

# vRealize Automation Code Stream 사용 및 관리

2022년 12월 14일

vRealize Automation 8.7

다음 VMware 웹 사이트에서 최신 기술 문서를 확인할 수 있습니다.

<https://docs.vmware.com/kr/>

**VMware, Inc.**  
3401 Hillview Ave.  
Palo Alto, CA 94304  
[www.vmware.com](http://www.vmware.com)

**VMware 코리아**  
서울시 강남구  
영동대로 517  
아셈타워 13층  
(우) 06164  
전화: +82 2 3016 6500  
팩스: +82 2 3016 6501  
[www.vmware.com/kr](http://www.vmware.com/kr)

Copyright © 2022 VMware, Inc. All rights reserved. 저작권 및 상표 정보

# 목차

<b>1</b>	<b>Code Stream 소개 및 작동 방식</b>	<b>5</b>
<b>2</b>	<b>릴리스 프로세스를 모델링하도록 설정</b>	<b>9</b>
	프로젝트를 추가하는 방법	13
	사용자 액세스 및 승인을 관리하는 방법	14
	사용자 작업 및 승인이란?	21
<b>3</b>	<b>파이프라인 생성 및 사용</b>	<b>24</b>
	파이프라인을 실행하고 결과를 보는 방법	26
	사용할 수 있는 작업 유형	31
	파이프라인에서 변수 바인딩을 사용하는 방법	36
	파이프라인의 실행 또는 중지를 위해 조건 작업에서 변수 바인딩을 사용하는 방법	45
	파이프라인 작업을 바인딩할 때 사용할 수 있는 변수 및 표현식	47
	내 파이프라인에 대한 알림을 보내는 방법	65
	파이프라인 작업이 실패한 경우에 Jira 티켓을 생성하는 방법	67
	배포를 롤백하는 방법	70
<b>4</b>	<b>기본적인 코드 구축, 통합 및 제공 계획</b>	<b>76</b>
	파이프라인 작업 공간 구성	76
	스마트 파이프라인 템플릿을 사용하기 전에 CICD 기본 구축 계획	79
	스마트 파이프라인 템플릿을 사용하기 전에 CI 네이티브 빌드 계획	85
	스마트 파이프라인 템플릿을 사용하기 전에 CD 네이티브 빌드 계획	86
	작업을 수동으로 추가하기 전에 CICD 기본 구축 계획	87
	롤백 계획	93
<b>5</b>	<b>자습서</b>	<b>95</b>
	GitHub 또는 GitLab 저장소의 코드를 파이프라인에 지속적으로 통합하는 방법	96
	YAML 클라우드 템플릿에서 배포하는 애플리케이션의 릴리스를 자동화하는 방법	100
	Kubernetes 클러스터에 애플리케이션의 릴리스를 자동화하는 방법	107
	애플리케이션을 블루-그린 배포에 배포하는 방법	115
	자체 빌드, 테스트 및 배포 도구를 통합하는 방법	120
	다음 작업에서 클라우드 템플릿 작업의 리소스 속성을 사용하는 방법	130
	REST API를 사용하여 다른 애플리케이션과 통합하는 방법	134
	파이프라인을 코드로 활용하는 방법	138
<b>6</b>	<b>끝점에 연결</b>	<b>143</b>

- 끝점 소개 143
- Jenkins와 통합하는 방법 145
- Git와 통합하는 방법 151
- Gerrit와 통합하는 방법 153
- vRealize Orchestrator와 통합하는 방법 156

## 7 파이프라인 트리거 162

- Docker 트리거를 사용하여 지속적 전달 파이프라인을 실행하는 방법 162
- Git 트리거를 사용하여 파이프라인을 실행하는 방법 170
- Gerrit 트리거를 사용하여 파이프라인을 실행하는 방법 177

