납니다.

- d. 상단에서 ARN 값을 복사합니다.

이제 적절한 권한으로 배포 역할을 생성하고 ARN을 획득했습니다.

빌드 역할을 생성하려면

1. 역할에 대한 정책을 다음과 같이 생성합니다.

- a. 에 로그인합니다 AWS.
- b. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
- c. 탐색 창에서 Policies를 선택합니다.
- d. 정책 생성을 선택합니다.
- e. JSON 탭을 선택합니다.
- f. 기존 코드를 삭제합니다.
- g. 다음 코드를 붙여넣습니다.

Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

- h. 다음: 태그를 선택합니다.
- i. 다음: 검토를 선택합니다.

- j. 이름에 다음과 같이 입력합니다.

codecatalyst-build-policy

- k. 정책 생성을 선택합니다.

이제 권한 정책을 생성했습니다.

2. 다음과 같이 빌드 역할을 생성합니다.

- a. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
- b. 사용자 지정 신뢰 정책을 선택합니다.
- c. 기존 사용자 지정 신뢰 정책을 삭제합니다.
- d. 다음 사용자 지정 신뢰 정책을 추가합니다.
- e. 다음을 선택합니다.
- f. 권한 정책에서 **codecatalyst-build-policy**를 검색하고 해당 확인란을 선택합니다.
- g. 다음을 선택합니다.
- h. 역할 이름에 다음과 같이 입력합니다.

codecatalyst-build-role

- i. 역할 설명에 다음과 같이 입력합니다.

CodeCatalyst build role

- j. 역할 생성을 선택합니다.

이제 신뢰 정책 및 권한 정책으로 빌드 역할을 생성했습니다.

3. 다음과 같이 빌드 역할 ARN을 가져옵니다.

- a. 탐색 창에서 Roles를 선택합니다.
- b. 검색 상자에 방금 생성한 역할의 이름을 입력합니다(**codecatalyst-build-role**).
- c. 목록에서 역할을 선택합니다.

역할의 요약 페이지가 나타납니다.

- d. 상단에서 ARN 값을 복사합니다.

이제 적절한 권한으로 빌드 역할을 생성하고 ARN을 획득했습니다.

스택 역할을 생성하려면

Note

스택 역할을 생성할 필요는 없지만 보안상의 이유로 그렇게 하는 것이 좋습니다. 스택 역할을 생성하지 않는 경우 이 절차에서 자세히 설명한 권한 정책을 배포 역할에 추가해야 합니다.

1. 스택을 배포하려는 계정을 AWS 사용하여 로그인합니다.
2. IAM 콘솔(<https://console.aws.amazon.com/iam/>)을 엽니다.
3. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
4. 상단에서 AWS 서비스를 선택합니다.
5. 서비스 목록에서 CloudFormation을 선택합니다.
6. 다음: 권한을 선택합니다.
7. 검색 상자에 스택의 리소스에 액세스하는 데 필요한 정책을 추가합니다. 예를 들어 스택에 AWS Lambda 함수가 포함된 경우 Lambda에 대한 액세스 권한을 부여하는 정책을 추가해야 합니다.

Tip

추가할 정책이 확실하지 않은 경우 지금은 생략할 수 있습니다. 작업을 테스트할 때 올바른 권한이 없는 경우는 추가해야 하는 권한을 표시하는 오류를 CloudFormation 생성합니다.

8. 다음: 태그를 선택합니다.
9. 다음: 검토를 선택합니다.
10. 역할 이름에 다음과 같이 입력합니다.

codecatalyst-stack-role

11. 역할 생성을 선택합니다.
12. 스택 역할의 ARN을 가져오려면 다음을 수행합니다.
 - a. 탐색 창에서 Roles를 선택합니다.

- b. 검색 상자에 방금 생성한 역할의 이름을 입력합니다(codecatalyst-stack-role).
- c. 목록에서 역할을 선택합니다.
- d. 요약 페이지에서 역할 ARN 값을 복사합니다.

AWS CloudFormation 를 사용하여 IAM에서 정책 및 역할 생성

AWS CloudFormation 템플릿을 생성하고 사용하여 CodeCatalyst 프로젝트 및 워크플로에서 리소스 AWS 계정에 액세스하는 데 필요한 정책 및 역할을 생성할 수 있습니다. CloudFormation 는 AWS 리소스를 모델링하고 설정하여 해당 리소스를 관리하는 데 소요되는 시간을 줄이고에서 실행되는 애플리케이션에 더 많은 시간을 할애할 수 있도록 지원하는 서비스입니다 AWS. 여러에서 역할을 생성하려는 경우 템플릿을 AWS 계정생성하면이 작업을 더 빠르게 수행하는 데 도움이 될 수 있습니다.

다음 예시 템플릿은 배포 작업 역할 및 정책을 생성합니다.

```
Parameters:
  CodeCatalystAccountId:
    Type: String
    Description: Account ID from the connections page
  ExternalId:
    Type: String
    Description: External ID from the connections page
Resources:
  CrossAccountRole:
    Type: 'AWS::IAM::Role'
    Properties:
      AssumeRolePolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: Allow
            Principal:
              AWS:
                - !Ref CodeCatalystAccountId
            Action:
              - 'sts:AssumeRole'
            Condition:
              StringEquals:
                sts:ExternalId: !Ref ExternalId
      Path: /
      Policies:
        - PolicyName: CodeCatalyst-CloudFormation-action-policy
          PolicyDocument:
```

```

Version: "2012-10-17"
Statement:
  - Effect: Allow
    Action:
      - 'cloudformation:CreateStack'
      - 'cloudformation>DeleteStack'
      - 'cloudformation:Describe*'
      - 'cloudformation:UpdateStack'
      - 'cloudformation:CreateChangeSet'
      - 'cloudformation>DeleteChangeSet'
      - 'cloudformation:ExecuteChangeSet'
      - 'cloudformation:SetStackPolicy'
      - 'cloudformation:ValidateTemplate'
      - 'cloudformation:List*'
      - 'iam:PassRole'
    Resource: '*'

```

웹 애플리케이션 블루프린트에 대한 역할 수동 생성

CodeCatalyst 웹 애플리케이션 블루프린트는 CDK의 빌드 역할, 배포 역할 및 스택 역할이라고 하는 사용자가 생성한 IAM 역할을 사용합니다.

IAM에서 역할을 생성하려면 이 단계를 따릅니다.

빌드 역할을 생성하려면

1. 역할에 대한 정책을 다음과 같이 생성합니다.
 - a. 에 로그인합니다 AWS.
 - b. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
 - c. 탐색 창에서 Policies를 선택합니다.
 - d. 정책 생성을 선택하세요.
 - e. JSON 탭을 선택합니다.
 - f. 기존 코드를 삭제합니다.
 - g. 다음 코드를 붙여넣습니다.

Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

- h. 다음: 태그를 선택합니다.
- i. 다음: 검토를 선택합니다.
- j. 이름에 다음과 같이 입력합니다.

```
codecatalyst-webapp-build-policy

```

- k. 정책 생성을 선택합니다.

이제 권한 정책을 생성했습니다.

2. 다음과 같이 빌드 역할을 생성합니다.

- a. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
- b. 사용자 지정 신뢰 정책을 선택합니다.
- c. 기존 사용자 지정 신뢰 정책을 삭제합니다.
- d. 다음 사용자 지정 신뢰 정책을 추가합니다.
- e. 다음을 선택합니다.
- f. 빌드 역할에 권한 정책을 연결합니다. 권한 추가 페이지의 권한 정책 섹션에서 `codecatalyst-webapp-build-policy`를 검색하고 해당 확인란을 선택합니다.
- g. 다음을 선택합니다.
- h. 역할 이름에 다음과 같이 입력합니다.

```
codecatalyst-webapp-build-role

```

- i. 역할 설명에 다음과 같이 입력합니다.

```
CodeCatalyst Web app build role

```

- j. 역할 생성을 선택합니다.

이제 신뢰 정책 및 권한 정책으로 빌드 역할을 생성했습니다.

3. 빌드 역할에 다음과 같이 권한 정책을 연결합니다.

- a. 탐색 창에서 역할을 선택한 후 `codecatalyst-webapp-build-role`을 검색합니다.

- b. 세부 정보를 표시하려면 `codecatalyst-webapp-build-role`을 선택합니다.
- c. 권한 탭에서 권한 추가, 정책 연결을 차례로 선택합니다.
- d. `codecatalyst-webapp-build-policy`을 검색하고 해당 확인란을 선택한 다음, 정책 연결을 선택합니다.

이제 빌드 역할에 권한 정책을 연결했습니다. 빌드 역할에는 권한 정책과 신뢰 정책이라는 2가지 정책이 있습니다.

4. 다음과 같이 빌드 역할 ARN을 가져옵니다.

- a. 탐색 창에서 Roles를 선택합니다.
- b. 검색 상자에 방금 생성한 역할의 이름을 입력합니다(`codecatalyst-webapp-build-role`).
- c. 목록에서 역할을 선택합니다.

역할의 요약 페이지가 나타납니다.

- d. 상단에서 ARN 값을 복사합니다.

이제 적절한 권한으로 빌드 역할을 생성하고 ARN을 획득했습니다.

SAM 블루프린트에 대한 역할 수동 생성

CodeCatalyst SAM 블루프린트는 CloudFormation 빌드 역할 및 SAM 배포 역할이라고 하는 사용자가 생성한 IAM 역할을 사용합니다.

IAM에서 역할을 생성하려면 이 단계를 따릅니다.

CloudFormation에 대한 빌드 역할을 생성하려면

1. 역할에 대한 정책을 다음과 같이 생성합니다.
 - a. 에 로그인합니다 AWS.
 - b. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
 - c. 탐색 창에서 Policies를 선택합니다.
 - d. 정책 생성을 선택하세요.
 - e. JSON 탭을 선택합니다.
 - f. 기존 코드를 삭제합니다.

- g. 다음 코드를 붙여넣습니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:*",
        "cloudformation:*"
      ],
      "Resource": "*"
    }
  ]
}
```

 Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

- h. 다음: 태그를 선택합니다.
- i. 다음: 검토를 선택합니다.
- j. 이름에 다음과 같이 입력합니다.

codecatalyst-SAM-build-policy

- k. 정책 생성을 선택합니다.

이제 권한 정책을 생성했습니다.

2. 다음과 같이 빌드 역할을 생성합니다.

- a. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.

- b. 사용자 지정 신뢰 정책을 선택합니다.
- c. 기존 사용자 지정 신뢰 정책을 삭제합니다.
- d. 다음 사용자 지정 신뢰 정책을 추가합니다.
- e. 다음을 선택합니다.
- f. 빌드 역할에 권한 정책을 연결합니다. 권한 추가 페이지의 권한 정책 섹션에서 `codecatalyst-SAM-build-policy`를 검색하고 해당 확인란을 선택합니다.
- g. 다음을 선택합니다.
- h. 역할 이름에 다음과 같이 입력합니다.

codecatalyst-SAM-build-role

- i. 역할 설명에 다음과 같이 입력합니다.

CodeCatalyst SAM build role

- j. 역할 생성을 선택합니다.

이제 신뢰 정책 및 권한 정책으로 빌드 역할을 생성했습니다.

3. 빌드 역할에 다음과 같이 권한 정책을 연결합니다.

- a. 탐색 창에서 역할을 선택한 후 `codecatalyst-SAM-build-role`을 검색합니다.
- b. 세부 정보를 표시하려면 `codecatalyst-SAM-build-role`을 선택합니다.
- c. 권한 탭에서 권한 추가, 정책 연결을 차례로 선택합니다.
- d. `codecatalyst-SAM-build-policy`를 검색하고 해당 확인란을 선택한 다음, 정책 연결을 선택합니다.

이제 빌드 역할에 권한 정책을 연결했습니다. 빌드 역할에는 권한 정책과 신뢰 정책이라는 2가지 정책이 있습니다.

4. 다음과 같이 빌드 역할 ARN을 가져옵니다.

- a. 탐색 창에서 Roles를 선택합니다.
- b. 검색 상자에 방금 생성한 역할의 이름을 입력합니다(`codecatalyst-SAM-build-role`).
- c. 목록에서 역할을 선택합니다.

역할의 요약 페이지가 나타납니다.

- d. 상단에서 ARN 값을 복사합니다.

이제 적절한 권한으로 빌드 역할을 생성하고 ARN을 획득했습니다.

SAM에 대한 배포 역할을 생성하려면

1. 역할에 대한 정책을 다음과 같이 생성합니다.
 - a. 에 로그인합니다 AWS.
 - b. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
 - c. 탐색 창에서 Policies를 선택합니다.
 - d. 정책 생성을 선택하세요.
 - e. JSON 탭을 선택합니다.
 - f. 기존 코드를 삭제합니다.
 - g. 다음 코드를 붙여넣습니다.

 Note

역할을 처음 사용하여 워크플로 작업을 실행할 때 리소스 정책 문에서 와일드카드를 사용한 다음, 사용 가능한 리소스 이름으로 정책 범위를 좁힙니다.

```
"Resource": "*"

```

- h. 다음: 태그를 선택합니다.
- i. 다음: 검토를 선택합니다.
- j. 이름에 다음과 같이 입력합니다.

```
codecatalyst-SAM-deploy-policy

```

- k. 정책 생성을 선택합니다.

이제 권한 정책을 생성했습니다.

2. 다음과 같이 빌드 역할을 생성합니다.
 - a. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
 - b. 사용자 지정 신뢰 정책을 선택합니다.

- c. 기존 사용자 지정 신뢰 정책을 삭제합니다.
- d. 다음 사용자 지정 신뢰 정책을 추가합니다.
- e. 다음을 선택합니다.
- f. 빌드 역할에 권한 정책을 연결합니다. 권한 추가 페이지의 권한 정책 섹션에서 `codecatalyst-SAM-deploy-policy`를 검색하고 해당 확인란을 선택합니다.
- g. 다음을 선택합니다.
- h. 역할 이름에 다음과 같이 입력합니다.

codecatalyst-SAM-deploy-role

- i. 역할 설명에 다음과 같이 입력합니다.

CodeCatalyst SAM deploy role

- j. 역할 생성을 선택합니다.

이제 신뢰 정책 및 권한 정책으로 빌드 역할을 생성했습니다.

3. 빌드 역할에 다음과 같이 권한 정책을 연결합니다.

- a. 탐색 창에서 역할을 선택한 후 `codecatalyst-SAM-deploy-role`을 검색합니다.
- b. 세부 정보를 표시하려면 `codecatalyst-SAM-deploy-role`을 선택합니다.
- c. 권한 탭에서 권한 추가, 정책 연결을 차례로 선택합니다.
- d. `codecatalyst-SAM-deploy-policy`을 검색하고 해당 확인란을 선택한 다음, 정책 연결을 선택합니다.

이제 빌드 역할에 권한 정책을 연결했습니다. 빌드 역할에는 권한 정책과 신뢰 정책이라는 2가지 정책이 있습니다.

4. 다음과 같이 빌드 역할 ARN을 가져옵니다.

- a. 탐색 창에서 Roles를 선택합니다.
- b. 검색 상자에 방금 생성한 역할의 이름을 입력합니다(`codecatalyst-SAM-deploy-role`).
- c. 목록에서 역할을 선택합니다.

역할의 요약 페이지가 나타납니다.

- d. 상단에서 ARN 값을 복사합니다.

이제 적절한 권한으로 빌드 역할을 생성하고 ARN을 획득했습니다.

Amazon CodeCatalyst에 대한 규정 준수 검증

AWS 서비스가 특정 규정 준수 프로그램의 범위에 포함되는지 알아보려면 [규정 준수 프로그램 제공 범위 내 AWS 서비스](#)를 참조하고 관심 있는 규정 준수 프로그램을 선택하십시오. 일반적인 정보는 [AWS 규정 준수 프로그램](#)을 참조하세요.

AWS Artifact(을)를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 [AWS Artifact에서 보고서 다운로드](#)를 참조하세요.

AWS 서비스 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률 및 규정에 따라 결정됩니다. AWS 서비스 사용 시 규정 준수 책임에 대한 자세한 내용은 [AWS 보안 설명서](#)를 참조하세요.

Amazon CodeCatalyst의 복원력

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. 리전은 물리적으로 분리되고 격리된 다수의 가용 영역을 제공하며, 이러한 영역은 짧은 지연 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크를 통해 연결되어 있습니다. 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 정보는 [AWS 글로벌 인프라](#)를 참조하세요. AWS 리전 전반에 복제되는 CodeCatalyst 데이터에 대한 자세한 내용은 [Amazon CodeCatalyst의 데이터 보호](#) 섹션을 참조하세요.

Amazon CodeCatalyst의 인프라 보안

관리형 서비스인 Amazon CodeCatalyst는 AWS 글로벌 네트워크 보안으로 보호됩니다. AWS 보안 서비스와 AWS의 인프라 보호 방법에 대한 자세한 내용은 [AWS Cloud Security](#)을 참조하세요. 인프라 보안에 대한 모범 사례를 사용하여 AWS 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 [인프라 보호](#)를 참조하세요.

AWS에서 게시한 API 직접 호출을 사용하여 네트워크를 통해 CodeCatalyst에 액세스합니다. 고객은 다음을 지원해야 합니다.

- Transport Layer Security(TLS) TLS 1.2는 필수이며 TLS 1.3을 권장합니다.

- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군 Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

Amazon CodeCatalyst의 구성 및 취약성 분석

구성 및 IT 컨트롤은 AWS와 고객 간의 공동 책임입니다. 자세한 내용은 AWS [공동 책임 모델](#)을 참조하세요.

Amazon CodeCatalyst의 데이터 및 개인 정보 보호

Amazon CodeCatalyst는 사용자의 개인 정보를 중요하게 생각하며, 사용자의 정보 보안을 최우선 과제로 삼고 있습니다. 사용자 정보를 처리하는 방법에 대한 자세한 내용은 [AWS 개인정보 취급방침](#)에서 확인할 수 있습니다.

데이터를 요청하고 확인하려면 AWS 일반 참조의 [데이터 요청](#)을 참조하세요.

AWS Builder ID 프로파일 삭제

프로파일 삭제는 되돌릴 수 없는 영구적인 조치입니다. 삭제 프로세스는 삭제를 선택한 직후 시작됩니다. Amazon CodeCatalyst가 프로파일 및 모든 관련 개인 정보를 삭제하기 시작합니다. 이 프로세스를 완료하는 데 최대 90일이 소요될 수 있습니다.

프로파일이 삭제되면 Amazon CodeCatalyst에서 데이터에 액세스하거나 복구할 수 없습니다. 여기에는 개인 액세스 토큰, 역할, 사용자 멤버십 및 귀하가 유일한 멤버인 모든 Amazon CodeCatalyst 스페이스가 포함됩니다. 더 이상 Amazon CodeCatalyst에 로그인할 수 없습니다.

AWS Builder ID 프로파일을 삭제하는 방법에 대한 자세한 내용은 AWS 일반 참조의 [AWS Builder ID 삭제](#)를 참조하세요.

Amazon CodeCatalyst의 워크플로 작업에 대한 모범 사례

CodeCatalyst에서 워크플로를 개발할 때 고려해야 할 여러 보안 모범 사례가 있습니다. 다음 모범 사례는 일반적인 지침이며 완벽한 보안 솔루션을 나타내지는 않습니다. 이러한 모범 사례는 환경에 적절하지 않거나 충분하지 않을 수 있으므로 참고용으로만 사용하세요.

주제

- [민감한 정보](#)
- [라이선스 조건](#)

- [신뢰할 수 없는 코드](#)
- [GitHub Actions](#)

민감한 정보

YAML에 민감한 정보를 포함시키지 마세요. YAML에 자격 증명, 키 또는 토큰을 임베딩하는 대신 CodeCatalyst 보안 암호를 사용하는 것이 좋습니다. 보안 암호는 YAML 내에서 민감한 정보를 쉽게 저장하고 참조할 수 있는 방법을 제공합니다.

라이선스 조건

사용하려는 작업의 라이선스 조건에 유의해야 합니다.

신뢰할 수 없는 코드

작업은 일반적으로 프로젝트, 스페이스 또는 더 광범위한 커뮤니티에서 공유할 수 있는 독립형 단일 목적 모듈입니다. 다른 사람의 코드를 사용하면 편리함과 효율성이 크게 향상될 수 있지만 새로운 위협 벡터도 도입됩니다. 다음 섹션을 검토하여 CI/CD 워크플로를 안전하게 유지하기 위한 모범 사례를 따르고 있는지 확인하세요.

GitHub Actions

GitHub Actions는 커뮤니티에서 구축하고 유지 관리하는 오픈 소스입니다. 당사는 [공동 책임 모델](#)을 따르며 GitHub Actions 소스 코드를 사용자에게 책임이 있는 고객 데이터로 간주합니다. GitHub Actions에는 보안 암호, 리포지토리 토큰, 소스 코드, 계정 링크 및 컴퓨팅 시간에 대한 액세스 권한을 부여할 수 있습니다. 실행하려는 GitHub Actions의 신뢰성과 보안에 확신이 있어야 합니다.

GitHub Actions에 대한 보다 구체적인 지침 및 보안 모범 사례:

- [보안 강화](#)
- [pwn 요청 방지](#)
- [신뢰할 수 없는 입력](#)
- [구성 요소를 신뢰하는 방법](#)

CodeCatalyst 신뢰 모델 이해

Amazon CodeCatalyst 트러스트 모델을 사용하면 CodeCatalyst가 연결된 AWS 계정에서 서비스 역할을 맡을 수 있습니다. 이 모델은 IAM 역할, CodeCatalyst 서비스 위탁자, CodeCatalyst 스페이스를 연결합니다. 신뢰 정책은 `aws:SourceArn` 조건 키를 사용하여 조건 키에 지정된 CodeCatalyst 스페이

스에 권한을 부여합니다. 이 조건 키에 대한 자세한 내용은 IAM 사용 설명서의 [aws:SourceArn](#)을 참조하세요.

신뢰 정책은 신뢰하는 위탁자가 역할을 맡도록 정의하는 JSON 정책 문서입니다. 역할 신뢰 정책은 IAM의 역할에 연결된 필수 리소스 기반 정책입니다. 자세한 내용은 IAM 사용 설명서의 [용어 및 개념](#)을 참조하세요. CodeCatalyst의 서비스 위탁자에 대한 자세한 내용은 [CodeCatalyst의 서비스 위탁자](#) 섹션을 참조하세요.

다음 신뢰 정책에서 Principal 요소에 나열된 서비스 위탁자는 리소스 기반 정책에서 권한을 부여받으며, Condition 블록은 범위가 지정된 리소스에 대한 액세스를 제한하는 데 사용됩니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "codecatalyst-runner.amazonaws.com",
          "codecatalyst.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:codecatalyst:::space/spaceId/project/*"
        }
      }
    }
  ]
}
```

신뢰 정책에서 CodeCatalyst 서비스 위탁자는 aws:SourceArn 조건 키를 통해 액세스 권한을 부여받으며, 여기에는 CodeCatalyst 스페이스 ID에 대한 Amazon 리소스 이름(ARN)이 포함되어 있습니다. ARN은 다음 형식을 사용합니다.

```
arn:aws:codecatalyst:::space/spaceId/project/*
```


⚠ Important

aws:SourceArn과 같은 조건 키에서만 스페이스 ID를 사용합니다. IAM 정책 문서에서 스페이스 ID를 리소스 ARN으로 사용하지 마세요.

모범 사례로 정책에서 가능한 한 권한 범위를 낮추는 것이 좋습니다.

- aws:SourceArn 조건 키에서 와일드카드(*)를 사용하여 project/*를 사용하여 스페이스의 모든 프로젝트를 지정할 수 있습니다.
- project/*projectId*를 사용하여 스페이스의 특정 프로젝트에 대한 aws:SourceArn 조건 키에 리소스 수준 권한을 지정할 수 있습니다.

CodeCatalyst의 서비스 위탁자

리소스 기반 JSON 정책의 Principal 요소를 사용하여 리소스에 대한 액세스가 허용되거나 거부되는 위탁자를 지정합니다. 신뢰 정책에서 지정할 수 있는 위탁자에는 사용자, 역할, 계정 및 서비스가 포함됩니다. ID 기반 정책에서는 Principal 요소를 사용할 수 없으며, 마찬가지로 그룹은 인증이 아니라 권한과 관련이 있고 위탁자는 인증된 IAM 엔터티이므로 정책(예: 리소스 기반 정책)에서 사용자 그룹을 위탁자로 식별할 수 없습니다.

신뢰 정책에서 리소스 기반 정책의 Principal 요소 또는 보안 주체 AWS 서비스 를 지원하는 조건 키를 지정할 수 있습니다. 서비스 위탁자는 서비스가 정의합니다. 다음은 CodeCatalyst 에 정의된 서비스 위탁자입니다.

- codecatalyst.amazonaws.com -이 서비스 보안 주체는 CodeCatalyst 액세스 권한을 부여할 역할에 사용됩니다 AWS.
- codecatalyst-runner.amazonaws.com -이 서비스 보안 주체는 CodeCatalyst 워크플로 배포의 AWS 리소스에 대한 CodeCatalyst 액세스 권한을 부여하는 역할에 사용됩니다.

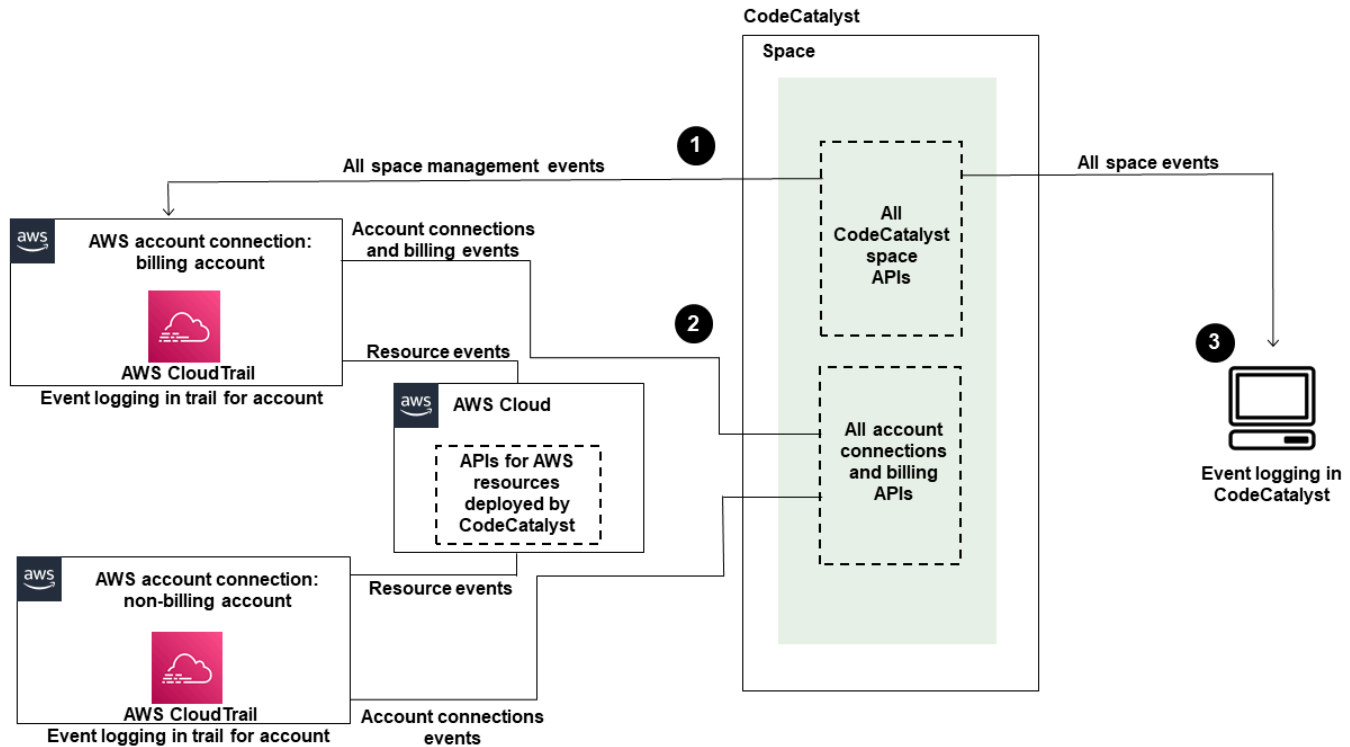
자세한 내용은 IAM 사용 설명서의 [AWS JSON 정책 요소: 위탁자](#)를 참조하세요.

로그를 사용하여 이벤트 및 API 직접 호출 모니터링

Amazon CodeCatalyst에서 스페이스에 대한 관리 이벤트는에서 수집 AWS CloudTrail 되며 스페이스에 대한 결제 계정의 추적에 로깅됩니다. CloudTrail 로깅은 CodeCatalyst 이벤트에 대한 로깅을 관리하는 기본 방법이며, 보조적인 방법으로 CodeCatalyst에서 이벤트 로깅을 보는 방법이 있습니다.

계정의 이벤트는 추적 및 AWS 계정에 대해 설정된 지정된 버킷과 함께 기록됩니다.

다음 다이어그램은 스페이스의 모든 관리 이벤트가 결제 계정의 CloudTrail에 기록되는 방법과 계정 연결/결제 이벤트 및 AWS 리소스 이벤트가 각 계정의 CloudTrail에 기록되는 방법을 보여줍니다.



다이어그램은 다음 단계들을 보여줍니다.

1. 스페이스가 생성되면 AWS 계정 가 스페이스에 연결되고 결제 계정으로 지정됩니다. 사용된 추적은 청구 계정에 대해 CloudTrail에서 생성된 추적으로 스페이스 이벤트가 기록됩니다. CloudTrail은 CodeCatalyst 스페이스가 직접 수행하거나 대신하여 수행된 API 직접 호출 및 관련 이벤트를 캡처하고, 사용자가 지정한 S3 버킷에 로그 파일을 전송합니다. 청구 계정이 다른 AWS 계정으로 변경되면 해당 계정의 추적 및 버킷에 스페이스 이벤트가 기록됩니다. CloudTrail에서 로깅하는 CodeCatalyst 관리 이벤트에 대한 자세한 내용은 [CloudTrail의 CodeCatalyst 정보](#) 섹션을 참조하세요.
2. 청구 계정을 포함하여 스페이스에 연결된 다른 계정은 계정 연결 및 청구 이벤트에 대한 이벤트 하위 집합을 로깅합니다. 해당 계정에 배포된 AWS 리소스에 대한 계정 이벤트를 생성하는 CodeCatalyst 워크플로도의 추적 및 버킷에 로깅됩니다 AWS 계정. CloudTrail은 CodeCatalyst 스페이스가 직접 수행하거나 대신하여 수행된 API 직접 호출 및 관련 이벤트를 캡처하고, 사용자가 지정한 S3 버킷에 로그 파일을 전송합니다. CloudTrail에서 로깅하는 CodeCatalyst 관리 이벤트에 대한 자세한 내용은 [이벤트 로깅을 사용하여 로깅된 이벤트에 액세스](#) 섹션을 참조하세요.

3. AWS CLI로 [list-event-logs](#) 명령을 사용하여 스페이스에서 특정 시간 내에 스페이스의 CodeCatalyst 작업을 모니터링할 수도 있습니다. 자세한 내용은 [Amazon CodeCatalyst API 참조 안내서](#)를 참조하세요. 스페이스 내 CodeCatalyst 작업에 대한 이벤트 목록을 직접적으로 호출하려면 스페이스 관리자 역할이 있어야 합니다. 자세한 내용은 [이벤트 로깅을 사용하여 로깅된 이벤트에 액세스](#) 단원을 참조하십시오.

 Note

ListEventLogs는 지정된 스페이스에서 지난 30일 동안의 이벤트를 보장합니다. 또한 이벤트 기록을 보거나, 이벤트 기록을 지난 90일까지 확장하여 생성하고 유지 관리하기 위한 추적을 생성하여, AWS CloudTrail 콘솔에서 CodeCatalyst에 대한 지난 90일 동안의 관리 이벤트 목록을 보고 검색할 수 있습니다. 자세한 설명은 [CloudTrail 이벤트 기록 작업](#) 및 [CloudTrail 추적 작업](#)을 참조하세요.

 Note

AWS CodeCatalyst 워크플로를 위해 연결된 계정에 배포된 리소스는 CodeCatalyst 스페이스에 대한 CloudTrail 로깅의 일부로 로깅되지 않습니다. 예를 들어 CodeCatalyst 리소스에는 스페이스 또는 프로젝트가 포함됩니다. AWS 리소스에는 Amazon ECS 서비스 또는 Lambda 함수가 포함됩니다. 리소스가 배포 AWS 계정 되는 각에 대해 별도로 CloudTrail 로깅을 구성해야 합니다.

다음은 CodeCatalyst에서 이벤트 모니터링을 할 때 가능한 흐름 중 하나입니다.

Mary Major는 CodeCatalyst 스페이스의 스페이스 관리자이며 CloudTrail에 기록된 스페이스의 스페이스 수준 및 프로젝트 수준 리소스에 대해 CodeCatalyst의 모든 관리 이벤트를 봅니다. CloudTrail에 기록된 이벤트의 예는 [CloudTrail의 CodeCatalyst 정보](#) 섹션을 참조하세요.

개발 환경과 같이 CodeCatalyst에서 생성된 리소스의 경우 Mary는 스페이스의 청구 계정에서 이벤트 기록을 보고 CodeCatalyst에서 프로젝트 멤버가 개발 환경을 생성한 이벤트를 조사합니다. 이벤트는 개발 환경을 생성한 사용자의 AWS Builder ID에 대한 자격 증명 스토어 IAM 자격 증명 유형 및 자격 증명을 제공합니다. 서버리스 배포를 위한 Lambda 함수와 같이 CodeCatalyst의 워크플로에 의해 배포될 AWS 때에서 생성되는 리소스의 경우 소유자는 워크플로 배포 작업에 대한 별도의 AWS 계정 (CodeCatalyst에 연결된 계정이기도 함)과 연결된 추적의 이벤트 기록을 볼 AWS 계정 수 있습니다.

추가 조사를 위해 Mary는 AWS CLI의 [list-event-logs](#) 명령을 사용하여 스페이스의 모든 CodeCatalyst API에 대한 이벤트를 볼 수도 있습니다.

주제

- [AWS CloudTrail 로깅을 사용하여 AWS 계정의 API 직접 호출 모니터링](#)
- [이벤트 로깅을 사용하여 로깅된 이벤트에 액세스](#)

AWS CloudTrail 로깅을 사용하여 AWS 계정의 API 직접 호출 모니터링

Amazon CodeCatalyst는 사용자, 역할 또는 AWS 서비스가 수행한 작업에 대한 레코드를 제공하는 서비스인 AWS CloudTrail과 통합됩니다. CloudTrail은 연결된 AWS 계정에서 이벤트로서 CodeCatalyst를 대신하여 수행된 API 직접 호출을 캡처합니다. 추적을 생성하면 CodeCatalyst 이벤트를 포함한 CloudTrail 이벤트를 지속적으로 S3 버킷에 배포할 수 있습니다. 트레일을 구성하지 않은 경우에도 CloudTrail 콘솔의 이벤트 기록에서 최신 이벤트를 볼 수 있습니다.

CodeCatalyst는 CloudTrail 로그 파일에 다음 작업을 이벤트로 로깅합니다.

- CodeCatalyst 스페이스의 관리 이벤트는 해당 스페이스의 지정된 청구 계정인 AWS 계정에 기록됩니다. 자세한 내용은 [CodeCatalyst 스페이스 이벤트](#) 섹션을 참조하세요.

Note

CodeCatalyst 스페이스에 대한 데이터 이벤트는 [이벤트 로깅을 사용하여 로깅된 이벤트에 액세스](#)에 설명된 대로 CLI를 사용하여 액세스할 수 있습니다.

- 연결된 AWS 계정에서 발생하는 CodeCatalyst 워크플로 작업에 사용되는 리소스의 이벤트는 해당 AWS 계정에서 이벤트로 기록됩니다. 자세한 내용은 [CodeCatalyst 계정 연결 및 청구 이벤트](#) 섹션을 참조하세요.

Important

여러 계정을 스페이스와 연결할 수 있지만 CodeCatalyst 스페이스 및 프로젝트의 이벤트에 대한 CloudTrail 로깅은 청구 계정에만 적용됩니다.

스페이스 청구 계정은 AWS 무료 계층을 벗어난 CodeCatalyst 리소스에 대해 요금이 청구되는 AWS 계정입니다. 여러 계정을 스페이스에 연결할 수 있지만 지정된 청구 계정은 하나뿐입니다. 스페이스에

대한 청구 계정 또는 추가 연결 계정에는 CodeCatalyst 워크플로에서 Amazon ECS 클러스터 또는 S3 버킷과 같은 AWS 리소스 및 인프라를 배포하는 데 사용되는 IAM 역할이 있을 수 있습니다. 워크플로 YAML을 사용하여 배포한 AWS 계정을 식별할 수 있습니다.

Note

CodeCatalyst 워크플로를 위해 연결된 계정에 배포되는 AWS 리소스는 CodeCatalyst 스페이스에 대한 CloudTrail 로깅의 일부로 로깅되지 않습니다. 예를 들어 CodeCatalyst 리소스에는 스페이스 또는 프로젝트가 포함됩니다. AWS 리소스에는 Amazon ECS 서비스 또는 Lambda 함수가 포함됩니다. CloudTrail 로깅은 리소스가 배포되는 각 AWS 계정에 대해 별도로 구성해야 합니다.

연결된 계정의 CodeCatalyst 로깅에는 다음 고려 사항이 포함됩니다.

- CloudTrail 이벤트에 대한 액세스는 CodeCatalyst 아닌 연결된 계정의 IAM으로 관리됩니다.
- GitHub 리포지토리 연결과 같은 타사 연결로 인해 타사 리소스 이름이 CloudTrail 로그에 기록됩니다.

Note

CodeCatalyst 이벤트에 대한 CloudTrail 로깅은 스페이스 수준에 있으며 프로젝트 경계별로 이벤트를 격리하지 않습니다.

CloudTrail에 대한 자세한 내용은 [AWS CloudTrail 사용 설명서](#)를 참조하십시오.

Note

이 섹션에서는 CodeCatalyst 스페이스에 로깅된 모든 이벤트와 CodeCatalyst에 연결된 AWS 계정에 대한 CloudTrail 로깅을 설명합니다. 또한 CodeCatalyst 스페이스에 기록된 모든 이벤트를 검토하려면 AWS CLI 및 `aws codecatalyst list-event-logs` 명령을 사용할 수도 있습니다. 자세한 내용은 [이벤트 로깅을 사용하여 로깅된 이벤트에 액세스](#) 섹션을 참조하세요.

CodeCatalyst 스페이스 이벤트

스페이스 수준 및 프로젝트 수준 리소스를 관리하기 위한 CodeCatalyst의 작업은 스페이스의 청구 계정에 기록됩니다. CodeCatalyst 스페이스에 대한 CloudTrail 로깅의 경우 이벤트는 다음 고려 사항과 함께 로깅됩니다.

- CloudTrail 이벤트는 전체 스페이스에 적용되며 단일 프로젝트에는 적용되지 않습니다.
- AWS 계정을 CodeCatalyst 스페이스에 연결하면 계정 연결에 대한 로깅 가능한 이벤트가 해당 AWS 계정에 기록됩니다. 이 연결을 활성화한 후에는 비활성화할 수 없습니다.
- CodeCatalyst 스페이스에 AWS 계정을 연결하고 해당 스페이스를 청구 계정으로 지정하면 이벤트가 해당 AWS 계정에 기록됩니다. 이 연결을 활성화한 후에는 비활성화할 수 없습니다.

스페이스 수준 및 프로젝트 수준 리소스에 대한 이벤트는 청구 계정에만 기록됩니다. CloudTrail 대상 계정을 변경하려면 CodeCatalyst에서 청구 계정을 업데이트합니다. 다음 월별 청구 주기가 시작되면 CodeCatalyst의 새 청구 계정에 변경 사항이 적용됩니다. 그런 다음 CloudTrail 대상 계정이 업데이트됩니다.

다음은 스페이스 수준 및 프로젝트 수준 리소스를 관리하기 위한 CodeCatalyst의 작업과 관련된 AWS의 이벤트 예시입니다. 다음 APIs는 SDK 및 CLI를 통해 릴리스됩니다. 이벤트는 CodeCatalyst 스페이스의 청구 계정으로 지정된 AWS 계정에 기록됩니다.

- [CreateDevEnvironment](#)
- [CreateProject](#)
- [DeleteDevEnvironment](#)
- [GetDevEnvironment](#)
- [GetProject](#)
- [GetSpace](#)
- [GetSubscription](#)
- [ListDevEnvironments](#)
- [ListDevEnvironmentSessions](#)
- [ListEventLogs](#)
- [ListProjects](#)
- [ListSourceRepositories](#)
- [StartDevEnvironment](#)

- [StartDevEnvironmentSession](#)
- [StopDevEnvironment](#)
- [StopDevEnvironmentSession](#)
- [UpdateDevEnvironment](#)

CodeCatalyst 계정 연결 및 청구 이벤트

다음은 계정 연결 또는 청구에 대한 CodeCatalyst의 작업과 관련된 AWS의 이벤트 예입니다.

- AcceptConnection
- AssociateIAMRoletoConnection
- DeleteConnection
- DissassociateIAMRolefromConnection
- GetBillingAuthorization
- GetConnection
- GetPendingConnection
- ListConnections
- ListIAMRolesforConnection
- PutBillingAuthorization
- RejectConnection

CloudTrail의 CodeCatalyst 정보

CloudTrail을 계정 생성 시 AWS 계정에서 사용할 수 있습니다. 이를 AWS 계정 CodeCatalyst 스페이스에 연결하면 AWS 계정에서 발생하는 해당 스페이스에 대한 이벤트가 해당 AWS 계정에 CloudTrail에 기록됩니다. CodeCatalyst의 로깅 가능 이벤트는 연결된 계정의 CloudTrail 로그 및 CloudTrail 콘솔의 이벤트 기록에서 해당 계정의 다른 로깅 가능 AWS 이벤트와 함께 CloudTrail 이벤트로 기록됩니다.