## 8 파이프라인 모니터링 185

- 파이프라인 대시보드에 표시되는 내용 185
- 사용자 지정 대시보드를 사용하여 주요 성능 지표를 추적하는 방법 188

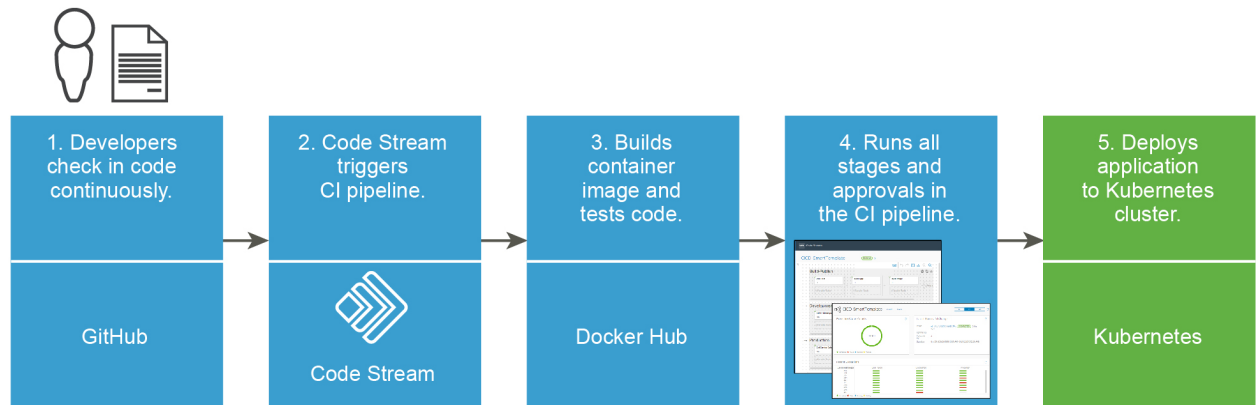
## 9 자세히 알아보기 191

- 검색이란? 191
- 관리자 및 개발자를 위한 추가 리소스 195

# Code Stream 소개 및 작동 방식

# 1

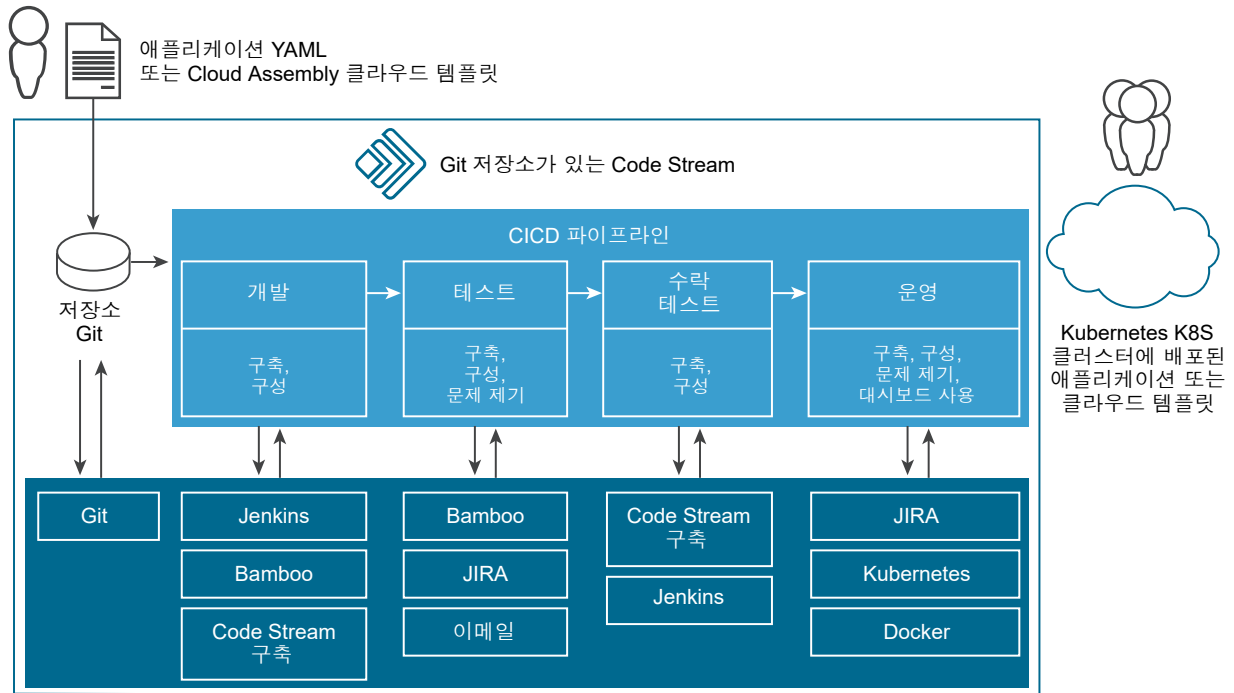
vRealize Automation Code Stream™은 CICD(지속적 통합 및 전달) 도구입니다. DevOps 수명 주기에서 소프트웨어 릴리스 프로세스를 모델링하는 파이프라인을 생성하면 소프트웨어를 신속하고 지속적으로 제공하는 코드 인프라를 구축할 수 있습니다.



Code Stream을 사용하여 소프트웨어를 전달할 때 DevOps 수명 주기에서 가장 중요한 두 가지 부분인 릴리스 프로세스와 개발자 툴을 통합합니다. Code Stream과 기존 배포 툴을 통합하는 초기 설정 후, 파이프라인은 전체 DevOps 수명 주기를 자동화합니다.

vRealize Automation 8.2부터 Blueprint를 VMware Cloud Templates라고 합니다.

소프트웨어를 구축, 테스트 및 릴리스하는 파이프라인을 생성합니다. Code Stream은 이 파이프라인을 사용하여 소스 코드 저장소에서 테스트를 거쳐 운영 환경으로 소프트웨어를 진행합니다.



지속적 통합 및 지속적 전달 파이프라인 계획에 대한 자세한 내용은 [장 4 Code Stream](#)에서 기본적인 코드 구축, 통합 및 제공 계획에서 확인할 수 있습니다.

## Code Stream 관리자가 Code Stream을 사용하는 방법

관리자는 끝점을 생성하고 개발자가 사용할 수 있는 작동 중인 인스턴스가 있는지 확인합니다. 파이프라인을 생성, 트리거, 관리하는 등의 작업을 수행할 수 있습니다. [Code Stream](#)에서 사용자 액세스 및 승인을 관리하는 방법에 설명된 대로 Administrator 역할이 있습니다.

표 1-1. Code Stream 관리자가 개발자를 지원하는 방법

개발자 지원을 위해...	수행할 수 있는 작업...
환경을 제공하고 관리합니다.	<p>개발자가 코드를 테스트하고 배포할 수 있는 환경을 조성합니다.</p> <ul style="list-style-type: none"> <li>■ 상태를 추적하고 이메일 알림을 보냅니다.</li> <li>■ 환경의 지속적인 작동을 보장하여 개발자의 생산성을 유지합니다.</li> </ul> <p>자세한 내용은 <a href="#">Code Stream</a> 관리자 및 개발자를 위한 추가 리소스 항목을 참조하십시오.</p> <p><a href="#">장 5 Code Stream</a> 사용을 위한 자습서 항목도 참조하십시오.</p>
끝점을 제공합니다.	개발자에게 해당 파이프라인에 연결할 수 있는 끝점의 작동 중인 인스턴스가 있는지 확인합니다.
다른 서비스와의 통합을 제공합니다.	<p>다른 서비스에 대한 통합이 작동되고 있는지 확인합니다.</p> <p>자세한 내용은 <a href="#">VMware Cloud Services</a> 설명서를 참조하십시오.</p>
파이프라인을 생성합니다.	<p>릴리스 프로세스를 모델링하는 파이프라인을 생성합니다.</p> <p>자세한 내용은 <a href="#">장 3 Code Stream</a>에서 파이프라인 생성 및 사용 항목을 참조하십시오.</p>

표 1-1. Code Stream 관리자가 개발자를 지원하는 방법 (계속)

개발자 지원을 위해...	수행할 수 있는 작업...
파이프라인을 트리거합니다.	<p>이벤트가 발생할 때 파이프라인이 실행되는지 확인합니다.</p> <ul style="list-style-type: none"> <li>■ 빌드 아티팩트가 생성되거나 업데이트될 때마다 독립형의 CD(지속적 전달) 파이프라인을 트리거하려면 <b>Docker</b> 트리거를 사용합니다.</li> <li>■ 개발자가 코드에 대한 변경을 커밋할 때 파이프라인을 트리거하려면 <b>Git</b> 트리거를 사용합니다.</li> <li>■ 개발자가 코드 검토, 병합 등을 수행할 때 파이프라인을 트리거하려면 <b>Gerrit</b> 트리거를 사용합니다.</li> <li>■ 빌드 아티팩트가 생성되거나 업데이트될 때마다 독립형 CD(지속적 전달) 파이프라인을 실행하려면 <b>Docker</b> 트리거를 사용합니다.</li> </ul> <p>자세한 내용은 <a href="#">장 7 Code Stream에서 파이프라인 트리거</a> 항목을 참조하십시오.</p>
파이프라인 및 승인을 관리합니다.	<p>파이프라인을 최신 상태로 유지합니다.</p> <ul style="list-style-type: none"> <li>■ 파이프라인 상태를 보고 파이프라인을 실행한 사람을 확인합니다.</li> <li>■ 파이프라인 실행에 대한 승인을 확인하고 활성 및 비활성 파이프라인 실행에 대한 승인을 관리합니다.</li> </ul> <p>자세한 내용은 <a href="#">Code Stream에서 사용자 지정 대시보드를 사용하여 파이프라인에 대한 주요 성능 지표를 추적하는 방법</a> 항목도 참조하십시오.</p>
개발자 환경을 모니터링합니다.	<p>파이프라인 상태, 추세, 메트릭 및 핵심 지표를 모니터링하는 사용자 지정 대시보드를 생성합니다. 사용자 지정 대시보드를 사용하여 개발자 환경에서 통과하거나 실패하는 파이프라인을 모니터링합니다. 충분히 사용되지 않은 리소스를 식별하여 보고하고 리소스를 확보할 수도 있습니다.</p> <p>다음 사항도 볼 수 있습니다.</p> <ul style="list-style-type: none"> <li>■ 파이프라인이 성공하기까지 경과한 실행 시간.</li> <li>■ 파이프라인이 승인을 위해 대기한 시간(승인 담당 사용자에게 알림).</li> <li>■ 가장 실패가 잦은 단계 및 작업.</li> <li>■ 실행 시간의 대부분을 차지하는 단계 및 작업.</li> <li>■ 개발 팀에서 진행 중인 릴리스.</li> <li>■ 배포 및 릴리스에 성공한 애플리케이션.</li> </ul> <p>자세한 내용은 <a href="#">장 8 Code Stream에서 파이프라인 모니터링</a> 항목을 참조하십시오.</p>
문제를 해결합니다.	<p>개발자 환경의 파이프라인 장애 문제를 해결합니다.</p> <ul style="list-style-type: none"> <li>■ CICD(지속적 통합 및 지속적 전달) 환경의 문제를 식별하고 해결합니다.</li> <li>■ 파이프라인 대시보드를 사용하고 사용자 지정 대시보드를 생성하면 더 많은 내용을 확인할 수 있습니다. <a href="#">장 8 Code Stream에서 파이프라인 모니터링</a>의 내용을 참조하십시오.</li> </ul> <p><a href="#">장 2 Code Stream</a>을 설정하여 릴리스 프로세스 모델링 항목도 참조하십시오.</p>

Code Stream은 VMware Cloud Services의 일부입니다.

- Cloud Assembly를 사용하여 클라우드 템플릿을 배포합니다.
- Service Broker를 사용하여 카탈로그에서 클라우드 템플릿을 가져옵니다.

수행할 수 있는 다른 작업에 대해 알아보려면 [VMware vRealize Automation 설명서](#)를 참조하십시오.

## 개발자가 Code Stream을 사용하는 방법

개발자는 Code Stream을 사용하여 파이프라인을 구축 및 실행하고 대시보드에서 파이프라인 작업을 모니터링합니다. [Code Stream에서 사용자 액세스 및 승인을 관리하는 방법](#)에 설명된 대로 User 역할이 있습니다.

파이프라인을 실행한 후 다음과 같은 내용을 알 수 있습니다.