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 대한 정보가 포함됩니다. 자격 증명을 이용하면 다음을 쉽게 판단할 수 있습니다.

- AWS Builder ID를 가진 사용자가 요청을 했는지 여부.

- 요청을 루트로 했는지 아니면 AWS Identity and Access Management(IAM) 사용자 보안 인증으로 했는지 여부입니다.
- 역할 또는 페더레이션 사용자에게 대한 임시 보안 인증을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지.

자세한 내용은 [CloudTrail userIdentity 요소](#)를 참조하세요.

CloudTrail 이벤트 액세스

AWS 계정의 CodeCatalyst 활동에 대한 이벤트를 포함하여 AWS 계정 계정에 이벤트를 지속적으로 기록하려는 경우 추적을 생성합니다. 추적은 CloudTrail이 S3 버킷으로 로그 파일을 전송할 수 있도록 합니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 리전에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 리전의 이벤트를 로깅하고 지정된 S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 처리하도록 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](#)

추적이란 지정한 S3 버킷에 이벤트를 로그 파일로 입력할 수 있도록 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스의 단일 요청을 나타내며 요청된 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보를 포함합니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출의 주문 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

AWS의 CodeCatalyst 계정 연결 이벤트 예

다음은 ListConnections 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다. 스페이스에 연결된 AWS 계정의 경우 ListConnections는 이 AWS 계정의 CodeCatalyst에 대한 모든 계정 연결을 보는 데 사용됩니다. 이벤트는 accountId에 지정된 AWS 계정에 기록되며 arn의 값은 작업에 사용되는 역할의 Amazon 리소스 이름(ARN)입니다.

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
```



```

    "principalId": "AKIAI44QH8DHBEXAMPLE",
    "arn": "role-ARN",
    "accountId": "account-ID",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "role-ARN",
        "accountId": "account-ID",
        "userName": "user-name"
      },
      "webIdFederationData": {},
      "attributes": {
        "creationDate": "2022-09-06T15:04:31Z",
        "mfaAuthenticated": "false"
      }
    },
    "eventTime": "2022-09-06T15:08:43Z",
    "eventSource": "account-ID",
    "eventName": "ListConnections",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.168.0.1",
    "userAgent": "aws-cli/1.18.147 Python/2.7.18 Linux/5.4.207-126.363.amzn2int.x86_64
botocore/1.18.6",
    "requestParameters": null,
    "responseElements": null,
    "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 ",
    "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 ",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "account-ID",
    "eventCategory": "Management"
  }
}

```

AWS의 CodeCatalyst 프로젝트 리소스 이벤트 예시

다음은 CreateDevEnvironment 작업을 설명하는 CloudTrail 로그 항목을 보여 주는 예시입니다. 스페이스에 연결되어 있고 스페이스의 지정된 청구 계정인 AWS 계정은 개발 환경 생성과 같은 스페이스의 프로젝트 수준 이벤트에 사용됩니다.

`userIdentity`의 `accountId` 필드에서 모든 AWS Builder ID에 대한 ID 풀을 호스팅하는 IAM Identity Center 계정 ID(432677196278)입니다. 이 계정 ID에는 이벤트의 CodeCatalyst 사용자에 대한 다음 정보가 포함되어 있습니다.

- `type` 필드는 요청에 대한 IAM 엔터티의 유형을 나타냅니다. 스페이스 및 프로젝트 리소스에 대한 CodeCatalyst 이벤트의 경우 이 값은 `IdentityCenterUser`입니다. `accountId` 필드는 자격 증명을 가져오기 위해 사용되는 개체를 소유한 계정을 지정합니다.
- `userId` 필드에는 사용자의 AWS Builder ID 식별자가 포함되어 있습니다.
- `identityStoreArn` 필드에는 Identity Store 계정 및 사용자의 역할 ARN이 포함됩니다.

`recipientAccountId` 필드에는 스페이스에 대한 청구 계정의 계정 ID가 포함되며, 여기에 예시 값은 111122223333입니다.

자세한 내용은 [CloudTrail userIdentity 요소](#)를 참조하세요.

```
{
  "eventVersion": "1.09",
  "userIdentity": {
    "type": "IdentityCenterUser",
    "accountId": "432677196278",
    "onBehalfOf": {
      "userId": "user-ID",
      "identityStoreArn": "arn:aws:identitystore::432677196278:identitystore/d-9067642ac7"
    },
    "credentialId": "ABCDDefGhiJKLMn11Lmn_1AbCDEFGHijk-AaBCdEFGHIjKLmnOPqrs11abEXAMPLE"
  },
  "eventTime": "2023-05-18T17:10:50Z",
  "eventSource": "codecatalyst.amazonaws.com",
  "eventName": "CreateDevEnvironment",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "192.168.0.1",
  "userAgent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:102.0) Gecko/20100101 Firefox/102.0",
  "requestParameters": {
    "spaceName": "MySpace",
    "projectName": "MyProject",
    "ides": [{
      "runtime": "public.ecr.aws/q6e8p2q0/cloud9-ide-runtime:2.5.1",
      "name": "Cloud9"
    }],
    "instanceType": "dev.standard1.small",
```

```

    "inactivityTimeoutMinutes": 15,
    "persistentStorage": {
      "sizeInGiB": 16
    }
  },
  "responseElements": {
    "spaceName": "MySpace",
    "projectName": "MyProject",
    "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 "
  },
  "requestID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "eventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "111122223333",
  "sharedEventID": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "eventCategory": "Management"
}

```

Note

경우에 따라 사용자 에이전트를 알 수 없을 수도 있습니다. 이 경우 CodeCatalyst는 CloudTrail 이벤트의 `userAgent` 필드에 `Unknown`의 값을 제공합니다.

CodeCatalyst 이벤트 추적 쿼리

Amazon Athena의 쿼리 테이블을 사용하여 CloudTrail 로그에 대한 쿼리를 생성하고 관리할 수 있습니다. 쿼리 생성에 대한 자세한 내용은 Amazon Athena 사용 설명서의 [AWS CloudTrail 로그 쿼리](#) 섹션을 참조하세요.

이벤트 로깅을 사용하여 로깅된 이벤트에 액세스

사용자가 Amazon CodeCatalyst에서 작업을 수행하면 이러한 작업이 이벤트로 기록됩니다. AWS CLI를 사용하여 지정된 기간 동안 스페이스의 이벤트 로그를 볼 수 있습니다. 이러한 이벤트를 보고 작업의 날짜 및 시간, 작업을 수행한 사용자의 이름, 사용자가 요청한 IP 주소를 포함하여 스페이스에서 수행된 작업을 검토할 수 있습니다.

Note

CodeCatalyst 스페이스의 관리 이벤트는 연결된 청구 계정의 CloudTrail에 기록됩니다. CloudTrail에서 로깅하는 CodeCatalyst 관리 이벤트에 대한 자세한 내용은 [CloudTrail의 CodeCatalyst 정보](#) 섹션을 참조하세요.

스페이스에 대한 이벤트 로그를 보려면 CodeCatalyst의 프로파일로 AWS CLI를 설치하고 구성해야 하며, 스페이스에 대한 스페이스 관리자 역할이 있어야 합니다. 자세한 내용은 [CodeCatalyst AWS CLI 에서를 사용하도록 설정](#) 및 [스페이스 관리자 역할](#) 섹션을 참조하세요.

Note

연결된 AWS 계정에서 CodeCatalyst를 대신하여 발생하는 이벤트에 대한 로깅을 보거나 연결된 청구 계정의 스페이스 또는 프로젝트 리소스에 대한 이벤트에 대한 로깅을 보려면 AWS CloudTrail를 사용할 수 있습니다. 자세한 내용은 [AWS CloudTrail 로깅을 사용하여 AWS 계정의 API 직접 호출 모니터링](#) 섹션을 참조하세요.

1. 터미널 또는 명령줄을 열고 다음을 지정하여 `aws codecatalyst list-event-logs` 명령을 실행합니다.
 - 스페이스의 이름(`--space-name` 옵션 사용).
 - `--start-time` 옵션을 사용하여 [RFC 3339](#)에 지정된 협정 세계시(UTC) 타임스탬프 형식으로 이벤트 검토를 시작하려는 날짜 및 시간입니다.
 - `--end-time` 옵션을 사용하여 [RFC 3339](#)에 지정된 협정 세계시(UTC) 타임스탬프 형식으로 이벤트 검토를 중지하려는 날짜 및 시간입니다.
 - (선택 사항) `--max-results` 옵션을 사용하여 단일 응답으로 반환할 최대 결과 수입니다. 결과 수가 지정한 수보다 크면 응답에 다음 결과를 반환하는 데 사용할 수 있는 `nextToken` 요소가 포함됩니다.
 - (선택 사항) `--event-name` 옵션을 사용하여 반환하려는 특정 이벤트 유형으로 결과를 제한합니다.

이 예시는 *ExampleCorp*이라는 스페이스에 **2022-11-30**부터 **2022-12-01**까지의 로깅된 이벤트를 반환하고 응답에 최대 **2**개의 이벤트를 반환합니다.

```
aws codecatalyst list-event-logs --space-name ExampleCorp --start-time 2022-11-30
--end-time 2022-12-01 --event-name list-event-logs --max-results 2
```

2. 이 기간에 이벤트가 발생한 경우 명령은 다음과 유사한 결과를 반환합니다.

```
{
  "nextToken": "EXAMPLE",
  "items": [
    {
      "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
      "eventName": "listEventLogs",
      "eventType": "AwsApiCall",
      "eventCategory": "MANAGEMENT",
      "eventSource": "manage",
      "eventTime": "2022-12-01T22:47:24.605000+00:00",
      "operationType": "READONLY",
      "userIdentity": {
        "userType": "USER",
        "principalId": "a1b2c3d4e5-678fgh90-1a2b-3c4d-e5f6-EXAMPLE11111"
        "userName": "MaryMajor"
      },
      "requestId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
      "requestPayload": {
        "contentType": "application/json",
        "data": "{\"spaceName\":\"ExampleCorp\",\"startTime\":
\\\"2022-12-01T00:00:00Z\\\",\\\"endTime\\\":\\\"2022-12-10T00:00:00Z\\\",\\\"maxResults\\\":
\\\"2\\\"}"
      },
      "sourceIpAddress": "127.0.0.1",
      "userAgent": "aws-cli/2.9.0 Python/3.9.11 Darwin/21.3.0 exe/x86_64
prompt/off command/codecatalyst.list-event-logs"
    },
    {
      "id": "a1b2c3d4-5678-90ab-cdef-EXAMPLEEaaaaa",
      "eventName": "createProject",
      "eventType": "AwsApiCall",
      "eventCategory": "MANAGEMENT",
      "eventSource": "manage",
      "eventTime": "2022-12-01T09:15:32.068000+00:00",
      "operationType": "MUTATION",
      "userIdentity": {
        "userType": "USER",
```