- 파이프라인의 모든 단계에서 코드가 성공했는지 여부. 확인하려면 파이프라인 실행의 결과를 관찰합니다.
- 파이프라인이 실패한 경우 수행할 작업 및 실패의 원인. 확인하려면 파이프라인 대시보드에서 최상위 오류를 관찰합니다.

표 1-2. Code Stream을 사용하는 개발자

코드 통합 및 릴리스를 위해	수행할 작업
파이프라인을 구축합니다.	코드를 테스트하고 배포합니다. 파이프라인이 실패하면 코드를 업데이트합니다.
파이프라인을 끝점에 연결합니다.	파이프라인의 작업을 GitHub 저장소와 같은 끝점에 연결합니다.
파이프라인을 실행합니다.	파이프라인을 다른 사용자가 특정 지점에서 승인할 수 있도록 '사용자 작업' 승인 작업을 추가합니다.
대시보드를 봅니다.	파이프라인 대시보드에서 결과를 봅니다. 추세, 기록, 실패 등을 확인할 수 있습니다.

시작에 대한 자세한 내용은 [VMware Code Stream 시작](#)을 참조하십시오.

## 제품 내 지원 패널에서 더 많은 설명서 찾기

여기에서 필요한 정보를 찾지 못한 경우 제품에서 더 많은 도움말을 볼 수 있습니다. 

- 상황에 맞는 정보를 필요한 위치와 시기에 보려면 사용자 인터페이스에서 포지판 및 도구 설명을 클릭하고 읽으십시오.
- 제품 내 지원 패널을 열고 활성 사용자 인터페이스 페이지에 표시되는 항목을 읽으십시오. 패널에서 검색하여 질문에 대한 답변을 얻을 수도 있습니다.

### Webhook에 대한 추가 정보

동일한 Git 끝점을 사용하고 Webhook 구성 페이지에서 분기 이름에 서로 다른 값을 제공하여 여러 분기에 대해 다수의 Webhook을 생성하고 할 수 있습니다. 동일한 Git 저장소의 다른 분기에 대해 또 다른 Webhook을 생성하기 위해, 여러 분기에 대해 Git 끝점을 여러 번 복제할 필요가 없습니다. 대신, Webhook에 분기 이름을 제공하면 Git 끝점을 재사용할 수 있습니다. Git Webhook의 분기가 끝점의 분기와 동일한 경우 Git Webhook 페이지에서 분기 이름을 제공하지 않아도 됩니다.



# Code Stream을 설정하여 릴리스 프로세스 모델링

## 2

릴리스 프로세스를 모델링하려면 소프트웨어 릴리스에 일반적으로 사용하는 단계, 작업 및 승인을 나타내는 파이프라인을 생성합니다. 그러면 **Code Stream**은 코드를 구축, 테스트, 승인 및 배포하는 프로세스를 자동화합니다.

소프트웨어 릴리스 프로세스를 모델링하기 위한 모든 것이 준비되었으면 이제 **Code Stream**에서 다음과 같이 작업을 수행할 수 있습니다.

### 사전 요구 사항

- 사용 가능한 끝점이 이미 있는지 확인합니다. **Code Stream**에서 **끝점**을 클릭합니다.
- 코드 구축하고 배포할 수 있는 기본적인 방법을 숙지합니다. **장 4 Code Stream**에서 기본적인 코드 구축, 통합 및 제공 계획 항목을 참조하십시오.
- 파이프라인에 사용할 리소스 중 일부를 제한된 리소스로 표시할지 여부를 결정합니다. **Code Stream**에서 **사용자 액세스 및 승인을 관리하는 방법** 항목을 참조하십시오.
- 관리자 역할 대신 사용자 역할이나 뷰어 역할이 있는 경우, **Code Stream** 인스턴스의 관리자를 결정합니다.

### 절차

- 1 **Code Stream**에서 사용할 수 있는 프로젝트를 검토하여 적절한 프로젝트를 선택합니다.
  - 프로젝트가 나타나지 않으면 프로젝트를 생성하고 나를 프로젝트의 멤버로 만들 수 있는 **Code Stream** 관리자에게 문의합니다. **Code Stream**에서 프로젝트를 추가하는 방법의 내용을 참조하십시오.

- 나열된 프로젝트의 멤버가 아니면 나를 프로젝트의 멤버로 추가할 수 있는 **Code Stream** 관리자에게 요청합니다.



- ## 2 파이프라인에 필요한 새 끝점을 추가합니다.

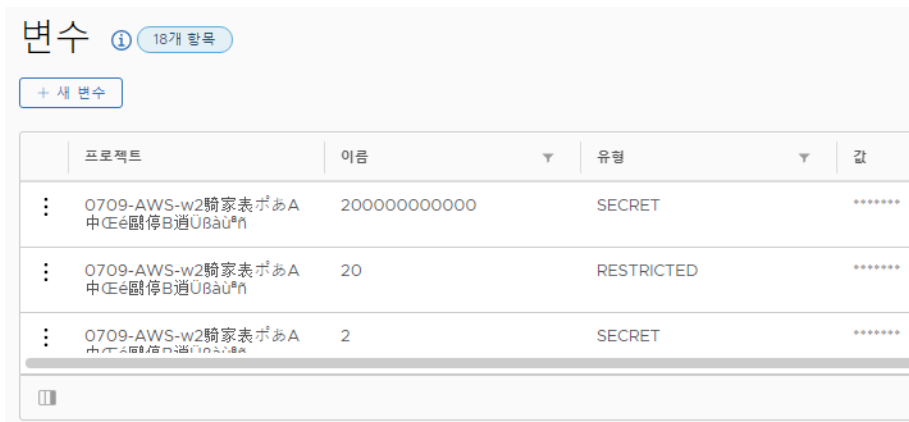
예를 들어 Git, Jenkins, Code Stream 빌드, Kubernetes 및 Jira 끝점이 필요할 수 있습니다.