```

        "principalId": "a1b2c3d4e5-678fgh90-1a2b-3c4d-e5f6-EXAMPLE11111",
        "userName": "MaryMajor"
    },
    "requestId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
    "requestPayload": {
        "contentType": "application/json",
        "data": "{\"spaceName\":\"ExampleCorp\",\"name\":\"MyFirstProject\", \"display\\Name\":\"MyFirstProject\"}"
    },
    "responsePayload": {
        "contentType": "application/json",
        "data": "{\"spaceName\":\"ExampleCorp\",\"name\":\"MyFirstProject\", \"display\\Name\":\"MyFirstProject\", \"id\":\"a1b2c3d4-5678-90ab-cdef-EXAMPLE4444\"}"
    },
    "sourceIpAddress": "192.0.2.23",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:102.0) Gecko/20100101 Firefox/102.0"
    }
}

```

3. --next-token 옵션과 반환된 토큰 값을 사용하여 list-event-logs 명령을 다시 실행하고 요청과 일치하는 다음 로그 이벤트 세트를 검색합니다.

CodeCatalyst의 ID, 권한 및 액세스 할당량

다음 표에서는 Amazon CodeCatalyst의 ID, 권한 및 액세스에 대한 할당량 및 제한에 대해 설명합니다. Amazon CodeCatalyst의 할당량에 대한 자세한 내용은 [CodeCatalyst 할당량](#) 섹션을 참조하세요.

Resource	정보
CodeCatalyst의 별칭	3~100자의 허용되는 문자 조합이며, 반드시 숫자가 아닌 문자로 시작해야 합니다. 유효한 문자: a~z, A~Z, 0~9 별칭은 다음을 수행할 수 없습니다. • 3자 미만으로 이루어진 별칭 • 공백 또는 허용되지 않는 문자(? ^ * [\ ~ :)의 사용

Resource	정보
사용자가 하루에 보낼 수 있는 최대 초대 수	500
하루에 이메일 주소로 전송할 수 있는 최대 초대 수	25
사용자당 최대 개인 액세스 토큰(PAT) 수	100
공급자 유형별로 모든 스페이스에서 각 사용자 ID(CodeCatalyst 별칭)에 대한 최대 개인 연결 수	1
CodeCatalyst의 암호	8~64자의 허용되는 문자 조합이면 됩니다. 유효한 문자: a~z, A~Z, 0~9 암호에는 영숫자가 아닌 문자(~ ! @ # \$ % ^ & * _ - + = ` \ { } [] : ; " ' < > , . ? /) 가 포함될 수 있습니다.
CodeCatalyst의 PAT 이름	1~100자의 허용되는 문자 조합이면 됩니다.
프로젝트 멤버 초대가 만료될 때까지의 시간	24시간 후 만료
스페이스 멤버 초대가 만료될 때까지의 시간	24시간 후 만료
이메일 주소 확인이 만료될 때까지의 시간	전송 후 10분 후 만료

문제 해결

이 섹션은 Amazon CodeCatalyst 프로파일에 액세스할 때 발생할 수 있는 몇 가지 일반적인 문제를 해결하는 데 유용할 수 있습니다.

가입 문제

가입하는 동안 몇 가지 문제가 발생할 수 있습니다. 여기에는 몇 가지 솔루션이 있습니다.

내 이메일 주소가 이미 사용 중입니다.

입력한 이메일이 이미 사용 중이고 해당 이메일을 자신의 이메일로 알고 있다면, 이미 여기에 프로필이 있을 수 있습니다. 이 기존의 ID로 로그인합니다. 기존 이메일을 소유하지 않는 경우, 사용하지 않은 다른 이메일을 사용하여 가입합니다.

이메일 인증을 완료할 수 없습니다

확인 이메일을 받지 못한 경우

1. 스팸, 정크, 삭제된 항목 폴더를 확인하세요.

 Note

이 확인 이메일은 주소 no-reply@signin.aws 또는 no-reply@login.awsapps.com에서 옵니다. 이러한 발신자 이메일 주소에서 오는 이메일을 수신하고 이를 정크 또는 스팸으로 처리하지 않도록 메일 시스템을 구성하는 것이 좋습니다.

2. 5분 동안 기다린 후 받은 편지함을 새로 고칩니다. 스팸, 정크, 삭제된 항목 폴더를 다시 확인하세요.
3. 그래도 확인 이메일이 보이지 않으면, 코드 재전송을 선택합니다. 해당 페이지를 이미 종료한 경우, [Amazon CodeCatalyst](#)에 가입하기 위해 워크플로를 다시 시작합니다.

암호가 최소 요구 사항을 충족하지 않습니다.

보안을 위해 암호에는 대문자와 소문자, 숫자로 구성된 8~20자의 문자가 포함되어야 합니다.

로그인 문제

암호를 잊어버렸습니다

[암호를 잊어버렸습니다](#) 섹션의 단계를 따르세요.

암호가 작동하지 않습니다

암호를 설정하거나 변경할 때마다 다음 요구 사항을 준수해야 합니다.

- 암호는 대/소문자를 구분합니다.

- 암호는 8~64자여야 하며 대문자와 소문자, 숫자, 영숫자가 아닌 문자를 하나 이상 포함해야 합니다.
- 최근 사용한 세 개의 암호는 다시 사용할 수 없습니다.

MFA를 활성화할 수 없습니다.

MFA를 활성화하려면 [다중 인증\(MFA\)으로 로그인하도록 AWS Builder ID 구성](#)의 단계에 따라 프로파일에 MFA 디바이스를 하나 이상 추가하세요.

MFA 디바이스를 추가할 수 없습니다

새 MFA 디바이스를 추가할 수 없는 경우, 등록할 수 있는 MFA 디바이스 수량 한도에 도달했을 수 있습니다. 새 MFA 디바이스를 추가하기 전에 기존 MFA 디바이스를 제거해야 할 수도 있습니다.

MFA 디바이스를 제거할 수 없습니다

MFA를 비활성화하려면 [MFA 디바이스 삭제](#)의 단계에 따라 MFA 디바이스 제거를 진행하세요. 하지만 MFA를 활성화된 상태로 유지하려면 기존 MFA 디바이스 삭제를 시도하기 전에 다른 MFA 디바이스를 추가해야 합니다. MFA 디바이스 추가에 대한 자세한 내용은 [다중 인증에 사용할 디바이스를 등록하는 방법](#) 섹션을 참조하세요.

로그아웃 문제

로그아웃할 위치를 찾을 수 없습니다.

페이지 오른쪽 상단에서 로그아웃을 선택합니다.

로그아웃을 해도 완전히 로그아웃되지 않습니다.

시스템은 즉시 로그아웃하도록 설계되어 있지만 완전 로그아웃에는 최대 1시간이 걸릴 수 있습니다.

실패한 워크플로에서 역할이 존재하지 않는다는 오류가 발생했습니다.

문제: 웹 애플리케이션 또는 서버리스 블루프린트에서 프로젝트를 생성한 후, 워크플로가 실패하고 다음 오류가 발생합니다.

CLIENT_ERROR: 역할이 존재하지 않음

해결 방법: 워크플로를 실행할 수 있는 권한을 가진 IAM 역할을 구성하고 워크플로 YAML에 IAM 역할을 추가한 후에도 워크플로가 여전히 실패하는 이유는 IAM 역할을 계정 연결에 추가해야 할 수 있기 때문입니다. [계정 연결에 IAM 역할 추가](#)에 설명된 대로 스페이스의 계정 연결에 IAM 역할을 추가합니다.

실패한 워크플로에 대한 역할 오류가 발생합니다.

문제: 웹 애플리케이션 또는 서버리스 블루프린트에서 프로젝트를 생성한 후, 워크플로가 실패하고 다음 오류가 발생합니다.

CLIENT_ERROR: 역할이 제대로 설정되지 않았거나 존재하지 않음

가능한 솔루션: 프로젝트가 생성된 공간은 AWS 계정 연결을 설정해야 하거나 계정 연결 요청을 완료해야 할 수 있습니다. 스페이스에 이미 활성 AWS 계정 연결이 있는 경우, 워크플로 작업을 실행할 권한이 있는 IAM 역할을 생성하고 추가합니다. [계정 연결에 IAM 역할 추가](#)에 설명된 대로 계정 연결에 IAM 역할을 추가합니다.

해결 방법: 연결을 지정하지 않고 프로젝트를 생성한 경우, 계정 연결을 배포 환경에 연결해야 합니다. 스페이스에 이미 활성 AWS 계정 연결이 있고 IAM 역할이 추가된 경우에 설명된 대로 배포 환경에 IAM 역할과의 계정 연결을 추가해야 합니다. [배포 환경에 계정 연결 및 IAM 역할 추가](#).

프로젝트 워크플로에서 IAM 역할을 업데이트해야 합니다.

AWS 계정 연결이 완전히 설정되고 IAM 역할이 생성되어 계정 연결에 추가되면 프로젝트 워크플로에서 IAM 역할을 업데이트할 수 있습니다.

1. CI/CD 옵션을 선택하고 워크플로를 선택합니다. YAML 버튼을 선택합니다.
2. 편집을 선택합니다.
3. ActionRoleArn: 필드에서 IAM 역할 ARN을 업데이트된 IAM 역할 ARN으로 변경합니다. 검증 중을 선택합니다.
4. 커밋을 선택합니다.

메인라인 브랜치에 있는 경우 워크플로가 자동으로 시작됩니다. 그렇지 않으면, 워크플로를 다시 실행하려면 실행을 선택합니다.

개인 연결을 생성한 후 GitHub 계정에 대한 검토 요청이 있습니다.

GitHub에 대한 개인 연결을 생성하면 GitHub 앱으로 GitHub 계정에 CodeCatalyst 앱이 설치됩니다. CodeCatalyst에 업데이트된 읽기 또는 쓰기 권한이 필요한 특정 리소스가 있는 경우, GitHub 계정에 액세스하여 설치된 앱의 권한을 업데이트해야 할 수 있습니다.

1. GitHub에 로그인하고 설치된 앱의 계정 설정으로 이동합니다. 프로필 아이콘을 선택하고 설정을 선택한 다음, 애플리케이션을 선택합니다.

2. 설치된 GitHub 앱 탭의 설치된 애플리케이션 목록에서 CodeCatalyst에 설치된 앱을 확인합니다. 검토할 권한이 있는 경우, 검토 요청 링크가 표시됩니다.
3. 해당 링크를 선택한 다음, 메시지가 표시되면 자격 증명을 확인합니다. 자격 증명을 입력하고 확인을 선택합니다.
4. 새 권한을 수락하고 권한을 적용할 리포지토리를 표시한 다음, 저장을 선택합니다.

지원 문의 양식은 어떻게 작성하나요?

[Amazon CodeCatalyst](#)로 이동하거나 [지원 피드백 양식](#)을 작성할 수 있습니다. 요청 정보 섹션에서 어떻게 도와드릴까요에 Amazon CodeCatalyst 고객이라는 점을 포함합니다. 문제를 가장 효율적으로 해결할 수 있도록 최대한 자세하게 설명해 주세요.

CodeCatalyst에서 확장 프로그램이 있는 프로젝트에 기능 추가

Amazon CodeCatalyst에는 기능을 추가하고 CodeCatalyst 외부의 제품과 통합하는 데 도움이 되는 확장 프로그램이 포함되어 있습니다. CodeCatalyst 카탈로그의 확장 프로그램을 통해 팀은 CodeCatalyst에서 경험을 사용자 지정할 수 있습니다.

주제

- [사용 가능한 타사 확장](#)
- [확장 프로그램 개념](#)
- [빠른 시작: CodeCatalyst에서 확장 프로그램 설치, 공급자 연결 및 리소스 연결](#)
- [스페이스에 확장 프로그램 설치](#)
- [스페이스에 확장 프로그램 제거](#)
- [GitHub 계정, Bitbucket 작업 영역, GitLab 사용자 및 Jira 사이트 CodeCatalyst 연결](#)
- [GitHub 계정, Bitbucket 작업 영역, GitLab 사용자 및 Jira 사이트 CodeCatalyst 연결 해제](#)
- [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결](#)
- [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#)
- [CodeCatalyst에서 타사 리포지토리 보기 및 Jira 문제 검색](#)
- [타사 리포지토리 이벤트 후 워크플로 실행 자동 시작](#)
- [타사 리포지토리 공급자를 통한 IP 액세스 제한](#)
- [워크플로 실패 시 타사 병합 차단](#)
- [Jira 문제를 CodeCatalyst 풀 요청에 연결](#)
- [Jira 문제에서 CodeCatalyst 이벤트 보기](#)

사용 가능한 타사 확장

리소스를 통합하도록 선택한 확장 프로그램에 따라 CodeCatalyst 프로젝트에 특정 기능을 추가할 수 있습니다.

CodeCatalyst에 GitHub 리포지토리 통합

GitHub는 개발자가 코드를 저장하고 관리하는 데 도움이 되는 클라우드 기반 서비스입니다. GitHub 리포지토리 확장 프로그램을 사용하면 Amazon CodeCatalyst 프로젝트에서 연결된 GitHub 리포지토리를 사용할 수 있습니다. 새 CodeCatalyst 프로젝트를 생성할 때 GitHub 리포지토리를 연결할 수도 있습니다. 자세한 내용은 [연결된 타사 리포지토리를 사용하여 프로젝트 생성](#) 섹션을 참조하세요.

Note

- 빈 GitHub 리포지토리 또는 아카이브된 GitHub 리포지토리를 CodeCatalyst 프로젝트와 함께 사용할 수 없습니다.
- GitHub 리포지토리 확장 프로그램은 GitHub Enterprise Server 리포지토리와 호환되지 않습니다.

GitHub 리포지토리 확장 프로그램을 설치하고 구성하면 다음을 수행할 수 있습니다.

- CodeCatalyst의 소스 리포지토리 목록에서 GitHub 리포지토리 보기
- GitHub 리포지토리에 워크플로 정의 파일 저장 및 관리
- CodeCatalyst 개발 환경에서 연결된 GitHub 리포지토리에 저장된 파일 생성, 읽기, 업데이트 및 삭제
- CodeCatalyst에서 연결된 GitHub 리포지토리의 파일 저장 및 인덱싱
- 연결된 GitHub 계정의 기존 리포지토리를 사용하여 CodeCatalyst 프로젝트 생성
- 블루프린트로 프로젝트를 생성하거나 블루프린트를 추가할 때 블루프린트로 생성된 코드를 사용하여 GitHub 리포지토리 생성
- 연결된 GitHub 리포지토리로 코드를 푸시하거나 연결된 GitHub 리포지토리에서 풀 요청이 생성, 수정 또는 종료될 때 CodeCatalyst 시작 워크플로가 자동으로 실행됩니다.
- CodeCatalyst 워크플로에서 연결된 GitHub 리포지토리 소스 파일 사용
- CodeCatalyst 워크플로에서 GitHub Actions 읽기 및 실행
- CodeCatalyst 워크플로 실행 상태를 연결된 GitHub 리포지토리로 전송하고 커밋 상태에 따라 GitHub 풀 요청 병합 차단

CodeCatalyst에 Bitbucket 리포지토리 통합

Bitbucket은 개발자가 코드를 저장하고 관리하는 데 도움이 되는 클라우드 기반 서비스입니다. Bitbucket 리포지토리 확장 프로그램을 사용하면 Amazon CodeCatalyst 프로젝트에서 연결된

Bitbucket 리포지토리를 사용할 수 있습니다. 새 CodeCatalyst 프로젝트를 생성할 때 Bitbucket 리포지토리를 연결할 수도 있습니다. 자세한 내용은 [연결된 타사 리포지토리를 사용하여 프로젝트 생성](#) 섹션을 참조하세요.

Note

- 빈 또는 아카이브된 Bitbucket 리포지토리는 CodeCatalyst 프로젝트와 함께 사용할 수 없습니다.
- Bitbucket 리포지토리 확장 프로그램은 Bitbucket Data Center 리포지토리와 호환되지 않습니다.

Bitbucket 리포지토리 확장 프로그램을 설치하고 구성하면 다음을 수행할 수 있습니다.

- CodeCatalyst의 소스 리포지토리 목록에서 Bitbucket 리포지토리 보기
- Bitbucket 리포지토리에 워크플로 정의 파일을 저장하고 관리합니다.
- CodeCatalyst 개발 환경에서 연결된 Bitbucket 리포지토리에 저장된 파일 생성, 읽기, 업데이트 및 삭제
- 연결된 Bitbucket 계정의 기존 리포지토리를 사용하여 CodeCatalyst 프로젝트 생성
- CodeCatalyst의 연결된 Bitbucket 리포지토리에서 파일을 저장하고 인덱싱합니다.
- 블루프린트로 프로젝트를 생성하거나 블루프린트를 추가할 때 블루프린트로 생성된 코드를 사용하여 Bitbucket 리포지토리 생성
- 연결된 Bitbucket 리포지토리에 코드를 푸시하거나 연결된 Bitbucket 리포지토리에서 풀 요청이 생성, 수정 또는 종료될 때 CodeCatalyst 시작 워크플로가 자동으로 실행됩니다.
- CodeCatalyst 워크플로에서 연결된 Bitbucket 리포지토리 소스 파일 사용
- CodeCatalyst 워크플로 실행 상태를 연결된 Bitbucket 리포지토리로 전송하고 커밋 상태에 따라 Bitbucket 풀 요청 병합 차단

CodeCatalyst에 GitLab 리포지토리 통합

GitLab은 개발자가 코드를 저장하고 관리하는 데 도움이 되는 클라우드 기반 서비스입니다. GitLab 리포지토리 확장 프로그램을 사용하면 Amazon CodeCatalyst 프로젝트에서 연결된 GitLab 리포지토리를 사용할 수 있습니다. 새 CodeCatalyst 프로젝트를 생성할 때 GitLab 프로젝트 리포지토리를 연결할 수도 있습니다. 자세한 내용은 [연결된 타사 리포지토리를 사용하여 프로젝트 생성](#) 섹션을 참조하세요.

Note

- 빈 GitLab 프로젝트 리포지토리는 CodeCatalyst 프로젝트와 함께 사용할 수 없습니다.
- GitLab 리포지토리 확장 프로그램은 GitLab 자체 관리형 리포지토리와 호환되지 않습니다.

GitLab 리포지토리 확장 프로그램을 설치하고 구성하면 다음을 수행할 수 있습니다.

- CodeCatalyst의 소스 리포지토리 목록에서 GitLab 프로젝트 리포지토리 보기
- GitLab 프로젝트 리포지토리에 워크플로 정의 파일을 저장하고 관리합니다.
- CodeCatalyst 개발 환경에서 연결된 GitLab 프로젝트 리포지토리에 저장된 파일 생성, 읽기, 업데이트 및 삭제
- 연결된 GitLab 사용자의 기존 리포지토리를 사용하여 CodeCatalyst 프로젝트 생성
- CodeCatalyst의 연결된 GitLab 프로젝트 리포지토리에서 파일을 저장하고 인덱싱합니다.
- 블루프린트로 프로젝트를 생성하거나 블루프린트를 추가할 때 블루프린트로 생성된 코드를 사용하여 GitLab 프로젝트 리포지토리 생성
- 연결된 GitLab 프로젝트 리포지토리에 코드를 푸시하거나 연결된 GitLab 프로젝트 리포지토리에서 풀 요청이 생성, 수정 또는 종료될 때 CodeCatalyst 시작 워크플로가 자동으로 실행됩니다.
- CodeCatalyst 워크플로에서 연결된 GitLab 프로젝트 리포지토리 소스 파일 사용
- CodeCatalyst 워크플로 실행 상태를 연결된 GitLab 프로젝트 리포지토리로 전송하고 커밋 상태에 따라 GitLab 병합 요청 차단

CodeCatalyst에 Jira 문제 통합

Jira는 애자일 개발 팀이 작업을 계획, 할당, 추적, 보고 및 관리할 수 있도록 지원하는 소프트웨어 애플리케이션입니다. Jira 소프트웨어 확장 프로그램을 사용하면 Amazon CodeCatalyst 프로젝트에서 Jira 프로젝트를 사용할 수 있습니다.

Note

CodeCatalyst는 Jira Software Cloud와만 호환됩니다.

Amazon CodeCatalyst 프로젝트에 대한 Jira 소프트웨어 확장 프로그램을 설치하고 구성하면 다음을 수행할 수 있습니다.

- CodeCatalyst 프로젝트와 연결하여 CodeCatalyst에서 Jira 프로젝트에 액세스
- CodeCatalyst 풀 요청으로 Jira 문제 업데이트
- Jira 문제에서 연결된 CodeCatalyst 풀 요청의 상태 및 워크플로 실행 보기

확장 프로그램 개념

다음은 CodeCatalyst에서 확장 프로그램 작업을 수행할 때 알아야 할 몇 가지 개념 및 용어입니다.

확장 프로그램

확장 프로그램은 CodeCatalyst 스페이스에 설치하여 프로젝트에 새 기능을 추가하고 CodeCatalyst 외부의 서비스와 통합할 수 있는 추가 기능입니다. 확장 프로그램은 CodeCatalyst 카탈로그에서 찾아보고 설치할 수 있습니다.

CodeCatalyst 카탈로그

CodeCatalyst 카탈로그는 CodeCatalyst에서 사용할 수 있는 모든 확장 프로그램의 중앙 집중식 목록입니다. CodeCatalyst 카탈로그를 검색하여 소스, 워크플로 등과 같은 CodeCatalyst 영역에서 팀의 경험을 개선할 수 있는 확장 프로그램을 찾을 수 있습니다.

연결 및 링크

사용하거나 관리하려는 타사 리소스에 따라 GitHub 계정, Bitbucket 작업 영역 또는 Jira 프로젝트를 연결해야 합니다. 그런 다음 GitHub 리포지토리, Bitbucket 리포지토리 또는 Jira 프로젝트를 CodeCatalyst 프로젝트에 링크해야 합니다.

- GitHub 리포지토리: GitHub 계정을 연결한 다음 GitHub 리포지토리를 링크합니다.
- Bitbucket 리포지토리: Bitbucket Work 스페이스를 연결한 다음 Bitbucket 리포지토리를 링크합니다.
- GitLab 리포지토리: GitLab 사용자를 연결한 다음 GitLab 프로젝트 리포지토리를 링크합니다.
- Jira 소프트웨어: Jira 사이트를 연결한 다음 Jira 프로젝트를 링크합니다.

빠른 시작: CodeCatalyst에서 확장 프로그램 설치, 공급자 연결 및 리소스 연결

이 자습서에서는 다음 세 가지 작업에 대한 연습을 제공합니다.

1. GitHub 리포지토리, Bitbucket 리포지토리, GitLab 리포지토리 또는 Jira 소프트웨어 확장 프로그램을 설치합니다. 외부 사이트에서 CodeCatalyst에 연결하고 타사 리소스에 대한 액세스 권한을 제공하라는 메시지가 표시되며, 이는 다음 단계의 일부로 수행됩니다.

 Important

GitHub 리포지토리, Bitbucket 리포지토리, GitLab 리포지토리 또는 Jira 소프트웨어 확장 프로그램을 CodeCatalyst 스페이스에 설치하려면 해당 스페이스에 스페이스 관리자 역할이 있는 계정으로 로그인해야 합니다.

2. GitHub 계정, Bitbucket Work스페이스, GitLab 사용자 또는 Jira 사이트를 CodeCatalyst에 연결합니다.

 Important

GitHub 계정, Bitbucket Work 스페이스, GitLab 사용자 또는 Jira 사이트를 CodeCatalyst 스페이스에 연결하려면 타사 소스의 관리자와 CodeCatalyst 스페이스 관리자여야 합니다.

 Important

리포지토리 확장 프로그램을 설치한 후 CodeCatalyst에 연결하는 모든 리포지토리의 코드가 CodeCatalyst에 인덱싱 및 저장됩니다. 이렇게 하면 CodeCatalyst에서 코드를 검색할 수 있습니다. CodeCatalyst에서 연결된 리포지토리를 사용할 때 코드에 대한 데이터 보호를 더 잘 이해하려면 Amazon CodeCatalyst 사용 설명서의 [데이터 보호](#)를 참조하세요.

 Note

GitHub 계정에 대한 연결을 사용하는 경우 개인 연결을 생성하여 CodeCatalyst ID와 GitHub ID 간에 ID 매핑을 설정해야 합니다. 자세한 내용은 [개인 연결](#) 및 [개인 연결을 사용하여 GitHub 리소스에 액세스](#) 섹션을 참조하세요.

3. GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 또는 Jira 프로젝트를 CodeCatalyst 프로젝트에 연결합니다.

⚠ Important

- GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 기고자로 연결할 수 있지만 타사 리포지토리를 스페이스 관리자 또는 프로젝트 관리자로서만 연결 해제할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#) 섹션을 참조하세요.
- Jira 프로젝트를 CodeCatalyst 프로젝트에 연결하려면 CodeCatalyst 스페이스 관리자 또는 CodeCatalyst 프로젝트 관리자여야 합니다.

⚠ Important

CodeCatalyst는 연결된 리포지토리에 대한 기본 브랜치의 변경 사항 감지를 지원하지 않습니다. 연결된 리포지토리의 기본 브랜치를 변경하려면 먼저 CodeCatalyst에서 연결을 해제하고 기본 브랜치를 변경한 다음 다시 연결해야 합니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결](#) 섹션을 참조하세요.

리포지토리를 연결하기 전에 항상 최신 버전의 확장 프로그램을 사용하는 것이 좋습니다.

i Note

- GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리는 한 스페이스에서 하나의 CodeCatalyst 프로젝트에만 연결할 수 있습니다.
- 비어있거나 보관된 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 CodeCatalyst 프로젝트와 함께 사용할 수 없습니다.
- CodeCatalyst 프로젝트의 리포지토리와 이름이 동일한 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 연결할 수 없습니다.
- GitHub 리포지토리 확장 프로그램은 GitHub Enterprise Server 리포지토리와 호환되지 않습니다.
- Bitbucket 리포지토리 확장 프로그램은 Bitbucket Data Center 리포지토리와 호환되지 않습니다.
- GitLab 리포지토리 확장 프로그램은 GitLab 자체 관리형 프로젝트 리포지토리와 호환되지 않습니다.

- 연결된 리포지토리에서는 설명 쓰기 또는 설명 요약 기능을 사용할 수 없습니다. 이러한 기능은 CodeCatalyst의 풀 요청에서만 사용할 수 있습니다.
- CodeCatalyst 프로젝트는 하나의 Jira 프로젝트에만 연결할 수 있습니다. Jira 프로젝트는 여러 CodeCatalyst 프로젝트에 연결할 수 있습니다.

또한 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 리포지토리 확장 프로그램을 설치하고, GitHub 계정, Bitbucket 작업 영역 또는 GitLab 사용자에게 연결하고, 새 CodeCatalyst 프로젝트를 생성할 때 타사 리포지토리를 연결할 수 있습니다. 자세한 내용은 [연결된 타사 리포지토리를 사용하여 프로젝트 생성](#) 섹션을 참조하세요.

주제

- [1단계: CodeCatalyst 카탈로그에서 타사 확장 프로그램 설치](#)
- [2단계: 타사 공급자를 CodeCatalyst 스페이스에 연결](#)
- [3단계: 타사 리소스를 CodeCatalyst 프로젝트에 연결](#)
- [다음 단계](#)

1단계: CodeCatalyst 카탈로그에서 타사 확장 프로그램 설치

CodeCatalyst에서 타사 리소스를 사용하는 첫 번째 단계는 CodeCatalyst 카탈로그에서 GitHub 리포지토리 확장 프로그램을 설치하는 것입니다. 확장 프로그램을 설치하려면 다음 단계를 수행하여 사용하는 타사 리소스의 확장 프로그램을 선택합니다. GitHub 리포지토리, Bitbucket 리포지토리 및 GitLab 리포지토리를 사용하면 CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 사용할 수 있습니다. Jira 소프트웨어를 사용하면 CodeCatalyst에서 Jira 문제를 관리할 수 있습니다.

CodeCatalyst 카탈로그에서 확장 프로그램을 설치하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다.
3. 상단 메뉴에서 카탈로그 아이콘



을

선택하여 CodeCatalyst 카탈로그로 이동합니다. GitHub 리포지토리, Bitbucket 리포지토리, GitLab 리포지토리 또는 Jira 소프트웨어를 검색할 수 있습니다. 범주를 기반으로 확장 프로그램을 필터링할 수도 있습니다.

4. (선택 사항) 확장 프로그램과 관련된 권한 등 확장 프로그램에 대한 자세한 내용을 보려면 확장 이름을 선택합니다.
5. 설치를 선택합니다. 확장 프로그램에 필요한 권한을 검토하고 계속하려면 설치를 다시 선택합니다.

확장 프로그램을 설치하면 확장 프로그램 세부 정보 페이지로 이동합니다. 설치한 확장 프로그램에 따라 연결된 공급자와 연결된 리소스를 보고 관리할 수 있습니다.

2단계: 타사 공급자를 CodeCatalyst 스페이스에 연결

GitHub 리포지토리, Bitbucket 리포지토리, GitLab 리포지토리 또는 Jira 소프트웨어 확장 프로그램을 설치한 후 다음 단계는 GitHub 계정, Bitbucket 작업 영역, GitLab 프로젝트 리포지토리 또는 Jira 사이트를 CodeCatalyst 스페이스에 연결하는 것입니다.

GitHub 계정, Bitbucket Work 스페이스, Jira 사이트를 CodeCatalyst에 연결하려면

- 설치한 타사 확장 프로그램에 따라 다음 중 하나를 수행합니다.
 - GitHub 리포지토리: GitHub 계정에 연결합니다.
 1. 연결된 GitHub 계정 탭에서 GitHub 계정 연결을 선택하여 GitHub의 외부 사이트로 이동합니다.
 2. GitHub 자격 증명으로 GitHub 계정에 로그인한 다음 Amazon CodeCatalyst를 설치할 계정을 선택합니다.

Tip

이전에 GitHub 계정을 스페이스에 연결한 경우 재인증하라는 메시지가 표시되지 않습니다. 대신 두 개 이상의 GitHub 조직의 멤버 또는 공동 작업자인 경우 확장 프로그램을 설치할 위치를 묻는 대화 상자가 표시되고, 하나의 GitHub 조직에만 속한 경우 Amazon CodeCatalyst 애플리케이션의 구성 페이지가 표시됩니다. 허용하려는 리포지토리 액세스에 대한 애플리케이션을 구성한 다음 저장을 선택합니다. 저장 버튼이 활성화되지 않은 경우 구성을 변경한 다음 다시 시도하세요.

3. CodeCatalyst가 현재 및 미래 리포지토리에 모두 액세스하도록 허용할지 여부를 선택하거나 CodeCatalyst에서 사용할 특정 GitHub 리포지토리를 선택합니다. 기본 옵션은 CodeCatalyst에서 액세스할 향후 리포지토리를 포함하여 GitHub 계정에 모든 GitHub 리포지토리를 포함하는 것입니다.

4. CodeCatalyst에 부여된 권한을 검토한 다음 설치를 선택합니다.

GitHub 계정을 CodeCatalyst에 연결하면 GitHub 리포지토리 확장 프로그램 세부 정보 페이지로 이동합니다. 여기서 연결된 GitHub 계정과 연결된 GitHub 리포지토리를 보고 관리할 수 있습니다.

- Bitbucket 리포지토리: Bitbucket 작업 영역에 연결합니다.
 1. 연결된 Bitbucket 작업 영역 탭에서 Bitbucket 작업 영역 연결을 선택하여 Bitbucket의 외부 사이트로 이동합니다.
 2. Bitbucket 자격 증명을 사용하여 Bitbucket 작업 영역에 로그인하고 CodeCatalyst에 부여된 권한을 검토합니다.
 3. 작업 영역에 대한 권한 부여 드롭다운 메뉴에서 CodeCatalyst 액세스를 제공할 Bitbucket 작업 영역을 선택한 다음 액세스 권한 부여를 선택합니다.

Tip

이전에 Bitbucket 작업 영역을 스페이스에 연결한 경우 재인증하라는 메시지가 표시되지 않습니다. 대신 하나 이상의 Bitbucket 작업 영역에서 멤버 또는 공동 작업자인 경우 확장 프로그램을 설치할 위치를 묻는 대화 상자가 표시되고, 하나의 Bitbucket 작업 영역에만 속한 경우 Amazon CodeCatalyst 애플리케이션의 구성 페이지가 표시됩니다. 작업 영역 액세스를 허용하려는 애플리케이션을 선택하고, 액세스 허용을 선택합니다. 액세스 허용 버튼이 활성화되지 않은 경우 구성을 변경한 다음 다시 시도하세요.

Bitbucket 워크스페이스를 CodeCatalyst에 연결하면 연결된 Bitbucket 워크스페이스와 연결된 Bitbucket 리포지토리를 보고 관리할 수 있는 Bitbucket 리포지토리 확장 프로그램 세부 정보 페이지로 이동합니다.

- GitLab 리포지토리: GitLab 사용자에게 연결합니다.
 1. GitLab 사용자 연결을 선택하여 GitLab 외부 사이트로 이동합니다.
 2. GitLab 자격 증명을 사용하여 GitLab 사용자에게 로그인하고 CodeCatalyst에 부여된 권한을 검토합니다.

i Tip

이전에 GitLab 사용자를 스페이스에 연결한 경우 재인증하라는 메시지가 표시되지 않습니다. 대신 CodeCatalyst 콘솔로 다시 이동합니다.

3. GitLab용 AWS 커넥터 권한 부여를 선택합니다.

GitLab 사용자를 CodeCatalyst에 연결하면 GitLab 리포지토리 확장 프로그램 세부 정보 페이지로 이동하며, 여기서 연결된 GitLab 사용자 및 연결된 GitLab 프로젝트 리포지토리를 보고 관리할 수 있습니다.

- Jira 소프트웨어: Jira 사이트를 연결합니다.

1. 연결된 Jira 사이트 탭에서 Jira 사이트 연결을 선택하여 Atlassian Marketplace의 외부 사이트로 이동합니다.
2. 지금 받기를 선택하여 Jira 사이트에 CodeCatalyst 설치를 시작합니다.

i Note

이전에 Jira 사이트에 CodeCatalyst를 설치한 경우 알림을 받게 됩니다. 시작하기를 선택하여 최종 단계로 이동합니다.

3. 역할에 따라 다음 중 하나를 수행합니다.

1. Jira 사이트 관리자인 경우 사이트 드롭다운 메뉴에서 CodeCatalyst 애플리케이션을 설치할 Jira 사이트를 선택한 다음 앱 설치를 선택합니다.

i Note

Jira 사이트가 하나 있는 경우 이 단계는 표시되지 않으며 자동으로 다음 단계로 이동합니다.

2. a. Jira 관리자가 아닌 경우 사이트 드롭다운 메뉴에서 Jira 사이트를 선택하여 CodeCatalyst 애플리케이션을 설치한 다음 앱 요청을 선택합니다. Jira 앱 설치에 대한 자세한 내용은 [앱을 설치할 수 있는 사람은 누구입니까?](#)를 참조하세요.
b. CodeCatalyst를 설치해야 하는 이유를 입력 텍스트 필드에 입력하거나 기본 텍스트를 유지한 다음 요청 제출을 선택합니다.

4. 애플리케이션이 설치될 때 CodeCatalyst에서 수행하는 작업을 검토한 다음 지금 받기를 선택합니다.
5. 애플리케이션을 설치한 후 CodeCatalyst로 돌아가기를 선택하여 CodeCatalyst로 돌아갑니다.

Jira 사이트를 CodeCatalyst 연결한 후 Jira 소프트웨어 확장 프로그램 세부 정보 페이지의 연결된 Jira 사이트 탭에서 연결된 사이트를 볼 수 있습니다.

3단계: 타사 리소스를 CodeCatalyst 프로젝트에 연결

GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 사용하거나 CodeCatalyst에서 Jira 문제를 관리하는 세 번째이자 마지막 단계는 이를 사용하려는 CodeCatalyst 프로젝트에 연결하는 것입니다.

확장 프로그램의 세부 정보 페이지에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 또는 Jira 프로젝트를 CodeCatalyst 프로젝트에 연결하려면

- 설치한 타사 확장 프로그램과 연결한 공급자에 따라 다음 중 하나를 수행합니다.
 - GitHub 리포지토리: GitHub 리포지토리를 연결합니다.
 1. 연결된 GitHub 리포지토리 탭에서 GitHub 리포지토리 연결을 선택합니다.
 2. GitHub 계정 드롭다운에서 연결할 리포지토리가 포함된 GitHub 계정을 선택합니다.
 3. GitHub 리포지토리 드롭다운에서 CodeCatalyst 프로젝트에 연결하려는 리포지토리를 선택합니다.
-  **Tip**

리포지토리 이름이 회색으로 표시된 경우 해당 리포지토리가 이미 스페이스의 다른 프로젝트에 연결되어 있으므로 해당 리포지토리를 연결할 수 없습니다.
4. (선택 사항) 리포지토리 목록에 GitHub 리포지토리가 표시되지 않는 경우 GitHub의 Amazon CodeCatalyst 애플리케이션에서 리포지토리 액세스에 대해 구성되지 않았을 수 있습니다. 연결된 계정의 CodeCatalyst에서 사용할 수 있는 GitHub 리포지토리를 구성할 수 있습니다.
 - a. [GitHub](#) 계정으로 이동하여 설정을 선택한 다음 애플리케이션을 선택합니다.

- b. 설치된 GitHub 앱 탭에서 Amazon CodeCatalyst 애플리케이션에 대해 구성을 선택합니다.
- c. 다음 중 하나를 수행하여 CodeCatalyst에서 연결하려는 GitHub 리포지토리의 액세스를 구성합니다.
 - 모든 현재 및 향후 리포지토리에 대한 액세스를 제공하려면 모든 리포지토리를 선택합니다.
 - 특정 리포지토리에 대한 액세스 권한을 제공하려면 선택된 리포지토리만을 선택하고 리포지토리 선택 드롭다운을 선택한 다음 CodeCatalyst에서 연결할 수 있도록 허용할 리포지토리를 선택합니다.
5. CodeCatalyst 프로젝트 드롭다운 메뉴에서 GitHub 리포지토리에 연결하려는 CodeCatalyst 프로젝트를 선택합니다.
6. 연결을 선택합니다.

CodeCatalyst에서 GitHub 리포지토리를 더 이상 사용하지 않으려면 CodeCatalyst 프로젝트에서 연결을 해제할 수 있습니다. 리포지토리가 연결 해제되면 해당 리포지토리의 이벤트는 워크플로를 시작하지 않으며 CodeCatalyst 개발 환경에서 해당 리포지토리를 사용할 수 없습니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#) 섹션을 참조하세요.

- Bitbucket 리포지토리: Bitbucket 리포지토리를 연결합니다.
 1. 연결된 Bitbucket 리포지토리 탭에서 Bitbucket 리포지토리 연결을 선택합니다.
 2. Bitbucket 작업 영역 드롭다운에서 연결하려는 리포지토리가 포함된 Bitbucket 작업 영역을 선택합니다.
 3. Bitbucket 리포지토리 드롭다운에서 CodeCatalyst 프로젝트에 연결하려는 리포지토리를 선택합니다.

 Tip

리포지토리 이름이 회색으로 표시된 경우 해당 리포지토리가 이미 스페이스의 다른 프로젝트에 연결되어 있으므로 해당 리포지토리를 연결할 수 없습니다.

4. CodeCatalyst 프로젝트 드롭다운 메뉴에서 Bitbucket 리포지토리에 연결하려는 CodeCatalyst 프로젝트를 선택합니다.
5. 연결을 선택합니다.

CodeCatalyst에서 Bitbucket 리포지토리를 더 이상 사용하지 않으려면 CodeCatalyst 프로젝트에서 연결을 해제할 수 있습니다. 리포지토리가 연결 해제되면 해당 리포지토리의 이벤트는 워크플로를 시작하지 않으며 CodeCatalyst 개발 환경에서 해당 리포지토리를 사용할 수 없습니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#) 섹션을 참조하세요.

- GitLab 리포지토리: GitLab 프로젝트 리포지토리를 연결합니다.
 1. 연결된 GitLab 프로젝트 리포지토리 탭에서 GitLab 프로젝트 리포지토리 연결을 선택합니다.
 2. GitLab 사용자 드롭다운에서 연결하려는 리포지토리가 포함된 GitLab 사용자를 선택합니다.
 3. GitLab 프로젝트 리포지토리 드롭다운에서 CodeCatalyst 프로젝트에 연결할 리포지토리를 선택합니다.

 Tip

리포지토리 이름이 회색으로 표시된 경우 해당 리포지토리가 이미 스페이스의 다른 프로젝트에 연결되어 있으므로 해당 리포지토리를 연결할 수 없습니다.

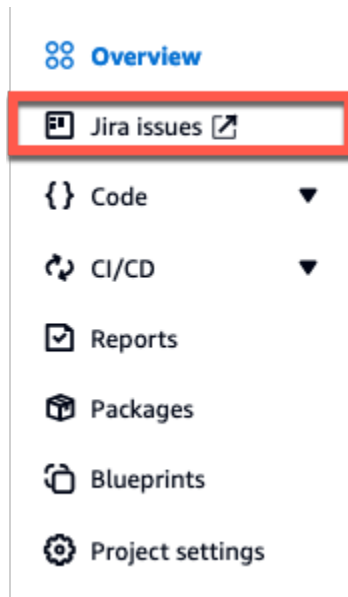
4. CodeCatalyst 프로젝트 드롭다운 메뉴에서 GitLab 프로젝트 리포지토리를 연결할 CodeCatalyst 프로젝트를 선택합니다.
5. 연결을 선택합니다.

CodeCatalyst에서 GitLab 프로젝트 리포지토리를 더 이상 사용하지 않으려면 CodeCatalyst 프로젝트에서 연결을 해제할 수 있습니다. 프로젝트 리포지토리가 연결 해제되면 해당 프로젝트 리포지토리의 이벤트는 워크플로를 시작하지 않으며 CodeCatalyst 개발 환경에서 해당 프로젝트 리포지토리를 사용할 수 없습니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#) 섹션을 참조하세요.

- Jira 소프트웨어: Jira 프로젝트를 연결합니다.
 1. 연결된 Jira 프로젝트 탭에서 Jira 프로젝트 연결을 선택합니다.
 2. Jira 사이트 드롭다운 메뉴에서 연결하려는 프로젝트가 포함된 Jira 사이트를 선택합니다.
 3. Jira 프로젝트 드롭다운 메뉴에서 CodeCatalyst 프로젝트에 연결할 프로젝트를 선택합니다.

4. CodeCatalyst 프로젝트 드롭다운 메뉴에서 Jira 프로젝트에 연결하려는 CodeCatalyst 프로젝트를 선택합니다.
5. 연결을 선택합니다.

Jira 프로젝트가 CodeCatalyst 프로젝트에 연결되면 CodeCatalyst 문제에 대한 액세스가 완전히 비활성화되고 CodeCatalyst 탐색 창의 문제가 Jira 프로젝트에 연결되는 Jira 문제 항목으로 대체됩니다.



CodeCatalyst에서 Jira 프로젝트를 더 이상 사용하지 않으려면 CodeCatalyst 프로젝트에서 연결을 해제할 수 있습니다. Jira 프로젝트가 연결 해제되면 CodeCatalyst 프로젝트에서 Jira 문제를 사용할 수 없으며 CodeCatalyst 문제가 다시 문제 공급자가 됩니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#) 섹션을 참조하세요.

GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 코드의 소스 리포지토리에서 프로젝트에 연결할 수도 있습니다. 자세한 내용은 [연결된 타사 공급자의 리소스 연결](#) 섹션을 참조하세요.

다음 단계

GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 리포지토리 확장 프로그램을 설치하고, 리소스 공급자를 연결하고, 타사 리포지토리를 CodeCatalyst 프로젝트에 연결한 후 CodeCatalyst 워크플로 및 개발 환경에서 사용할 수 있습니다. 블루프린트에서 생성된 코드를 사용하여 연결된 GitHub 계정,

Bitbucket 작업 영역 또는 GitLab 사용자에서 타사 리포지토리를 생성할 수도 있습니다. 자세한 내용은 [타사 리포지토리 이벤트 후 워크플로 실행 자동 시작](#) 및 [개발 환경 생성](#) 섹션을 참조하세요.

Jira 소프트웨어 확장 프로그램을 설치하고, Jira 사이트를 연결하고, Jira 프로젝트를 CodeCatalyst 프로젝트에 연결하고, 풀 요청을 연결하면 CodeCatalyst의 업데이트가 Jira 프로젝트에 반영됩니다. 풀 요청을 Jira 문제에 연결하는 방법에 대한 자세한 내용은 [Jira 문제를 CodeCatalyst 풀 요청에 연결](#) 섹션을 참조하세요. Jira에서 CodeCatalyst 이벤트 보기에 대한 자세한 내용은 [Jira 문제에서 CodeCatalyst 이벤트 보기](#) 섹션을 참조하세요.

스페이스에 확장 프로그램 설치

CodeCatalyst 스페이스의 확장 프로그램을 설치하여 해당 스페이스의 프로젝트에 기능을 추가할 수 있습니다. 카탈로그



아이콘을 선택하여 CodeCatalyst 카탈로그를 볼 수 있습니다. 확장 프로그램 및 기능에 대한 자세한 내용은 [사용 가능한 타사 확장](#) 섹션을 참조하세요.

Important

확장 프로그램을 설치하려면 스페이스에 스페이스 관리자 역할이 있는 계정으로 로그인해야 합니다.

Important

리포지토리 확장 프로그램을 설치한 후 CodeCatalyst에 연결하는 모든 리포지토리의 코드가 CodeCatalyst에 인덱싱 및 저장됩니다. 이렇게 하면 CodeCatalyst에서 코드를 검색할 수 있습니다. CodeCatalyst에서 연결된 리포지토리를 사용할 때 코드에 대한 데이터 보호를 더 잘 이해하려면 Amazon CodeCatalyst 사용 설명서의 [데이터 보호](#)를 참조하세요.

CodeCatalyst 카탈로그에서 확장 프로그램을 설치하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다.

3. 상단 메뉴 바에서 카탈로그



아이콘을 선택하여 카탈로그로 이동합니다. GitHub 리포지토리, Bitbucket 리포지토리, GitLab 리포지토리 또는 Jira 소프트웨어를 검색할 수 있습니다. 범주를 기반으로 확장 프로그램을 필터링할 수도 있습니다.

4. (선택 사항) 확장 프로그램과 관련된 권한 등 확장 프로그램에 대한 자세한 내용을 보려면 확장 이름을 선택합니다.
5. 설치를 선택합니다. 확장 프로그램에 필요한 권한을 검토하고 계속하려면 설치를 다시 선택합니다.

익스텐션을 설치하면 설치된 확장 프로그램의 세부 정보 페이지가 표시됩니다. 확장 프로그램에 대한 추가 정보는 탭을 참조하세요. 또한 세부 정보 페이지에서 필요한 경우 확장 프로그램을 추가로 구성할 수 있습니다.

스페이스에 확장 프로그램 제거

이전에 CodeCatalyst 스페이스에 설치한 확장 프로그램을 제거할 수 있습니다. 확장 프로그램을 제거하면 해당 확장 프로그램과 관련된 리소스가 CodeCatalyst 스페이스 또는 프로젝트에서 제거될 수 있습니다.

Important

확장 프로그램을 제거하려면 스페이스에 스페이스 관리자 역할이 있는 계정으로 로그인해야 합니다.

CodeCatalyst 스페이스에서 확장 프로그램 제거

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다.
3. 다음 중 하나를 수행하여 스페이스에 설치된 확장 프로그램 목록을 봅니다.
 - a. 설정을 선택한 다음 설치된 확장 프로그램을 선택합니다.

- b. 상단 메뉴에서 카탈로그



아이콘을 선택합니다.

4. 제거하려는 확장 프로그램에서 구성을 선택합니다.
5. 확장 프로그램 세부 정보 페이지에서 제거를 선택합니다.
6. 확장 프로그램 제거 대화 상자의 정보를 검토합니다. 지침에 따라 제거를 선택하여 확장 프로그램을 제거합니다.

GitHub 계정, Bitbucket 작업 영역, GitLab 사용자 및 Jira 사이트 CodeCatalyst 연결

GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 사용하거나 CodeCatalyst에서 Jira 프로젝트를 관리하려면 먼저 타사 소스를 CodeCatalyst 스페이스에 연결해야 합니다. 확장 프로그램 및 기능에 대한 자세한 내용은 [사용 가능한 타사 확장](#) 섹션을 참조하세요.

Important

GitHub 계정, Bitbucket Work 스페이스, GitLab 사용자 또는 Jira 사이트를 CodeCatalyst 스페이스에 연결하려면 타사 소스의 관리자와 CodeCatalyst 스페이스 관리자여야 합니다.

Note

GitHub 계정에 대한 연결을 사용하는 경우 개인 연결을 생성하여 CodeCatalyst ID와 GitHub ID 간에 ID 매핑을 설정해야 합니다. 자세한 내용은 [개인 연결](#) 및 [개인 연결을 사용하여 GitHub 리소스에 액세스](#) 섹션을 참조하세요.

GitHub 계정, Bitbucket Work 스페이스, GitLab 사용자 또는 Jira 사이트를 CodeCatalyst에 연결하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다.
3. 다음 중 하나를 수행하여 스페이스에 설치된 확장 프로그램 목록을 봅니다.
 - a. 설정을 선택한 다음 설치된 확장 프로그램을 선택합니다.

b. 상단 메뉴에서 카탈로그



아이콘을 선택합니다.

4. 구성하려는 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 리포지토리 또는 Jira 소프트웨어 확장 프로그램 중 하나에 대해 구성을 선택합니다.

5. 구성하도록 선택한 타사 확장 프로그램에 따라 다음 중 하나를 수행합니다.

- GitHub 리포지토리: GitHub 계정에 연결합니다.

1. 연결된 GitHub 계정 탭에서 GitHub 계정 연결을 선택하여 GitHub의 외부 사이트로 이동합니다.
2. GitHub 자격 증명으로 GitHub 계정에 로그인한 다음 Amazon CodeCatalyst를 설치할 계정을 선택합니다.

 Tip

이전에 GitHub 계정을 스페이스에 연결한 경우 재인증하라는 메시지가 표시되지 않습니다. 대신 둘 이상의 GitHub 스페이스에 멤버 또는 공동 작업자인 경우 확장 프로그램을 설치할 위치를 묻는 대화 상자가 표시되고, GitHub 스페이스에만 속한 경우 Amazon CodeCatalyst 애플리케이션의 구성 페이지가 표시됩니다. 허용하려는 리포지토리 액세스에 대한 애플리케이션을 구성한 다음 저장을 선택합니다. 저장 버튼이 활성화되지 않은 경우 구성을 변경한 다음 다시 시도하세요.

3. CodeCatalyst가 현재 및 미래 리포지토리에 모두 액세스하도록 허용할지 여부를 선택하거나 CodeCatalyst에서 사용할 특정 GitHub 리포지토리를 선택합니다. 기본 옵션은 CodeCatalyst에서 액세스할 향후 리포지토리를 포함하여 GitHub 계정에 모든 GitHub 리포지토리를 포함하는 것입니다.
4. CodeCatalyst에 부여된 권한을 검토한 다음 설치를 선택합니다.

GitHub 계정을 CodeCatalyst에 연결하면 GitHub 리포지토리 확장 프로그램 세부 정보 페이지로 이동합니다. 여기서 연결된 GitHub 계정과 연결된 GitHub 리포지토리를 보고 관리할 수 있습니다.

- Bitbucket 리포지토리: Bitbucket 작업 영역에 연결합니다.

1. 연결된 Bitbucket 작업 영역 탭에서 Bitbucket 작업 영역 연결을 선택하여 Bitbucket의 외부 사이트로 이동합니다.

2. Bitbucket 자격 증명을 사용하여 Bitbucket 작업 영역에 로그인하고 CodeCatalyst에 부여된 권한을 검토합니다.
3. 작업 영역에 대한 권한 부여 드롭다운 메뉴에서 CodeCatalyst 액세스를 제공할 Bitbucket 작업 영역을 선택한 다음 액세스 권한 부여를 선택합니다.

 Tip

이전에 Bitbucket 작업 영역을 스페이스에 연결한 경우 재인증하라는 메시지가 표시되지 않습니다. 대신 하나 이상의 Bitbucket 작업 영역에서 멤버 또는 공동 작업자인 경우 확장 프로그램을 설치할 위치를 묻는 대화 상자가 표시되고, 하나의 Bitbucket 작업 영역에만 속한 경우 Amazon CodeCatalyst 애플리케이션의 구성 페이지가 표시됩니다. 작업 영역 액세스를 허용하려는 애플리케이션을 선택하고, 액세스 허용을 선택합니다. 액세스 허용 버튼이 활성화되지 않은 경우 구성을 변경한 다음 다시 시도하세요.

Bitbucket 워크스페이스를 CodeCatalyst에 연결하면 연결된 Bitbucket 워크스페이스와 연결된 Bitbucket 리포지토리를 보고 관리할 수 있는 Bitbucket 리포지토리 확장 프로그램 세부 정보 페이지로 이동합니다.

- GitLab 리포지토리: GitLab 사용자에게 연결합니다.
 1. GitLab 사용자 연결을 선택하여 GitLab 외부 사이트로 이동합니다.
 2. GitLab 자격 증명을 사용하여 GitLab 사용자에게 로그인하고 CodeCatalyst에 부여된 권한을 검토합니다.

 Tip

이전에 GitLab 사용자를 스페이스에 연결한 경우 재인증하라는 메시지가 표시되지 않습니다. 대신 CodeCatalyst 콘솔로 다시 이동합니다.

3. GitLab용 AWS 커넥터 권한 부여를 선택합니다.

GitLab 사용자를 CodeCatalyst에 연결하면 GitLab 리포지토리 확장 프로그램 세부 정보 페이지로 이동하며, 여기서 연결된 GitLab 사용자 및 연결된 GitLab 프로젝트 리포지토리를 보고 관리할 수 있습니다.

- Jira 소프트웨어: Jira 사이트를 연결합니다.

1. 연결된 Jira 사이트 탭에서 Jira 사이트 연결을 선택하여 Atlassian Marketplace의 외부 사이트로 이동합니다.
2. 지금 받기를 선택하여 Jira 사이트에 CodeCatalyst 설치를 시작합니다.

 Note

이전에 Jira 사이트에 CodeCatalyst를 설치한 경우 알림을 받게 됩니다. 시작하기를 선택하여 최종 단계로 이동합니다.

3. 역할에 따라 다음 중 하나를 수행합니다.
 1. Jira 사이트 관리자인 경우 사이트 드롭다운 메뉴에서 CodeCatalyst 애플리케이션을 설치할 Jira 사이트를 선택한 다음 앱 설치를 선택합니다.

 Note

Jira 사이트가 하나 있는 경우 이 단계는 표시되지 않으며 자동으로 다음 단계로 이동합니다.

2. a. Jira 관리자가 아닌 경우 사이트 드롭다운 메뉴에서 Jira 사이트를 선택하여 CodeCatalyst 애플리케이션을 설치한 다음 앱 요청을 선택합니다. Jira 앱 설치에 대한 자세한 내용은 [앱을 설치할 수 있는 사람은 누구입니까?](#)를 참조하세요.
b. CodeCatalyst를 설치해야 하는 이유를 입력 텍스트 필드에 입력하거나 기본 텍스트를 유지한 다음 요청 제출을 선택합니다.
4. 애플리케이션이 설치될 때 CodeCatalyst에서 수행하는 작업을 검토한 다음 지금 받기를 선택합니다.
5. 애플리케이션을 설치한 후 CodeCatalyst로 돌아가기를 선택하여 CodeCatalyst로 돌아갑니다.

Jira 사이트를 CodeCatalyst 연결한 후 Jira 소프트웨어 확장 프로그램 세부 정보 페이지의 연결된 Jira 사이트 탭에서 연결된 사이트를 볼 수 있습니다.

GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 더 이상 사용하지 않거나 CodeCatalyst에서 Jira 문제를 관리하지 않으려면 타사 소스를 연결 해제할 수 있습니다. GitHub 계정, Bitbucket 작업 영역 또는 GitLab 사용자의 연결이 끊어지면 타사 리포지토리의 이벤트는 워크플로 실행을 시작하지 않으며 CodeCatalyst 개발 환경에서 이러한 리포지토리를 사용할 수 없습니다.

Jira 사이트 연결이 끊어지면 사이트의 프로젝트에서 발생하는 Jira 문제는 CodeCatalyst 프로젝트에서 사용할 수 없으며 CodeCatalyst 문제가 다시 문제 공급자가 됩니다. 자세한 내용은 [GitHub 계정](#), [Bitbucket 작업 영역](#), [GitLab 사용자 및 Jira 사이트 CodeCatalyst 연결 해제](#) 섹션을 참조하세요.

GitHub 계정, Bitbucket 작업 영역, GitLab 사용자 및 Jira 사이트 CodeCatalyst 연결 해제

GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 더 이상 사용하지 않거나 CodeCatalyst에서 Jira 문제를 관리하지 않으려면 타사 소스를 연결 해제할 수 있습니다. GitHub 계정, Bitbucket 작업 영역 또는 GitLab 사용자의 연결이 끊어지면 리포지토리의 이벤트는 워크플로 실행을 시작하지 않으며 CodeCatalyst 개발 환경에서 이러한 리포지토리를 사용할 수 없습니다. Jira 사이트 연결이 끊어지면 사이트의 프로젝트에서 발생하는 Jira 문제는 CodeCatalyst 프로젝트에서 사용할 수 없으며 CodeCatalyst 문제가 다시 문제 공급자가 됩니다.

Note

- GitHub 계정을 연결 해제하려면 먼저 해당 계정에서 링크된 모든 GitHub 리포지토리의 링크를 해제해야 합니다.
- Bitbucket 작업 영역을 연결 해제하려면 먼저 링크된 모든 Bitbucket 리포지토리를 해당 작업 영역에서 링크 해제해야 합니다.
- GitLab 사용자를 연결 해제하려면 먼저 링크된 모든 GitLab 프로젝트 리포지토리를 해당 작업 영역에서 링크 해제해야 합니다.
- Jira 사이트를 연결 해제하려면 먼저 링크된 모든 Jira 프로젝트를 해당 계정에서 링크 해제해야 합니다.

자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리](#), [Bitbucket 리포지토리](#), [GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#) 섹션을 참조하세요.

GitHub 프로젝트, Bitbucket Work 스페이스, GitLab 사용자 또는 Jira 사이트 연결을 해제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다.
3. 다음 중 하나를 수행하여 스페이스에 설치된 확장 프로그램 목록을 봅니다.
 - a. 설정을 선택한 다음 설치된 확장 프로그램을 선택합니다.

b. 상단 메뉴에서 카탈로그



아이콘을 선택합니다.

4. 구성하려는 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 리포지토리 또는 Jira 소프트웨어 확장 프로그램 중 하나에 대해 구성을 선택합니다.
5. 구성하도록 선택한 타사 확장 프로그램에 따라 다음 중 하나를 수행합니다.

- GitHub 리포지토리: GitHub 계정을 연결 해제합니다.

연결된 GitHub 계정 탭에서 연결 해제하려는 GitHub 계정을 선택한 다음 GitHub 계정 연결 해제를 선택합니다.

- Bitbucket 리포지토리: Bitbucket 작업 영역 연결을 해제합니다.

연결된 Bitbucket 작업 영역 탭에서 연결 해제하려는 Bitbucket 작업 영역을 선택한 다음 Bitbucket 작업 영역 연결 해제를 선택합니다.

- GitLab 리포지토리: GitLab 사용자를 연결 해제합니다.

연결된 GitLab 사용자 탭에서 연결 해제하려는 GitLab 사용자를 선택한 다음 GitLab 사용자 연결 해제를 선택합니다.

- Jira 소프트웨어: Jira 사이트를 연결 해제합니다.

연결된 Jira 사이트 탭에서 연결 해제하려는 Jira 사이트를 선택한 다음 Jira 사이트 연결 해제를 선택합니다.

6. 연결 해제 대화 상자에서 계정 연결 해제에 영향을 검토합니다.
7. 텍스트 입력 필드에 disconnect를 입력한 다음 연결 해제를 선택합니다.

CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결

GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 사용하거나 Jira 프로젝트를 관리하려면 먼저 리포지토리 또는 프로젝트가 속한 타사 소스를 CodeCatalyst 스페이스에 연결해야 합니다. 자세한 내용은 [GitHub 계정](#), [Bitbucket 작업 영역](#), [GitLab 사용자 및 Jira 사이트](#) [CodeCatalyst 연결](#) 섹션을 참조하세요.

워크플로에서 연결된 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 사용할 수 있습니다. 여기서 연결된 리포지토리의 이벤트는 워크플로 구성에 따라 코드를 빌드, 테스트 또는 배포할 수 있는 워크플로를 시작합니다. 연결된 GitHub 또는 Bitbucket 리포지토리를 사용하는 워크플로의 워크플로 구성 파일은 연결된 리포지토리에 저장됩니다. 연결된 리포지토리를 개발 환경과 함께 사용하여 연결된 리포지토리에서 파일을 생성, 업데이트 및 삭제할 수도 있습니다. GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 리포지토리 확장 프로그램의 세부 정보 페이지 또는 프로젝트 자체의 코드에 있는 소스 리포지토리 보기에서 CodeCatalyst 프로젝트에 연결할 수 있습니다.

Important

GitHub 또는 Bitbucket 리포지토리를 기고자로 연결할 수 있지만, 타사 리포지토리는 스페이스 관리자 또는 프로젝트 관리자 만 연결 해제할 수 있습니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#) 섹션을 참조하세요.

Important

리포지토리 확장 프로그램을 설치한 후 CodeCatalyst에 연결하는 모든 리포지토리의 코드가 CodeCatalyst에 인덱싱 및 저장됩니다. 이렇게 하면 CodeCatalyst에서 코드를 검색할 수 있습니다. CodeCatalyst에서 연결된 리포지토리를 사용할 때 코드에 대한 데이터 보호를 더 잘 이해하려면 Amazon CodeCatalyst 사용 설명서의 [데이터 보호](#)를 참조하세요.

Important

CodeCatalyst는 연결된 리포지토리에 대한 기본 브랜치의 변경 사항 감지를 지원하지 않습니다. 연결된 리포지토리의 기본 브랜치를 변경하려면 먼저 CodeCatalyst에서 연결을 해제하고 기본 브랜치를 변경한 다음 다시 연결해야 합니다.

리포지토리를 연결하기 전에 항상 최신 버전의 확장 프로그램을 사용하는 것이 좋습니다.

연결된 Jira 프로젝트를 사용하여 문제를 관리하고 CodeCatalyst 풀 요청을 Jira 문제에 연결할 수 있습니다. 풀 요청의 요약 상태 및 연결된 CodeCatalyst 워크플로 이벤트의 상태는 Jira 문제에 반영됩니다.

 Important

Jira 프로젝트를 CodeCatalyst 프로젝트에 연결하려면 CodeCatalyst 스페이스 관리자 또는 CodeCatalyst 프로젝트 관리자여야 합니다.

 Note

- GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리는 한 스페이스에서 하나의 CodeCatalyst 프로젝트에만 연결할 수 있습니다.
- 비어있거나 보관된 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 CodeCatalyst 프로젝트와 함께 사용할 수 없습니다.
- CodeCatalyst 프로젝트의 리포지토리와 이름이 동일한 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 리포지토리를 연결할 수 없습니다.
- GitHub 리포지토리 확장 프로그램은 GitHub Enterprise Server 리포지토리와 호환되지 않습니다.
- Bitbucket 리포지토리 확장 프로그램은 Bitbucket Data Center 리포지토리와 호환되지 않습니다.
- GitLab 리포지토리 확장 프로그램은 GitLab 자체 관리형 프로젝트 리포지토리와 호환되지 않습니다.
- 연결된 리포지토리에서는 설명 쓰기 또는 설명 요약 기능을 사용할 수 없습니다. 이러한 기능은 CodeCatalyst의 풀 요청에서만 사용할 수 있습니다.
- CodeCatalyst 프로젝트는 하나의 Jira 프로젝트에만 연결할 수 있습니다. Jira 프로젝트는 여러 CodeCatalyst 프로젝트에 연결할 수 있습니다.

주제

- [연결된 타사 공급자의 리소스 연결](#)
- [CodeCatalyst 프로젝트 생성 중에 타사 리포지토리 연결](#)

연결된 타사 공급자의 리소스 연결

확장 프로그램의 세부 정보 페이지에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 또는 Jira 프로젝트를 CodeCatalyst 프로젝트에 연결하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다.
3. 다음 중 하나를 수행하여 스페이스에 설치된 확장 프로그램 목록을 봅니다.

- a. 설정을 선택한 다음 설치된 확장 프로그램을 선택합니다.
- b. 상단 메뉴에서 카탈로그



아이콘을 선택합니다.

4. GitHub 리포지토리, Bitbucket 리포지토리, GitLab 리포지토리 또는 Jira 소프트웨어 중 하나에 대해 구성을 선택합니다.
5. 구성하도록 선택한 타사 확장 프로그램에 따라 다음 중 하나를 수행합니다.

- GitHub 리포지토리: GitHub 리포지토리를 연결합니다.

1. 연결된 GitHub 리포지토리 탭에서 GitHub 리포지토리 연결을 선택합니다.
2. GitHub 계정 드롭다운에서 연결할 리포지토리가 포함된 GitHub 계정을 선택합니다.
3. GitHub 리포지토리 드롭다운에서 CodeCatalyst 프로젝트에 연결하려는 리포지토리를 선택합니다.

Tip

리포지토리 이름이 회색으로 표시된 경우 해당 리포지토리가 이미 스페이스의 다른 프로젝트에 연결되어 있으므로 해당 리포지토리를 연결할 수 없습니다.

4. (선택 사항) 리포지토리 목록에 GitHub 리포지토리가 표시되지 않는 경우 GitHub의 Amazon CodeCatalyst 애플리케이션에서 리포지토리 액세스에 대해 구성되지 않았을 수 있습니다. 연결된 계정의 CodeCatalyst에서 사용할 수 있는 GitHub 리포지토리를 구성할 수 있습니다.
 - a. [GitHub](#) 계정으로 이동하여 설정을 선택한 다음 애플리케이션을 선택합니다.

- b. 설치된 GitHub 앱 탭에서 Amazon CodeCatalyst 애플리케이션에 대해 구성을 선택합니다.
- c. 다음 중 하나를 수행하여 CodeCatalyst에서 연결하려는 GitHub 리포지토리의 액세스를 구성합니다.
 - 모든 현재 및 향후 리포지토리에 대한 액세스를 제공하려면 모든 리포지토리를 선택합니다.
 - 특정 리포지토리에 대한 액세스 권한을 제공하려면 선택된 리포지토리만을 선택하고 리포지토리 선택 드롭다운을 선택한 다음 CodeCatalyst에서 연결할 수 있도록 허용할 리포지토리를 선택합니다.
5. CodeCatalyst 프로젝트 드롭다운 메뉴에서 GitHub 리포지토리에 연결하려는 CodeCatalyst 프로젝트를 선택합니다.
6. 연결을 선택합니다.

CodeCatalyst에서 GitHub 리포지토리를 더 이상 사용하지 않으려면 CodeCatalyst 프로젝트에서 연결을 해제할 수 있습니다. 리포지토리가 연결 해제되면 해당 리포지토리의 이벤트는 워크플로를 시작하지 않으며 CodeCatalyst 개발 환경에서 해당 리포지토리를 사용할 수 없습니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#) 섹션을 참조하세요.

- Bitbucket 리포지토리: Bitbucket 리포지토리를 연결합니다.
 1. 연결된 Bitbucket 리포지토리 탭에서 Bitbucket 리포지토리 연결을 선택합니다.
 2. Bitbucket 작업 영역 드롭다운에서 연결하려는 리포지토리가 포함된 Bitbucket 작업 영역을 선택합니다.
 3. Bitbucket 리포지토리 드롭다운에서 CodeCatalyst 프로젝트에 연결하려는 리포지토리를 선택합니다.

 Tip

리포지토리 이름이 회색으로 표시된 경우 해당 리포지토리가 이미 스페이스의 다른 프로젝트에 연결되어 있으므로 해당 리포지토리를 연결할 수 없습니다.

4. CodeCatalyst 프로젝트 드롭다운 메뉴에서 Bitbucket 리포지토리에 연결하려는 CodeCatalyst 프로젝트를 선택합니다.
5. 연결을 선택합니다.

CodeCatalyst에서 Bitbucket 리포지토리를 더 이상 사용하지 않으려면 CodeCatalyst 프로젝트에서 연결을 해제할 수 있습니다. 리포지토리가 연결 해제되면 해당 리포지토리의 이벤트는 워크플로를 시작하지 않으며 CodeCatalyst 개발 환경에서 해당 리포지토리를 사용할 수 없습니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#) 섹션을 참조하세요.

- GitLab 리포지토리: GitLab 프로젝트 리포지토리를 연결합니다.
 1. 연결된 GitLab 프로젝트 리포지토리 탭에서 GitLab 프로젝트 리포지토리 연결을 선택합니다.
 2. GitLab 사용자 드롭다운에서 연결하려는 프로젝트 리포지토리가 포함된 GitLab 사용자를 선택합니다.
 3. GitLab 프로젝트 리포지토리 드롭다운에서 CodeCatalyst 프로젝트에 연결할 리포지토리를 선택합니다.

 Tip

리포지토리 이름이 회색으로 표시된 경우 해당 리포지토리가 이미 스페이스의 다른 프로젝트에 연결되어 있으므로 해당 리포지토리를 연결할 수 없습니다.

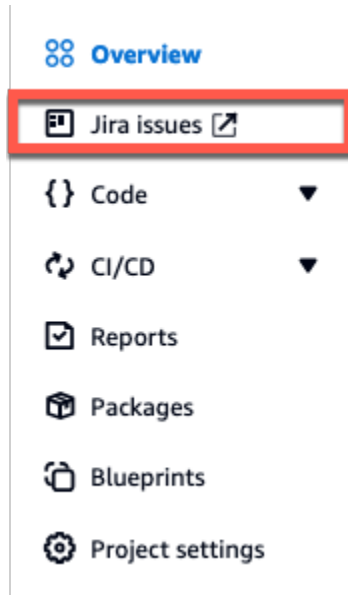
4. CodeCatalyst 프로젝트 드롭다운 메뉴에서 GitLab 프로젝트 리포지토리를 연결할 CodeCatalyst 프로젝트를 선택합니다.
5. 연결을 선택합니다.

CodeCatalyst에서 GitLab 프로젝트 리포지토리를 더 이상 사용하지 않으려면 CodeCatalyst 프로젝트에서 연결을 해제할 수 있습니다. 프로젝트 리포지토리가 연결 해제되면 해당 프로젝트 리포지토리의 이벤트는 워크플로를 시작하지 않으며 CodeCatalyst 개발 환경에서 해당 프로젝트 리포지토리를 사용할 수 없습니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#) 섹션을 참조하세요.

- Jira 소프트웨어: Jira 프로젝트를 연결합니다.
 1. 연결된 Jira 프로젝트 탭에서 Jira 프로젝트 연결을 선택합니다.
 2. Jira 사이트 드롭다운 메뉴에서 연결하려는 프로젝트가 포함된 Jira 사이트를 선택합니다.
 3. Jira 프로젝트 드롭다운 메뉴에서 CodeCatalyst 프로젝트에 연결할 프로젝트를 선택합니다.

4. CodeCatalyst 프로젝트 드롭다운 메뉴에서 Jira 프로젝트에 연결하려는 CodeCatalyst 프로젝트를 선택합니다.
5. 연결을 선택합니다.

Jira 프로젝트가 CodeCatalyst 프로젝트에 연결되면 CodeCatalyst 문제에 대한 액세스가 완전히 비활성화되고 CodeCatalyst 탐색 창의 문제가 Jira 프로젝트에 연결되는 Jira 문제 항목으로 대체됩니다.



CodeCatalyst에서 Jira 프로젝트를 더 이상 사용하지 않으려면 CodeCatalyst 프로젝트에서 연결을 해제할 수 있습니다. Jira 프로젝트가 연결 해제되면 CodeCatalyst 프로젝트에서 Jira 문제를 사용할 수 없으며 CodeCatalyst 문제가 다시 문제 공급자가 됩니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#) 섹션을 참조하세요.

프로젝트의 소스 리포지토리 페이지에서 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 CodeCatalyst 프로젝트에 연결하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 프로젝트로 이동합니다.
3. 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
4. 리포지토리 추가를 선택하고 리포지토리 연결을 선택합니다.

5. 리포지토리 공급자 드롭다운 메뉴에서 타사 리포지토리 공급자 GitHub, Bitbucket, GitLab 중 하나를 선택합니다.
6. 연결하도록 선택한 타사 리포지토리 공급자에 따라 다음 중 하나를 수행합니다.

- GitHub 리포지토리: GitHub 리포지토리를 연결합니다.

1. GitHub 계정 드롭다운 메뉴에서 연결할 리포지토리가 포함된 GitHub 계정을 선택합니다.
2. GitHub 리포지토리 드롭다운 메뉴에서 CodeCatalyst 프로젝트를 연결할 GitHub 리포지토리를 선택합니다.

 Tip

리포지토리 이름이 회색으로 표시된 경우 Amazon CodeCatalyst 의 다른 프로젝트에 이미 연결되어 있으므로 해당 리포지토리를 연결할 수 없습니다.

3. (선택 사항) 리포지토리 목록에 GitHub 리포지토리가 표시되지 않는 경우 GitHub의 Amazon CodeCatalyst 애플리케이션에서 리포지토리 액세스에 대해 구성되지 않았을 수 있습니다. 연결된 계정의 CodeCatalyst에서 사용할 수 있는 GitHub 리포지토리를 구성할 수 있습니다.
 - a. [GitHub](#) 계정으로 이동하여 설정을 선택한 다음 애플리케이션을 선택합니다.
 - b. 설치된 GitHub 앱 탭에서 Amazon CodeCatalyst 애플리케이션에 대해 구성을 선택합니다.
 - c. 다음 중 하나를 수행하여 CodeCatalyst에서 연결하려는 GitHub 리포지토리의 액세스를 구성합니다.
 - 모든 현재 및 향후 리포지토리에 대한 액세스를 제공하려면 모든 리포지토리를 선택합니다.
 - 특정 리포지토리에 대한 액세스 권한을 제공하려면 선택된 리포지토리만을 선택하고 리포지토리 선택 드롭다운을 선택한 다음 CodeCatalyst에서 연결할 수 있도록 허용할 리포지토리를 선택합니다.
- Bitbucket 리포지토리: Bitbucket 리포지토리를 연결합니다.
 1. Bitbucket 작업 영역 드롭다운 메뉴에서 연결하려는 리포지토리가 포함된 Bitbucket 작업 영역을 선택합니다.
 2. Bitbucket 리포지토리 드롭다운 메뉴에서 CodeCatalyst 프로젝트를 연결할 Bitbucket 리포지토리를 선택합니다.

i Tip

리포지토리 이름이 회색으로 표시된 경우 Amazon CodeCatalyst 의 다른 프로젝트에 이미 연결되어 있으므로 해당 리포지토리를 연결할 수 없습니다.

- GitLab 리포지토리: GitLab 프로젝트 리포지토리를 연결합니다.
 1. GitLab 사용자 드롭다운 메뉴에서 연결하려는 프로젝트 리포지토리가 포함된 GitLab 사용자를 선택합니다.
 2. GitLab 프로젝트 리포지토리 드롭다운 메뉴에서 CodeCatalyst 프로젝트를 연결하려는 GitLab 프로젝트 리포지토리를 선택합니다.

i Tip

프로젝트 리포지토리 이름이 회색으로 표시된 경우 Amazon CodeCatalyst 의 다른 프로젝트에 이미 연결되어 있으므로 해당 프로젝트 리포지토리를 연결할 수 없습니다.

7. 연결을 선택합니다.

CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 더 이상 사용하지 않으려면 CodeCatalyst 프로젝트에서 연결을 해제할 수 있습니다. 리포지토리가 연결 해제되면 해당 리포지토리의 이벤트는 워크플로를 시작하지 않으며 CodeCatalyst 개발 환경에서 해당 리포지토리를 사용할 수 없습니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제](#) 섹션을 참조하세요.

GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 CodeCatalyst 프로젝트에 연결한 후 CodeCatalyst 워크플로 및 개발 환경에서 사용할 수 있습니다. Amazon Q Developer, 블루프린트 등과 함께 연결된 리포지토리를 사용할 수도 있습니다. 자세한 내용은 [타사 리포지토리 이벤트 후 워크플로 실행 자동 시작](#) 및 [개발 환경 생성](#) 섹션을 참조하세요.

Jira 프로젝트를 CodeCatalyst 프로젝트에 연결하고 풀 요청을 연결하면 CodeCatalyst의 업데이트가 Jira 프로젝트에 반영됩니다. 풀 요청을 Jira 문제에 연결하는 방법에 대한 자세한 내용은 [Jira 문제를 CodeCatalyst 풀 요청에 연결](#) 섹션을 참조하세요. Jira에서 CodeCatalyst 이벤트 보기에 대한 자세한 내용은 [Jira 문제에서 CodeCatalyst 이벤트 보기](#) 섹션을 참조하세요.

CodeCatalyst 프로젝트 생성 중에 타사 리포지토리 연결

새 CodeCatalyst 프로젝트를 생성할 때 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 새 CodeCatalyst 프로젝트에 연결할 수 있습니다. 자세한 내용은 [연결된 타사 리포지토리를 사용하여 프로젝트 생성](#) 섹션을 참조하세요.

CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결 해제

더 이상 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 사용하거나 CodeCatalyst에서 Jira 프로젝트를 관리하지 않으려면 CodeCatalyst 프로젝트에서 리포지토리 또는 프로젝트의 연결을 해제할 수 있습니다.

GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리의 연결을 해제해도 리포지토리가 삭제되거나 변경되지 않습니다. 연결된 리포지토리에 저장된 워크플로 구성 파일은 삭제되지 않습니다. 그러나 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리의 연결을 해제하면 해당 리포지토리의 이벤트는 더 이상 워크플로 실행을 시작하지 않으며 개발 환경과 함께 리포지토리를 사용할 수 없습니다. GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 리포지토리 확장 프로그램의 세부 정보 페이지 또는 프로젝트 자체의 코드에 있는 소스 리포지토리 보기를 통해 CodeCatalyst 프로젝트에서 연결 해제할 수 있습니다.

Jira 프로젝트의 연결을 해제해도 계획 항목 또는 개발 정보를 포함하여 프로젝트가 삭제되거나 변경되지 않습니다. 그러나 Jira 프로젝트의 연결을 해제하면 프로젝트의 Jira 문제를 CodeCatalyst 프로젝트에 더 이상 연결할 수 없으며 CodeCatalyst 문제가 다시 문제 공급자가 됩니다.

Important

CodeCatalyst 프로젝트에서 GitHub 리포지토리, Bitbucket 리포지토리 또는 Gitlab 프로젝트 리포지토리의 연결을 해제하려면 스페이스 관리자 또는 프로젝트 관리자여야 합니다.

확장 프로그램 세부 정보 페이지에서 CodeCatalyst 프로젝트의 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 또는 Jira 프로젝트의 연결을 해제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다.
3. 다음 중 하나를 수행하여 스페이스에 설치된 확장 프로그램 목록을 봅니다.

- a. 설정을 선택한 다음 설치된 확장 프로그램을 선택합니다.
- b. 상단 메뉴에서 카탈로그



아이콘을 선택합니다.

4. 구성하려는 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 리포지토리 또는 Jira 소프트웨어 확장 프로그램 중 하나에 대해 구성을 선택합니다.
5. 구성하도록 선택한 타사 확장 프로그램에 따라 다음 중 하나를 수행합니다.

- GitHub 리포지토리: GitHub 리포지토리를 연결 해제합니다.

GitHub 리포지토리 탭에서 연결을 해제하려는 GitHub 리포지토리를 선택한 다음 GitHub 리포지토리 연결 해제를 선택합니다.

- Bitbucket 리포지토리: Bitbucket 리포지토리를 연결 해제합니다.

Bitbucket 리포지토리 탭에서 연결을 해제하려는 Bitbucket 리포지토리를 선택한 다음 Bitbucket 리포지토리 연결 해제를 선택합니다.

- GitLab 리포지토리: GitLab 프로젝트 리포지토리를 연결 해제합니다.

GitLab 프로젝트 리포지토리 탭에서 연결을 해제하려는 GitLab 프로젝트 리포지토리를 선택한 다음 GitLab 프로젝트 리포지토리 연결 해제를 선택합니다.

- Jira 소프트웨어: Jira 프로젝트의 연결을 해제합니다.

Jira 프로젝트 탭에서 연결을 해제하려는 Jira 프로젝트를 선택한 다음 Jira 프로젝트 연결 해제를 선택합니다.

6. 연결 해제 대화 상자에서 리포지토리 연결 해제의 영향을 검토합니다.
7. 텍스트 입력 필드에 연결 해제를 입력하고 연결 해제를 선택합니다.

소스 리포지토리 페이지에서 CodeCatalyst 프로젝트의 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리 연결을 해제하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 프로젝트로 이동합니다.
3. 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.
4. 연결을 해제하려는 리포지토리의 라디오 버튼을 선택한 다음 리포지토리 연결 해제를 선택합니다.

5. 대화 상자의 정보를 검토합니다. 지침에 따라 연결 해제를 선택하여 리포지토리 연결을 해제합니다.

CodeCatalyst에서 타사 리포지토리 보기 및 Jira 문제 검색

GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 연결한 후 CodeCatalyst에서 해당 리포지토리를 보고 리소스를 확인하고 구성할 수 있습니다. CodeCatalyst에서 연결된 Jira 문제를 검색할 수도 있습니다.

주제

- [CodeCatalyst에서 타사 리포지토리 보기](#)
- [CodeCatalyst에서 Jira 문제 검색](#)

CodeCatalyst에서 타사 리포지토리 보기

연결된 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 프로젝트의 소스 리포지토리 목록 또는 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 리포지토리 확장 프로그램 세부 정보 페이지에서 볼 수 있습니다. 리포지토리 목록에서 리포지토리를 선택하면 CodeCatalyst에서 열리지 않습니다. 대신 연결된 리포지토리의 코드를 보고 작업할 수 있는 타사 리포지토리 공급자에서 열립니다.

CodeCatalyst에서 연결된 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 보려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 프로젝트로 이동합니다.
3. 탐색 창에서 코드를 선택한 다음 소스 리포지토리를 선택합니다.

확장 프로그램 세부 정보 페이지에서 연결된 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 보려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동한 다음 설치된 확장 프로그램 탭을 선택합니다.
3. 보려는 타사 리포지토리에 따라 다음 중 하나를 수행합니다.

- GitHub 리포지토리에서 구성을 선택한 다음 연결된 GitHub 리포지토리를 선택하여 CodeCatalyst 스페이스의 CodeCatalyst 프로젝트에 연결된 모든 GitHub 리포지토리를 봅니다.
- Bitbucket 리포지토리에서 구성을 선택한 다음 연결된 Bitbucket 리포지토리를 선택하여 CodeCatalyst 스페이스의 CodeCatalyst 프로젝트에 연결된 모든 Bitbucket 리포지토리를 봅니다.
- GitLab 리포지토리에서 구성을 선택한 다음 연결된 GitLab 리포지토리를 선택하여 CodeCatalyst 스페이스의 CodeCatalyst 프로젝트에 연결된 모든 GitLab 리포지토리를 봅니다.

CodeCatalyst 프로젝트에 연결된 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리가 목록에 표시됩니다. GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 선택하여 타사 리포지토리 공급자의 파일을 보고 편집합니다.

Note

워크플로가 소스 작업에서 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 사용하는 경우 시각적 편집기의 워크플로 YAML 또는 CodeCatalyst의 YAML에 대한 변경 사항이 자동으로 커밋되어 타사 리포지토리로 푸시됩니다.

CodeCatalyst에서 Jira 문제 검색

Jira 프로젝트를 연결한 후 CodeCatalyst 글로벌 검색 창을 사용하여 연결된 Jira 프로젝트에서 문제를 검색할 수 있습니다. 풀 요청의 문제에 연결하는 동안 CodeCatalyst에서 Jira 문제를 검색할 수도 있습니다. Jira 문제를 CodeCatalyst 풀 요청에 연결하는 방법에 대한 자세한 내용은 [Jira 문제를 CodeCatalyst 풀 요청에 연결](#) 섹션을 참조하세요.

연결된 Jira 프로젝트에서 Jira 문제를 검색하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 프로젝트로 이동합니다.
3. 글로벌 검색 표시줄에서 풀 요청에 연결하려는 문제 또는 Jira 문제에 대해 연결된 Jira 프로젝트를 검색합니다.

타사 리포지토리 이벤트 후 워크플로 실행 자동 시작

연결된 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 워크플로의 소스로 사용할 수 있습니다. 이 경우 연결된 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리의 지정된 브랜치에 대한 변경 사항이 자동으로 워크플로 실행을 시작합니다.

워크플로는 지속적 통합 및 지속적 전송(CI/CD) 시스템의 일부로 코드를 빌드, 테스트 및 배포하는 방법을 설명하는 자동화된 절차입니다. 워크플로는 워크플로 실행 중에 수행할 일련의 단계 또는 작업을 정의합니다. 또한 워크플로는 워크플로를 시작하게 하는 이벤트 또는 트리거를 정의합니다. 워크플로를 설정하려면 CodeCatalyst 콘솔의 [시각적 또는 YAML 편집기](#)를 사용하여 워크플로 정의 파일을 생성합니다.

Tip

프로젝트에서 워크플로를 사용하는 방법을 간략하게 알아보려면 [블루프린트가 있는 프로젝트를 생성](#)합니다. 각 블루프린트는 검토, 실행 및 실험할 수 있는 기능 워크플로를 배포합니다.

연결된 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리를 사용하도록 워크플로를 구성하면 워크플로 구성 파일이 해당 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리에 저장됩니다. 워크플로 구성은 워크플로 이름, 트리거, 리소스, 아티팩트 및 작업을 정의하는 YAML 파일입니다. 워크플로 구성 파일에 대한 자세한 내용은 [워크플로 YAML 정의](#) 섹션을 참조하세요.

워크플로 구성 파일은 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리의 `./codecatalyst/workflows/` 디렉터리에 있어야 합니다.

워크플로 편집기를 사용하여 워크플로를 생성하고 구성할 수 있습니다. 자세한 내용은 [워크플로 시작하기](#) 및 [워크플로에 소스 리포지토리 연결](#) 섹션을 참조하세요.

워크플로 실행을 시작하기 위한 트리거 추가

코드를 GitHub 또는 Bitbucket 리포지토리의 지정된 브랜치로 푸시할 때 자동으로 실행을 시작하도록 CodeCatalyst 워크플로를 구성할 수 있습니다. 워크플로 실행을 자동으로 시작하려면 워크플로 구성 파일의 Triggers 섹션에 트리거를 추가합니다.

예시: 간단한 코드 푸시 트리거

다음 예시는 소스 리포지토리의 모든 브랜치로 코드가 푸시될 때마다 워크플로 실행을 시작하는 트리거를 보여줍니다.