- ### 3 파이프라인 작업에 값을 재사용할 수 있도록 변수를 생성합니다.

파이프라인에서 사용되는 리소스(예: 호스트 시스템)를 제한하려면 제한된 변수를 사용합니다. 다른 사용자가 명시적으로 승인할 때까지 파이프라인이 계속 실행되지 않도록 제한할 수 있습니다.

관리자는 비밀 변수와 제한된 변수를 생성할 수 있습니다. 사용자는 비밀 변수를 생성할 수 있습니다.

여러 파이프라인에서 원하는 횟수만큼 변수를 재사용할 수 있습니다. 예를 들어, 호스트 시스템을 정의하는 변수는 `HostIPAddress`일 수 있습니다. 파이프라인 작업에서 변수를 사용하려면 `$` {var.HostIPAddress}를 입력합니다.



- 4** 관리자는 비즈니스에 꼭 필요한 끝점 및 변수를 제한된 리소스로 표시할 수 있습니다.

관리자가 아닌 사용자가 제한된 리소스를 포함하는 파이프라인을 실행하려고 하면 제한된 리소스를 사용하는 작업에서 파이프라인이 중지됩니다. 그러면 관리자가 파이프라인을 재개해야 합니다.

## 5 기본 CICD, CI 또는 CD 파이프라인의 구축 전략을 계획합니다.

코드를 연속적으로 통합(CI)하고 연속적으로 배포(CD)하는 파이프라인을 생성하려면 먼저 구축 전략을 계획해야 합니다. 빌드 계획은 코드를 기본적으로 구축, 통합, 테스트 및 배포할 수 있도록 Code Stream에 필요한 사항을 결정하는 데 도움이 됩니다.

### Code Stream 기본 빌드를 생성하는 방법

#### 이 구축 전략의 결과

#### 스마트 파이프라인 템플릿 중 하나 사용

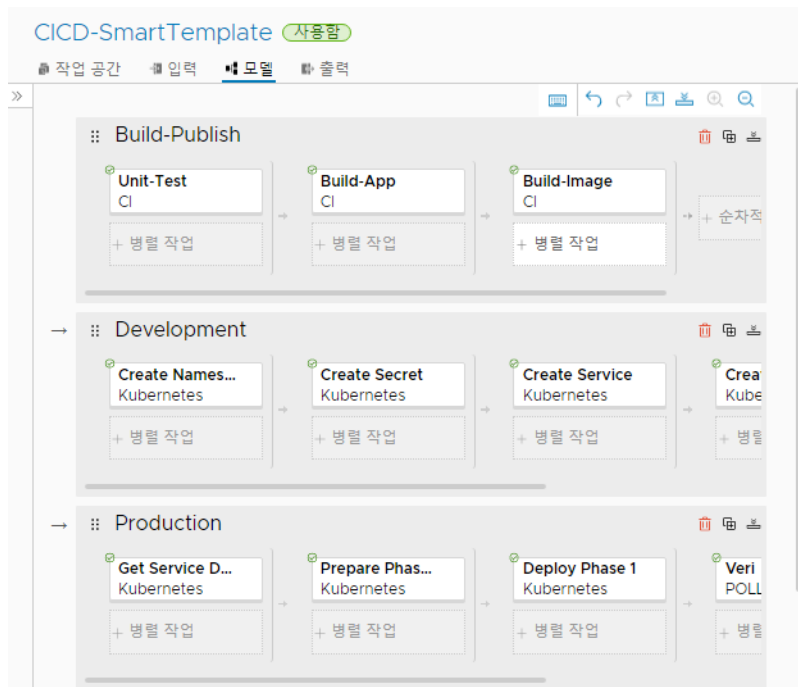
- 모든 단계와 작업이 자동으로 구축됩니다.
- 소스 저장소가 복제됩니다.
- 코드 구축 및 테스트가 수행됩니다.
- 배포를 위해 코드 컨테이너가 생성됩니다.
- 선택한 옵션에 기반하여 파이프라인 작업 단계를 채웁니다.

#### 수동으로 단계 및 작업 추가

직접 단계 및 작업을 추가하고, 여기에 채워질 정보를 입력합니다.

## 6 스마트 파이프라인 템플릿을 사용하거나 파이프라인에 스테이지 및 작업을 수동으로 추가하여 파이프라인을 생성합니다.

그런 다음 원하는 리소스를 제한된 리소스로 표시합니다. 필요한 위치에 승인을 추가합니다. 일반 변수, 제한된 변수 또는 비밀 변수를 적용합니다. 작업 간 바인딩을 추가합니다.



## 7 파이프라인을 검증 및 사용하도록 설정하고 실행합니다.

## 8 파이프라인 실행을 봅니다.

실행 468개 항목

+ 새 실행

검색

단계별 설정

작업

Demo-Jenkins-... #93 COMPLETED 스테이지: 4/4

실행한 사람 sestervil 날짜: 2020년 1월 19일 오후 4:17:14

Execution Completed.

★ 입력: 6d82d079a8b8921a911

☆ 출력: -

Demo-Jenkins-K... #13 COMPLETED 스테이지: 4/4

실행한 사람 sestervil 날짜: 2020년 1월 19일 오후 4:14:35

Execution Completed.

★ 입력: 6d82d079a8b8921a9

☆ 출력: -

Demo-Jenkins-... #92 COMPLETED 스테이지: 4/4

실행한 사람 sestervil 날짜: 2020년 1월 19일 오후 4:11:06

Execution Completed.

★ 입력: 8b3a29fdf\_\_

☆ 출력: -

Demo-CICD-Simp#48 FAILED 스테이지: 4/4

실행한 사람 sestervil 날짜: 2020년 1월 19일 오후 4:09:20

Production.Deploy Phase 1: Failed script execution: Failed to execu...

☆ 입력: -

☆ 출력: -

## 9 상태 및 KPI(주요 성능 지표)를 추적하려면 파이프라인 대시보드를 사용하고, 사용자 지정 대시보드를 생성합니다.

< 뒤로

CICD-SmartTemplate 작업

최근 성공한 실행

Demo-Jenkins-K8s #110 COMPLETED 14분 전

실행한 사람 pmartini

기간 25 초

★ 입력: 8b3a29fdf... ☆ 출력: -

실행 상태 수 지난 14일

합계: 107

COMPLETED FAILED WAITING

실행 통계 지난 14일

MITD 1분

MITF 3일

MTBD 47분

MITR 15분

최근 실행

실행 #/스테이지	Dev
#110	COMPLETED
#109	COMPLETED
#107	COMPLETED
#104	COMPLETED
#103	COMPLETED
#102	COMPLETED
#101	COMPLETED
#100	COMPLETED
#99	COMPLETED
#98	COMPLETED

COMPLETED FAILED RUNNING WAITING CANCELED ROLLBACK\_COMPLETED ROLLBACK\_FAILED

## 결과

선택한 프로젝트에서 사용할 수 있는 파이프라인을 생성했습니다.

파이프라인 YAML을 내보낸 후 가져와서 다른 프로젝트에 재사용할 수도 있습니다.

## 다음에 수행할 작업

환경에 적용할 수 있는 사용 사례에 대해 알아보니다. [장 5 Code Stream](#) 사용을 위한 자습서 항목을 참조하십시오.

## Code Stream에서 프로젝트를 추가하는 방법

프로젝트를 생성하고 여기에 관리자와 멤버를 추가합니다. 프로젝트 멤버는 파이프라인 생성 및 끝점 추가와 같은 기능을 사용할 수 있습니다. 개발 팀의 프로젝트를 생성, 삭제 또는 업데이트하려면 Code Stream 관리자여야 합니다.

파이프라인을 생성하려면 프로젝트가 있어야 합니다. 파이프라인을 생성할 때 모든 파이프라인 정보를 함께 그룹화하는 프로젝트를 선택합니다. 끝점 및 변수에 대한 정의도 기존 프로젝트에 따라 다릅니다.

### 사전 요구 사항

- Code Stream 관리자 역할이 있는지 확인합니다. [Code Stream에서의 역할 소개](#)의 내용을 참조하십시오.

Code Stream 관리자 역할이 없어도 Cloud Assembly의 관리자이면 Cloud Assembly UI를 사용하여 프로젝트를 생성, 업데이트 또는 삭제할 수 있습니다. [Cloud Assembly 개발 팀용 프로젝트를 추가하는 방법](#)을 참조하십시오.

- 프로젝트에 Active Directory 그룹을 추가하는 경우 조직에 대한 Active Directory 그룹을 구성했는지 확인합니다. vRealize Automation에서 프로젝트에 대해 Active Directory 그룹을 사용하도록 설정하는 방법을 참조하십시오. 그룹이 동기화되지 않은 경우 프로젝트에 추가하려고 할 때 사용할 수 없습니다.

### 절차

- 1 **프로젝트**를 선택하고 **새 프로젝트**를 클릭합니다.
- 2 프로젝트 이름을 입력합니다.
- 3 **생성**을 클릭합니다.
- 4 새로 생성된 프로젝트의 카드를 선택하고 **열기**를 클릭합니다.
- 5 **사용자** 탭을 클릭하고 사용자를 추가하고 역할을 할당합니다.
  - 프로젝트 관리자는 멤버를 추가할 수 있습니다.
  - 서비스 역할이 있는 프로젝트 멤버는 서비스를 사용할 수 있습니다.
  - 프로젝트 뷰어는 프로젝트를 볼 수 있지만 생성, 업데이트 또는 삭제할 수는 없습니다.

프로젝트 역할에 대한 자세한 내용은 [Code Stream에서 사용자 액세스 및 승인을 관리하는 방법](#) 항목을 참조하십시오.
- 6 **저장**을 클릭합니다.

## 다음에 수행할 작업

프로젝트를 사용하는 끝점 및 파이프라인을 추가합니다. [장 6 끝점에 Code Stream 연결](#) 및 [장 3 Code Stream에서 파이프라인 생성 및 사용](#) 항목을 참조하십시오.

파이프라인을 생성하면 모든 파이프라인 정보를 그룹화하는 프로젝트의 이름이 파이프라인 카드 및 파이프라인 실행 카드에 나타납니다.

## Code Stream에서 사용자 액세스 및 승인을 관리하는 방법

Code Stream는 사용자가 소프트웨어 애플리케이션을 릴리스하는 파이프라인을 사용하는 데 필요한 적절한 권한과 동의를 얻을 수 있도록 하기 위한 여러 가지 방법을 제공합니다.

팀의 각 멤버에게는 할당된 역할이 있으며 이러한 역할은 파이프라인, 끝점 및 대시보드에 대한 특정 사용 권한과 리소스를 제한된 리소스로 표시할 수 있는 권한을 제공합니다.