```
Triggers:
- Type: PUSH
```

예시: 간단한 풀 요청 트리거

다음 예시는 소스 리포지토리의 브랜치에 대해 풀 요청이 생성될 때마다 워크플로 실행을 시작하는 트리거를 보여줍니다.

```
Triggers:
- Type: PULLREQUEST
Events:
- OPEN
```

자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 섹션을 참조하세요.

타사 리포지토리 공급자를 통한 IP 액세스 제한

규칙 또는 구성을 설정하여 IP 주소를 기반으로 GitHub 리포지토리, Bitbucket 리포지토리 또는 GitLab 프로젝트 리포지토리에 대한 액세스를 제한할 수 있습니다. 타사 공급자의 설정 또는 액세스 제어 기능을 통해 이 작업을 수행할 수 있습니다.

사용 중인 타사 리포지토리 제공업체에 따라 다음 중 하나를 참조하세요.

- Amazon CodeCatalyst GitHub 리포지토리 확장 프로그램은 [GitHub Enterprise Cloud IP 액세스 제한](#)과 호환됩니다. 특정 IP 주소에 대한 액세스를 제한하도록 GitHub Enterprise Cloud 조직을 구성할 때 [GitHub 앱이 허용 목록을 구성하도록 활성화](#)할 수도 있습니다. 그러면 CodeCatalyst가 GitHub에 IP 주소를 자동으로 등록할 수 있습니다. 또는 [CodeCatalyst IP 주소를 수동으로 추가](#)할 수 있습니다.
- Amazon CodeCatalyst Bitbucket 리포지토리 확장 프로그램은 [Bitbucket Cloud Premium 액세스 제한](#)과 호환됩니다. 특정 IP 주소에 대한 액세스를 제한하도록 Bitbucket Cloud Premium 작업 영역을 구성할 때 [IP 주소 세트의 IP 주소 또는 네트워크 블록을 허용 목록에 추가](#)할 수도 있습니다.
- Amazon CodeCatalyst GitLab 리포지토리 확장 프로그램은 [GitLab IP 주소 제한](#)과 호환됩니다. 특정 IP 주소에 대한 액세스를 제한하도록 GitLab Premium 또는 Ultimate 그룹을 구성할 때 [IP 주소 세트의 IP 주소 또는 네트워크 블록을 허용 목록에 추가](#)할 수도 있습니다.

CodeCatalyst IP 주소가 타사 리포지토리의 허용 목록에 없는 경우 Amazon CodeCatalyst 앱은 타사 리포지토리에 액세스할 수 없습니다. 자세한 내용은 [타사 리포지토리 확장 프로그램에 사용되는 IP 주소](#) 섹션을 참조하세요.

타사 리포지토리 확장 프로그램에 사용되는 IP 주소

다음 IP 주소는 타사 확장 프로그램에서 타사 리소스에 액세스하는 데 사용됩니다.

- GitHub 리포지토리:

```
us-west-2
  52.32.242.246
  54.148.176.49
  35.164.118.94
eu-west-1
  34.241.64.10
  34.246.255.80
  3.248.38.7
```

- Bitbucket 리포지토리 및 GitLab 리포지토리:

```
us-west-2
  35.160.210.199
  54.71.206.108
  54.71.36.205
eu-west-1
  34.242.64.82
  52.18.37.201
  54.77.75.62
```

워크플로 실패 시 타사 병합 차단

GitHub 또는 Bitbucket 리포지토리를 CodeCatalyst에 연결한 후 풀 요청에 CodeCatalyst 워크플로를 추가할 수 있습니다. 마찬가지로 GitLab 프로젝트 리포지토리를 CodeCatalyst에 연결한 후 병합 요청에 CodeCatalyst 워크플로를 추가할 수 있습니다. 특정 커밋에서 하나 이상의 워크플로 실행이 발생할 수 있으며 CodeCatalyst의 각 워크플로 실행 상태도 GitHub, Bitbucket 또는 GitLab의 커밋 상태의 일부로 반영됩니다. 새 커밋이 푸시되면 새 커밋에 대한 새 워크플로 [실행 상태](#)가 GitHub, Bitbucket 또는 GitLab에 반영됩니다. 커밋에 대해 워크플로를 다시 실행하면 새 워크플로 실행 상태가 해당 커밋 및 워크플로의 이전 상태를 재정의합니다.

GitHub 또는 Bitbucket에서 브랜치 보호 규칙을 설정하여 풀 요청 병합을 차단하거나 GitLab에서 최신 커밋의 워크플로 실행 상태가 실패할 때 병합 요청을 차단할 수 있습니다. 브랜치 보호 규칙을 사용하면 최신 커밋 상태가 GitHub, Bitbucket 또는 GitLab에서 풀 요청을 병합하는 기능에 영향을 미칩니다.

워크플로에 대한 자세한 내용은 [워크플로 실행](#) 및 [트리거를 사용하여 워크플로 실행 자동 시작](#) 섹션을 참조하세요.

사용 중인 타사 리포지토리 제공업체에 따라 다음을 참조하세요.

- GitHub 리포지토리: GitHub의 [상태 확인](#) 및 [보호 브랜치](#) 설명서.
- Bitbucket 리포지토리: Bitbucket의 [Bitbucket Cloud에서 브랜치 권한 사용](#) 및 [브랜치 권한으로 제어](#) 설명서.
- GitLab 리포지토리: GitLab의 [자동 병합](#) 및 [보호 브랜치](#) 설명서.

Jira 문제를 CodeCatalyst 풀 요청에 연결

CodeCatalyst 소스 리포지토리에서 생성된 풀 요청을 Jira 문제에 연결할 수 있습니다. Jira 문제를 연결하면 해당 문제가 풀 요청의 속성으로 표시됩니다. 따라서, 풀 요청 이벤트, 워크플로 이벤트 및 배포 이벤트가 Jira로 전송되어 Jira 문제에 추가됩니다. 풀 요청은 하나 이상의 Jira 문제에 연결할 수 있습니다. GitHub과 같은 타사 리포지토리에 있는 요청이 아니라, CodeCatalyst 소스 리포지토리에 있는 풀 요청만 연결할 수 있습니다. Jira 문제를 풀 요청에 연결하려면, 먼저 Jira 프로젝트를 CodeCatalyst 프로젝트에 연결해야 합니다. Jira 프로젝트를 CodeCatalyst 프로젝트에 연결하는 방법에 대한 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결](#) 섹션을 참조하세요.

Note

CodeCatalyst 프로젝트에서 두 개의 브랜치가 있는 소스 리포지토리 없이는 풀 요청을 생성할 수 없습니다. 풀 요청에 대한 자세한 내용은 [CodeCatalyst의 풀 요청 작업](#)을 참조하세요.

Jira 문제를 CodeCatalyst 풀 요청에 연결하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 프로젝트로 이동합니다.
3. 탐색 창에서 코드를 선택한 다음, 풀 요청을 선택합니다.
4. 풀 요청 생성을 선택하여 풀 요청 세부 정보를 입력합니다.
5. 소스 리포지토리 드롭다운 메뉴에서 풀 요청을 연결할 소스 리포지토리를 선택합니다.
6. 소스 브랜치 드롭다운 메뉴에서 검토하려는 변경 사항이 포함된 브랜치를 선택합니다.
7. 대상 브랜치 드롭다운 메뉴에서 검토된 변경 사항을 병합할 브랜치를 선택합니다.

8. 풀 요청 제목 텍스트 입력 필드에 풀 요청의 제목을 입력합니다.
9. Jira 문제에 대한 문제 연결 - 선택적 필드를 선택하고, 드롭다운을 선택한 다음 연결된 Jira 프로젝트에서 추가하려는 Jira 문제를 검색합니다.
10. 풀 요청에 추가하려는 Jira 문제를 선택합니다.
11. 생성을 선택하여 풀 리퀘스트를 생성합니다.

Jira 문제를 CodeCatalyst 풀 요청에 연결하면, 풀 요청 요약에 사용할 수 있습니다. 요약에는 워크플로 실행, 연결된 문제, 필수 검토자, 선택적 검토자 및 작성자가 포함됩니다.

Note

Jira 문제와 관련된 담당자 및 작성자 정보는 CodeCatalyst에서 사용할 수 없습니다.

풀 요청을 연결한 후, 동기화된 CodeCatalyst 프로젝트와 Jira 프로젝트로 CodeCatalyst의 업데이트를 Jira 프로젝트에 반영합니다. Jira에서 볼 때 연결된 풀 요청의 상태와 풀 요청과 관련된 모든 워크플로 이벤트는 Jira 문제에 표시됩니다. Jira에서 CodeCatalyst 이벤트 보기에 대한 자세한 내용은 [Jira 문제에서 CodeCatalyst 이벤트 보기](#) 섹션을 참조하세요.

Jira 문제에서 CodeCatalyst 이벤트 보기

풀 요청의 요약 상태 및 연결된 CodeCatalyst 워크플로 이벤트의 상태는 CodeCatalyst 프로젝트와 Jira 프로젝트가 연결되어 있는 경우 Jira 문제에 반영됩니다. 예를 들어, CodeCatalyst에서 풀 요청을 닫거나 병합하는 경우 상태 업데이트가 Jira 문제에 반영됩니다. CodeCatalyst 풀 요청과 관련된 CodeCatalyst 워크플로 CI/CD 이벤트는 동기화되므로 워크플로 실행이 성공하면 Jira 문제에도 전송됩니다.

Jira 문제에서 CodeCatalyst 이벤트 보기

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 프로젝트로 이동합니다.
3. CodeCatalyst 탐색 창에서 코드, 풀 요청을 선택한 다음 Jira 프로젝트에서 보려는 Jira 문제가 있는 풀 요청을 선택합니다.
4. 추가 정보 창에서 Jira 프로젝트에서 보려는 Jira 이슈를 선택합니다.
5. Jira 프로젝트의 세부 정보 창에서 개발에 나열된 풀 요청을 선택하여 풀 요청의 세부 정보를 확인합니다.

6. (선택 사항) 최신 빌드를 보려면 빌드 탭을 선택합니다.
7. (선택 사항) 개발 상태를 보려면 배포 탭을 선택합니다.

CodeCatalyst에서 코드, 문제, 프로젝트 및 사용자 검색

CodeCatalyst의 검색 표시줄 또는 전용 검색 결과 창을 사용하여 CodeCatalyst 내 코드, 문제, 프로젝트 및 사용자를 검색합니다.

검색 창에 이름, 설명 및 상태와 같은 쿼리를 입력하여 스페이스 및 프로젝트 전체에서 리소스를 찾을 수 있습니다. 검색 쿼리 언어를 사용하여 검색 쿼리를 구체화할 수도 있습니다.

주제

- [검색 쿼리 구체화](#)
- [검색 작업 시 고려 사항](#)
- [검색 가능한 필드 참조](#)

검색하려면

1. 상단 탐색 모음의 검색 창에 검색 쿼리를 입력합니다.
2. (선택 사항) CodeCatalyst 검색 쿼리 언어를 사용하여 검색 쿼리를 구체화합니다. 자세한 내용은 [검색 쿼리 구체화](#) 섹션을 참조하세요.
3. 다음 중 하나를 수행합니다.
 - 현재 진행 중인 프로젝트 내에서 리소스를 검색하려면 이 프로젝트를 선택합니다.
 - 현재 있는 스페이스의 모든 프로젝트 내에서 리소스를 검색하려면 이 스페이스를 선택합니다.
4. 다음 중 하나를 수행하여 전용 검색 결과 창에서 검색 결과를 봅니다.
 - 빠른 검색 결과 창 하단에서 project-name | space-name의 모든 결과 보기를 선택하여 모든 검색 결과를 봅니다.
 - Enter를 눌러 모든 검색 결과를 봅니다.

Tip

표시 이름 또는 사용자 이름 앞에 @ 기호를 사용하여 풀 요청 주석 또는 설명 또는 문제 주석 또는 설명에서 다른 프로젝트 사용자를 언급합니다. 문제 또는 코드 파일 이름 앞에 @ 기호를 사용하여 문제 또는 코드 파일과 같은 리소스에 연결할 수도 있습니다.

검색 쿼리 구체화

검색 후 원하는 내용을 찾을 수 없는 경우 CodeCatalyst 특수 쿼리 언어를 사용하여 검색을 세분화할 수 있습니다. 개별 필드에는 문자 제한이 없지만, 전체 쿼리에서는 1,024자로 제한됩니다.

주제

- [유형별 구체화](#)
- [필드별 구체화](#)
- [부울 연산자를 사용한 구체화](#)
- [프로젝트별 구체화](#)

유형별 구체화

검색 범위를 특정 유형의 정보로 구체화하려면 검색에 *type:result-type*를 포함하세요. 여기서 *##*은 code, issue, project 또는 user 입니다.

예시:

- type:code AND java - “java”가 포함된 코드 관련 필드에 코드 결과를 표시합니다.

자세한 내용은 [코드 필드](#) 섹션을 참조하세요.

- type:issue AND Bug - “버그”가 포함된 문제 관련 필드에 문제 결과를 표시합니다.

자세한 내용은 [문제 필드](#) 섹션을 참조하세요.

- type:user AND MaryMajor - “MaryMajor”가 포함된 사용자 관련 필드에 사용자 결과를 표시합니다.

자세한 내용은 [사용자 필드](#) 섹션을 참조하세요.

- type:project AND Datafeeder - “Datafeeder”가 포함된 프로젝트 결과를 표시합니다.

자세한 내용은 [프로젝트 필드](#) 섹션을 참조하세요.

필드별 구체화

검색 범위를 특정 필드로 구체화하려면 검색에 *field-name:query*를 포함하세요. 여기서 *## ###* title, username, projectdescription, 등이고 *##*는 검색하려는 텍스트입니다. 필드 목록은 [검색 가능한 필드 참조](#) 섹션을 참조하세요. 괄호를 사용하여 여러 쿼리를 검색할 수 있습니다.

예시:

- `title:bug` - 제목에 “버그”가 포함된 결과를 표시합니다.
- `username:John` - 사용자 이름에 “John”이 포함된 결과를 표시합니다.
- `project:DataFeeder` - “DataFeeder” 프로젝트의 결과를 표시합니다. 쿼리는 대/소문자를 구분하지 않습니다.
- `description:overview` - 설명에 “개요”가 포함된 결과를 표시합니다.

부울 연산자를 사용한 구체화

검색 구문에 제약 조건을 지정하려면 부울 연산자 AND, OR 및 NOT을 사용할 수 있습니다. 여러 구문을 나열하는 경우, CodeCatalyst는 기본적으로 해당 구문을 OR과 조인합니다. 괄호를 사용하여 검색 구문을 그룹화할 수 있습니다.

- `exception AND type:code` - “예외”에 대한 코드 결과만 표시합니다.
- `path:README.md AND repo:ServerlessAPI` - 리포지토리 이름이 “ServerlessAPI”인 “README.md”를 사용하여 경로에 대한 결과를 표시합니다.
- `buildspec.yml AND (repo:ServerlessAPI OR ServerlessWebApp)` - 리포지토리가 “ServerlessAPI” 또는 “ServerlessWebApp”인 “buildspec.yml”에 대한 결과를 표시합니다.
- `path:java NOT (path:py OR path:ts)` - 경로에 “java”가 포함되어 있지만 “py” 또는 “ts”가 포함되어 있지 않은 결과를 표시합니다.

프로젝트별 구체화

검색 범위를 특정 프로젝트로 구체화하려면 검색에 *project:name AND query*를 포함합니다. 여기서 *###* 검색하려는 프로젝트이고 *##*는 검색하려는 콘텐츠입니다.

- `project:name AND query` - 경로에 쿼리와 프로젝트 이름이 포함된 결과를 표시합니다.

검색 작업 시 고려 사항

지연된 콘텐츠 업데이트 - 이름 변경 또는 문제 재할당과 같은 콘텐츠 업데이트가 검색 결과에 반영되는 데 몇 분 정도 걸릴 수 있습니다. 코드 기반 마이그레이션과 같은 대규모 업데이트가 검색 결과에 나타나는 데 더 오래 걸릴 수 있습니다.

특수 문자 이스케이프 처리 - 검색 쿼리에서 + - & & || ! () { } [] ^ " ~ * ? : \와 같은 특수 문자를 특별히 고려해야 합니다. 특수 문자는 쿼리에 영향을 주지 않으므로 특수 문자를 제거하거나 이스케이프 처리해야 합니다. 문자를 이스케이프 처리하려면 앞에 백슬래시(\)를 추가합니다. 예를 들어, 검색 쿼리 [Feature]는 Feature 또는 \[Feature\]여야 합니다.

검색 범위 좁히기 - 검색은 대/소문자를 구분하지 않습니다. 소문자로만 검색하면 쿼리에서 대소문자 변경에 따른 단어 분할이 일어나지 않습니다. 예를 들어 MyService를 쿼리하거나 MyService만 쿼리하려면, my 또는 service만 포함된 결과를 피하기 위해 myservice를 쿼리하는 방법을 고려하세요.

검색은 기본적으로 OR을 기준으로 단어와 단어의 일부를 조인합니다. 예를 들어, new function은 new 및 function을 모두 포함하는 결과를 반환할 수 있으며, 결과는 new 또는 function로만 반환할 수 있습니다. 후자를 방지하려면 여러 단어를 AND와 결합합니다. 예를 들어, new AND function을 검색합니다.

기본 브랜치 - 검색은 소스 리포지토리의 기본 브랜치에 대한 최신 커밋의 코드 결과만 반환합니다. 다른 브랜치 또는 커밋에서 코드를 찾으려면 [리포지토리를 로컬로 복제](#)하거나, [개발 환경에서 브랜치를 열거나](#), [CodeCatalyst UI에서 브랜치 및 세부 정보를 보는](#) 방법을 고려합니다. 기본 브랜치를 변경하면 검색으로 확인할 수 있는 파일이 업데이트됩니다. 자세한 내용은 [리포지토리의 기본 브랜치 관리](#) 섹션을 참조하세요.

Important

CodeCatalyst는 연결된 리포지토리에 대한 기본 브랜치의 변경 사항 감지를 지원하지 않습니다. 연결된 리포지토리의 기본 브랜치를 변경하려면 먼저 CodeCatalyst에서 연결을 해제하고 기본 브랜치를 변경한 다음 다시 연결해야 합니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리](#), [Bitbucket 리포지토리](#), [GitLab 프로젝트 리포지토리](#) 및 [Jira 프로젝트 연결](#) 섹션을 참조하세요.

리포지토리를 연결하기 전에 항상 최신 버전의 확장 프로그램을 사용하는 것이 좋습니다.

검색 가능한 필드 참조

CodeCatalyst는 검색 쿼리를 입력할 때 다음 필드를 검색합니다. 별칭은 고급 쿼리 언어로 필드를 참조하는 데 사용할 수 있는 또 다른 이름입니다.

코드 필드

필드	별칭	설명
branchName	branch	코드 파일이 있는 브랜치의 이름입니다.
코드	N/A	검색과 일치하는 소스 코드의 일부를 나타내는 코드 조각 형태의 코드 내용에 대한 정보입니다.
commitId	N/A	가장 마지막으로 업데이트된 상태의 반환된 코드 파일에 있는 커밋의 커밋 ID입니다. branchName 에 지정된 브랜치 이름의 최신 커밋 ID일 수도 있고 아닐 수도 있습니다.
commitMessage	N/A	가장 마지막으로 업데이트된 상태의 반환된 코드 파일에 있는 커밋의 커밋 메시지입니다. branchName 에 지정된 브랜치 이름의 최신 커밋의 커밋 메시지를일 수도 있고 아닐 수도 있습니다. 커밋 메시지가 제공되지 않은 경우, 이 값은 빈 문자열이 됩니다.
filePath	경로	이 코드 파일의 파일 경로입니다.
lastUpdatedBy	N/A	코드 파일을 마지막으로 업데이트한 CodeCatalyst 사용자입니다. 사용자 이름을 사용할 수 없는 경우, 이 값은 Git 구성 파일에 구성된 사용자의 이메일 주소를 사용합니다.

필드	별칭	설명
lastUpdatedById	N/A	코드 파일을 마지막으로 업데이트한 사용자의 고유한 시스템 생성 ID입니다. 사용자 ID를 사용할 수 없는 경우, 이 값은 사용자의 이메일 주소일 수 있습니다.
lastUpdatedTime	N/A	코드 파일이 포함된 커밋으로 검색 데이터를 마지막으로 업데이트한 시간[협정 세계시(UTC) 타임스탬프 기준]
projectId	N/A	작업의 고유한 시스템 생성 ID입니다.
projectName	projectNames, project	코드 파일이 커밋된 소스 리포지토리가 포함된 프로젝트의 이름을 표시합니다.
repositoryId	repold	리포지토리의 고유한 시스템 생성 ID
repositoryName	repository, repo	코드 파일이 커밋된 소스 리포지토리의 이름을 표시합니다.

문제 필드

필드	별칭	설명
assigneelds	assigneeld	문제에 할당된 사용자의 고유한 시스템 생성 ID입니다.
assignees	assignee	문제에 할당된 사용자의 이름입니다.

필드	별칭	설명
createdBy	N/A	문제를 생성한 사용자의 이름을 표시합니다.
createdByld	N/A	문제를 생성한 사용자의 고유한 시스템 생성 ID입니다.
createdTime	N/A	문제가 생성된 시간[협정 세계시(UTC) 타임스탬프 기준]
설명	N/A	문제에 대한 설명입니다.
isArchived	archived	보관된 상태에서 문제를 생성할지 여부를 나타내는 부울 값입니다.
isBlocked	blocked	문제가 차단됨으로 표시되는지 여부를 나타내는 부울 값입니다.
labelIds	labelId	문제에 대한 레이블의 고유한 시스템 생성 ID입니다.
lastUpdatedBy	N/A	문제를 마지막으로 업데이트한 사용자의 이름을 표시합니다.
lastUpdatedByld	N/A	문제를 마지막으로 업데이트한 사용자의 고유한 시스템 생성 ID입니다.
lastUpdatedTime	N/A	문제가 마지막으로 업데이트된 시간[협정 세계시(UTC) 타임스탬프 기준].
priority	N/A	문제가 할당된 경우 문제의 우선순위입니다.

필드	별칭	설명
projectId	N/A	작업의 고유한 시스템 생성 ID입니다.
projectName	projectNames, project	이 문제를 찾을 수 있는 프로젝트입니다.
shortId	N/A	문제에 대한 단축된 자동 증가 식별자입니다.
status	N/A	문제가 백로그 또는 보드의 열에 있는지 여부를 나타내는 문제의 상태입니다.
statusId	N/A	상태의 시스템 식별자입니다.
title	N/A	문제의 제목입니다.

프로젝트 필드

필드	별칭	Description
설명	N/A	객체 설명입니다.
lastUpdatedTime	N/A	프로젝트 메타데이터가 마지막으로 업데이트된 시간[협정 세계시(UTC) 타임스탬프 기준].
projectName	project	스페이스에 있는 프로젝트의 이름입니다.
projectPath	N/A	프로젝트 생성 중에 정의된 프로젝트의 URL 라우팅이 가능한 이름입니다. 프로젝트 이름이 필요한 URL에 사용됩니다.

사용자 필드

필드	별칭	설명
displayName	N/A	CodeCatalyst에서 사용자에게 사용되는 이름입니다. 표시 이름은 고유하지 않습니다.
이메일	N/A	사용자의 이메일 주소입니다.
lastUpdatedTime	N/A	사용자 메타데이터가 마지막으로 업데이트된 시간[협정 세계시(UTC) 타임스탬프 기준].
userName	사용자 이름	사용자가 CodeCatalyst에 가입할 때 선택한 사용자 이름입니다. 표시 이름과 달리 사용자 이름은 변경할 수 없습니다.

Amazon CodeCatalyst 문제 해결

다음 정보는 CodeCatalyst의 일반적인 문제를 해결하는 데 도움이 될 수 있습니다. Amazon CodeCatalyst 상태 보고서를 사용하여 경험에 영향을 미칠 수 있는 서비스 문제가 있는지 확인할 수도 있습니다.

주제

- [일반적인 액세스 문제 해결](#)
- [지원 문제 관련 문제 해결](#)
- [Amazon CodeCatalyst의 일부 또는 전부를 사용할 수 없습니다](#)
- [CodeCatalyst에서 프로젝트를 생성할 수 없습니다](#)
- [사용자 이름이 잘려서 새 사용자로 BID 스페이스에 액세스할 수 없거나 새 SSO 사용자로 추가할 수 없습니다](#)
- [SSO 사용자를 새 사용자로 페더레이션 스페이스에 추가하면 중복된 사용자가 생성됩니다](#)
- [CodeCatalyst에서 피드백을 제출하고 싶습니다.](#)
- [소스 리포지토리 문제 해결](#)
- [프로젝트 및 블루프린트 문제 해결](#)
- [개발 환경 문제 해결](#)
- [워크플로 관련 문제 해결](#)
- [문제 관련 문제 해결](#)
- [CodeCatalyst의 검색 문제 해결](#)
- [확장 프로그램 관련 문제 해결](#)
- [스페이스와 연결된 계정의 문제 해결](#)
- [Amazon CodeCatalyst와 AWS SDK 또는 AWS CLI 간의 문제 해결](#)

일반적인 액세스 문제 해결

암호를 잊어버렸습니다

문제: AWS Builder ID 및 Amazon CodeCatalyst에 사용하는 암호를 잊어버렸습니다.

가능한 해결 방법: 이 문제를 해결하는 가장 쉬운 방법은 암호를 재설정하는 것입니다.

1. [Amazon CodeCatalyst](#)를 열고 이메일 주소를 입력합니다. 그런 다음 계속을 선택합니다.

2. Forgot Password?를 선택합니다.
3. 암호를 변경할 수 있는 링크가 포함된 이메일을 보내드리겠습니다. 계정으로 이메일이 오지 않았으면 스팸 폴더를 확인하세요.

Amazon CodeCatalyst의 일부 또는 전부를 사용할 수 없습니다

문제: CodeCatalyst 콘솔로 이동하거나 링크를 따랐지만 오류가 발생했습니다.

가능한 해결 방법: 이 문제의 가장 일반적인 이유는 초대받지 않은 프로젝트 또는 스페이스에 대한 링크를 따르거나 서비스에 일반적인 가용성 문제가 있기 때문입니다. [상태 보고서](#)를 확인하여 서비스에 알려진 문제가 있는지 확인합니다. 그렇지 않은 경우 프로젝트 또는 스페이스에 초대된 사람에게 연락하여 다른 초대를 요청합니다. 프로젝트 또는 스페이스에 초대되지 않은 경우 등록하고 [자체 스페이스 및 프로젝트를 생성](#)할 수 있습니다.

CodeCatalyst에서 프로젝트를 생성할 수 없습니다

문제: 프로젝트를 생성하려고 하지만 프로젝트 생성 버튼이 사용 불가능으로 표시되거나 오류 메시지가 표시됩니다.

가능한 해결 방법: 이 문제의 가장 일반적인 이유는 스페이스 관리자 역할이 없는 AWS Builder ID로 콘솔에 로그인했기 때문입니다. 스페이스에서 프로젝트를 만들려면 이 역할이 있어야 합니다.

이 역할이 있고 버튼이 사용 가능한 것으로 표시되지 않는 경우 서비스에 일시적인 문제가 있을 수 있습니다. 브라우저를 새로 고치고 다시 시도하세요.

지원 문제 관련 문제 해결

Amazon CodeCatalyst용 지원에 액세스할 때 오류가 발생합니다

문제: Amazon CodeCatalyst용 지원 옵션을 선택하면 다음과 같은 오류 메시지가 표시됩니다.

Unable to assume role

To access support cases, you must add the role
AWSRoleForCodeCatalystSupport to the AWS ## that is the billing account for
the space.

가능한 해결 방법: 스페이스의 청구 계정인 AWS 계정에 필요한 역할을 추가합니다. 스페이스의 청구 계정으로 지정된 계정은 AWSRoleForCodeCatalystSupport 역할 및

AmazonCodeCatalystSupportAccess 관리형 정책을 사용합니다. 자세한 내용은 [계정 및 스페이스의 AWSRoleForCodeCatalystSupport 역할 생성](#) 섹션을 참조하세요.

 Note

AWS Builder ID는 인증된 별칭에 대한 지원만 받을 수 있으며 CodeCatalyst 권한에 따라 리소스에 대한 지원만 받을 수 있습니다. 계정 및 청구 지원은 스페이스의 모든 사용자가 사용할 수 있습니다. 그러나 빌더는 CodeCatalyst에서 권한이 있는 리소스 및 정보에 대해서만 지원을 받을 수 있습니다.

내 스페이스에 대한 기술 지원 사례를 생성할 수 없습니다

문제: 내 스페이스에 대한 기술 지원 사례를 생성할 수 없습니다.

해결 방법: 스페이스 내 사용자가 기술 지원 사례를 생성하려면 스페이스 청구 계정에 Business Support 또는 Enterprise Support 플랜을 추가해야 합니다. 스페이스 관리자에게 스페이스 청구 계정에 지원 플랜을 추가하도록 요청하거나 <https://repost.aws/> 방문하여 AWS 커뮤니티에 문의하세요.

지원 사례에 대한 내 계정이 CodeCatalyst의 내 스페이스에 더 이상 연결되지 않습니다

문제: 지원 사례에 대한 내 계정이 CodeCatalyst의 내 스페이스에 더 이상 연결되지 않습니다.

해결 방법: 스페이스 관리자 역할이 있는 사용자가 스페이스 청구 계정을 전환하면 지원 계획과 모든 관련 케이스가 해당 스페이스에서 연결 해제됩니다. Amazon CodeCatalyst용 지원에 대해 이전 스페이스 청구 계정과 연결된 지원 사례는 더 이상 에서 볼 수 없습니다. 해당 청구 계정의 루트 사용자는 AWS Management Console에서 이전 사례를 보고 해결할 수 있으며 다른 사용자가 이전 사례를 보고 해결할 수 있도록 지원에 대한 IAM 권한을 설정할 수 있습니다. AWS Management Console을 통해 이전 스페이스 청구 계정에서 CodeCatalyst에 대한 기술 지원을 계속 받을 수 없지만 지원 플랜이 취소될 때까지 다른 서비스에 대한 기술 지원을 받을 수 있습니다.

자세한 내용은 지원 사용 설명서의 [Updating, resolving, and reopening your case](#)를 참조하세요.

Amazon CodeCatalyst용 지원에서 다른 AWS 서비스에 대한 지원 사례를 열 수 없습니다.

문제: CodeCatalyst용 AWS 서비스에서 다른 지원에 대한 지원 사례를 열 수 없습니다.

가능한 해결 방법: CodeCatalyst용 지원에서 CodeCatalyst 지원 사례만 열 수 있습니다. CodeCatalyst에서 다른 AWS, Amazon 또는 기타 타사 서비스로 배포된 서비스 또는 리소스에 대한 지원이 필요한 경우, AWS Management Console 또는 타사 서비스 지원 채널을 통해 케이스를 만들어야 합니다. 자세한 내용은 지원 사용 설명서의 [Creating support cases and case management](#)를 참조하세요.

Amazon CodeCatalyst의 일부 또는 전부를 사용할 수 없습니다

문제: CodeCatalyst 콘솔로 이동하거나 링크를 따랐지만 오류가 발생했습니다.

가능한 해결 방법: 이 문제의 가장 일반적인 이유는 초대받지 않은 프로젝트 또는 스페이스에 대한 링크를 따르거나 서비스에 일반적인 가용성 문제가 있기 때문입니다. [상태 보고서](#)를 확인하여 서비스에 알려진 문제가 있는지 확인합니다. 그렇지 않은 경우 프로젝트 또는 스페이스에 초대된 사람에게 연락하여 다른 초대를 요청합니다. 프로젝트 또는 스페이스에 초대되지 않은 경우 등록하고 [자체 스페이스 및 프로젝트를 생성](#)할 수 있습니다.

CodeCatalyst에서 프로젝트를 생성할 수 없습니다

문제: 프로젝트를 생성하려고 하지만 프로젝트 생성 버튼이 사용 불가능으로 표시되거나 오류 메시지가 표시됩니다.

가능한 해결 방법: 이 문제의 가장 일반적인 이유는 스페이스 관리자 역할이 없는 AWS Builder ID로 콘솔에 로그인했기 때문입니다. 스페이스에서 프로젝트를 만들려면 이 역할이 있어야 합니다.

이 역할이 있고 버튼이 사용 가능한 것으로 표시되지 않는 경우 서비스에 일시적인 문제가 있을 수 있습니다. 브라우저를 새로 고치고 다시 시도하세요.

사용자 이름이 잘려서 새 사용자로 BID 스페이스에 액세스할 수 없거나 새 SSO 사용자로 추가할 수 없습니다

문제: CodeCatalyst는 100자 후의 사용자 이름을 잘라내기 때문에 일부 사용자 이름이 동일해 보일 수 있습니다. CodeCatalyst 스페이스에 액세스하는 새 사용자로서 다음과 같이 스페이스 유형에 따라 이 문제가 발생합니다.

- CodeCatalyst에 로그인하는 데 사용할 AWS Builder ID가 있습니다. 스페이스에 로그인하려고 하면 사용자 이름이 유효하지 않다는 메시지가 표시됩니다.
- ID 페더레이션을 지원하는 CodeCatalyst 스페이스의 페더레이션 ID 관리자입니다. IAM Identity Center의 SSO 사용자 및 그룹에 새 사용자를 추가할 때 해당 사용자가 유효하지 않다는 메시지가 표시됩니다.

가능한 해결 방법: CodeCatalyst에 로그인하거나 주어진 잘린 사용자 이름을 가진 SSO 사용자로 스페이스에 추가되는 첫 번째 사용자가 성공합니다. AWS Builder ID로 가입하거나 이후 IAM Identity Center에 추가되는 사용자는 이름이 중복된 것으로 나타나므로 로그인할 수 없습니다. 스페이스 유형에 따라 다음 중 하나를 수행합니다.

- AWS Builder ID 스페이스에 로그인하려면 다른 사용자 이름으로 가입합니다.
- IAM Identity Center에서 새 사용자를 추가하려면 다른 사용자 이름을 가진 사용자를 추가합니다.

Note

사용자 이름이 잘린 것처럼 보이더라도 CodeCatalyst는 이름이 잘린 사용자 이름에 영향을 받지 않는 방식으로 ID에 매핑합니다. 그러나 잘린 사용자 이름과 동일한 사용자 이름을 생성하면, 동일한 스페이스 또는 IAM Identity Center 애플리케이션과 연결된 다른 사용자가 이미 해당 잘린 사용자 이름으로 CodeCatalyst에 가입한 경우 해당 사용자 이름을 사용할 수 없습니다.

SSO 사용자를 새 사용자로 페더레이션 스페이스에 추가하면 중복된 사용자가 생성됩니다

문제: CodeCatalyst SSO 사용자가 CodeCatalyst 스페이스에 추가되었다가 제거되면 사용자 이름의 재사용이 시도되는 문제가 발생할 수 있습니다. 이 경우 사용자가 다시 생성되었음에도 불구하고 해당 사용자의 중복이 허용되지 않아 다음과 유사한 오류가 발생할 수 있습니다.

Unable to assume role

To access support cases, you must add the role `AWSRoleForCodeCatalystSupport` to the AWS ## that is the billing account for the space.

가능한 해결 방법: 기존 IDC 사용자가 삭제되고 새 사용자가 동일한 사용자 이름으로 다시 생성되면 이전 사용자와 충돌하는 사용자 이름으로 인해 새 사용자가 로그인할 수 없습니다. 사용자 이름이 SSO 사용자로 스페이스에 추가된 후에는 해당 이름을 다시 사용할 수 없습니다. AWS Builder ID로 가입하거나 이후 IAM Identity Center에 추가되는 사용자는 이름이 중복된 것으로 나타나므로 로그인할 수 없습니다.

스페이스 유형에 따라 다음 중 하나를 수행합니다.

- AWS Builder ID 스페이스에 로그인하려면 다른 사용자 이름으로 가입합니다.
- IAM Identity Center에서 새 사용자를 추가하려면 다른 사용자 이름을 가진 사용자를 추가합니다.

CodeCatalyst에서 피드백을 제출하고 싶습니다.

문제: CodeCatalyst에서 버그가 발견되어 피드백을 제출하려고 합니다.

가능한 해결 방법: CodeCatalyst에서 직접 피드백을 제출할 수 있습니다.

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 탐색 창에서 피드백 제공을 선택합니다.
3. 드롭다운 메뉴에서 피드백 유형을 선택하고 피드백을 입력합니다.

소스 리포지토리 문제 해결

다음 정보는 CodeCatalyst의 소스 리포지토리와 관련된 일반적인 문제 해결에 도움이 될 수 있습니다.

주제

- [내 스페이스의 스토리지가 최대 용량에 도달하여 경고 또는 오류가 표시됩니다](#)
- [Amazon CodeCatalyst 소스 리포지토리를 복제하거나 푸시하려고 할 때 오류가 발생합니다.](#)
- [Amazon CodeCatalyst 소스 리포지토리를 커밋하거나 푸시하려고 할 때 오류가 발생합니다.](#)
- [프로젝트를 위한 소스 리포지토리가 필요합니다.](#)
- [소스 리포지토리가 완전히 새 것이지만 커밋이 포함되어 있습니다.](#)
- [다른 브랜치를 기본 브랜치로 사용하겠습니다.](#)
- [풀 요청의 활동에 대한 이메일을 받고 있습니다.](#)
- [개인 액세스 토큰\(PAT\)을 잊어버렸습니다.](#)
- [풀 요청에는 예상 변경 사항이 표시되지 않습니다.](#)
- [풀 요청은 병합할 수 없음 상태를 표시합니다.](#)

내 스페이스의 스토리지가 최대 용량에 도달하여 경고 또는 오류가 표시됩니다

문제: CodeCatalyst에서 하나 이상의 소스 리포지토리에 코드를 커밋하려고 하지만 오류가 표시됩니다. 콘솔의 소스 리포지토리 페이지에 스페이스의 스토리지 한도에 도달했다는 메시지가 표시됩니다.

가능한 해결 방법: 프로젝트 또는 스페이스의 역할에 따라 소스 리포지토리 중 하나 이상의 크기를 줄이거나, 사용하지 않는 소스 리포지토리를 삭제하거나, 청구 계층을 스토리지가 더 많은 계층으로 변경할 수 있습니다.

- 프로젝트의 소스 리포지토리 크기를 줄이려면 사용하지 않는 브랜치를 삭제할 수 있습니다. 자세한 내용은 [브랜치 삭제](#) 및 [기고자 역할](#) 섹션을 참조하세요.
- 스페이스의 전체 스토리지를 줄이려면 미사용 소스 리포지토리를 삭제할 수 있습니다. 자세한 내용은 [소스 리포지토리 삭제](#) 및 [프로젝트 관리자 역할](#) 섹션을 참조하세요.
- 스페이스에 사용할 수 있는 스토리지 양을 늘리려면 청구 계층을 스토리지가 더 많은 계층으로 변경할 수 있습니다. 자세한 내용은 Amazon CodeCatalyst 관리자 안내서의 [Changing your CodeCatalyst billing tier](#)를 참조하세요.

Amazon CodeCatalyst 소스 리포지토리를 복제하거나 푸시하려고 할 때 오류가 발생합니다.

문제: 로컬 컴퓨터 또는 통합 개발 환경(IDE)에 소스 리포지토리를 복제하려고 하면 권한 오류가 발생합니다.

가능한 해결 방법: AWS Builder ID에 대한 개인 액세스 토큰(PAT)이 없거나, PAT로 자격 증명 관리 시스템을 구성하지 않았거나, PAT가 만료되었을 수 있습니다. 다음 해결 방법 중 하나 이상을 시도해 보세요.

- 개인용 액세스 토큰(PAT)을 생성합니다. 자세한 내용은 [개인 액세스 토큰을 사용하여 사용자 리포지토리 액세스 권한 부여](#) 섹션을 참조하세요.
- 소스 리포지토리가 포함된 프로젝트에 초대를 수락했는지, 그리고 그 프로젝트의 멤버인지 확인합니다. 해당 프로젝트의 활성 멤버가 아닌 경우 소스 리포지토리를 복제할 수 없습니다. 콘솔에 로그인하여 소스 리포지토리를 복제하려는 스페이스와 프로젝트로 이동합니다. 스페이스의 프로젝트 목록에서 프로젝트를 볼 수 없는 경우 해당 프로젝트의 멤버가 아니거나 해당 프로젝트에 대한 초대 수락을 수락하지 않은 것입니다. 자세한 내용은 [초대 수락 및 AWS Builder ID 생성](#) 섹션을 참조하세요.
- 복제 명령의 형식이 올바르고 AWS Builder ID가 포함되어 있는지 확인합니다. 예:

```
https://LiJuan@git.us-west-2.codecatalyst.aws/v1/ExampleCorp/MyExampleProject/MyExampleRepo
```

- AWS CLI를 사용하여 AWS Builder ID와 연결된 PAT가 있고 만료되지 않았는지 확인합니다. PAT가 없거나 PAT가 만료된 경우 PAT를 생성합니다. 자세한 내용은 [개인 액세스 토큰을 사용하여 사용자 리포지토리 액세스 권한 부여](#) 섹션을 참조하세요.

- 소스 리포지토리의 코드를 로컬 리포지토리나 IDE에 복제하는 대신 개발 환경을 만들어서 작업해 봅니다. 자세한 내용은 [개발 환경 생성](#) 섹션을 참조하세요.

Amazon CodeCatalyst 소스 리포지토리를 커밋하거나 푸시하려고 할 때 오류가 발생합니다.

문제: 소스 리포지토리로 푸시하려고 하면 권한 오류가 발생합니다.

가능한 해결 방법: 프로젝트에 코드 변경을 커밋하고 푸시할 수 있는 역할이 없을 수 있습니다. 소스 리포지토리에 변경 사항을 푸시하려는 프로젝트에서 역할을 봅니다. 자세한 내용은 [멤버 및 해당 프로젝트 역할 목록 가져오기](#) 및 [사용자 역할로 액세스 권한 부여](#) 섹션을 참조하세요.

변경 사항을 커밋하고 푸시할 수 있는 역할을 가지고 있다면, 변경 사항을 커밋하려는 브랜치에 코드 변경 사항을 푸시하지 못하도록 하는 브랜치 규칙이 설정되어 있을 수 있습니다. 대신 브랜치를 생성하고 코드를 해당 브랜치로 푸시해 봅니다. 자세한 내용은 [브랜치 규칙을 사용하여 브랜치에 대한 작업 관리](#) 섹션을 참조하세요.

프로젝트를 위한 소스 리포지토리가 필요합니다.

문제: 프로젝트에 소스 리포지토리가 없거나 프로젝트에 다른 소스 리포지토리가 필요합니다.

가능한 해결 방법: 일부 프로젝트는 리소스 없이 생성됩니다. 프로젝트의 멤버인 경우 CodeCatalyst에서 해당 프로젝트에 대한 소스 리포지토리를 생성할 수 있습니다. 스페이스 관리자 역할을 가진 사람이 GitHub 리포지토리를 설치하고 GitHub 계정에 연결하는 경우 프로젝트 관리자 역할이 있는 경우 사용할 수 있는 GitHub 리포지토리에 연결하여 프로젝트에 추가할 수 있습니다. 자세한 내용은 [Creating a source repository](#) 및 [Linking a source repository](#)를 참조하세요.

소스 리포지토리가 완전히 새 것이지만 커밋이 포함되어 있습니다.

문제: 소스 리포지토리를 방금 생성했습니다. 비어 있어야 하지만 커밋, 브랜치 및 README.md 파일이 있어야 합니다.

가능한 해결 방법: 정상적인 동작입니다. CodeCatalyst의 모든 소스 리포지토리에는 기본 브랜치를 main으로 설정하는 초기 커밋이 포함되어 있으며, 샘플 코드(해당 리포지토리가 샘플 코드를 포함하는 블루프린트를 사용하는 프로젝트를 위해 생성된 경우) 또는 리포지토리 README 파일에 대한 템플릿 마크다운 파일이 포함되어 있습니다. 콘솔 및 Git 클라이언트에서 추가 브랜치를 생성할 수 있습니다. 콘솔에서 파일을 생성 및 편집하고 개발 환경 및 Git 클라이언트에서 파일을 삭제할 수 있습니다.

다른 브랜치를 기본 브랜치로 사용하겠습니다.

문제: 소스 리포지토리에는 main이라는 기본 브랜치가 있지만 다른 브랜치를 기본 브랜치로 사용하려고 합니다.

가능한 해결 방법: CodeCatalyst의 소스 리포지토리에서 기본 브랜치를 변경하거나 삭제할 수 없습니다. 추가 브랜치를 생성하고 워크플로의 소스 작업에서 해당 브랜치를 사용할 수 있습니다. GitHub 리포지토리를 연결하여 프로젝트의 리포지토리로 사용하도록 선택할 수도 있습니다.

풀 요청의 활동에 대한 이메일을 받고 있습니다.

문제: 풀 요청 활동에 대한 이메일 알림을 등록하거나 구성하지 않았지만 어쨌든 수신하고 있습니다.

가능한 해결 방법: 풀 요청 활동에 대한 이메일 알림이 자동으로 전송됩니다. 자세한 내용은 [Amazon CodeCatalyst에서 풀 요청을 사용하여 코드 검토](#) 섹션을 참조하세요.

개인 액세스 토큰(PAT)을 잊어버렸습니다.

문제: 소스 리포지토리의 코드를 복제, 푸시 및 가져오는 데 PAT를 사용했지만 토큰의 값을 잃어 CodeCatalyst 콘솔에서 찾을 수 없습니다.

가능한 해결 방법: 이 문제를 해결하는 가장 빠른 방법은 다른 PAT를 생성하고 이 새 PAT를 사용하도록 자격 증명 관리자 또는 IDE를 구성하는 것입니다. PAT는 생성할 때만 표시됩니다. 이 값을 잃으면 검색할 수 없습니다. 자세한 내용은 [개인 액세스 토큰을 사용하여 사용자 리포지토리 액세스 권한 부여](#) 섹션을 참조하세요.

풀 요청에는 예상 변경 사항이 표시되지 않습니다.

문제: 풀 요청을 생성했지만 소스 브랜치와 대상 브랜치 간에 예상되는 변경 사항이 보이지 않습니다.

가능한 해결 방법: 이는 여러 가지 문제로 인해 발생할 수 있습니다. 다음 해결 방법 중 하나 이상을 시도해 보세요.

- 이전 개정 간의 변경 사항을 검토하거나 최신 변경 사항을 확인하지 못할 수 있습니다. 브라우저를 새로 고치고 보려는 개정 간의 비교를 선택했는지 확인합니다.
- 콘솔에 풀 요청의 모든 변경 사항을 표시할 수 있는 것은 아닙니다. 예를 들어, 콘솔에서 Git 하위 모듈을 볼 수 없으므로 풀 요청에서 하위 모듈의 차이를 볼 수 없습니다. 일부 차이는 표시하기에 너무 클 수 있습니다. 자세한 내용은 [CodeCatalyst의 소스 리포지토리 할당량](#) 및 [파일 보기](#) 섹션을 참조하세요.

- 풀 요청은 병합 기반과 선택한 개정 간의 차이를 표시합니다. 풀 요청을 생성할 때 표시되는 차이는 소스 브랜치의 팁과 대상 브랜치의 팁 간의 차이입니다. 풀 요청이 생성되면 표시된 차이는 개정과 병합 기반 간에 발생합니다. 병합 기반은 개정이 생성될 때 대상 브랜치의 팁인 커밋입니다. 병합 기반은 개정 간에 변경될 수 있습니다. Git의 차이점 및 병합 기반에 대한 자세한 내용은 Git 설명서의 [git-merge-base](#)를 참조하세요.

풀 요청은 병합할 수 없음 상태를 표시합니다.

문제: 풀 요청을 병합하려고 하지만 상태가 병합할 수 없음으로 표시됩니다.

가능한 해결 방법: 하나 이상의 문제로 인해 발생할 수 있습니다.

- 풀 요청에 필요한 모든 검토자는 풀 요청을 승인해야 병합할 수 있습니다. 이름 옆에 시계 아이콘이 있는 모든 검토자의 필수 검토자 목록을 검토합니다. 시계 아이콘은 검토자가 풀 요청을 승인하지 않았음을 나타냅니다.

Note

풀 요청을 승인하기 전에 프로젝트에서 필수 검토자를 제거한 경우 풀 요청을 병합할 수 없습니다. 풀 요청을 닫고 새 풀 요청을 생성합니다.

- 소스 브랜치와 대상 브랜치 간에 병합 충돌이 발생할 수 있습니다. CodeCatalyst는 가능한 모든 Git 병합 전략 및 옵션을 지원하지 않습니다. 개발 환경에서 병합 충돌에 대해 브랜치를 평가하거나 리포지토리를 복제하고 IDE 또는 Git 도구를 사용하여 병합 충돌을 찾아 해결할 수 있습니다. 자세한 내용은 [풀 요청 병합](#) 섹션을 참조하세요.

프로젝트 및 블루프린트 문제 해결

이 섹션은 Amazon CodeCatalyst에서 프로젝트 및 블루프린트로 작업하는 동안 발생할 수 있는 몇 가지 일반적인 문제를 해결하는 데 도움이 될 수 있습니다.

apache-maven-3.8.6에 대한 종속성이AWS Fargate 블루프린트에서 누락된 Java API

문제: AWS Fargate 블루프린트가 있는 Java API에서 생성된 프로젝트의 경우, apache-maven-3.8.6 종속성이 누락되어 오류가 발생하여 워크플로가 실패합니다. 다음 예시와 유사한 출력으로 워크플로가 실패합니다.

```

Step 8/25 : RUN wget https://dlcdn.apache.org/maven/maven-3/3.8.6/binaries/apache-
maven-3.8.6-bin.tar.gz -P /tmp
---> Running in 1851ce6f4d1b
[91m--2023-03-10 01:24:55-- https://dlcdn.apache.org/maven/maven-3/3.8.6/binaries/
apache-maven-3.8.6-bin.tar.gz
[0m[91mResolving dlcdn.apache.org (dlcdn.apache.org)...
[0m[91m151.101.2.132, 2a04:4e42::644
Connecting to dlcdn.apache.org (dlcdn.apache.org)|151.101.2.132|:443...
[0m[91mconnected.
[0m[91mHTTP request sent, awaiting response... [0m[91m404 Not Found
2023-03-10 01:24:55 ERROR 404: Not Found.
[0mThe command '/bin/sh -c wget https://dlcdn.apache.org/maven/maven-3/3.8.6/binaries/
apache-maven-3.8.6-bin.tar.gz -P /tmp' returned a non-zero code: 8
[Container] 2023/03/10 01:24:55 Command failed with exit status 8

```

해결 방법: 다음 단계를 따라 블루프린트 Dockerfile을 업데이트합니다.

1. 검색 창에서 apache-maven-3.8.6를 입력하여 AWS Fargate Java API와 블루프린트로 생성된 프로젝트 내에서 Dockerfile을 찾습니다.
2. 기본 이미지로 maven:3.9.0-amazoncorretto-11를 사용하도록 Dockerfile(/static-assets/app/Dockerfile)을 업데이트하고 apache-maven-3.8.6 패키지에 대한 종속성을 제거합니다.
3. (권장 사항) 또한 Maven 힙 크기를 6GB로 업데이트하는 것을 권장합니다.

다음은 Dockerfile의 예입니다.

```

FROM maven:3.9.0-amazoncorretto-11 AS builder

COPY ./pom.xml ./pom.xml
COPY src ./src/

ENV MAVEN_OPTS='-Xmx6g'

RUN mvn -Dmaven.test.skip=true clean package

FROM amazoncorretto:11-alpine

COPY --from=builder target/CustomService-0.0.1.jar CustomService-0.0.1.jar
EXPOSE 80

```



```
CMD ["java","-jar","-Dspring.profiles.active=prod","/CustomerService-0.0.1.jar", "-server.port=80"]
```

Amazon CodeGuru에 대한 권한 오류로 최신 3계층 웹 애플리케이션 블루프린트 OnPullRequest 워크플로 실패

문제: 프로젝트의 워크플로를 실행하려고 하면 다음 메시지와 함께 워크플로가 실행되지 않습니다.

```
Failed at codeguru_codereview: The action failed during runtime. View the action's logs for more details.
```

해결 방법: 이 작업 실패의 한 가지 가능한 원인은 연결된에서 CodeCatalyst가 사용하는 서비스 역할의 버전에 codeguru_codereview 작업이 성공적으로 실행되기 위해 필요한 권한이 AWS 계정 없는 IAM 역할 정책의 권한 누락 때문일 수 있습니다. 이 문제를 해결하려면 서비스 역할이 업데이트하여 필요한 권한을 갖추거나, 워크플로에 사용되는 서비스 역할을 Amazon CodeGuru 및 Amazon CodeGuru Reviewer에 필요한 권한이 있는 역할로 변경해야 합니다. 다음 단계를 사용하여 역할을 찾고 역할 정책 권한을 업데이트하여 워크플로가 성공적으로 실행되도록 합니다.

Note

이러한 단계는 CodeCatalyst의 다음 워크플로에 적용됩니다.

- CodeCatalyst에서 최신 3계층 웹 애플리케이션 블루프린트로 생성된 프로젝트에 제공되는 OnPullRequest 워크플로입니다.
- Amazon CodeGuru 또는 Amazon CodeGuru Reviewer에 액세스하는 작업으로 CodeCatalyst의 프로젝트에 워크플로가 추가되었습니다.

각 프로젝트에는 CodeCatalyst의 프로젝트에 AWS 계정 연결된에서 제공하는 역할 및 환경을 사용하는 작업이 포함된 워크플로가 포함되어 있습니다. 해당 작업 및 지정된 정책이 포함된 워크플로는 디렉터리 `/.codecatalyst/workflow`의 소스 리포지토리에 저장됩니다. 기존 워크플로에 새 역할 ID를 추가하지 않는 한 워크플로 YAML을 수정할 필요는 없습니다. YAML 템플릿 요소와 형식에 대한 정보는 [워크플로 YAML 정의](#) 섹션을 참조하세요.

다음은 역할 정책을 편집하고 워크플로 YAML을 확인하기 위해 따라야 하는 상위 단계입니다.

워크플로 YAML에서 역할 이름을 참조하고 정책을 업데이트하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다. 프로젝트로 이동합니다.
3. CI/CD를 선택한 다음 워크플로를 선택합니다.
4. OnPullRequest라는 제목의 워크플로를 선택합니다. 정의 탭을 선택합니다.
5. 워크플로 YAML에서 codeguru_codereview 작업 아래에 있는 Role: 필드에 역할 이름을 기록해 둡니다. 이 역할에는 IAM에서 수정할 정책이 있습니다. 다음 예시는 역할 이름을 보여줍니다.

The screenshot displays the Amazon CodeCatalyst console interface for a workflow named 'OnPullRequest'. The top navigation bar includes tabs for CI/CD, Workflows, Environments, Compute, Secrets, and Change tracking. The workflow is associated with the 'mysfitscvb63' workspace and the 'test' environment. The workflow definition is shown in a YAML format on the right side of the console. The workflow definition includes the following details:

- Name:** OnPullRequest
- SchemaVersion:** "1.0"
- Triggers:**
 - Type: PULLREQUEST
 - Branches: main
- Events:**
 - OPEN
 - REVISION
- Actions:**
 - codeguru_codereview:
 - Identifier: aws/build@v1
 - Inputs:
 - Sources:
 - WorkflowSource
 - Variables:
 - Name: AWS_DEFAULT_REGION
 - Value: us-west-2
 - Outputs:
 - Artifacts:
 - Name: codereview
 - Files:
 - ./code-guru/*
 - Configuration:**
 - Steps:
 - Run: curl -OL https://github.com/aws/aws-codeguru-cli
 - Run: unzip aws-codeguru-cli.zip
 - Run: export PATH=\$PATH:./aws-codeguru-cli/bin
 - Run: aws-codeguru-cli --root-dir ./src --no-prompt
 - Environment:**
 - Name: development
 - Connections:
 - Name: connection-11-30
 - Role: CodeCatalystPreviewDevelopmentAdministrator-j

6. 다음 중 하나를 수행하세요.

- (권장 사항) Amazon CodeGuru 및 Amazon CodeGuru Reviewer에 필요한 권한으로 프로젝트에 연결된 서비스 역할을 업데이트합니다. 역할에는 고유한 식별자가 추가된 CodeCatalystWorkflowDevelopmentRole-*spaceName* 이름이 있습니다. 역할 및 역할 정

책에 대한 자세한 내용은 [CodeCatalystWorkflowDevelopmentRole-*spaceName* 서비스 역할 이해](#) 섹션을 참조하세요. 다음 단계로 진행하여 IAM에서 정책을 업데이트합니다.

Note

역할 및 정책을 AWS 계정 사용하여에 대한 AWS 관리자 액세스 권한이 있어야 합니다.

- 워크플로에 사용되는 서비스 역할을 Amazon CodeGuru 및 Amazon CodeGuru Reviewer에 필요한 권한이 있는 역할로 변경하거나 필요한 권한이 있는 새 역할을 생성합니다.
7. 에 로그인 AWS Management Console 하고 <https://console.aws.amazon.com/iam/> IAM 콘솔을 엽니다.
- IAM 콘솔에서 5단계에서 CodeCatalystPreviewDevelopmentRole과 같은 역할을 찾습니다.
8. 5단계의 역할에서 codeguru-reviewer:* 및 codeguru:* 권한을 포함하도록 권한 정책을 변경합니다. 이러한 권한을 추가한 후에는 권한 정책이 다음과 비슷해야 합니다.
 9. 정책을 수정한 후 CodeCatalyst로 돌아가 워크플로 실행을 다시 시작합니다.

여전히 문제가 해결되지 않았나요?

[Amazon CodeCatalyst](#)로 이동하거나 [지원 피드백 양식](#)을 작성할 수 있습니다. 요청 정보 섹션에서 어떻게 도와드릴까요에 Amazon CodeCatalyst 고객이라는 점을 포함합니다. 문제를 가장 효율적으로 해결할 수 있도록 최대한 자세하게 설명해 주세요.

개발 환경 문제 해결

다음 섹션을 참조하여 개발 환경과 관련된 문제를 해결합니다. 개발 환경에 대한 자세한 내용은 [CodeCatalyst에서 개발 환경으로 코드 작성 및 수정](#) 섹션을 참조하세요.

주제

- [할당량 문제로 인해 내 개발 환경 생성에 실패했습니다.](#)
- [개발 환경에서 리포지토리의 특정 브랜치로 변경 사항을 푸시할 수 없습니다.](#)
- [내 개발 환경이 재개되지 않습니다.](#)
- [내 개발 환경 연결이 해제됩니다.](#)
- [VPC 연결 개발 환경이 실패합니다.](#)
- [프로젝트가 있는 디렉터리를 찾을 수 없습니다.](#)
- [SSH를 통해 개발 환경에 연결할 수 없습니다.](#)

- [로컬 SSH 구성이 누락되어 SSH를 통해 내 개발 환경에 연결할 수 없습니다.](#)
- [codecatalyst 프로파일에 대한 AWS Config에 문제가 있어 SSH를 통해 개발 환경에 연결할 수 없습니다.](#)
- [Single Sign-On 계정을 사용하여 CodeCatalyst에 로그인할 때는 개발 환경을 생성할 수 없습니다.](#)
- [IDE 문제 해결](#)
- [devfile 문제 해결](#)

할당량 문제로 인해 내 개발 환경 생성에 실패했습니다.

문제: CodeCatalyst에서 개발 환경을 생성하려고 하지만 오류가 발생했습니다. 콘솔의 개발 환경 페이지에 스페이스의 스토리지 한도에 도달했다는 메시지가 표시됩니다.

가능한 수정 사항: 프로젝트 또는 스페이스에서의 역할에 따라 하나 이상의 자체 개발 환경을 삭제하거나, 스페이스 관리자 역할이 있는 경우 다른 사용자가 생성한 미사용 개발 환경을 삭제할 수 있습니다. 또한 청구 계층을 더 많은 스토리지가 포함된 계층으로 변경할 수도 있습니다.

- 스토리지 한도를 보려면 Amazon CodeCatalyst 스페이스의 청구 탭을 보고 사용 할당량이 최대 허용 한도에 도달했는지 확인하세요. 할당량이 최대값에 도달한 경우 스페이스 관리자 역할을 가진 사람에게 문의하여 불필요한 개발 환경을 제거하거나 청구 계층 변경을 고려하세요.
- 더 이상 필요하지 않은 생성한 개발 환경을 제거하려면 [개발 환경 삭제](#) 섹션을 참조하세요.

문제가 계속되고 IDE에 오류가 발생하면 개발 환경을 생성할 수 있는 CodeCatalyst 역할이 있는지 확인합니다. 스페이스 관리자 역할, 프로젝트 관리자 역할 및 기고자 역할 모두 개발 환경을 생성할 수 있는 권한이 있습니다. 자세한 내용은 [사용자 역할로 액세스 권한 부여](#) 섹션을 참조하세요.

개발 환경에서 리포지토리의 특정 브랜치로 변경 사항을 푸시할 수 없습니다.

문제: 소스 리포지토리의 브랜치에 개발 환경의 코드 변경 사항을 커밋하고 푸시하려고 하지만 오류가 발생합니다.

가능한 해결 방법: 프로젝트 또는 스페이스의 역할에 따라 코드를 푸시하여 프로젝트의 리포지토리를 소싱할 권한이 없을 수 있습니다. 스페이스 관리자 역할, 프로젝트 관리자 역할 및 기고자 역할은 모두 프로젝트의 리포지토리에 코드를 푸시할 수 있는 권한이 있습니다.

기고자 역할이 있지만 특정 브랜치에 코드를 푸시할 수 없는 경우 해당 역할이 있는 사용자가 해당 브랜치에 코드를 푸시하지 못하도록 브랜치 규칙이 특정 브랜치에 대해 구성되어 있을 수 있습니다. 변경

사항을 다른 브랜치로 푸시하거나 브랜치를 생성한 다음 코드를 해당 브랜치로 푸시해 보세요. 자세한 내용은 [브랜치 규칙을 사용하여 브랜치에 대한 작업 관리](#) 섹션을 참조하세요.

내 개발 환경이 재개되지 않습니다.

문제: 중지한 후 내 개발 환경이 재개되지 않았습니다.

가능한 수정 사항: 문제를 해결하려면 Amazon CodeCatalyst 스페이스의 청구 탭을 보고 사용량 할당량이 최대 한도에 도달했는지 확인하세요. 할당량이 최대 한도에 도달한 경우 스페이스 관리자에게 문의하여 청구 계층을 높입니다.

내 개발 환경 연결이 해제됩니다.

문제: 사용 중 개발 환경 연결이 끊겼습니다.

가능한 수정 사항: 문제를 해결하려면 인터넷 연결을 확인합니다. 인터넷에 연결되지 않은 경우 개발 환경에서 인터넷에 연결하고 작업을 재개합니다.

VPC 연결 개발 환경이 실패합니다.

문제: VPC 연결을 개발 환경에 연결했는데 오류가 발생했습니다.

가능한 수정 사항: Docker는 동일한 브리지 네트워크에 연결된 컨테이너가 통신할 수 있도록 하는 브리지 네트워크라는 링크 계층 디바이스를 사용합니다. 기본 브리지는 일반적으로 172.17.0.0/16 서브넷을 컨테이너 네트워킹에 사용합니다. 환경 인스턴스의 VPC 서브넷이 Docker가 이미 사용하는 것과 동일한 주소 범위를 사용하는 경우, IP 주소 충돌이 발생할 수 있습니다. Amazon VPC 및 동일한 IPv4 CIDR 주소 블록 Docker 사용으로 인해 발생하는 IP 주소 충돌을 해결하려면 172.17.0.0/16와 다른 CIDR 블록을 구성합니다.

Note

(기존 VPC 또는 서브넷의 IP 주소 범위는 변경할 수 없습니다.)

프로젝트가 있는 디렉터리를 찾을 수 없습니다.

문제: 프로젝트가 있는 디렉터리를 찾을 수 없습니다.

가능한 수정 사항: 프로젝트를 찾으려면 디렉터리를 /projects로 변경합니다. 프로젝트를 찾을 수 있는 디렉터리입니다.

SSH를 통해 개발 환경에 연결할 수 없습니다.

SSH를 통해 개발 환경에 대한 연결을 해결하려면 `-vvv` 옵션을 사용하여 `ssh` 명령을 실행하여 문제를 해결하는 방법에 대한 자세한 정보를 표시할 수 있습니다.

```
ssh -vvv codecatalyst-dev-env=<space-name>=<project-name>=<dev-environment-id>
```

로컬 SSH 구성이 누락되어 SSH를 통해 내 개발 환경에 연결할 수 없습니다.

로컬 SSH 구성(`~/.ssh/config`)이 누락되었거나 Host `codecatalyst-dev-env*` 섹션의 내용이 오래된 경우 SSH를 통해 개발 환경에 연결할 수 없습니다. 이 문제를 해결하려면 Host `codecatalyst-dev-env*` 섹션을 삭제하고 SSH 액세스 모달에서 첫 번째 명령을 다시 실행합니다. 자세한 내용은 [SSH를 사용하여 개발 환경에 연결](#) 섹션을 참조하세요.

codecatalyst 프로파일에 대한 AWS Config에 문제가 있어 SSH를 통해 개발 환경에 연결할 수 없습니다.

codecatalyst 프로파일의 AWS Config(`~/.aws/config`)가 [CodeCatalyst AWS CLI 에서를 사용하도록 설정](#)에 설명된 것과 일치하는지 확인합니다. 그렇지 않은 경우 codecatalyst의 프로파일을 삭제하고 SSH 액세스 모달에서 첫 번째 명령을 다시 실행합니다. 자세한 내용은 [SSH를 사용하여 개발 환경에 연결](#) 섹션을 참조하세요.

Single Sign-On 계정을 사용하여 CodeCatalyst에 로그인할 때는 개발 환경을 생성할 수 없습니다.

문제: CodeCatalyst 콘솔에 SSO 사용자로 로그인하면 스페이스에 개발 환경을 생성하도록 선택하면 알 수 없는 예외 오류가 발생합니다. 개발 환경을 생성하고 AWS Cloud9 같은 액세스용 IDE를 선택하면 다음과 유사한 문제가 발생합니다.

- CodeCatalyst 콘솔의 개발 환경 페이지에는 FAILED 상태의 목록에 개발 환경이 표시됩니다.
- 다음과 유사한 오류 메시지가 표시됩니다.

An unknown exception happened

We encountered an unknown exception when launching your Dev Environment. Mention your Dev Environment id `error_message_ID` if you want to report or need any help.

수정 방법:

Active Directory가 ID 제공업체로 사용되는 스페이스의 사용자는 개발 환경을 사용할 수 없습니다. 스페이스 관리자는 대체 ID 제공업체를 사용하여 IAM Identity Center와 같은 개발 환경에 액세스할 수 있습니다. ID 페더레이션을 지원하는 스페이스 계획에 대한 자세한 내용은 CodeCatalyst 관리자 안내서의 [ID 페더레이션을 지원하는 스페이스 계획](#)을 참조하세요.

IDE 문제 해결

CodeCatalyst의 IDEs와 관련된 문제를 해결하려면 다음 섹션을 참조하세요. IDE에 대한 자세한 내용은 [IDE에서 개발 환경 생성](#) 섹션을 참조하세요.

주제

- [AWS Cloud9에서 런타임 이미지 버전이 일치하지 않습니다.](#)
- [AWS Cloud9에서 /projects/projects 파일에 액세스할 수 없습니다.](#)
- [AWS Cloud9에서 사용자 지정 devfile을 사용하여 개발 환경을 시작할 수 없습니다.](#)
- [AWS Cloud9에 문제가 있습니다.](#)
- [JetBrains에서는 CodeCatalyst를 통해 개발 환경에 연결할 수 없습니다.](#)
- [내 IDE에 AWS Toolkit을 설치할 수 없습니다.](#)
- [IDE에서 개발 환경을 시작할 수 없습니다.](#)

AWS Cloud9에서 런타임 이미지 버전이 일치하지 않습니다.

AWS Cloud9는 다양한 버전의 프런트엔드 자산과 백엔드 런타임 이미지를 사용합니다. 다른 버전을 사용하면 Git 확장 프로그램 및 AWS Toolkit이 잘못 작동할 수 있습니다. 문제를 해결하려면 개발 환경 대시보드로 이동하여 개발 환경을 중지한 다음 다시 시작합니다. API를 사용하여 문제를 해결하려면 UpdateDevEnvironment API를 사용하여 런타임을 업데이트합니다. 자세한 내용은 Amazon CodeCatalyst API 참조의 [UpdateDevEnvironment](#)를 참조하세요.

AWS Cloud9에서 **/projects/projects** 파일에 액세스할 수 없습니다.

AWS Cloud9 편집기가 /projects/projects 디렉터리의 파일에 액세스할 수 없습니다. 문제를 해결하려면 AWS Cloud9 터미널을 사용하여 파일에 액세스하거나 다른 디렉터리로 이동합니다.

AWS Cloud9에서 사용자 지정 devfile을 사용하여 개발 환경을 시작할 수 없습니다.

devfile 이미지가 AWS Cloud9와 호환되지 않을 수 있습니다. 문제를 해결하려면 리포지토리 및 해당 개발 환경의 devfile을 검토하고 계속하려면 새 파일을 생성합니다.

AWS Cloud9에 문제가 있습니다.

다른 문제는 [AWS Cloud9 사용 설명서](#)의 문제 해결 섹션을 참조하세요.

JetBrains에서는 CodeCatalyst를 통해 개발 환경에 연결할 수 없습니다.

문제를 해결하려면 최신 버전의 JetBrains만 설치되어 있는지 확인합니다. 버전이 여러 개인 경우 이전 버전을 제거하고 IDE와 브라우저를 닫아 프로토콜 핸들러를 다시 등록합니다. 그런 다음 JetBrains를 열고 프로토콜 핸들러를 다시 등록합니다.

내 IDE에 AWS Toolkit을 설치할 수 없습니다.

VS 코드에 대한 이 문제를 해결하려면 [GitHub](#)에서 수동으로 AWS Toolkit for Visual Studio Code을 설치합니다.

JetBrains에 대한 이 문제를 해결하려면 [GitHub](#)에서 수동으로 AWS Toolkit for JetBrains을 설치합니다.

IDE에서 개발 환경을 시작할 수 없습니다.

VS 코드에 대한 이 문제를 해결하려면 최신 버전의 VS 코드 및 AWS Toolkit for Visual Studio Code이 설치되어 있는지 확인합니다. 최신 버전이 없는 경우 개발 환경을 업데이트하고 시작합니다. 자세한 내용은 [Amazon CodeCatalyst for VS Code](#) 사용 설명서를 참조하세요.

JetBrains에 대해 이 문제를 해결하려면 JetBrains 및 AWS Toolkit for JetBrains의 최신 버전이 설치되어 있는지 확인합니다. 최신 버전이 없는 경우 개발 환경을 업데이트하고 시작합니다. 자세한 내용은 [Amazon CodeCatalyst for JetBrains](#)를 참조하세요.

devfile 문제 해결

CodeCatalyst의 devfile과 관련된 문제를 해결하려면 다음 섹션을 참조하세요. devfile에 대한 자세한 내용은 [개발 환경용 devfile 구성](#) 섹션을 참조하세요.

주제

- [내 개발 환경은 사용자 지정 devfile에서 사용자 지정 이미지를 구현한 경우에도 기본 범용 devfile을 사용합니다.](#)
- [프로젝트가 기본 범용 devfile을 사용하여 내 개발 환경에서 빌드되지 않습니다.](#)
- [개발 환경용 리포지토리 devfile 이동](#)
- [devfile을 시작하는 데 문제가 있는 경우](#)

- [devfile 상태를 확인하는 방법을 잘 모르겠습니다.](#)
- [내 devfile이 최신 이미지에 제공된 도구와 호환되지 않습니다.](#)

내 개발 환경은 사용자 지정 devfile에서 사용자 지정 이미지를 구현한 경우에도 기본 범용 devfile을 사용합니다.

사용자 지정 devfile을 사용하는 개발 환경을 시작하는 동안 CodeCatalyst에 오류가 발생하면 개발 환경은 기본 범용 devfile로 기본 설정됩니다. 문제를 해결하려면 `/aws/mde/logs/devfile.log` 아래의 로그에서 정확한 오류를 확인할 수 있습니다. 로그(`/aws/mde/logs/devfileCommand.log`)에서 `postStart` 실행이 성공했는지 확인할 수도 있습니다.

프로젝트가 기본 범용 devfile을 사용하여 내 개발 환경에서 빌드되지 않습니다.

문제를 해결하려면 사용자 지정 devfile을 사용하지 않는지 확인합니다. 사용자 지정 devfile을 사용하지 않는 경우 프로젝트의 소스 리포지토리에서 `devfile.yaml` 파일을 보고 오류를 찾아 수정합니다.

개발 환경용 리포지토리 devfile 이동

`/projects/devfile.yaml`의 기본 devfile을 소스 코드 리포지토리로 이동할 수 있습니다.

devfile의 위치를 업데이트하려면 다음 명령을 사용합니다. `/aws/mde/mde start --location repository-name/devfile.yaml`

devfile을 시작하는 데 문제가 있는 경우

devfile을 시작하는 데 문제가 있는 경우 복구 모드로 전환되므로 환경에 계속 연결하고 devfile을 수정할 수 있습니다. 복구 모드에 있는 동안 `/aws/mde/mde status` 실행에는 devfile의 위치가 포함되지 않습니다.