사용자 작업 및 승인을 통해 파이프라인 실행 시기 및 승인을 위해 중지해야 하는 시기를 제어할 수 있습니다. 역할은 파이프라인을 재개할 수 있는지 여부 그리고 제한된 끝점 또는 변수를 포함하는 파이프라인을 실행할 수 있는지 여부를 결정합니다.

비밀 변수를 사용하여 중요한 정보를 숨기고 암호화합니다. 제한된 변수는 숨기고 암호화되어야 하는 문자열, 암호 및 URL에 사용하고, 실행에서 사용을 제한하기 위해 사용합니다. 예를 들어 암호 또는 URL에 대해 비밀 변수를 사용합니다. 파이프라인의 모든 작업 유형에서 비밀 변수와 제한된 변수를 사용할 수 있습니다.

## Code Stream에서의 역할 소개

Code Stream에서의 역할에 따라 특정 작업을 수행하고 특정 영역에 액세스할 수 있습니다. 예를 들어 역할에 따라 파이프라인에 대한 생성, 업데이트 및 실행이 가능할 수 있습니다. 또는 파이프라인을 볼 수 있는 사용 권한만 가질 수도 있습니다.

제한된 항목을 제외한 모든 작업은 엔티티(제한된 변수와 끝점은 제외)에 대한 생성, 읽기, 업데이트 및 삭제 작업을 수행할 수 있는 권한이 이 역할에 있다는 의미입니다.

**표 2-1. Code Stream의 서비스 및 프로젝트 수준 액세스 권한**

Code Stream 역할					
액세스 수준	Code Stream 관리자	Code Stream 개발자	Code Stream Executor	Code Stream 뷰어	Code Stream 사용자
Code Stream 서비스 수준 액세스	모든 작업	제한된 항목을 제외한 모든 작업	실행 작업	읽기 전용	없음
프로젝트 수준 액세스: 프로젝트 관리자	모든 작업	모든 작업	모든 작업	모든 작업	모든 작업

표 2-1. Code Stream의 서비스 및 프로젝트 수준 액세스 권한 (계속)

Code Stream 역할					
액세스 수준	Code Stream 관리자	Code Stream 개발자	Code Stream Executor	Code Stream 뷰어	Code Stream 사용자
프로젝트 수준 액세스: 프로젝트 멤버	모든 작업	제한된 항목을 제외한 모든 작업	제한된 항목을 제외한 모든 작업	제한된 항목을 제외한 모든 작업	제한된 항목을 제외한 모든 작업
프로젝트 수준 액세스: 프로젝트 뷰어	모든 작업	제한된 항목을 제외한 모든 작업	실행 작업	읽기 전용	읽기 전용

프로젝트 관리자 역할이 있는 사용자는 자신이 프로젝트 관리자인 프로젝트에서 모든 작업을 수행할 수 있습니다.

프로젝트 관리자는 파이프라인, 변수, 끝점, 대시보드 및 트리거를 생성하고, 읽고, 업데이트하고, 삭제할 수 있으며, 해당 리소스가 프로젝트 관리자가 속한 프로젝트에 있는 경우 제한된 끝점 또는 변수를 포함하는 파이프라인을 시작할 수 있습니다.

서비스 뷰어 역할이 있는 사용자는 관리자가 사용할 수 있는 모든 정보를 볼 수 있습니다. 관리자가 이들을 프로젝트 관리자나 프로젝트 멤버로 만들지 않는 한, 아무 작업도 수행할 수 없습니다. 사용자가 프로젝트와 연관된 경우에는 역할과 관련된 권한이 있습니다. 프로젝트 뷰어는 관리자 또는 멤버 역할이 수행하는 방식으로 권한을 확장하지 않습니다. 이 역할은 모든 프로젝트에서 읽기 전용입니다.

프로젝트에서 읽기 권한이 있는 경우에도 제한된 리소스를 볼 수 있습니다.

- 끝점 카드에 잠금 아이콘을 표시하는 제한된 끝점을 보려면 **구성 > 끝점**을 클릭합니다.
- **유형** 열에 RESTRICTED 또는 SECRET을 표시하는 제한된 변수와 비밀 변수를 보려면 **구성 > 변수**를 클릭합니다.

표 2-2. Code Stream 서비스 역할 기능

UI 컨텍스트	기능	Code Stream 관리자 역할	Code Stream 개발자 역할	Code Stream Executor 역할	Code Stream 뷰어 역할	Code Stream 사용자 역할
파이프라인						
	파이프라인 보기	예	예	예	예	
	파이프라인 생성	예	예			
	파이프라인 실행	예	예	예		
	제한된 끝점 또는 변수를 포함하는 파이프라인 실행	예				
	파이프라인 업데이트	예	예			
	파이프라인 삭제	예	예			

표 2-2. Code Stream 서비스 역할 기능 (계속)

UI 컨텍스트	기능	Code Stream 관리자 역할	Code Stream 개발자 역할	Code Stream Executor 역할	Code Stream 뷰어 역할	Code Stream 사용자 역할
<b>파이프라인 실행</b>						
	파이프라인 실행 보기	예	예	예	예	
	파이프라인 실행 재개, 일시 중지 및 취소	예	예	예		
	제한된 리소스에 대한 승인을 위해 중지되는 파이프라인 재개	예				
<b>사용자 지정 통합</b>						
	사용자 지정 통합 생성	예	예			
	사용자 지정 통합 읽기	예	예	예	예	
	사용자 지정 통합 업데이트	예	예			
<b>끝점</b>						
	실행 보기	예	예	예	예	
	실행 생성	예	예			
	실행 업데이트	예	예			
	실행 삭제	예	예			
<b>리소스를 제한된 리소스로 표시</b>						
	끝점 또는 변수를 제한된 리소스로 표시	예				
<b>대시보드</b>						
	대시보드 보기	예	예	예	예	
	대시보드 생성	예	예			
	대시보드 업데이트	예	예			
	대시보드 삭제	예	예			