```
{
  "status": "STABLE"
}
```

`/aws/mde/logs`의 로그에서 오류를 확인하고, devfile을 수정하고, 다시 `/aws/mde/mde start` 실행을 시도할 수 있습니다.

devfile 상태를 확인하는 방법을 잘 모르겠습니다.

`/aws/mde/mde status`를 실행하여 devfile 상태를 확인할 수 있습니다. 이 명령을 실행한 후 다음 중 하나가 표시될 수 있습니다.

- {"status": "STABLE", "location": "devfile.yaml" }

이는 devfile이 정확함을 나타냅니다.

- {"status": "STABLE" }

이는 devfile을 시작할 수 없고 복구 모드로 전환되었음을 나타냅니다.

/aws/mde/logs/devfile.log의 로그에서 정확한 오류를 확인할 수 있습니다.

로그(/aws/mde/logs/devfileCommand.log)에서 postStart 실행이 성공했는지 확인할 수도 있습니다.

자세한 내용은 [개발 환경에 대한 범용 devfile 이미지 지정](#) 섹션을 참조하세요.

내 devfile이 최신 이미지에 제공된 도구와 호환되지 않습니다.

개발 환경에서 devfile 또는 devfile postStart는 latest 도구에 특정 프로젝트에 필요한 도구가 없는 경우 실패할 수 있습니다. 문제를 해결하려면 다음을 수행합니다.

1. devfile로 이동합니다.
2. devfile에서 latest 대신 세분화된 이미지 버전으로 업데이트합니다. 내용은 다음처럼 보일 것입니다.

```
components:
  - container:
      image: public.ecr.aws/amazonlinux/universal-image:1.0
```

3. 업데이트된 devfile을 사용하여 새 개발 환경을 생성합니다.

워크플로 관련 문제 해결

다음 섹션을 참조하여 Amazon CodeCatalyst의 워크플로와 관련된 문제를 해결하세요. 워크플로에 대한 자세한 내용은 [워크플로를 사용하여 빌드, 테스트 및 배포](#) 섹션을 참조하세요.

주제

- ['Workflow is inactive' 메시지를 해결하려면 어떻게 해야 하나요?](#)
- ['Workflow definition has n errors' 오류를 해결하려면 어떻게 해야 하나요?](#)
- ['Unable to locate credentials' 및 'ExpiredToken' 오류를 해결하려면 어떻게 해야 하나요?](#)

- ['Unable to connect to the server' 오류를 해결하려면 어떻게 해야 하나요?](#)
- [시각적 편집기에서 CodeDeploy 필드가 누락된 이유는 무엇인가요?](#)
- [IAM 기능 오류를 해결하려면 어떻게 해야 하나요?](#)
- ['npm install' 오류를 해결하려면 어떻게 해야 하나요?](#)
- [여러 워크플로의 이름이 같은 이유는 무엇인가요?](#)
- [워크플로 정의 파일을 다른 폴더에 저장할 수 있나요?](#)
- [워크플로에 작업을 순차적으로 추가하려면 어떻게 해야 하나요?](#)
- [워크플로가 성공적으로 검증되었지만 런타임에 실패하는 이유는 무엇인가요?](#)
- [자동 검색이 내 작업에 대한 보고서를 찾지 못함](#)
- [성공 기준을 구성한 후 내 작업이 자동 검색된 보고서에 대한 작업이 실패해요.](#)
- [자동 검색이 내가 원하지 않는 보고서를 생성해요.](#)
- [자동 검색이 단일 테스트 프레임워크에 대해 작은 보고서를 많이 생성함](#)
- [CI/CD에 나열된 워크플로가 소스 리포지토리의 워크플로와 일치하지 않음](#)
- [워크플로를 생성하거나 업데이트할 수 없음](#)

'Workflow is inactive' 메시지를 해결하려면 어떻게 해야 하나요?

문제: CodeCatalyst 콘솔의 CI/CD, 워크플로에서 워크플로가 다음 메시지와 함께 나타납니다.

Workflow is inactive.

이 메시지는 워크플로 정의 파일에 현재 사용 중인 브랜치에 적용되지 않는 트리거가 포함되어 있음을 나타냅니다. 예를 들어 워크플로 정의 파일에 PUSH 브랜치를 참조하는 main 트리거가 포함되어 있지만 현재 특성 브랜치에 있을 수 있습니다. 특성 브랜치에서 변경한 내용은 main 브랜치에 적용되지 않고 main 브랜치에서 워크플로 실행을 시작하지 않으므로 CodeCatalyst는 브랜치에서 워크플로를 해제하고 Inactive로 표시합니다.

수정 방법:

특성 브랜치에서 워크플로를 시작하려면 다음을 수행하면 됩니다 .

- 특성 브랜치의 워크플로 정의 파일에서 Triggers 섹션의 Branches 속성을 제거하여 다음과 같이 표시되도록 합니다.

Triggers:

- Type: PUSH

이 구성을 사용하면 특성 브랜치를 포함한 모든 브랜치로 푸시할 때 트리거가 활성화됩니다. 트리거가 활성화되면 CodeCatalyst는 푸시하는 브랜치에 있는 워크플로 정의 파일과 소스 파일을 사용하여 워크플로 실행을 시작합니다.

- 특성 브랜치의 워크플로 정의 파일에서 Triggers 섹션을 제거하고 워크플로를 수동으로 실행합니다.
- 특성 브랜치의 워크플로 정의 파일에서 다른 브랜치(예시: main 등)가 아닌 특성 브랜치를 참조하도록 PUSH 섹션을 변경합니다.

Important

변경 내용을 다시 main 브랜치에 병합하지 않으려는 경우에는 이러한 변경 내용을 커밋하지 않도록 주의하세요.

워크플로 정의 파일 편집에 대한 자세한 내용은 [워크플로 생성](#) 섹션을 참조하세요.

트리거에 대한 자세한 내용은 [트리거를 사용하여 워크플로 실행 자동 시작](#) 주제를 참조하세요.

'Workflow definition has *n* errors' 오류를 해결하려면 어떻게 해야 하나요?

문제: 다음 오류 메시지 중 하나가 표시됩니다.

오류 1:

CI/CD, 워크플로 페이지의 워크플로 이름 아래에 다음과 같은 메시지가 표시됩니다.

Workflow definition has *n* errors

오류 2:

워크플로를 편집하는 동안 검증 버튼을 선택하면 CodeCatalyst 콘솔 상단에 다음 메시지가 표시됩니다.

The workflow definition has errors. Fix the errors and choose Validate to verify your changes.

오류 3:

워크플로의 세부 정보 페이지로 이동한 후 워크플로 정의 필드에 다음 오류가 표시됩니다.

n errors

수정 방법:

- CI/CD를 선택하고 워크플로를 선택한 다음 오류가 발생한 워크플로의 이름을 선택합니다. 상단 근처의 워크플로 정의 필드에서 오류에 대한 링크를 선택합니다. 페이지 하단에 오류에 대한 자세한 내용이 표시됩니다. 오류의 문제 해결 팁에 따라 문제를 해결합니다.
- 워크플로 정의 파일이 YAML 파일인지 확인합니다.
- 워크플로 정의 파일의 YAML 속성이 올바른 수준에서 중첩되어 있는지 확인합니다. 워크플로 정의 파일에서 속성을 중첩하는 방법을 확인하려면 [워크플로 YAML 정의](#) 섹션을 참조하거나 [워크플로에 작업 추가](#) 섹션에서 연결된 작업 설명서를 참조하세요.
- 별표(*) 및 기타 특수 문자가 올바르게 이스케이프 처리되었는지 확인합니다. 이스케이프 처리하려면 작은따옴표 또는 큰따옴표를 추가하세요. 예:

```
Outputs:
  Artifacts:
    - Name: myartifact
      Files:
        - "***/*"
```

워크플로 정의 파일의 특수 문자에 대한 자세한 내용은 [구문 지침 및 규칙](#) 섹션을 참조하세요.

- 워크플로 정의 파일의 YAML 속성이 올바른 대문자를 사용하는지 확인합니다. 대/소문자 규칙에 대한 자세한 내용은 [구문 지침 및 규칙](#) 섹션을 참조하세요. 각 속성의 올바른 대/소문자를 확인하려면 [워크플로 YAML 정의](#) 섹션을 참조하거나 [워크플로에 작업 추가](#) 섹션에서 연결된 작업 설명서를 참조하세요.
- 워크플로 정의 파일에 SchemaVersion 속성이 있고 올바른 버전으로 설정되어 있는지 확인합니다. 자세한 내용은 [SchemaVersion](#) 섹션을 참조하세요.
- 워크플로 정의 파일의 Triggers 섹션에 필요한 모든 속성이 포함되어 있는지 확인합니다. 필요한 속성을 확인하려면 [시각적 편집기](#)에서 트리거를 선택하고 누락된 정보가 있는 필드를 찾거나 [Triggers](#)에서 트리거 참조 문서를 참조하세요.
- 워크플로 정의 파일의 DependsOn 속성이 올바르게 구성되어 있고 순환 종속성을 유발하지 않는지 확인합니다. 자세한 내용은 [작업 순서 지정](#) 섹션을 참조하세요.
- 워크플로 정의 파일의 Actions 섹션에 적어도 하나의 작업이 포함되어 있는지 확인합니다. 자세한 내용은 [작업](#) 섹션을 참조하세요.

- 각 작업에 모든 필수 속성이 포함되어 있는지 확인합니다. 필요한 속성을 확인하려면 [시각적 편집기](#)에서 작업을 선택하고 정보가 누락된 필드를 찾거나 [워크플로에 작업 추가](#) 섹션에서 연결된 작업 설명서를 참조하세요.
- 모든 입력 아티팩트에 해당하는 출력 아티팩트가 있는지 확인합니다. 자세한 내용은 [출력 아티팩트 정의](#) 섹션을 참조하세요.
- 한 작업에서 정의된 변수를 다른 작업에서 사용할 수 있도록 내보내야 합니다. 자세한 내용은 [다른 작업에서 사용할 수 있도록 변수 내보내기](#) 섹션을 참조하세요.

'Unable to locate credentials' 및 'ExpiredToken' 오류를 해결하려면 어떻게 해야 하나요?

문제: [자습서: Amazon EKS에 애플리케이션 배포](#)를 진행하는 동안 개발 컴퓨터의 터미널 창에 다음 오류 메시지 중 하나 또는 둘이 표시됩니다.

Unable to locate credentials. You can configure credentials by running "aws configure".

ExpiredToken: The security token included in the request is expired

수정 방법:

이러한 오류는 AWS 서비스에 액세스하는 데 사용 중인 자격 증명이 만료되었음을 나타냅니다. 이 경우 `aws configure` 명령을 실행하지 않습니다. 대신 다음 지침에 따라 AWS 액세스 키와 세션 토큰을 새로 고치세요.

AWS 액세스 키 및 세션 토큰을 새로 고치려면 다음을 수행하세요.

1. Amazon EKS 자습서를 사용 중인 사용자의 AWS 액세스 포털 URL, 사용자 이름 및 암호 (codecatalyst-eks-user)가 있는지 확인합니다. 자습서의 [1단계: 개발 머신 설정](#)를 완료했다면 이러한 항목이 구성되었을 것입니다.

Note

이 정보가 없는 경우 IAM Identity Center의 codecatalyst-eks-user 세부 정보 페이지로 이동하여 암호 재설정, 일회용 암호 생성 [...]을 선택한 후 암호 재설정을 다시 선택하면 화면에 해당 정보가 표시됩니다.

2. 다음 중 하나를 수행합니다.

- AWS 액세스 포털 URL을 브라우저의 주소 표시줄에 붙여 넣습니다.

Or

- AWS 액세스 포털 페이지가 이미 로드된 경우 새로 고칩니다.
3. 아직 로그인하지 않은 경우 codecatalyst-eks-user의 사용자 이름과 암호로 로그인합니다.
 4. AWS 계정을 선택한 다음 codecatalyst-eks-user 사용자 및 권한 세트를 할당한 AWS 계정의 이름을 선택합니다.
 5. 권한 세트 이름(codecatalyst-eks-permission-set) 옆에서 명령줄 또는 프로그래밍 액세스를 선택합니다.
 6. 페이지 중간에 있는 명령을 복사합니다. 다음처럼 보일 것입니다.

```
export AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
export AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
export AWS_SESSION_TOKEN="session-token"
```

...여기서 *## ##*은 긴 무작위 문자열입니다.

7. 개발 시스템의 터미널 프롬프트에 명령을 붙여넣고 Enter 키를 누릅니다.

새 키와 세션 토큰이 로드됩니다.

이제 자격 증명을 새로 고쳤습니다. 이제 AWS CLI, eksctl 및 kubectl 명령이 작동할 것입니다.

'Unable to connect to the server' 오류를 해결하려면 어떻게 해야 하나요?

문제: [자습서: Amazon EKS에 애플리케이션 배포](#)에 설명된 자습서를 진행하는 동안 개발 시스템의 터미널 창에 다음과 유사한 오류 메시지가 표시됩니다.

```
Unable to connect to the server: dial tcp: lookup long-string.gr7.us-west-2.eks.amazonaws.com on 1.2.3.4:5: no such host
```

수정 방법:

이 오류는 일반적으로 kubectl 유틸리티가 Amazon EKS 클러스터에 연결하는 데 사용하는 자격 증명이 만료되었음을 나타냅니다. 이 문제를 해결하려면 터미널 프롬프트에 다음 명령을 입력하여 자격 증명을 새로 고칩니다.

```
aws eks update-kubeconfig --name codecatalyst-eks-cluster --region us-west-2
```

위치:

- *codecatalyst-eks-cluster*는 Amazon EKS 클러스터의 이름으로 대체됩니다.
- *us-west-2*는 클러스터가 배포된 AWS 리전으로 대체됩니다.

시각적 편집기에서 CodeDeploy 필드가 누락된 이유는 무엇인가요?

문제: [Amazon ECS에 배포](#) 작업을 사용 중인데 워크플로의 시각적 편집기에서 CodeDeploy AppSpec과 같은 CodeDeploy 필드가 보이지 않습니다. 이 문제는 서비스 필드에서 지정한 Amazon ECS 서비스가 블루/그린 배포를 수행하도록 구성되지 않았기 때문에 발생할 수 있습니다.

수정 방법:

- Amazon ECS에 배포 작업의 구성 탭에서 다른 Amazon ECS 서비스를 선택합니다. 자세한 내용은 [워크플로를 사용하여 Amazon ECS에 배포](#) 섹션을 참조하세요.
- 블루/그린 배포를 수행하기 위해 선택한 Amazon ECS 서비스를 구성합니다. 자세한 내용은 Amazon Elastic 컨테이너 서비스 개발자 안내서의 [CodeDeploy로 제어되는 블루/그린 배포](#)를 참조하세요.

IAM 기능 오류를 해결하려면 어떻게 해야 하나요?

문제: [CloudFormation 스택 배포](#) 작업을 사용 중인데 `##[error] requires capabilities: [capability-name]` 메시지가 CloudFormation 스택 배포 작업의 로그에 표시됩니다.

가능한 해결 방법: 다음 절차를 완료하여 워크플로 정의 파일에 기능을 추가합니다. IAM 기능에 대한 자세한 내용은 IAM 사용 설명서의 [CloudFormation 템플릿에서 IAM 리소스 승인](#)을 참조하세요.

Visual

시각적 편집기를 사용하여 IAM 기능을 추가하려면 다음을 수행하세요.

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.

5. 편집을 선택합니다.
6. 비주얼을 선택합니다.
7. 워크플로 다이어그램에서 CloudFormation 스택 배포 작업을 선택합니다.
8. 구성 탭을 선택합니다.
9. 하단에서 고급 - 선택 사항을 선택합니다.
10. 기능 드롭다운 목록에서 오류 메시지에 언급된 기능 옆에 있는 확인란을 선택합니다. 목록에서 기능을 사용할 수 없는 경우 YAML 편집기를 사용하여 추가합니다.
11. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
12. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.
13. 새 워크플로 실행이 자동으로 시작되지 않으면 워크플로를 수동으로 실행하여 변경 사항으로 오류가 수정되는지 확인합니다. 워크플로를 수동으로 실행하는 방법에 대한 자세한 내용 [워크플로 수동 실행 시작](#) 섹션을 참조하세요.

YAML

YAML 편집기를 사용하여 IAM 기능을 추가하려면 다음을 수행하세요.

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. 프로젝트를 선택합니다.
3. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
4. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
5. 편집을 선택합니다.
6. YAML을 선택합니다.
7. CloudFormation 스택 배포 작업에서 다음과 같이 capabilities 속성을 추가합니다.

```
DeployCloudFormationStack:
  Configuration:
    capabilities: capability-name
```

*capability-name*을 오류 메시지에 표시된 IAM 기능의 이름으로 바꿉니다. 여러 기능을 나열하려면 공백 없이 쉼표를 사용하세요. 자세한 내용은 '[스 CloudFormation 탭 배포' 작업 YAML](#)의 capabilities 속성에 관한 설명을 참조하세요.

8. (선택 사항) 커밋하기 전에 워크플로의 YAML 코드를 검증하려면 검증을 선택합니다.
9. 커밋을 선택하고 커밋 메시지를 입력한 다음 커밋을 다시 선택합니다.
10. 새 워크플로 실행이 자동으로 시작되지 않으면 워크플로를 수동으로 실행하여 변경 사항으로 오류가 수정되는지 확인합니다. 워크플로를 수동으로 실행하는 방법에 대한 자세한 내용 [워크플로 수동 실행 시작](#) 섹션을 참조하세요.

'npm install' 오류를 해결하려면 어떻게 해야 하나요?

문제: [AWS CDK 배포 작업](#) 또는 [AWS CDK 부트스트랩 작업](#)이 npm install 오류가 발생하여 실패합니다. 이 오류는 작업에서 액세스할 수 없는 프라이빗 노드 패키지 관리자(npm) 레지스트리에 AWS CDK 앱 종속성을 저장하고 있기 때문에 발생할 수 있습니다.

가능한 해결 방법: 다음 지침에 따라 추가 레지스트리 및 인증 정보로 AWS CDK 앱의 cdk.json 파일을 업데이트합니다.

시작하기 전 준비 사항

1. 인증 정보에 대한 보안 암호를 생성합니다. 명확한 텍스트에 해당하는 내용을 제공하는 대신 cdk.json 파일에서 이러한 보안 암호를 참조하게 됩니다. 보안 암호를 생성하려면 다음을 수행하세요.
 - a. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
 - b. 프로젝트를 선택합니다.
 - c. 탐색 창에서 CI/CD를 선택한 후, 보안 암호를 선택합니다.
 - d. 다음 속성을 가진 두 개의 보안 암호를 다시 생성합니다.

첫 번째 보안 암호	두 번째 보안 암호
이름: npmUsername 값: <i>npm-username</i> , 여기서 <i>npm-username</i> 은 프라이빗 npm 레지스트리에 인증하는 데 사용되는 사용자 이름입니다. (선택 사항) 설명: The username used to authenticate to the private npm registry.	이름: npmAuthToken 값: <i>npm-auth-token</i> , 여기서 <i>npm-auth-token</i> 은 프라이빗 npm 레지스트리에 인증하는 데 사용되는 액세스 토큰입니다. npm 액세스 토큰에 대한 자세한 내용은 npm 설명서의 액세스 토큰 정보를 참조하세요.

첫 번째 보안 암호	두 번째 보안 암호
	(선택 사항) 설명: The access token used to authenticate to the private npm registry.

보안 암호에 대한 자세한 내용은 [보안 암호를 사용하여 데이터 마스킹](#) 섹션을 참조하세요.

2. 보안 암호를 AWS CDK 작업에 환경 변수로 추가합니다. 이 작업은 변수가 실행될 때 변수를 실제 값으로 바꿉니다. 보안 암호를 추가하려면 다음을 수행하세요.
 - a. 탐색 창에서 CI/CD를 선택한 다음 워크플로를 선택합니다.
 - b. 워크플로의 이름을 선택합니다. 소스 리포지토리 또는 워크플로가 정의된 브랜치 이름을 기준으로 필터링하거나, 워크플로 이름 또는 상태를 기준으로 필터링할 수 있습니다.
 - c. 편집을 선택합니다.
 - d. 비주얼을 선택합니다.
 - e. 워크플로 다이어그램에서 AWS CDK 작업을 선택합니다.
 - f. 입력 탭을 선택합니다.
 - g. 다음 속성을 가진 두 변수를 추가합니다.

첫 번째 변수	두 번째 변수
이름: NPMUSER	이름: NPMTOKEN
값: <code>\${Secrets.npmUsername}</code>	값: <code>\${Secrets.npmAuthToken}</code>

이제 보안 암호에 대한 참조가 포함된 두 개의 변수가 생겼습니다.

이제 워크플로 정의 파일 YAML 코드는 다음과 비슷하게 보일 것입니다.

Note

다음 코드 샘플은 AWS CDK 부트스트랩 작업에서 가져온 것으로 AWS CDK 배포 작업은 비슷한 내용입니다.

```

Name: CDK_Bootstrap_Action
SchemaVersion: 1.0
Actions:
  CDKBootstrapAction:
    Identifier: aws/cdk-bootstrap@v2
    Inputs:
      Variables:
        - Name: NPMUSER
          Value: ${Secrets.npmUsername}
        - Name: NPMTOKEN
          Value: ${Secrets.npmAuthToken}
      Sources:
        - WorkflowSource
    Environment:
      Name: Dev2
      Connections:
        - Name: account-connection
          Role: codecatalystAdmin
    Configuration:
      Parameters:
        Region: "us-east-2"

```

이제 `cdk.json` 파일에서 `NPMUSER` 및 `NPMTOKEN` 변수를 사용할 준비가 되었습니다. 다음 절차로 이동합니다.

`cdk.json` 파일을 업데이트하려면 다음을 수행하세요.

1. AWS CDK 프로젝트의 루트 디렉터리로 변경하고 `cdk.json` 파일을 엽니다.
2. `"app":` 속성을 찾아 `### ####`로 표시된 코드를 포함하도록 변경합니다.

Note

다음 샘플 코드는 TypeScript 프로젝트에서 가져온 것입니다. JavaScript 프로젝트를 사용하는 경우 코드는 동일하지는 않지만 비슷하게 보일 것입니다.

```
{
```

```
"app": "npm set registry=https://your-registry/folder/CDK-package/ --
userconfig .npmrc && npm set //your-registry/folder/CDK-package/:always-auth=true
--userconfig .npmrc && npm set //your-registry/folder/CDK-package/:_authToken=
\"${NPMUSER}\"\":\"${NPM_TOKEN}\" && npm install && npx ts-node --prefer-ts-exts bin/
hello-cdk.ts/js",
"watch": {
  "include": [
    "*"
  ],
  "exclude": [
    "README.md",
    "cdk*.json",
    "**/*.d.ts",
    "**/*.js",
    "tsconfig.json",
    "package*.json",
    ...
  ]
}
```

3. **### ####**로 강조 표시된 코드에서

- **your-registry/folder/CDK-package/**를 프라이빗 레지스트리의 AWS CDK 프로젝트 종속성 경로로 바꿉니다.
- **hello-cdk.ts/js**를 엔트리포인트 파일 이름으로 바꿉니다. 이 파일은 사용 중인 언어에 따라 .ts(TypeScript) 또는 .js(JavaScript) 파일일 수 있습니다.

Note

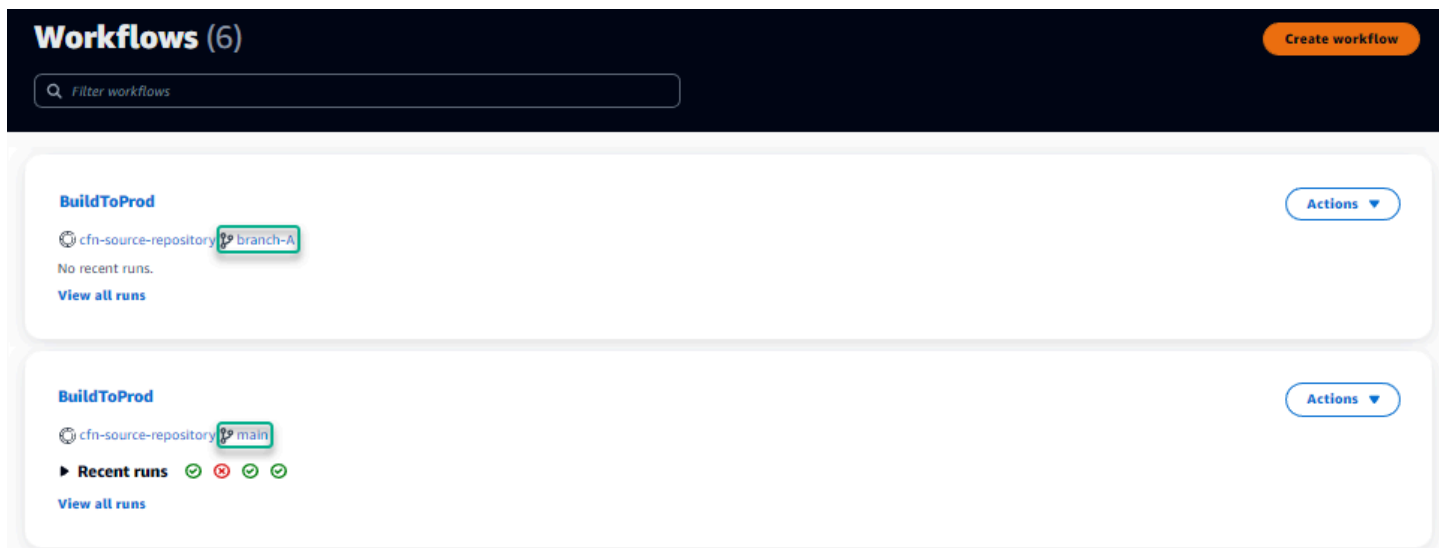
이 작업은 보안 암호에서 지정한 npm 사용자 이름 및 액세스 토큰으로 **NPMUSER** 및 **NPM_TOKEN** 변수를 바꿉니다.

4. cdk.json 파일을 저장합니다.
5. 작업을 수동으로 다시 실행하여 변경한 내용으로 오류가 해결되는지 확인합니다. 수동으로 작업을 실행하는 방법에 대한 자세한 내용은 [워크플로 수동 실행 시작](#) 섹션을 참조하세요.

여러 워크플로의 이름이 같은 이유는 무엇인가요?

워크플로는 리포지토리당 브랜치별로 저장됩니다. 서로 다른 브랜치에 있는 두 개의 워크플로가 같은 이름을 가질 수 있습니다. 워크플로 페이지에서 브랜치 이름을 보고 같은 이름의 워크플로를 구분할 수

있습니다. 자세한 내용은 [Amazon CodeCatalyst의 브랜치를 사용하여 소스 코드 작업 구성](#) 섹션을 참조하세요.



워크플로 정의 파일을 다른 폴더에 저장할 수 있나요?

아니요, 모든 워크플로 정의 파일은 `.codecatalyst/workflows` 폴더 또는 해당 폴더의 하위 폴더에 저장해야 합니다. 여러 논리 프로젝트가 있는 모노 리포지토리를 사용하는 경우 모든 워크플로 정의 파일을 `.codecatalyst/workflows` 폴더 또는 그 하위 폴더 중 하나에 배치한 다음 트리거 내부의 파일 변경됨 필드(시각적 편집기) 또는 `FilesChanged` 속성(YAML 편집기)을 사용하여 지정된 프로젝트 경로에서 워크플로를 자동으로 트리거합니다. 자세한 내용은 [워크플로에 트리거 추가](#) 및 [예시: 푸시, 브랜치 및 파일이 있는 트리거](#) 섹션을 참조하세요.

워크플로에 작업을 순차적으로 추가하려면 어떻게 해야 하나요?

기본적으로 워크플로에 작업을 추가하면 종속성이 없으며 다른 작업과 병렬로 실행됩니다.

작업을 순서대로 정렬하려면 `DependsOn` 필드를 설정하여 다른 작업에 대한 종속성을 설정할 수 있습니다. 다른 작업의 결과물인 아티팩트나 변수를 사용하도록 작업을 구성할 수도 있습니다. 자세한 내용은 [작업 순서 지정](#) 섹션을 참조하세요.

워크플로가 성공적으로 검증되었지만 런타임에 실패하는 이유는 무엇인가요?

Validate 버튼을 사용하여 워크플로를 검증했지만 워크플로가 검증에 실패했다면 검사기의 제한 때문일 수 있습니다.

워크플로 구성에서 보안 암호, 환경 또는 플릿과 같은 CodeCatalyst 리소스에 대한 참조 오류는 커밋 중에 등록되지 않습니다. 유효하지 않은 참조가 사용된 경우 워크플로가 실행될 때만 오류가 식별됩니다. 마찬가지로 필수 필드가 누락되었거나 작업 속성에 오타가 있는 등 작업 구성에 오류가 있는 경우 워크플로가 실행될 때만 오류가 식별됩니다. 자세한 내용은 [워크플로 생성](#) 섹션을 참조하세요.

자동 검색이 내 작업에 대한 보고서를 찾지 못함

문제: 테스트를 실행하는 작업에 대해 자동 검색을 구성했지만 CodeCatalyst에서 보고서를 검색하지 못합니다.

가능한 해결 방법: 이는 여러 가지 문제로 인해 발생할 수 있습니다. 다음 해결 방법 중 하나 이상을 시도해 보세요.

- 테스트를 실행하는 데 사용되는 도구가 CodeCatalyst가 이해하는 형식 중 하나로 출력을 생성하는지 확인합니다. 예를 들어, pytest를 통해 CodeCatalyst가 테스트 및 코드 커버리지 보고서를 검색할 수 있도록 하려면 다음 인수를 포함하세요.

```
--junitxml=test_results.xml --cov-report xml:test_coverage.xml
```

자세한 내용은 [품질 보고서 유형](#) 섹션을 참조하세요.

- 출력의 파일 확장자가 선택한 형식과 일치하는지 확인합니다. 예를 들어, JUnitXML 형식으로 결과를 생성하도록 pytest를 구성하는 경우 파일 확장자가 .xml인지 확인하세요. 자세한 내용은 [품질 보고서 유형](#) 섹션을 참조하세요.
- 특정 폴더를 일부러 제외하지 않는 한 IncludePaths가 전체 파일 시스템(**/*)을 포함하도록 구성되었는지 확인합니다. 마찬가지로, 보고서가 위치할 것으로 예상되는 디렉터리를 제외하지 않도록 ExcludePaths가 구성되어 있는지 확인합니다.
- 특정 출력 파일을 사용하도록 보고서를 수동으로 구성한 경우에는 자동 검색에서 제외됩니다. 자세한 내용은 [품질 보고서 YAML 예시](#) 섹션을 참조하세요.
- 출력이 생성되기 전에 작업이 실패하여 자동 검색에서 보고서를 찾지 못할 수도 있습니다. 예를 들어 단위 테스트가 실행되기 전에 빌드가 실패했을 수 있습니다.

성공 기준을 구성한 후 내 작업이 자동 검색된 보고서에 대한 작업이 실패해요.

문제: 자동 검색을 활성화하고 성공 기준을 구성하면 일부 보고서가 성공 기준을 충족하지 못하여 결과적으로 작업이 실패합니다.

가능한 해결 방법: 이 문제를 해결하려면 다음 솔루션 중 하나 이상을 시도해 보세요.

- IncludePaths 또는 ExcludePaths를 수정하여 관심 없는 보고서를 제외합니다.
- 모든 보고서가 전달되도록 성공 기준을 업데이트합니다. 예를 들어, 라인 커버리지가 50%인 보고서와 70%인 보고서 두 개가 발견된 경우 최소 라인 커버리지를 50%로 조정합니다. 자세한 내용은 [성공 기준](#) 섹션을 참조하세요.
- 실패한 보고서를 수동으로 구성된 보고서로 전환합니다. 이렇게 하면 특정 보고서에 대해 다양한 성공 기준을 구성할 수 있습니다. 자세한 내용은 [보고서의 성공 기준 구성](#) 섹션을 참조하세요.

자동 검색이 내가 원하지 않는 보고서를 생성해요.

문제: 자동 검색을 활성화하면 원하지 않는 보고서가 생성됩니다. 예를 들어, CodeCatalyst는 node_modules에 저장된 애플리케이션의 종속성에 포함된 파일에 대한 코드 커버리지 보고서를 생성합니다.

가능한 해결 방법: 원치 않는 파일을 제외하도록 ExcludePaths 구성을 조정할 수 있습니다. 예를 들어, node_modules를 제외하려면 node_modules/**/*를 추가합니다. 자세한 내용은 [경로 포함/제외](#) 섹션을 참조하세요.

자동 검색이 단일 테스트 프레임워크에 대해 작은 보고서를 많이 생성함

문제: 특정 테스트 및 코드 커버리지 보고 프레임워크를 사용할 때 자동 검색이 많은 수의 보고서를 생성하는 것을 발견했습니다. 예를 들어, [Maven Surefire 플러그인](#)을 사용할 때 자동 검색은 각 테스트 클래스에 대해 서로 다른 보고서를 생성합니다.

가능한 해결 방법: 프레임워크에서 출력을 단일 파일로 집계할 수 있습니다. 예를 들어, Maven Surefire 플러그인을 사용하는 경우 npx junit-merge를 사용하여 파일을 수동으로 집계할 수 있습니다. 전체 표현식은 다음과 같습니다.

```
mvn test; cd test-package-path/surefire-reports && npx junit-merge -d ./ && rm
*Test.xml
```

CI/CD에 나열된 워크플로가 소스 리포지토리의 워크플로와 일치하지 않음

문제: CI/CD, 워크플로 페이지에 표시된 워크플로가 [소스 리포지토리](#)의 ~/.codecatalyst/workflows/ 폴더에 있는 워크플로와 일치하지 않습니다. 다음과 같은 불일치가 발생할 수 있습니다.

- 워크플로 페이지에 워크플로가 표시되지만 해당 워크플로 정의 파일이 소스 리포지토리에 존재하지 않습니다.

- 워크플로 정의 파일이 소스 리포지토리에 존재하지만 해당 워크플로가 워크플로 페이지에 나타나지 않습니다.
- 워크플로는 소스 리포지토리와 워크플로 페이지에 모두 존재하지만 서로 다릅니다.

워크플로 페이지를 새로 고칠 시간이 없거나 워크플로 할당량을 초과한 경우 이 문제가 발생할 수 있습니다.

수정 방법:

- 대기합니다. 일반적으로 소스에 커밋한 후 2~3초 정도 기다려야 워크플로 페이지에서 변경 내용을 볼 수 있습니다.
- 워크플로 할당량을 초과한 경우 다음 중 하나를 수행하세요.

Note

워크플로 할당량을 초과했는지 확인하려면 [CodeCatalyst의 워크플로 할당량](#)을 검토하고 문서화된 할당량을 소스 리포지토리 또는 워크플로 페이지의 워크플로와 교차 확인합니다. 할당량이 초과되었음을 나타내는 오류 메시지는 없으므로 직접 조사해야 합니다.

- 스페이스당 최대 워크플로 수 할당량을 초과한 경우 일부 워크플로를 삭제한 다음 워크플로 정의 파일에 대해 테스트 커밋을 수행합니다. 테스트 커밋의 예시로는 파일에 스페이스를 추가하는 것이 있습니다.
- 최대 워크플로 정의 파일 크기 할당량을 초과한 경우에는 워크플로 정의 파일을 변경하여 길이를 줄입니다.
- 단일 소스 이벤트 할당량에서 처리되는 워크플로 파일의 최대 개수를 초과한 경우에는 여러 번 테스트 커밋을 수행합니다. 각 커밋에서 최대 워크플로 수보다 적은 수의 워크플로를 수정하세요.

워크플로를 생성하거나 업데이트할 수 없음

문제: 워크플로를 만들거나 업데이트하고 싶은데 변경 내용을 커밋하려고 할 때 오류가 발생합니다.

가능한 해결 방법: 프로젝트 또는 스페이스의 역할에 따라 코드를 푸시하여 프로젝트의 리포지토리를 소싱할 권한이 없을 수 있습니다. 워크플로용 YAML 파일은 리포지토리에 저장됩니다. 자세한 내용은 [워크플로 정의 파일](#) 섹션을 참조하세요. 스페이스 관리자 역할, 프로젝트 관리자 역할 및 기여자 역할은 모두 프로젝트의 리포지토리에 코드를 커밋하고 푸시할 수 있는 권한이 있습니다.

기여자 역할이 있지만 특정 브랜치에서 워크플로 YAML 변경 내용을 만들거나 커밋할 수 없는 경우 해당 브랜치에 대해 해당 역할이 있는 사용자가 특정 브랜치에 코드를 푸시하지 못하도록 하는 브랜치 규칙이 구성되어 있을 수 있습니다. 다른 브랜치에서 워크플로를 생성하거나 다른 브랜치에 변경 사항을 커밋해 보세요. 자세한 내용은 [브랜치 규칙을 사용하여 브랜치에 대한 작업 관리](#) 섹션을 참조하세요.

문제 관련 문제 해결

다음은 CodeArtifact에서 일반적으로 발생하는 issues 문제를 해결하는 데 유용한 정보입니다.

주제

- [내 문제에 대한 담당자를 선택할 수 없습니다.](#)

내 문제에 대한 담당자를 선택할 수 없습니다.

문제: 문제를 생성할 때 담당자 목록이 비어 있습니다.

가능한 해결 방법: 담당자 목록은 프로젝트의 멤버로 등록된 CodeCatalyst 사용자에게 직접 연결됩니다. 사용자 프로파일 액세스가 제대로 작동하는지 확인하려면 프로파일 아이콘을 선택한 다음 사용자 프로파일을 선택합니다. 사용자 프로파일 정보가 채워지지 않으면 상태 보고서에서 인시던트가 있는지 확인하세요. 채워지면 서비스 티켓을 제출합니다.

CodeCatalyst의 검색 문제 해결

다음 섹션을 참조하여 CodeCatalyst의 검색과 관련된 문제를 해결합니다. 워크플로에 대한 자세한 내용은 [CodeCatalyst에서 코드, 문제, 프로젝트 및 사용자 검색](#) 섹션을 참조하세요.

주제

- [프로젝트에서 사용자를 찾을 수 없음](#)
- [프로젝트 또는 공간에서 원하는 것이 보이지 않음](#)
- [페이지를 탐색할 때 검색 결과 수가 계속 변경됨](#)
- [검색 쿼리가 완료되지 않음](#)

프로젝트에서 사용자를 찾을 수 없음

문제: 사용자의 세부 정보를 보려고 하면 프로젝트에 해당 사용자의 정보가 표시되지 않습니다.

가능한 해결 방법: 검색은 현재 프로젝트 내 사용자 검색을 지원하지 않습니다. 스페이스에 액세스할 수 있는 사용자를 검색하려면 빠른 검색에서 이 스페이스로 전환하거나 고급 쿼리 언어를 사용하여 지정된 프로젝트 필터를 제거하세요.

프로젝트 또는 공간에서 원하는 것이 보이지 않음

문제: 특정 정보를 검색하려고 해도 결과가 나타나지 않습니다.

가능한 해결 방법: 콘텐츠 업데이트는 검색 결과에서 업데이트하는 데 몇 초 정도 걸릴 수 있습니다. 대규모 업데이트는 몇 분 정도 걸릴 수 있습니다.

최근에 업데이트되지 않은 리소스의 경우 검색 범위를 좁혀야 할 수도 있습니다. 키워드를 더 추가하거나 고급 쿼리 언어를 사용하여 구체화할 수 있습니다. 쿼리 구체화에 대한 자세한 내용은 [검색 쿼리 구체화](#) 섹션을 참조하세요.

페이지를 탐색할 때 검색 결과 수가 계속 변경됨

문제: 다음 페이지로 이동하면 검색 결과 수가 변경되는 것처럼 보여서 총 검색 결과 수가 얼마나 되는지 명확하지 않습니다.

가능한 해결 방법: 검색 결과 페이지를 탐색할 때 검색어와 일치하는 검색 결과의 수가 변경되는 것을 볼 수 있습니다. 페이지를 탐색할 때 검색된 일치 항목의 수가 더 정확하게 반영되도록 결과 수가 업데이트될 수 있습니다.

결과를 탐색할 때 '테스트'에 대한 결과 없음 메시지가 표시될 수 있습니다. 나머지 결과에 액세스할 수 없는 경우 메시지가 표시됩니다.

검색 쿼리가 완료되지 않음

문제: 검색 쿼리 결과가 표시되지 않고 너무 오래 걸리는 것 같습니다.

가능한 해결 방법: 프로그램적으로 또는 팀 활동량이 많아서 스페이스에서 동시에 많은 검색이 이루어지고 있는 경우 검색이 완료되지 않을 수 있습니다. 프로그래밍 검색을 실행하는 경우 검색을 일시 중지하거나 줄입니다. 그렇지 않으면 몇 초 후에 다시 시도합니다.

확장 프로그램 관련 문제 해결

다음 섹션을 참조하여 CodeCatalyst의 확장 프로그램과 관련된 문제를 해결합니다. 확장 프로그램에 대한 자세한 내용은 [CodeCatalyst에서 확장 프로그램이 있는 프로젝트에 기능 추가](#) 섹션을 참조하세요.

주제

- [연결된 타사 리포지토리의 변경 내용을 보거나 해당 변경 결과를 검색할 수 없습니다.](#)

연결된 타사 리포지토리의 변경 내용을 보거나 해당 변경 결과를 검색할 수 없습니다.

문제: 타사 리포지토리의 변경 내용이 CodeCatalyst에 표시되지 않습니다.

가능한 해결 방법: 현재 CodeCatalyst는 연결된 리포지토리에 대한 기본 브랜치의 변경 사항 감지를 지원하지 않습니다. 연결된 리포지토리의 기본 브랜치를 변경하려면 먼저 CodeCatalyst에서 연결을 해제하고 기본 브랜치를 변경한 다음 다시 연결해야 합니다. 자세한 내용은 [CodeCatalyst에서 GitHub 리포지토리, Bitbucket 리포지토리, GitLab 프로젝트 리포지토리 및 Jira 프로젝트 연결](#) 섹션을 참조하세요.

스페이스와 연결된 계정의 문제 해결

CodeCatalyst에서 스페이스에 AWS 계정을 추가하여 리소스에 권한을 부여하고 청구 목적으로 사용할 수 있습니다. 다음 정보는 CodeCatalyst에서 관련 계정과 관련된 일반적인 문제를 해결하는 데 도움이 될 수 있습니다.

주제

- [내 AWS 계정 연결 요청에 잘못된 토큰 오류가 발생했습니다](#)
- [내 Amazon CodeCatalyst 프로젝트 워크플로가 구성된 계정, 환경 또는 IAM 역할에 대한 오류로 실패합니다](#)
- [프로젝트를 생성하려면 연결된 계정, 역할 및 환경이 필요해요](#)
- [AWS Management Console에서 Amazon CodeCatalyst 스페이스 페이지에 액세스할 수 없어요](#)
- [다른 계정을 청구 계정으로 사용하고 싶어요](#)
- [연결 이름 오류와 함께 프로젝트 워크플로가 실패합니다.](#)

내 AWS 계정 연결 요청에 잘못된 토큰 오류가 발생했습니다

문제: 연결 토큰으로 연결 요청을 생성할 때 페이지는 토큰을 수락하지 않으며 토큰이 유효하지 않다는 오류를 표시합니다.

가능한 해결 방법: 스페이스에 추가할 계정 ID를 제공해야 합니다. AWS 계정에 대한 관리 권한이 있거나 관리자와 협력하여 계정을 추가할 수 있어야 합니다.

계정 확인을 선택하면 AWS Management Console에서 새 브라우저 창이 열립니다. 콘솔 측에 동일한 계정을 로그인해야 합니다. 다음을 확인한 후 다시 시도하세요.

- 스페이스에 추가하려는 것과 동일한 AWS 계정을 사용하여 AWS Management Console에 로그인합니다.
- 리전이 해당 스페이스에 맞는 리전으로 설정된 상태로 AWS Management Console에 로그인합니다.
- 청구 페이지에서 AWS 계정을 스페이스의 지정된 청구 계정으로 추가하려는 경우 계정이 다른 스페이스 또는 스페이스의 청구 계정으로 할당량에 도달하지 않았는지 확인하세요.

내 Amazon CodeCatalyst 프로젝트 워크플로가 구성된 계정, 환경 또는 IAM 역할에 대한 오류로 실패합니다

문제: 워크플로가 실행되고 스페이스와 연결된 구성된 계정 또는 IAM 역할을 찾지 못하면 워크플로 YAML에서 역할, 연결 및 환경 필드를 수동으로 입력해야 합니다. 실패한 워크플로 작업을 보고 오류 메시지가 다음과 같은지 확인합니다.

- 이 역할은 환경과 연결된 연결과 함께 사용할 수 없습니다.
- 작업이 성공하지 못했습니다. 상태: FAILED, 계정 연결 또는 환경에 대해 제공된 값이 유효하지 않습니다. 연결이 스페이스와 연결되어 있고 환경이 프로젝트와 연결되어 있는지 확인합니다.
- 작업이 성공하지 못했습니다. 상태: FAILED, IAM 역할에 대해 제공된 값이 유효하지 않습니다. 이름이 존재하고, IAM 역할이 계정 연결에 추가되고, 연결이 이미 Amazon CodeCatalyst 스페이스와 연결되어 있는지 확인합니다.

가능한 해결 방법: 워크플로 YAML 필드에 [환경](#), [연결](#) 및 [역할](#)에 대한 정확한 값이 있는지 확인합니다. 환경이 필요한 CodeCatalyst 워크플로 작업은 AWS 리소스를 실행하거나 AWS 리소스 스택을 생성하는 빌드 또는 배포 작업입니다.

실패한 워크플로 작업 블록을 선택한 다음 비주얼을 선택합니다. 구성 탭을 선택합니다. 환경, 연결 이름 및 역할 이름 필드가 채워지지 않은 경우 워크플로를 수동으로 업데이트해야 합니다. 워크플로 YAML을 편집하려면 다음 단계를 따르세요.

- /.codecatalyst 디렉터리를 확장한 다음 /workflows 디렉터리를 확장합니다. 워크플로 YAML 파일을 엽니다. 워크플로에 대해 구성한 YAML에 IAM 역할 및 계정 정보가 지정되어 있는지 확인합니다. 예시

Actions:

```

cdk_bootstrap:
  Identifier: action-@v1
  Inputs:
    Sources:
      - WorkflowSource
  Environment:
    Name: Staging
  Connections:
    - Name: account-connection
      Role: build-role

```

CodeCatalyst 워크플로 빌드를 실행하고 AWS 리소스를 사용하여 작업을 배포하려면 Environment, Connection, Role 속성이 필요합니다. 예시는 [Environment](#), [Connections](#) 및 [Role](#)에 대한 CodeCatalyst 빌드 작업 참조 YAML 파라미터를 참조하세요.

- 스페이스에 계정이 추가되어 있는지 확인하고 계정에 적절한 IAM 역할 또는 계정에 추가된 역할이 있는지 확인합니다. 스페이스 관리자 역할이 있는 경우 계정을 조정하거나 추가할 수 있습니다. 자세한 내용은 [연결된 리소스를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 섹션을 참조하세요.

프로젝트를 생성하려면 연결된 계정, 역할 및 환경이 필요해요

문제: 프로젝트 생성 옵션에서 프로젝트에 내 스페이스에 사용 가능한 계정이 추가되지 않았거나 프로젝트를 사용하려면 내 스페이스에 다른 계정을 추가해야 합니다.

가능한 해결 방법: 스페이스 관리자 역할이 있는 경우 스페이스에 권한이 있는 AWS 계정을 추가하여 프로젝트에 추가할 수 있습니다. 관리자 권한이 있거나 AWS 관리자와 함께 작업할 수 있는 AWS 계정도 있어야 합니다.

프로젝트 생성 화면에서 계정과 역할을 사용할 수 있도록 하려면 먼저 계정과 역할을 추가해야 합니다. 자세한 내용은 [연결된 리소스를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#) 섹션을 참조하세요.

CodeCatalystWorkflowDevelopmentRole-**spaceName** 역할 정책이라는 역할 정책을 사용하여 서비스 역할을 생성하도록 선택할 수 있습니다. 역할에는 고유한 식별자가 추가된 CodeCatalystWorkflowDevelopmentRole-**spaceName** 이름이 있습니다. 역할 및 역할 정책에 대한 자세한 내용은 [CodeCatalystWorkflowDevelopmentRole-**spaceName** 서비스 역할 이해](#) 섹션을 참조하세요. 역할을 생성하는 단계는 [계정 및 스페이스의 CodeCatalystWorkflowDevelopmentRole-**spaceName** 역할 생성](#) 섹션을 참조하세요. 역할은 계정에 추가되며 CodeCatalyst의 프로젝트 생성 페이지에서 사용할 수 있습니다.

AWS Management Console에서 Amazon CodeCatalyst 스페이스 페이지에 액세스할 수 없어요

문제: AWS Management Console의 Amazon CodeCatalyst 페이지에 액세스하여 내 CodeCatalyst 스페이스에 계정을 추가하거나 AWS의 계정에 역할을 추가하려고 하면 권한 오류가 발생합니다.

수정 방법:

스페이스 관리자 역할이 있는 경우 스페이스에 권한이 있는 AWS 계정을 추가하여 프로젝트에 추가할 수 있습니다. 관리자 권한이 있거나 AWS 관리자와 함께 작업할 수 있는 AWS 계정도 있어야 합니다. 먼저 관리하려는 것과 동일한 계정으로 AWS Management Console에 로그인해야 합니다. AWS Management Console에 로그인한 후 콘솔을 열고 다시 시도할 수 있습니다.

<https://us-west-2.console.aws.amazon.com/codecatalyst/home?region=us-west-2#/>의 AWS Management Console에서 Amazon CodeCatalyst 페이지를 엽니다.

다른 계정을 청구 계정으로 사용하고 싶어요

문제: CodeCatalyst 로그인을 설정할 때 스페이스를 설정하고 권한이 부여된 AWS 계정을 연결하는 여러 단계를 완료했습니다. 이제 다른 계정에 청구 권한을 부여하려고 합니다.

가능한 해결 방법: 스페이스 관리자 역할이 있는 경우 스페이스에 대해 청구 계정을 승인할 수 있습니다. 관리자 권한이 있거나 AWS 관리자와 함께 작업할 수 있는 AWS 계정도 있어야 합니다.

자세한 내용은 Amazon CodeCatalyst 관리자 안내서의 [Managing billing](#)을 참조하세요.

연결 이름 오류와 함께 프로젝트 워크플로가 실패합니다.

문제: 프로젝트를 생성한 다음 프로젝트 워크플로를 실행할 때 워크플로가 실패하고 다음과 같이 연결 이름이 유효하지 않다는 오류가 표시됩니다.

<action_name>에서 실패: 연결 이름이 유효하지 않습니다.

가능한 해결 방법: 스페이스에 추가할 계정 ID를 제공하고 계정이 프로젝트 제한 계정 연결에 대해 활성화되지 않았는지 확인합니다. 계정이 프로젝트 제한 계정 연결에 대해 활성화된 경우 새 프로젝트에 대한 액세스를 활성화하여 계정 연결을 업데이트해야 할 수 있습니다. 자세한 내용은 [Configuring project-restricted account connections](#)를 참조하세요.

Amazon CodeCatalyst와 AWS SDK 또는 AWS CLI 간의 문제 해결

다음 정보는 CodeCatalyst AWS CLI 및 AWS SDK로 작업할 때 발생하는 일반적인 문제를 해결하는데 도움이 될 수 있습니다.

주제

- [명령줄 또는 터미널에서 aws codecatalyst라고 입력하면 잘못된 선택이라는 메시지가 표시됩니다.](#)
- [aws codecatalyst 명령을 실행할 때 자격 증명 오류가 발생합니다.](#)

명령줄 또는 터미널에서 aws codecatalyst라고 입력하면 잘못된 선택이라는 메시지가 표시됩니다.

문제: AWS CLI를 CodeCatalyst에서 사용하려고 할 때 하나 이상의 aws codecatalyst 명령이 유효한 것으로 인식되지 않습니다.

해결 방법: 이 문제의 가장 일반적인 원인은 최신 서비스 및 명령에 대한 최신 업데이트가 포함되지 않은 버전의 AWS CLI를 사용하고 있기 때문입니다. AWS CLI 설치를 업데이트한 다음 다시 시도하세요. 자세한 내용은 [CodeCatalyst AWS CLI 에서를 사용하도록 설정](#)을 참조하세요.

aws codecatalyst 명령을 실행할 때 자격 증명 오류가 발생합니다.

문제: CodeCatalyst에서 AWS CLI를 사용하려고 할 때 You can configure credentials by running "aws configure". 또는 Unable to locate authorization token이라는 메시지가 표시됩니다.

해결 방법: CodeCatalyst 명령으로 작동하도록 AWS CLI 프로파일을 구성해야 합니다. 자세한 내용은 [CodeCatalyst AWS CLI 에서를 사용하도록 설정](#) 섹션을 참조하세요.

CodeCatalyst 상태 보고서를 사용하여 현재 서비스 상태 이해

Amazon CodeCatalyst 상태 보고서는 사용자에게 광범위한 영향을 미치는 CodeCatalyst의 리소스 성능 및 서비스 가용성에 대한 최신 알림의 집계 목록을 제공하는 퍼블릭 대시보드입니다. CodeCatalyst의 애플리케이션에 영향을 미칠 수 있는 문제가 있는 리소스를 확인할 수 있습니다. 이를 통해 시스템 전체에서 가동 중단 및 기타 리소스 가동 중지 시간을 추적할 수 있습니다. 인시던트가 발생하면 상태 보고서 아이콘에 파란색 표시기가 나타납니다. 또한 CodeCatalyst는 프로젝트에서 스페이스 관리자 역할을 가진 모든 사용자에게 알림 및 이메일 알림을 자동으로 전송하여 인시던트의 세부 정보와 기록을 거의 실시간으로 제공합니다.

대시보드는 모든 활성 이벤트 목록과 지난 30일 동안 발생한 최대 100건의 이전 인시던트 기록을 제공합니다. 인시던트가 업데이트된 날짜를 기준으로 인시던트 목록을 구성할 수 있습니다. 최대 분의 업데이트까지 인시던트 목록을 새로 고칠 수도 있습니다.

CodeCatalyst 상태 보고서를 사용하기 위한 워크플로는 다음과 같습니다.

Mateo Jackson은 스페이스 관리자 권한이 있는 Budding 스페이스의 개발자입니다. 풀 요청을 생성하려고 시도하는 동안 오류 메시지가 계속 표시됩니다. 그는 이메일을 확인하고 CodeCatalyst로부터 자신의 스페이스에 영향을 미치는 시스템 문제에 대한 자세한 기록을 제공하는 자동 생성된 시스템 인시던트 이메일을 받았음을 알게 됩니다. 그는 업데이트 보기를 선택하고 CodeCatalyst 상태 보고서로 이동하여 모든 시스템 보고 인시던트를 볼 수 있습니다. 그는 목록에서 인시던트를 선택하여 자세한 정보를 찾습니다. 마지막 업데이트, 기록, 영향을 받는 기능, 시작 시간 및 인시던트의 현재 상태에 대한 타임스탬프를 제공하는 분할 화면이 열립니다. 또한 문제가 진행 중이지만 서비스 팀이 문제를 해결하기 시작했음을 알 수 있습니다. 인시던트의 기록 또는 상태에 대한 업데이트가 있을 때마다 그는 이메일을 받습니다. 이메일에 액세스할 수 없는 경우 상단 패널에서 종 모양 아이콘을 선택하여 CodeCatalyst 상태 보고서로 이동할 수 있습니다.

CodeCatalyst 상태 보고서 개념

다음 개념을 학습하면 CodeCatalyst 상태 보고서와 이를 통해 애플리케이션, 서비스 및 리소스의 상태를 추적할 수 있는 방법을 이해하는 데 도움이 됩니다.

인시던트

인시던트는 CodeCatalyst 내의 애플리케이션 및 리소스에 영향을 미치는 시스템 이벤트입니다. 인시던트를 선택하여 이벤트가 시작된 시간 및 서비스 팀이 문제를 해결하기 위해 작업 중인지 여부를 포함하여 이벤트의 자세한 기록을 볼 수 있습니다.

상태 표시기

상태는 인시던트의 실시간 상태입니다. 진행 중 또는 해결됨으로 표시됩니다.

영향을 받는 기능

영향을 받는 기능은 인시던트의 영향을 받는 리소스 또는 애플리케이션입니다. 단일 인시던트는 풀 요청, 문제, 워크플로, 테스트, 배포 및 소스 등 시스템의 여러 영역에 영향을 미칠 수 있습니다.

업데이트 시기

업데이트 시기는 인시던트의 마지막 업데이트 타임스탬프가 제공됩니다.

Amazon CodeCatalyst의 지원

스페이스를 만들 때 AWS 계정을 연결하고 해당 계정을 스페이스의 청구 계정으로 지정해야 합니다. 청구 계정으로 지정한 AWS 계정은 Amazon CodeCatalyst용 지원 플랜에 액세스하는 곳이기도 합니다. 지원이 필요한 경우 이 지정된 AWS 계정에서 지원 사례를 생성할 수 있습니다.

한 스페이스의 CodeCatalyst 사용자는 CodeCatalyst의 Amazon CodeCatalyst용 지원 페이지를 사용하여 지원 사례를 관리합니다. Business Support 또는 Enterprise Support와 같은 지원 플랜으로 업그레이드하여 CodeCatalyst에서 기술 지원 사례를 생성하고 관리할 수 있습니다. 지원 사례에 대해 전화, 웹 또는 채팅을 통해 지원을 받을 수 있습니다.

Amazon CodeCatalyst용 지원을 통해 CodeCatalyst 서비스 및 리소스와 관련된 사례만 지원할 수 있습니다. CodeCatalyst 리소스에는 CodeCatalyst 내에 배포된 리소스와 CodeCatalyst의 사용자가 배포한 리소스가 포함되지만, 다른 AWS 또는 타사 서비스를 위해 배포된 리소스는 포함되지 않습니다. 다른 AWS 서비스에 대한 지원이 필요한 경우 AWS Management Console을 통해 해당 서비스를 열어야 합니다.

지원 플랜을 변경하려면 [Changing support plans](#)를 참조하세요.

Note

개발자 지원 플랜은 프로덕션 환경을 위해 설계되지 않았습니다. 스페이스 청구 계정에 개발자 지원 플랜이 있는 경우, 이 플랜은 CodeCatalyst의 지원 내의 모든 스페이스 관리자 및 스페이스 멤버에게 캐스케이드되지 않습니다.

Amazon CodeCatalyst의 지원에 대한 청구

CodeCatalyst에서 스페이스를 만들면 해당 스페이스의 사용자가 Amazon CodeCatalyst용 지원에서 지원 사례를 만들고 관리할 수 있습니다. 두 가지 유형의 고객 사례를 생성할 수 있습니다.

- 계정 및 청구 지원 사례는 해당 스페이스의 모든 CodeCatalyst 사용자가 사용할 수 있습니다. CodeCatalyst의 권한에 따라 청구 및 계정 관련 질문에 대한 도움을 받을 수 있습니다.
- 기술 지원 사례는 서비스 관련 기술 문제 및 타사 애플리케이션 확장 프로그램에 대한 도움을 받기 위해 기술 지원 엔지니어와 연결해 줍니다. 기본 지원 플랜에 가입한 경우 기술 지원 사례를 생성할 수 없습니다.

스페이스의 청구 계정으로 지정된 AWS 계정에 해당 스페이스에 대한 Business Support 또는 Enterprise Support 플랜이 있어야 기술 사례에 대해 CodeCatalyst용 지원을 사용할 수 있습니다.

Note

스페이스에서 Business Support 또는 Enterprise Support 플랜이 없는 계정에서 Amazon CodeCatalyst용 지원을 사용하는 경우에도 계정 및 청구 사례에 대해 Amazon CodeCatalyst용 지원을 계속 사용할 수 있습니다.

기술 지원을 받으려면 CodeCatalyst 콘솔을 통해 모든 사례를 열어야 합니다. AWS Management Console의 [지원](#)에서는 CodeCatalyst에 대한 기술 지원 사례를 만들 수 없습니다.

Note

서비스 제한 증가 요청은 Amazon CodeCatalyst용 지원에서 사용할 수 없습니다. 이러한 요청은 AWS Support Center Console의 스페이스 청구 계정에 대한 루트 사용자만 제출할 수 있습니다.

Amazon CodeCatalyst용 지원에는 지원과 동일한 지원 계약이 적용되며 다음과 같은 고려 사항이 있습니다.

- [심각도 선택](#)에 자세히 설명된 대로 지원의 심각도 목록, 응답 시간 및 SLA는 CodeCatalyst용 지원의 지원 사례에 적용됩니다.
- 스페이스 관리자 및 스페이스 멤버는 Slack의 지원 API 또는 AWS SDK 또는 지원 앱을 사용하여 CodeCatalyst용 사례를 만들 수 없습니다. CodeCatalyst 지원 사례는 CodeCatalyst에서만 제출할 수 있습니다.

Note

CodeCatalyst는 AWS Trusted Advisor 또는 AWS 사고 탐지 및 대응과 완전히 통합되지 않았습니다. CodeCatalyst가 통합되는 방법을 검증하여 비즈니스 방식이 현재 통합과 일치하는지 확인하세요.

지원을 요청하려는 스페이스의 사용자여야 합니다.

Note

스페이스에 두 명 이상의 빌더가 있는 경우 Business Support 또는 Enterprise Support 플랜을 구매하는 것이 좋습니다. 이러한 플랜은 최대 5,000명의 빌더에게 스페이스에 대한 기술 지원을 제공합니다.

스페이스의 청구 계정으로 지정된 AWS 계정은 `AWSRoleForCodeCatalystSupport` 역할 및 [AmazonCodeCatalystSupportAccess](#) 관리형 정책을 사용합니다. 이를 통해 스페이스의 CodeCatalyst 사용자는 Amazon CodeCatalyst용 지원 페이지에 액세스할 수 있습니다. 이 역할 및 정책에 대한 자세한 내용은 [AmazonCodeCatalystSupportAccess](#) 참조하세요. 청구에 대한 기타 고려 사항은 Amazon CodeCatalyst 관리자 안내서의 [Managing billing](#)을 참조하세요.

다음은 빌더가 CodeCatalyst에서 지원 사례를 만드는 가능한 흐름입니다.

Mateo Jackson은 CodeCatalyst에서 프로젝트를 진행 중인 개발자입니다. Amazon CodeCatalyst용 지원로 청구를 관리하는 AWS 계정을 등록하고 Business Support 플랜으로 업그레이드한 후 해당 스페이스의 모든 빌더가 기술 지원 사례를 만들 수 있습니다. Mateo는 프로젝트에서 실패한 워크플로에 대한 기술 지원 사례를 제출합니다. Mateo는 Amazon CodeCatalyst용 지원 페이지를 사용하여 양식을 작성하고 요청에 워크플로 ID 및 기타 세부 정보를 제공하여 사례를 생성합니다. 사례는 사례 ID로 생성되며 청구 계정으로 지정된 AWS 계정의 계정 ID를 포함하고 스페이스에 대한 지원 플랜과 연결됩니다.

모든 빌더가 에서 CodeCatalyst용 지원에서 지원 사례를 생성할 수 있지만 생성된 각 사례에 대해 비용이 부과되지는 않습니다. 스페이스 청구 계정에서 구매한 지원 Premium 플랜에 따라 사실상 무제한으로 사례와 연락처를 열 수 있습니다.

Note

스페이스 청구 계정은 CodeCatalyst 사용자 및 리소스에 대해 요금이 청구되는 AWS 계정입니다. 추가 AWS 계정에 배포한 경우, 다른 서비스에 배포된 리소스에 대한 지원은 AWS Management Console을 통해 지원에 문의하세요.
워크플로에서 배포한 AWS 계정을 식별할 수 있습니다.

Amazon CodeCatalyst용 지원의 스페이스 설정

Amazon CodeCatalyst용 지원은 CodeCatalyst와 지원 API 통합의 일부로 지원 사례를 관리합니다.

AWSRoleForCodeCatalystSupport 역할은 스페이스의 지원 사례에 사용되는 서비스 역할입니다. 이 역할은 스페이스의 지정된 청구 계정에 추가해야 합니다. 자세한 내용은 [계정 및 스페이스의 AWSRoleForCodeCatalystSupport 역할 생성](#) 섹션을 참조하세요.

Note

2023년 4월 20일 이전에 생성된 스페이스의 경우, 해당 스페이스에 CodeCatalyst 지원을 받으려면 역할을 생성해야 합니다. 2023년 4월 20일 이후에 스페이스를 생성하는 경우 스페이스를 생성하는 동안, CodeCatalyst의 청구 세부 정보 페이지에서 또는 CodeCatalyst에서 지원 배너 링크를 클릭하여 역할을 생성할 수 있습니다.

스페이스에 대한 지원 설정

1. CodeCatalyst 스페이스를 생성하면 청구 계정을 연결하라는 메시지가 표시됩니다. 스페이스에 대해 지정된 청구 계정으로 AWS에서 청구됩니다. 스페이스 생성에 대한 자세한 내용은 [새 스페이스 및 개발 역할 생성\(초대 없이 시작\)](#) 섹션을 참조하세요.
2. CodeCatalyst 스페이스를 생성할 때 CodeCatalyst 사용자가 지원에 액세스할 수 있도록 허용하는 AWSRoleForCodeCatalystSupport 서비스 역할을 생성할 수 있는 옵션을 사용할 수 있습니다. 이 역할에는 관리형 정책 AmazonCodeCatalystSupportAccess가 사용됩니다. 이 역할은 스페이스의 청구 계정으로 지정된 AWS 계정에 추가해야 합니다. 이 역할 생성에 관한 자세한 내용은 [계정 및 스페이스의 AWSRoleForCodeCatalystSupport 역할 생성](#) 섹션을 참조하세요.
3. 스페이스에 지정된 청구 계정의 경우, 스페이스 관리자는 AWS 계정에 대한 Business Support 또는 Enterprise Support 플랜을 구매하는 것이 좋습니다. 스페이스의 모든 멤버는 Amazon CodeCatalyst용 지원에서 지원 사례를 관리할 수 있으며, 지원 채널은 통합이 완료된 경우 구매한 지원 플랜에 맞춰 조정됩니다.
4. CodeCatalyst에서 지원 사례를 생성하고 관리하려면 [CodeCatalyst에서 CodeCatalyst 지원 사례 생성](#) 섹션을 참조하세요.

AWS Management Console에서 CodeCatalyst에 대한 지원 액세스

스페이스에 대한 지원 사용 청구 계정의 연결이 해제되면 이전 스페이스 청구 계정 및 관련 지원 플랜과 연결된 지원 사례가 더 이상 Amazon CodeCatalyst용 지원에 표시되지 않습니다. 해당 청구 계정의 루트 사용자는 AWS Management Console에서 이전 사례를 보고 해결할 수 있으며 다른 사용자가 이전 사례를 보고 해결할 수 있도록 지원에 대한 IAM 권한을 설정할 수 있습니다. 다른 모든 AWS

서비스에 대해 AWS Management Console에서 지원 계획의 혜택에 참여하고 이전에 해결되지 않은 CodeCatalyst 지원 사례를 완료할 수 있습니다.

자세한 내용은 지원 사용 설명서의 [Updating, resolving, and reopening your case](#)를 참조하세요.

CodeCatalyst에 대한 일반적인 사용 방법 정보에 대한 지원 사례도 AWS Management Console에서 열 수 있지만 CodeCatalyst 대한 이 채널을 통해 기술 지원을 받을 수는 없습니다. 자세한 내용은 지원 사용 설명서의 [Creating support cases and case management](#)를 참조하세요.

다음은 AWS Management Console에서 CodeCatalyst에 대한 지원 사례를 해결하는 사용자의 가능한 흐름입니다.

모든 빌더가 Amazon CodeCatalyst용 지원을 사용하여 대한 지원 사례를 생성할 수 있지만, 지원 요청은 스페이스의 청구 계정으로 지정된 계정에서 청구됩니다. Mateo Jackson은 CodeCatalyst 프로젝트의 개발자로서 프로젝트의 워크플로 실패에 대한 기술 지원 사례를 열었습니다. 그러나 Amazon CodeCatalyst용 지원에 가입하고 Business Support 플랜을 구매한 스페이스에 대한 청구 계정이 스페이스에서 연결이 해제되었습니다. Mateo가 최신 커뮤니케이션을 확인하고 CodeCatalyst에 대해 개설된 사례를 해결하는 유일한 방법은 AWS Management Console의 지원 센터에서 사례 ID를 관리하는 것입니다. 이를 위해 Mateo는 지원 사례에 연결된 이전 스페이스 청구 계정의 루트 사용자로부터 IAM 권한을 부여받고 콘솔의 지원을 통해 사례를 해결합니다.

Important

스페이스에 대해 지정된 청구 계정을 변경하는 경우, AWS Management Console을 통해서만 해당 월말까지 지원 플랜에 계속 액세스할 수 있습니다. 이전에 CodeCatalyst에서 생성한 지원 사례에 계속 액세스하려면 업데이트된 청구 계정으로 지원을 재구매해야 합니다. 모든 지원 사례가 해결될 때까지 기다렸다가 Amazon CodeCatalyst용 지원을 통해 지원 사례에 액세스하는데 영향을 미치지 않도록 공간 청구 계정을 변경하는 것이 좋습니다.

CodeCatalyst에서 CodeCatalyst 지원 사례 생성

Amazon CodeCatalyst용 지원 페이지에서 지원 사례를 생성할 수 있습니다.

AWS Builder ID는 인증된 별칭에 대한 지원만 받을 수 있으며 권한에 따라 리소스에 대한 지원만 받을 수 있습니다. 계정 및 청구 옵션은 모든 스페이스 관리자 및 스페이스 멤버가 사용할 수 있습니다. 그러나 사용자는 CodeCatalyst에서 액세스할 수 있는 리소스에 대한 지원만 받을 수 있으며 계정에 대한 청구 관리와 관련된 지원은 받을 수 없습니다.

스페이스에 대한 for CodeCatalyst 페이지를 사용하여 지원 CodeCatalyst 리소스에 대한 계정 및 청구 사례 또는 기술 지원 사례를 생성할 수 있습니다.

 Note

Amazon CodeCatalyst용 지원을 통해 CodeCatalyst 서비스 및 리소스와 관련된 사례만 지원할 수 있습니다. CodeCatalyst 리소스에는 CodeCatalyst 내에 배포된 리소스와 CodeCatalyst의 사용자가 배포한 리소스가 포함되지만, 다른 AWS 또는 타사 서비스를 위해 배포된 리소스는 포함되지 않습니다. 다른 AWS 서비스에 대한 지원이 필요한 경우 AWS Management Console을 통해 해당 서비스를 열어야 합니다.

CodeCatalyst에서 지원 사례를 생성하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다.

 Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

3. 페이지 상단에서 ? 아이콘을 선택한 다음 지원을 선택합니다.
4. 사례 생성(Create case)을 선택하세요.
5. 다음 옵션 중 하나를 선택하세요:
 - 계정 및 청구(Account and billing)
 - 기술(Technical)

 Note

Amazon CodeCatalyst용 지원에서 비즈니스 지원 또는 엔터프라이즈 지원 플랜이 스페이스 청구 계정에 추가되면 모든 스페이스 관리자 및 스페이스 멤버가 CodeCatalyst 기술 사례 지원을 사용할 수 있습니다. 문제 해결 정보는 [내 스페이스에 대한 기술 지원 사례를 생성할 수 없습니다](#)를 참조하세요.

지원 요금제는 스페이스에 전체에 적용되지 않습니다. 여러 스페이스의 멤버인 경우 스페이스 관리자가 모든 스페이스에서 기술 지원을 받으려면 모든 스페이스에 대한 지원 프리미엄 플랜을 구매해야 합니다.

6. 서비스(Service), 범주(Category), 심각도(Severity)를 선택합니다. 심각도 선택에 대한 자세한 내용은 [심각도 선택](#)을 참조하세요.
 - 일반 지침
 - 시스템 손상
 - 프로덕션 시스템 손상
 - 프로덕션 시스템 중단
 - 비즈니스 크리티컬 시스템 중단
7. 다음 단계: 추가 정보를 선택합니다
8. 추가 정보(Additional information) 페이지의 제목(Subject)에 해당 문제에 대한 제목을 입력합니다.
9. 설명(Description)에서 프롬프트를 따라 다음과 같이 사례를 설명합니다:
 - 워크플로 ID, 로그 또는 스크린샷과 같은 CodeCatalyst 관련 정보 문제 해결
 - 수신한 오류 메시지
 - 수행한 문제 해결 단계

 Note

자격 증명, 신용 카드 데이터, 서명된 URL 또는 개인 식별 정보와 같은 민감한 정보는 서신에 공유하지 않습니다.

10. (선택 사항) 파일 첨부(Attach files)를 선택하여 오류 로그 또는 스크린샷과 같은 관련 파일을 사례에 추가합니다. 최대 3개의 파일을 첨부할 수 있습니다. 각 파일의 크기는 최대 5MB까지입니다.
11. 스페이스 이름에 스페이스 이름이 표시됩니다.
12. 빌더 이름에서 AWS Builder ID와 연결된 전체 이름이 자동으로 채워집니다.
13. (선택 사항) 프로젝트 이름에서 프로젝트를 선택합니다(해당하는 경우).

Note

권한이 있는 프로젝트만 표시됩니다. 다른 프로젝트에 대한 액세스 권한이 필요한 경우 지원 사례를 생성하기 전에 프로젝트 관리자에게 액세스 권한을 제공하도록 요청합니다.

14. 다음 단계: 문의를 선택합니다.
15. 기본 연락처 언어에서 기본값을 선택합니다. 현재 영어만 사용할 수 있습니다.
16. 연락 방법으로 웹, 전화 또는 채팅 옵션을 선택합니다.
17. 사례 세부 정보를 검토한 다음 제출을 선택합니다. 사례 ID 번호와 요약이 표시됩니다.

지원 사례는 스페이스 수준에서 생성되며 지원 사례에 정의된 스페이스 및 프로젝트(선택한 경우)에 액세스할 수 있는 모든 멤버가 볼 수 있습니다. 현재 개별 사용자의 지원 사례를 생략할 수 있는 방법은 없습니다.

CodeCatalyst에서 지원 사례 해결

Amazon CodeCatalyst용 지원 페이지에서 열려있는 지원 사례를 확인할 수 있습니다.

지원 사례를 해결하려는 스페이스의 스페이스 관리자 또는 스페이스 멤버 역할이 있어야 합니다. 스페이스 관리자 역할이 없거나, 사례가 생성될 때 프로젝트를 선택한 경우, 사례를 보고 해결하려면 프로젝트에 대한 멤버십도 있어야 합니다.

CodeCatalyst에서 열린 지원 사례를 해결하려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다.

Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

3. 페이지 상단에서 ? 아이콘을 선택한 다음 Amazon CodeCatalyst용 지원을 선택합니다.
4. 관리하려는 지원 사례의 링크를 선택합니다. Resolve case(사례 해결)를 선택합니다.

CodeCatalyst에서 지원 사례 다시 열기

Amazon CodeCatalyst용 지원 페이지에서 해결된 지원 사례 다시 열기를 사용할 수 있습니다.

Note

문제가 해결된 날로부터 최대 14일 이내에 지원 사례를 다시 열 수 있습니다. 그러나 14일 이상 비활성 상태인 사례는 다시 열 수 없습니다. 사례를 다시 열 수 없는 경우, 새 사례를 열고 이전 사례 ID를 참조로 포함합니다.

현재 문제와는 다른 정보가 있는 기존 사례를 다시 열면 지원 상담원이 새 사례 생성을 사용자에게 요청할 수 있습니다.

CodeCatalyst에서 지원 사례를 다시 열려면

1. <https://codecatalyst.aws/>에서 CodeCatalyst 콘솔을 엽니다.
2. CodeCatalyst 스페이스로 이동합니다.

Tip

둘 이상의 스페이스에 속해 있는 경우 상단 탐색 표시줄에서 보려는 스페이스를 선택할 수 있습니다.

3. 페이지 상단에서 ? 아이콘을 선택한 다음, CodeCatalyst용 AWS Support를 선택합니다.
4. 관리하려는 지원 사례의 링크를 선택합니다. Reopen(다시 열기)을 선택합니다. 확인 화면에서 확인을 선택한 다음 제출을 선택합니다.
5. 동일한 문제에 대한 최신 정보로 설명을 작성합니다. 자격 증명, 신용 카드 데이터, 서명된 URL 또는 개인 식별 정보와 같은 민감한 정보는 서신에 공유하지 않습니다.

CodeCatalyst 할당량

다음 표에서는 Amazon CodeCatalyst의 할당량 및 제한에 대해 설명합니다. 다음 주제에서 CodeCatalyst의 특정 측면에 대한 추가 정보를 찾을 수 있습니다.

- [CodeCatalyst의 소스 리포지토리 할당량](#)
- [CodeCatalyst의 ID, 권한 및 액세스 할당량](#)
- [CodeCatalyst의 워크플로 할당량](#)
- [CodeCatalyst에서의 개발 환경 할당량](#)
- [프로젝트 할당량](#)
- [CodeCatalyst의 블루프린트 할당량](#)
- [스페이스 할당량](#)
- [CodeCatalyst 문제에 대한 할당량](#)

한 계정의 최대 스페이스 수	5
사용자가 한 달에 생성할 수 있는 최대 스페이스 수	5
스페이스의 최소 AWS 계정 수	1
스페이스에 대한 최대 계정 연결 수	5,000
스페이스의 청구 계정으로 지정되는 최대 AWS 계정 수	1
단일 AWS 계정이 청구 계정으로 지정될 수 있는 프리 티어에서 최대 스페이스 수	3
단일 AWS 계정이 청구 계정으로 지정될 수 있는 유료 티어에서 최대 스페이스 수	15
스페이스에 대한 최대 VPC 연결 수	100
스페이스의 최대 프로젝트 수	100

사용자가 속할 수 있는 최대 프로젝트 수	1,000
스페이스 설명	스페이스 설명은 선택 사항입니다. 지정할 경우, 설명의 길이는 0~200자여야 합니다. 문자, 숫자, 공백, 마침표, 밑줄, 쉼표, 대시 및 다음과 같은 특수 문자의 조합을 포함할 수 있습니다. ? & \$ % + = / \ ; : \n \t \r
스페이스 이름	스페이스 이름은 CodeCatalyst 전체에서 고유해야 합니다. 삭제된 스페이스의 이름은 재사용할 수 없습니다. 스페이스 이름은 3~63자 이내로 작성해야 합니다. 또한 영숫자 문자로 시작해야 합니다. 스페이스 이름은 문자, 숫자, 마침표, 대시 및 밑줄의 조합을 포함할 수 있습니다. 이름에는 다음 문자를 포함할 수 없습니다. ! ? @ # \$ % ^ & * () + = { } [] / \ > < ~ ` ' " ; :

Amazon CodeCatalyst의 문서 기록

다음 표에서는 CodeCatalyst에 대한 전체 설명서의 설명서 기록 및 업데이트를 설명합니다.

변경 사항	설명	날짜
Amazon CodeCatalyst는 더 이상 신규 고객에게 공개되지 않습니다	Amazon CodeCatalyst는 더 이상 신규 고객에게 공개되지 않습니다. 자세한 내용은 Amazon CodeCatalyst에서 마이그레이션하는 방법 을 참조하세요.	2025년 11월 7일
새로운 내용: 새 범용 devfile 이미지	새 범용 devfile 이미지 (Universal image 5.0)에 대한 설명서가 추가되었습니다.	2025년 7월 30일
업데이트된 콘텐츠: 데이터 프라이버시	IAM Identity Center와 관련된 데이터 프라이버시에 대한 추가 정보를 포함하도록 설명서를 업데이트했습니다. CodeCatalyst의 데이터 프라이버시 참조	2025년 7월 28일
업데이트된 콘텐츠: CodeCatalyst 블루프린트 사용 중단	사용 가능한 블루프린트 테이블에서 AWS 프로젝트 개발 키트(AWS PDK) 콘텐츠를 제거했습니다. https://docs.aws.amazon.com/codecatalyst/latest/userguide/project-blueprints.html	2024년 12월 6일
새로운 내용: 새 범용 devfile 이미지	새 범용 devfile 이미지 (Universal image 4.0)에 대한 설명서가 추가되었습니다.	2024년 8월 26일

새로운 내용: 'AWS CDK 배포' 작업에서 사용하는 런타임 이미지, 'AWS CDK 부트스트랩' 작업에서 사용하는 런타임 이미지	두 가지 주제('AWS CDK 배포' 작업에서 사용하는 런타임 이미지 및 'AWS CDK 부트스트랩' 작업에서 사용하는 런타임 이미지)가 추가되었습니다.	2024년 8월 20일
새로운 내용: 더 이상 사용되지 않는 범용 devfile 이미지	더 이상 사용되지 않는 범용 devfile 이미지에 대한 설명서가 추가되었습니다(Universal image 1.0 및 Universal image 2.0).	2024년 8월 16일
업데이트된 콘텐츠: CodeCatalyst 블루프린트 사용 중단	사용 가능한 블루프린트 테이블 에서 서버리스 애플리케이션 모델(SAM) 및 주문형 비디오 웹 서비스 콘텐츠를 제거했습니다.	2024년 8월 14일
업데이트 내용: 활성 이미지	특정 작업에서 2022년 11월 또는 2024년 3월 이미지를 사용할 수 있음을 나타내도록 활성 이미지 주제를 업데이트했습니다. 또한 사용된 이미지를 나타내도록 각 작업의 설명서도 업데이트했습니다.	2024년 8월 9일
업데이트된 콘텐츠: 잘린 사용자 이름 문제 해결	특정 경우에 잘릴 수 있는 사용자 이름 문제 해결에 대한 정보를 포함하도록 문제 해결 주제를 업데이트했습니다.	2024년 7월 31일
업데이트 내용: 환경 생성	CodeCatalyst 역할을 언급하도록 환경 생성 섹션을 업데이트했습니다.	2024년 7월 25일

[업데이트 내용: 작업 간 아티팩트 및 파일 공유](#)

작업 그룹에 대한 정보를 포함하도록 [아티팩트의 파일 참조](#) 하위 주제를 업데이트했습니다. [예시: 작업 그룹이 있을 때 아티팩트의 파일 참조](#) 하위 주제도 추가했습니다.

2024년 7월 18일

[업데이트 내용: 워크플로를 사용하여 빌드, 테스트 및 배포](#)

워크플로 정의의 .yaml 또는 .yam1 파일 확장자가 소문자여야 함을 나타내도록 문서를 업데이트했습니다.

2024년 7월 15일

[업데이트된 콘텐츠: CodeCatalyst 스페이스에서 개발 환경을 생성할 때 SSO 사용자를 위한 문제 해결 추가](#)

개발 환경을 생성할 때 SSO 사용자가 ID 페더레이션을 사용하여 스페이스에 로그인한 문제에 대한 정보를 포함하도록 문제 해결 주제를 추가했습니다.

2024년 7월 12일

[업데이트 내용: 워크플로 생성](#)

워크플로 정의 파일을 `~/.codecatalyst/workflows/` 의 하위 디렉터리에 저장할 수 있음을 나타내도록 주제를 업데이트했습니다.

2024년 7월 12일

[업데이트 내용: CodeCatalyst의 워크플로 상태](#)

CodeCatalyst 콘솔에서 비활성 워크플로를 숨기는 방법에 대한 지침이 추가되었습니다.

2024년 7월 12일

[새로운 내용: 수동 전용 트리거 구성](#)

[수동 전용 트리거 구성](#) 주제가 추가되었습니다.

2024년 6월 26일

[업데이트 내용: 환경 생성](#)

CI/CD 환경을 생성할 때 기본 IAM 역할을 지정할 수 있는 새 기능을 반영하도록 [환경 생성](#) 섹션 및 기타 여러 섹션을 업데이트했습니다. 이 역할은 환경과 연결된 모든 워크플로 작업에 할당됩니다. 더 이상 IAM 역할을 작업에 직접 할당할 필요가 없습니다.

2024년 6월 21일

[새로운 내용: 자습서: 패키지 리포지토리에서 가져오기](#)

CodeCatalyst 패키지 리포지토리에서 종속성을 가져오도록 워크플로를 구성하는 방법을 설명하는 자습서를 추가했습니다.

2024년 6월 20일

[업데이트된 콘텐츠: 블루프린트와 함께 GitLab 프로젝트 리포지토리 사용](#)

[블루프린트를 사용하여 프로젝트 생성](#), [프로젝트에 블루프린트를 추가하여 리소스 통합](#) 또는 [사용자 지정 블루프린트 생성](#) 시 GitLab 프로젝트 리포지토리를 생성하는 기능에 대한 설명서가 업데이트되었습니다.

2024년 6월 19일

[업데이트된 콘텐츠: 자습서: 생성형 AI 기능 사용](#)

프로젝트를 생성하거나 기존 프로젝트에 블루프린트를 추가할 때 Amazon Q를 사용하여 블루프린트를 선택할 수 있는 기능을 반영하도록 자습서를 업데이트했습니다.

2024년 6월 18일

[업데이트된 콘텐츠: 소스 리포지토리 생성](#)

빈 리포지토리 생성에 대한 정보를 포함하도록 설명서를 업데이트했습니다.

2024년 6월 18일

[새로운 내용: Amazon Q를 사용하여 CodeCatalyst에서 프로젝트 생성](#)

Amazon Q를 사용하여 프로젝트를 생성하고 블루프린트를 추가하는 지침과 함께 제목이 [Amazon Q를 사용하여 프로젝트를 생성하거나 블루프린트를 사용하여 기능을 추가할 때의 모범 사례](#) 및 [프로젝트에서 블루프린트를 사용하는 모범 사례](#)인 새 섹션을 [프로젝트 생성](#)에 추가했습니다.

2024년 6월 18일

[새로운 내용: 기존 Git 리포지토리를 소스 리포지토리로 복제](#)

Git을 사용하여 미러 복제본 또는 로컬 리포지토리를 CodeCatalyst의 빈 소스 리포지토리로 푸시하는 방법에 대한 새 주제를 추가했습니다.

2024년 6월 18일

[새로운 내용: Maven, NuGet 및 Python 패키지 유형 지원](#)

CodeCatalyst에서 Maven, NuGet 및 Python 패키지를 사용하는 방법에 대한 설명서가 추가되었습니다.

2024년 6월 17일

[스페이스 및 계정 연결 업데이트](#)

이제 결제 계정으로 AWS 계정 지정된 여러 CodeCatalyst 스페이스에 연결할 수 있습니다. ID 페더레이션을 위해 설정된 스페이스의 경우 하나의 Identity Center 인스턴스를 여러 스페이스에 연결할 수 있습니다. [연결된를 사용하여 AWS 리소스에 대한 액세스 허용 AWS 계정](#), [스페이스와 연결된 계정의 문제 해결](#) 및 [CodeCatalyst 할당량](#) 섹션을 참조하세요.

2024년 6월 13일

업데이트된 콘텐츠: 역할 작업	Amazon Q를 사용하여 문제에 대한 작업을 추천하고 생성할 수 있는 권한을 포함하도록 역할에 대한 설명서를 업데이트했습니다.	2024년 6월 13일
업데이트된 콘텐츠: 역할 작업	문제 간 링크를 생성, 업데이트 및 제거할 수 있는 권한을 포함하도록 역할에 대한 설명서를 업데이트했습니다.	2024년 6월 13일
업데이트된 콘텐츠: 자습서: 생성형 AI 기능 사용	Amazon Q를 사용하여 문제에 대한 작업을 추천하는 방법에 대한 정보를 포함하도록 자습서를 업데이트했습니다.	2024년 6월 13일
업데이트된 콘텐츠: 문제에 대한 작업 관리	Amazon Q를 사용하여 CodeCatalyst의 문제에 대한 작업을 추천하는 방법에 대한 정보가 추가되었습니다.	2024년 6월 13일
새로운 내용: 문제를 다른 문제에 연결	CodeCatalyst의 다른 문제에 문제를 연결하는 방법에 대한 새 주제가 추가되었습니다.	2024년 6월 13일
업데이트 내용: 'AWS CDK 배포' 작업 YAML	Region 속성에 대한 설명을 수정했습니다.	2024년 6월 12일
업데이트된 콘텐츠: 블루프린트에 타사 리포지토리 사용	블루프린트를 사용하여 프로젝트 생성 , 프로젝트에 블루프린트를 추가하여 리소스 통합 또는 사용자 지정 블루프린트 생성 시 GitHub 리포지토리를 생성하는 기능에 대한 설명서가 업데이트되었습니다.	2024년 6월 6일

[새로운 내용: 개인 연결 작업 단계 추가](#)

개인 연결을 생성하고 삭제하는 단계가 추가되었습니다. 개인 연결을 사용하면 Amazon CodeCatalyst에서 프로젝트 및 블루프린트에 대한 GitHub 리소스를 관리할 수 있습니다.

2024년 6월 6일

[새로운 내용: Bitbucket 리포지토리 확장 프로그램](#)

CodeCatalyst에서 Bitbucket 리포지토리 확장 프로그램을 사용하기 위한 새로운 내용이 추가되었습니다.

2024년 6월 5일

[새로운 내용: 작업 유형](#)

SonarCloud Scan 작업을 언급하도록 [타사 작업](#) 주제를 업데이트했습니다.

2024년 5월 29일

[업데이트 내용: 'AWS CDK 배포' 작업 YAML](#)

CdkRootRootPath 예시를 수정했습니다.

2024년 5월 28일

[콘텐츠 업데이트](#)

주제 제목을 업데이트하고 콘텐츠를 재구성하여 가독성과 검색 기능을 개선했습니다. 이러한 변경 사항에 대한 피드백을 제공하려면 [피드백 제공 링크](#)를 사용하세요.

2024년 5월 17일

[새로운 내용: 파일 변경 내역 보기](#)

소스 리포지토리의 파일 변경 기록을 볼 수 있는 새 기능을 반영하도록 설명서를 업데이트했습니다.

2024년 5월 1일

[업데이트된 콘텐츠: 자습서: 생성형 AI 기능 사용](#)

Amazon Q Developer와의 통합을 반영하도록 자습서를 업데이트했습니다.

2024년 4월 29일

[업데이트된 콘텐츠: 자습서: 생성형 AI 기능 사용](#)

Amazon Q가 문제의 복잡성을 분석하고, 작업을 제안 및 생성하고, 문제의 작업을 수행하는 방법을 반영하도록 자습서를 업데이트했습니다.

2024년 4월 22일

[새로운 내용: 워크플로 게이팅](#)

워크플로 승인과 관련된 [워크플로 게이팅](#), [워크플로 실행에 대한 승인 요구](#) 및 기타 여러 주제를 추가했습니다.

2024년 4월 22일

[새로운 내용: 작업을 사용하여 문제를 더 작은 목표로 세분화](#)

문제에서 작업의 시작을 지원하는 콘텐츠를 추가했습니다. 문제에 작업을 추가하여 해당 문제의 작업을 더 세분화, 구성 및 추적할 수 있습니다.

2024년 4월 4일

[업데이트 내용: 작업 간 아티팩트 및 파일 공유](#)

두 가지 새로운 하위 주제를 포함하도록 [작업 간 아티팩트 및 파일 공유](#) 주제를 업데이트했습니다. [아티팩트를 출력 및 입력으로 지정하지 않고 공유할 수 있나요?](#) [워크플로 간에 아티팩트를 공유할 수 있나요?](#)

2024년 4월 2일

[업데이트 내용: CodeCatalyst의 GitHub Actions 제한 사항](#)

GitHub Actions가 이전 런타임 환경 Docker 이미지에서 실행됨을 나타내도록 [CodeCatalyst의 GitHub Actions 제한 사항](#) 주제를 업데이트했습니다.

2024년 4월 2일

[새로운 내용: 'AWS CDK 배포' 작업 YAML](#)

['AWS CDK 배포' 작업 YAML](#)에 새 CloudAssemblyRootPath 속성을 추가했습니다.

2024년 4월 1일

업데이트 내용: 런타임 환경 이미지 지정

새로운 2024년 3월 런타임 환경 이미지에 대한 정보를 포함하도록 [런타임 환경 이미지 지정](#) 주제를 업데이트했습니다.

2024년 3월 26일

업데이트된 콘텐츠: 역할 작업

역할 권한 정보를 단일 테이블로 통합했습니다. 해당 표는 [새 각 역할이 사용할 수 있는 권한 보기](#) 주제에 있습니다.

2024년 3월 18일

새로운 내용: 사용자의 모든 스페이스 및 프로젝트 보기

CodeCatalyst에서 로그인된 사용자의 CodeCatalyst 스페이스 또는 프로젝트를 보여주는 사용자 홈 페이지의 목록을 보는 방법에 대한 정보가 추가되었습니다. [사용자의 모든 스페이스 및 프로젝트 보기](#)을(를) 참조하세요.

2024년 3월 18일

새로운 내용: 예시: 풀 및 브랜치가 있는 트리거

풀 요청 트리거의 예를 추가했습니다. [트리거를 사용하여 워크플로 실행 자동 시작](#) 주제를 전반적으로 약간 수정했습니다.

2024년 3월 11일

업데이트된 콘텐츠: 역할 작업

환경에 대한 생성, 삭제 및 보기 권한을 포함하도록 역할에 대한 설명서를 업데이트했습니다.

2024년 3월 4일

업데이트된 콘텐츠: 자습서: 생성형 AI 기능 사용

Amazon Q에 문제를 생성하고 할당할 때 변경 사항을 반영하도록 자습서를 업데이트했습니다.

2024년 3월 4일

새로운 내용: 구성 요소 문제	문제 구성 요소를 사용자 지정 블루프린트 개발자로 사용하는 방법에 대한 새로운 내용이 추가되었습니다.	2024년 2월 27일
업데이트 내용: 작업 유형	CodeCatalyst Labs 작업 목록을 포함하도록 CodeCatalyst Labs 작업 주제를 업데이트했습니다.	2024년 2월 21일
업데이트된 콘텐츠: 풀 요청 작업	승인 규칙이 포함된 새 기능을 반영하고 풀 요청을 병합하기 위한 요구 사항을 재정의하도록 설명서를 업데이트했습니다.	2024년 2월 15일
업데이트된 콘텐츠: 풀 요청 병합	필수 검토자로부터 아직 승인을 받지 않았거나 승인 규칙을 충족하지 못한 풀 요청을 병합하기 위한 병합 요구 사항 재정의에 대한 정보를 포함하도록 풀 요청에 대한 설명서를 추가했습니다.	2024년 2월 15일
새로운 내용: 승인 규칙 관리	승인 규칙 생성 및 관리에 대한 정보를 포함하도록 풀 요청에 대한 설명서를 추가했습니다.	2024년 2월 15일
업데이트된 콘텐츠: 역할 작업	승인 규칙 및 풀 요청 작업 권한을 포함하도록 역할에 대한 설명서를 업데이트했습니다.	2024년 2월 14일
업데이트 내용: 'Workflow definition has <i>n</i> errors' 오류를 해결하려면 어떻게 해야 하나요?	더 많은 문제 해결 팁을 포함하도록 'Workflow definition has <i>n</i> errors' 오류를 해결하려면 어떻게 해야 하나요? 섹션을 업데이트했습니다.	2024년 2월 9일

새로운 내용: 워크플로의 상태 보기	워크플로 상태를 설명하는 섹션을 추가했습니다.	2024년 2월 9일
새로운 내용: 워크플로의 상태 보기	워크플로 상태를 설명하는 섹션을 추가했습니다.	2024년 2월 9일
업데이트 내용: CodeCatalyst의 워크플로 할당량	CodeCatalyst의 워크플로 할당량 주제에서 워크플로당 최대 작업 수와 스페이스당 AWS 계정으로 연결된 최대 환경 수 할당량을 업데이트했습니다.	2024년 2월 7일
업데이트 내용: 환경 생성	환경당 최대 하나의 계정 연결을 사용할 수 있음을 나타내도록 환경 생성 섹션을 업데이트했습니다.	2024년 1월 31일
새로운 내용: 사용자 지정 블루프린트 GitHub 리포지토리	공개적으로 사용할 수 있는 GitHub 리포지토리에 대한 새로운 내용이 추가되었습니다.	2024년 1월 10일
업데이트된 콘텐츠: CodeCatalyst를 사용하여 npm 구성	CodeCatalyst에서 npm을 사용하는 방법에 대한 일반 구성 지침을 업데이트하고 always-auth=true 옵션에 대한 명확성을 추가했습니다.	2024년 1월 5일
업데이트된 콘텐츠: 풀 요청 작업	CodeCatalyst의 생성형 AI 기능과 함께 새로운 기능을 반영하도록 설명서를 업데이트했습니다.	2023년 11월 28일
업데이트된 콘텐츠: 문제 생성	CodeCatalyst의 생성형 AI 기능과 함께 새로운 기능을 반영하도록 설명서를 업데이트했습니다.	2023년 11월 28일

[새로운 내용: 자습서: 생성형 AI 기능 사용](#)

Amazon CodeCatalyst에서 생성형 AI 기능을 사용하기 위한 자습서가 추가되었습니다.

2023년 11월 28일

[새로운 내용: 사용자 지정 블루프린트 및 수명 주기 관리](#)

Amazon CodeCatalyst에서 사용자 지정 블루프린트 및 수명 주기 관리 기능을 사용하기 위한 새로운 내용이 추가되었습니다.

2023년 11월 27일

[업데이트 내용: 자습서: 최신 3계층 웹 애플리케이션 블루프린트로 프로젝트 만들기](#)

수정 및 문제 해결 정보로 자습서를 업데이트했습니다.

2023년 11월 22일

[업데이트 내용: 트리거를 사용하여 워크플로 실행 자동 시작](#)

풀 요청 트리거와 관련된 몇 가지 예시 및 설명을 수정했습니다. [트리거 및 브랜치 사용 지침](#) 섹션이 추가되었습니다.

2023년 11월 22일

[새로운 내용: SSO로 로그인](#)

AWS Single Sign-On(SSO)으로 로그인하는 방법에 대한 정보와 ID 페더레이션을 지원하는 CodeCatalyst 스페이스 설정 및 관리 정보에 대한 링크를 추가했습니다. [CodeCatalyst 설정 및 로그인](#) 및 [SSO로 로그인](#) 섹션을 참조하세요.

2023년 11월 17일

[업데이트된 콘텐츠: 역할 작업](#)

팀 작업, VPC 연결, SSO(Single Sign-On) 및 시스템 리소스에 대한 권한을 포함하도록 역할에 대한 설명서를 업데이트했습니다.

2023년 11월 16일

[업데이트된 콘텐츠: 풀 요청 작업](#)

풀 요청에 대한 변경 사항이 표시되는 방법의 변경 사항을 반영하도록 설명서를 업데이트했습니다.

2023년 11월 16일

업데이트된 콘텐츠: CodeCatalyst 할당량	CodeCatalyst 할당량 주제에서 스페이스에 대한 최대 VPC 연결 수 할당량을 업데이트했습니다.	2023년 11월 16일
CodeCatalyst에 대한 새로운 마이그레이션 주제	새로운 마이그레이션 주제입니다.	2023년 11월 16일
새로운 내용: 스페이스 및 CodeCatalyst 프로젝트에 대한 팀 관리	스페이스의 팀 사용에 대한 정보가 추가되었습니다. 팀을 사용하여 스페이스 액세스 허용 및 팀을 통해 프로젝트 액세스 허용 섹션을 참조하세요.	2023년 11월 16일
새로운 내용: 스페이스 내 블루프린트 및 워크플로를 위한 시스템 리소스 관리	스페이스의 시스템 리소스 사용에 대한 정보가 추가되었습니다. 시스템 리소스에 대한 스페이스 액세스 허용 을(를) 참조하세요.	2023년 11월 16일
새로운 내용: CodeCatalyst 프로젝트 내 블루프린트 및 워크플로를 위한 시스템 리소스 관리	CodeCatalyst 프로젝트에서 시스템 리소스를 사용하는 방법에 대한 정보가 추가되었습니다. 시스템 리소스에 대한 프로젝트 액세스 허용 을(를) 참조하세요.	2023년 11월 16일
새로운 내용: VPC 연결을 환경과 연결	VPC 연결을 워크플로에 사용할 수 있는 환경과 연결하기 위한 설명서가 추가되었습니다.	2023년 11월 16일
새로운 내용: VPC 연결을 개발 환경에 연결	VPC 연결에서 개발 환경을 사용하는 방법에 대한 설명서가 추가되었습니다.	2023년 11월 16일
새로운 내용	Amazon CodeCatalyst 관리자 안내서 의 최초 발행.	2023년 11월 16일

새로운 내용: 'AWS CDK 배포' 작업 YAML	'AWS CDK 배포' 작업 YAML 및 'AWS CDK 부트스트랩' 작업 YAML 에 새 CdkCliVersion 속성을 추가했습니다.	2023년 11월 14일
업데이트된 콘텐츠: 역할 작업	브랜치 규칙 작업에 대한 권한을 포함하도록 역할에 대한 설명서를 업데이트했습니다.	2023년 11월 13일
업데이트된 콘텐츠: 소스 리포지토리, 워크플로 및 개발 환경 문제 해결	브랜치 규칙 작업에 대한 정보를 포함하도록 문제 해결 주제를 업데이트했습니다.	2023년 11월 13일
업데이트 내용: 빌드 및 테스트 작업 YAML	Environment 속성에 대한 설명서가 업데이트되었습니다. 이제 빌드 및 테스트 작업을 위한 선택적 필드입니다.	2023년 11월 13일
새로운 내용: 브랜치 규칙 관리	소스 리포지토리에서 브랜치에 대한 규칙을 보고 브랜치 규칙을 생성하고 관리하는 방법에 대한 정보를 포함하도록 브랜치에 대한 설명서를 추가했습니다.	2023년 11월 13일
업데이트된 콘텐츠: 풀 요청 작업	풀 요청에 대한 정보가 표시되는 방법의 변경 사항을 반영하도록 설명서를 업데이트했습니다.	2023년 11월 10일
업데이트 내용: 워크플로 실행 간 파일 캐싱	파일 캐싱 제한을 포함하도록 설명서를 업데이트했습니다.	2023년 11월 10일
업데이트 내용: 자습서: Amazon EKS에 애플리케이션 배포	EKS 앱 배포 블루프린트를 언급하도록 설명서를 업데이트했습니다.	2023년 11월 9일

[새로운 내용: CodeCatalyst의 패키지](#)

CodeCatalyst에서 패키지를 사용하는 방법에 대한 설명서가 추가되었습니다.

2023년 11월 1일

[새 콘텐츠 및 업데이트된 콘텐츠: 역할 작업](#)

CodeCatalyst의 네 가지 새 역할인 파워 유저, 제한된 액세스, 검토자 및 읽기 전용에 대한 설명서를 업데이트했습니다.

2023년 11월 1일

[업데이트 내용: GitHub 출력 파라미터 내보내기](#)

set-output 명령 대신 GITHUB_OUTPUT 환경 파일을 사용하도록 예시를 업데이트했습니다. 환경 파일을 사용하는 것이 GitHub 권장 출력 파라미터 설정 방법입니다.

2023년 10월 24일

[새로운 내용: 트리거를 사용하여 워크플로 실행 자동 시작](#)

일정 트리거에 대한 설명서가 추가되었습니다.

2023년 10월 16일

[업데이트 내용: 'Kubernetes 클러스터에 배포' 작업 YAML](#)

'[Kubernetes 클러스터에 배포](#)' 작업 YAML 및 [자습서: Amazon EKS에 애플리케이션 배포](#) 주제에 CodeCatalystWorkflowDevelopmentRole- *spaceName* 역할 사용에 대한 정보가 추가되었습니다.

2023년 9월 22일

[업데이트된 콘텐츠: CodeCatalystWorkflowDevelopmentRole-**spaceName** 역할에 대한 새 역할 이름 및 정책](#)

개발자 역할 이름 변경에 대한 단계 및 역할 설명을 CodeCatalystWorkflowDevelopmentRole-**spaceName** 으로 업데이트했습니다. 이제 개발자 역할이 AdministratorAccess AWS 관리형 정책을 사용합니다. [CodeCatalystWorkflowDevelopmentRole-**spaceName** 서비스 역할 이해 및 새 스페이스 및 개발 역할 생성\(초대 없이 시작\)](#) 섹션을 참조하세요.

2023년 9월 20일

[업데이트 내용: 워크플로에서 변수 사용](#)

사용자 정의 변수와 사전 정의된 변수라는 두 가지 새로운 개념을 도입했습니다. 이러한 개념을 사용하면 [워크플로에서 변수 사용](#) 섹션을 더 쉽게 읽고 이해할 수 있습니다.

2023년 9월 19일

[업데이트된 콘텐츠: 커밋 작업](#)

표시된 정보의 변경 사항을 반영하고 여러 상위 엔티티로 커밋을 보는 방법에 대한 세부 정보를 제공하도록 설명서를 업데이트했습니다.

2023년 9월 7일

[새로운 내용: 트리거를 사용하여 워크플로 실행 자동 시작](#)

[트리거를 사용하여 워크플로 실행 자동 시작](#) 주제에 다음 예시가 추가되었습니다. [예시: 간단한 '메인으로 푸시' 트리거](#)

2023년 9월 6일

[업데이트된 콘텐츠: 풀 요청 작업](#)

풀 요청을 생성할 때 소스 브랜치 및 대상 브랜치의 표시 순서 변경을 반영하도록 설명서를 업데이트했습니다.

2023년 8월 30일

새로운 내용: 기본 브랜치 보기 및 변경

소스 리포지토리의 기본 브랜치를 보고 변경하는 방법에 대한 정보를 포함하도록 브랜치에 대한 설명서를 추가했습니다.

2023년 8월 30일

업데이트 내용: 'Kubernetes 클러스터에 배포' 작업 YAML

'Kubernetes 클러스터에 배포' 작업 YAML의 [Manifests](#) 속성 설명에 Helm 및 Kustomize에 대한 참고 사항이 추가되었습니다.

2023년 8월 15일

새로운 내용: 문제 첨부 파일 관리

문제에 대한 첨부 파일 작업 및 관리에 대한 설명서가 추가되었습니다.

2023년 8월 15일

업데이트 내용: 트리거를 사용하여 워크플로 실행 자동 시작

워크플로 트리거와 관련된 설명서를 개선하고 확장했습니다.

2023년 8월 11일

새로운 내용: 역할 권한 문제 해결

Amazon CodeGuru 액세스가 필요한 워크플로를 실행하기 위해 역할 권한을 업데이트하는 방법에 대한 정보가 추가되었습니다. [Amazon CodeGuru에 대한 권한 오류로 최신 3계층 웹 애플리케이션 블루프린트 OnPullRequest 워크플로 실패](#)(를) 참조하세요.

2023년 8월 11일

새로운 내용: 'Workflow is inactive' 메시지를 해결하려면 어떻게 해야 하나요?

다음 문제 해결 주제가 추가되었습니다. ['Workflow is inactive' 메시지를 해결하려면 어떻게 해야 하나요?](#)

2023년 8월 11일

[새로운 내용: 워크플로를 사용하여 Amazon EKS에 배포](#)

Kubernetes 클러스터에 배포 작업에 대한 설명서가 추가되었습니다. 자세한 내용은 [워크플로를 사용하여 Amazon EKS에 배포](#) 및 [자습서: Amazon EKS에 애플리케이션 배포](#) 섹션을 참조하세요.

2023년 7월 27일

[CodeCatalyst 스페이스에 대한 관리 이벤트 로깅 방법에 대한 업데이트](#)

를 사용하여 CodeCatalyst 스페이스의 특정 작업에 대해 관리 이벤트를 로깅하는 방법에 대한 정보가 추가되었습니다. AWS CloudTrail. list-event-logs 명령을 사용하여 스페이스의 모든 이벤트를 보는 방법에 대한 정보가 추가되었습니다. [로깅을 사용하여 이벤트 및 API 직접 호출 모니터링](#)(를) 참조하세요.

2023년 7월 20일

[업데이트 내용: 트리거를 사용하여 워크플로 실행 자동 시작](#)

이제 풀 요청 트리거가 GitHub 소스 리포지토리에서 지원됨을 나타내도록 설명서를 업데이트했습니다. 이전에는 풀 요청 트리거가 CodeCatalyst 소스 리포지토리에서만 지원되었습니다.

2023년 7월 14일

[업데이트 내용: CodeCatalyst의 워크플로 할당량](#)

[CodeCatalyst의 워크플로 할당량](#) 주제에서 작업이 실행할 수 있는 최대 시간 할당량을 업데이트했습니다.

2023년 6월 27일

[업데이트 내용: 워크플로 YAML 정의](#)

Compute 코드 블록의 형식 지정 오류를 수정했습니다.

2023년 6월 27일

업데이트된 콘텐츠: 데이터 보호	데이터 복제에 대한 추가 정보를 포함하도록 설명서를 업데이트했습니다.	2023년 6월 26일
새로운 내용: 사용할 작업 버전 지정	사용할 작업 버전 지정 주제가 추가되었습니다.	2023년 6월 21일
업데이트 내용: AWS 계정 및 VPCs에 배포	CodeCatalyst에 배포 정보가 표시되는 것을 지원하는 작업은 무엇입니까? 섹션을 명확히 했습니다.	2023년 6월 14일
업데이트된 콘텐츠: 재구성된 문제 설명서	전체 설명서 세트 및 사용자 흐름에 맞게 대부분의 문제 설명서를 재구성했습니다.	2023년 5월 31일
업데이트된 콘텐츠: 문제 보기 전환기	업데이트된 문제 보기 전환기와 일치하도록 다양한 사용자 흐름을 업데이트했습니다.	2023년 5월 31일
업데이트된 콘텐츠: 알림 관리	개인 Slack 알림 구성에 대한 정보를 포함하도록 알림 설명서를 업데이트했습니다.	2023년 5월 30일
업데이트된 콘텐츠: 알림 관리	개인 Slack 알림 구성에 대한 정보를 포함하도록 알림 설명서를 업데이트했습니다.	2023년 5월 30일
새로운 내용: CodeCatalyst 신뢰 모델	CodeCatalyst가 연결된 AWS 계정에서 서비스 역할을 받아들이도록 허용하는 신뢰 모델에 대한 정보가 포함된 새 주제를 추가했습니다. CodeCatalyst에 대해 정의된 서비스 위탁자에 대한 새 섹션이 추가되었습니다. CodeCatalyst 신뢰 모델 이해 (를) 참조하세요.	2023년 5월 20일

업데이트 내용: 워크플로에 소스 리포지토리 연결	소스 리포지토리 파일 참조 의 지침을 간소화했습니다.	2023년 5월 10일
업데이트 내용: CodeCatalyst의 워크플로 할당량	CodeCatalyst의 워크플로 할당량 주제의 출력 변수 값 최대 길이 할당량을 업데이트했습니다.	2023년 5월 10일
새로운 내용: 작업 간 아티팩트 및 파일 공유	두 가지 예시(예시: 단일 아티팩트에서 파일 참조 및 예시: WorkflowSource가 있을 때 아티팩트의 파일 참조)가 추가되었습니다.	2023년 5월 10일
업데이트된 콘텐츠: 풀 요청 작업	풀 요청 이벤트에 대한 이메일 기본 설정 구성 정보를 포함하도록 풀 요청에 대한 설명서를 업데이트했습니다.	2023년 4월 21일
업데이트된 콘텐츠: 알림 관리	풀 요청 이벤트에 대한 이메일 기본 설정 구성 정보를 포함하도록 알림에 대한 설명서를 업데이트했습니다.	2023년 4월 21일
관리형 정책으로 업데이트	AWS 관리형 정책: AmazonCodeCatalyst FullAccess , AWS 관리형 정책: AmazonCodeCatalyst ReadOnlyAccess 및 AWS 관리형 정책: AmazonCodeCatalystSupportAccess 관리형 정책이 추가되었습니다. AWS 관리형 정책에 대한 CodeCatalyst 업데이트 을(를) 참조하세요.	2023년 4월 20일
새로운 내용: 배포 대상 제거	배포 대상 제거 주제가 추가되었습니다.	2023년 4월 20일

새로운 내용: 작업 유형	CodeCatalyst 작업 주제가 추가되었습니다.	2023년 4월 20일
스페이스에서 스페이스 관리자 역할인 사용자를 관리하기 위한 업데이트	스페이스에서 스페이스 관리자 역할이 있는 사용자의 역할을 제거 또는 변경하는 방법에 대한 정보가 추가되었습니다. 스페이스 관리자 역할이 있는 사용자의 역할 제거 또는 변경 (를) 참조하세요.	2023년 4월 19일
개발 환경 관리를 위한 업데이트	개발 환경을 스페이스 관리자로서 관리하는 방법에 대한 정보가 추가되었습니다. 스페이스의 개발 환경 관리 (를) 참조하세요.	2023년 4월 19일
업데이트된 콘텐츠: 문제 찾기 및 보기	문제 찾기 및 보기 주제와 하위 주제를 재구성했습니다.	2023년 4월 19일
업데이트된 콘텐츠: 연결된 GitLab 프로젝트 리포지토리를 사용하여 CodeCatalyst에서 프로젝트 생성	연결된 타사 리포지토리를 사용하여 프로젝트 생성 섹션의 GitLab 통합을 포함하도록 프로젝트 생성 이 업데이트되었습니다.	2023년 4월 19일
업데이트 내용: 컴퓨팅 및 런타임 이미지 구성	Amazon Linux 2의 Arm64 아키텍처에 대한 지원이 추가되었습니다.	2023년 4월 19일
새로운 내용: 그룹 내에서 문제 이동	보드의 그룹 내에서 문제를 이동하기 위한 설명서 와 모든 문제 보기가 추가되었습니다.	2023년 4월 19일

업데이트 내용: CodeCatalyst의 워크플로 할당량	CodeCatalyst의 워크플로 할당량 주제의 누락된 할당량을 업데이트하고 단일 작업 출력 변수의 최대 총 크기 할당량을 2KB에서 120KB로 업데이트했습니다.	2023년 4월 18일
새로운 내용: 작업의 소스 코드 보기	작업의 소스 코드 보기 주제가 추가되었습니다.	2023년 4월 18일
새로운 내용: 워크플로 실행 중지	워크플로 실행 중지 주제가 추가되었습니다.	2023년 4월 10일
새 콘텐츠: AWS 와 Amazon CodeCatalyst 간의 계정 연결을 위한 리소스 태그 지정 섹션 추가	계정 연결 리소스에 태그를 지정하고 연결 리소스에 대한 IAM 정책을 관리하기 위한 정보가 추가되었습니다. 태그를 사용하여 계정 연결 리소스에 대한 액세스 제어 및 CodeCatalyst 권한 참조 섹션을 참조하세요.	2023년 4월 6일
새로운 내용: 작업 유형	작업 유형 주제를 추가했습니다.	2023년 4월 6일
업데이트 내용: '스 CloudFormation 템플릿 배포' 작업 YAML	parameter-overrides 속성 설명이 업데이트되었습니다. 이제 JSON 파일을 지원합니다.	2023년 4월 5일
새로운 내용: 연결된 GitHub 리포지토리를 사용하여 CodeCatalyst에서 프로젝트 생성	GitHub 리포지토리에 연결하는 프로젝트를 생성하는 지침과 함께 제목이 연결된 타사 리포지토리를 사용하여 프로젝트 생성 인 새 섹션을 프로젝트 생성 에 추가했습니다.	2023년 4월 5일

업데이트된 콘텐츠: 알림 작업	프로젝트 이벤트에 대한 이메일 구성 정보를 포함하도록 알림 설명서를 업데이트했습니다.	2023년 3월 31일
새로운 내용	Amazon CodeCatalyst Action Development Kit 가이드 의 최초 발행.	2023년 3월 31일
업데이트된 콘텐츠: Amazon CodeCatalyst의 스페이스 섹션 재구성	랜딩 페이지를 제거하고 주제를 통합하여 스페이스 섹션을 업데이트했습니다.	2023년 3월 29일
업데이트 내용: 자습서: Amazon ECS에 애플리케이션 배포	AWS IAM Identity Center 대신에서 사용자를 생성하는 방법을 설명 1단계: AWS 사용자 설정 및 AWS CloudShell 하도록 변경되었습니다 AWS Identity and Access Management. IAM 사용자를 생성하는 것은 더 이상 권장되지 않습니다.	2023년 3월 23일
업데이트된 콘텐츠: 역할 작업	문제를 풀 요청과 연결할 수 있는 권한을 포함하도록 스페이스 관리자, 프로젝트 관리자 및 기고자 역할에 대한 설명서를 업데이트했습니다.	2023년 3월 13일
업데이트된 콘텐츠: 풀 요청 작업	풀 요청에 문제 연결에 대한 정보를 포함하도록 풀 요청에 대한 설명서를 업데이트했습니다.	2023년 3월 13일
업데이트된 콘텐츠: 문제 작업	풀 요청에 문제 연결에 대한 정보를 포함하도록 문제에 대한 설명서를 업데이트했습니다.	2023년 3월 13일

[새로운 내용: 프로젝트의 모든 실행 상태 및 세부 정보 보기](#)

새로운 집계된 워크플로 실행 페이지를 설명하는 섹션을 추가했습니다.

2023년 3월 8일

[새로운 내용: 'Workflow definition has *n* errors' 오류를 해결하려면 어떻게 해야 하나요?](#)

'The workflow definition has errors' 오류를 해결하는 방법에 대한 섹션을 추가했습니다.

2023년 3월 7일

[업데이트 내용: 워크플로 생성](#)

새 UI를 반영하도록 지침을 업데이트했습니다.

2023년 3월 3일

[새로운 내용: universal-test-runner와 통합](#)

[universal-test-runner](#)와 통합 주제가 추가되었습니다.

2023년 3월 3일

[업데이트 내용: 워크플로를 사용하여 빌드, 테스트 및 배포](#)

워크플로 요약 페이지의 새 소스 리포지토리, 브랜치 및 워크플로 이름 필터를 반영하도록 다양한 섹션을 업데이트했습니다.

2023년 3월 2일

[새로운 내용: 커밋별 배포 상태 추적](#)

커밋별 코드 품질 및 배포 상태 보기 섹션을 추가했습니다.

2023년 2월 27일

[새로운 내용: 'BranchName' 및 'CommitId' 변수](#)

사전 정의된 새 BranchName 변수가 추가되었습니다.

2023년 2월 16일

[업데이트된 콘텐츠: Amazon CodeCatalyst의 스페이스 멤버 관리](#)

CodeCatalyst에서 사용자가 할당한 역할에 따라 멤버 역할 변경, 멤버 초대 및 두 개의 새 테이블에서 멤버 제거 방법에 대한 정보가 업데이트되었습니다.

2023년 2월 15일

[업데이트된 콘텐츠: Amazon CodeCatalyst 콘솔에 PAT 관리 단계 추가](#)

콘솔에서 PAT를 보고, 생성하고, 삭제하는 단계가 추가되었습니다.

2023년 2월 15일

업데이트 내용: 런타임 환경 이미지 지정	기본 이미지 도구 버전 테이블에 도구를 더 추가했습니다.	2023년 1월 10일
업데이트 내용: 작업 간 아티팩트 및 파일 공유	아티팩트 경로를 수정했습니다.	2023년 1월 3일
업데이트 내용: 'GitHub Actions' 작업 YAML	Steps 섹션의 코드 조각을 수정했습니다.	2023년 1월 3일
업데이트 내용: 워크플로에 소스 리포지토리 연결	소스 경로를 수정했습니다.	2023년 1월 3일
업데이트된 콘텐츠: 풀 요청 업데이트	풀 요청에 필요한 검토자 또는 선택적 검토자 업데이트에 대한 정보를 포함하도록 설명서를 업데이트했습니다.	2022년 12월 23일
새로운 내용: 워크플로 실행 간 파일 캐싱	워크플로에서의 파일 캐싱을 위한 페이지가 추가되었습니다.	2022년 12월 20일
업데이트된 콘텐츠: 풀 요청 작업	알림에 대한 정보를 포함하도록 풀 요청에 대한 설명서를 업데이트했습니다.	2022년 12월 16일
새로운 내용: 'AWS CDK 배포' 작업 YAML	새 CdkRootPath 속성을 추가했습니다.	2022년 12월 16일
새로운 내용: 작업 간에 컴퓨팅 공유	작업 간에 컴퓨팅 공유 주제가 추가되었습니다.	2022년 12월 14일
업데이트 내용: 작업 간 아티팩트 및 파일 공유	입력 아티팩트를 지정하는 방법을 보여주는 예시가 수정되었습니다.	2022년 12월 13일
새로운 내용: 'GitHub Actions' 작업 YAML	GitHub Actions 작업에 대한 전용 참조 페이지가 추가되었습니다.	2022년 12월 13일

[업데이트된 콘텐츠: CodeCatalyst의 프로젝트 할당량](#)

한 스페이스에 최대 100개의 프로젝트로 설명서를 업데이트했습니다.

2022년 12월 2일

[새로운 내용](#)

Amazon CodeCatalyst 사용 설명서 최초 발행.

2022년 12월 1일

[새로운 내용: 확장 프로그램 관련 문제 해결](#)

타사 확장 프로그램을 사용할 때 사용자가 직면할 수 있는 가능한 문제에 대한 문제 해결 주제를 추가했습니다.

2022년 6월 18일

AWS 용어집

최신 AWS 용어는 AWS 용어집 참조의 [AWS 용어집](#)을 참조하세요.