## Code Stream에서의 사용자 지정 역할 및 사용 권한

Cloud Assembly에서 사용자 지정 역할을 생성하여 파이프라인을 사용하는 사용자의 권한을 확장할 수 있습니다. Code Stream 파이프라인에 대한 사용자 지정 역할을 생성하는 경우 **파이프라인** 사용 권한을 하나 이상 선택합니다.

이 사용자 지정 역할을 할당할 사용자에게 필요한 **파이프라인** 사용 권한의 최대한 적게 선택합니다.

사용자가 프로젝트에 할당된 후 해당 프로젝트에서 역할이 지정되고, 해당 사용자에게 하나 이상의 **파이프라인** 사용 권한이 포함된 사용자 지정 역할이 할당되면 사용 권한이 허용하는 모든 작업을 수행할 수 있습니다. 예를 들어 제한된 변수를 생성하고, 제한된 파이프라인을 관리하고, 사용자 지정 통합을 생성하고 관리하는 등의 작업을 수행할 수 있습니다.

표 2-3. 사용자 지정 역할에 할당할 수 있는 파이프라인 사용 권한

파이프라인 사용 권한	Code Stream 관리자	Code Stream 개발자	Code Stream Executor	Code Stream 뷰어	Code Stream 사용자	프로젝트 관리자	프로젝트 멤버	프로젝트 뷰어
파이프라인 관리	예	예				예	예	
제한된 파이프라인 관리	예					예		
사용자 지정 통합 관리	예	예						
파이프라인 실행	예	예	예			예	예	
제한된 파이프라인 실행	예					예		
실행 관리	예					예		
읽기. 이 사용 권한은 표시되지 않습니다.	예	예	예	예		예	예	예

표 2-4. 사용자 지정 역할에서 파이프라인 사용 권한을 사용할 수 있는 방법

사용 권한	수행할 수 있는 작업
파이프라인 관리	<ul style="list-style-type: none"> <li>■ 파이프라인을 생성, 업데이트, 삭제 및 복제합니다.</li> <li>■ VMware Service Broker에 파이프라인을 릴리스/릴리스 취소합니다.</li> <li>■ 끝점을 생성, 업데이트 및 삭제합니다.</li> <li>■ 일반 변수와 비밀 변수를 생성, 업데이트 및 삭제합니다.</li> <li>■ Gerrit 수신기를 생성, 복제, 업데이트 및 삭제합니다.</li> <li>■ Gerrit 수신기를 연결하고 연결을 끊습니다.</li> <li>■ Gerrit 트리거를 생성, 복제, 업데이트 및 삭제합니다.</li> <li>■ Git Webhook을 생성, 업데이트 및 삭제합니다.</li> <li>■ Docker Webhook을 생성, 업데이트 및 삭제합니다.</li> <li>■ 스마트 파이프라인 템플릿을 사용하여 파이프라인을 생성합니다.</li> <li>■ YAML에서 파이프라인을 가져와서 YAML로 내보냅니다.</li> <li>■ 사용자 지정 대시보드를 생성, 업데이트 및 삭제합니다.</li> <li>■ 모든 사용자 지정 통합을 읽습니다.</li> <li>■ 모든 제한된 끝점 및 변수를 읽습니다. 해당 값을 볼 수는 없습니다.</li> </ul>
제한된 파이프라인 관리	<ul style="list-style-type: none"> <li>■ 끝점을 생성, 업데이트 및 삭제합니다.</li> <li>■ 끝점을 제한된 리소스로 표시하고 제한된 끝점을 업데이트 및 삭제합니다.</li> <li>■ 일반 변수와 비밀 변수를 생성, 업데이트 및 삭제합니다.</li> <li>■ 제한된 변수를 생성, 업데이트 및 삭제합니다.</li> <li>■ 파이프라인 관리를 통해 수행할 수 있는 모든 사용 권한.</li> </ul>
사용자 지정 통합 관리	<ul style="list-style-type: none"> <li>■ 사용자 지정 통합을 생성하고 업데이트합니다.</li> <li>■ 사용자 지정 통합에 버전을 지정하고 릴리스합니다.</li> <li>■ 사용자 지정 통합 버전을 삭제하고 폐기합니다.</li> <li>■ 사용자 지정 통합을 삭제합니다.</li> </ul>
파이프라인 실행	<ul style="list-style-type: none"> <li>■ 파이프라인을 실행합니다.</li> <li>■ 파이프라인 실행을 일시 중지, 재개 및 취소합니다.</li> <li>■ 파이프라인 실행을 다시 실행합니다.</li> <li>■ Gerrit 트리거 이벤트를 재개하고 다시 실행하고 수동으로 트리거합니다.</li> <li>■ 사용자 작업을 승인하고 사용자 작업의 일괄 승인을 수행할 수 있습니다.</li> </ul>

표 2-4. 사용자 지정 역할에서 파이프라인 사용 권한을 사용할 수 있는 방법 (계속)

사용 권한	수행할 수 있는 작업
제한된 파이프라인 실행	<ul style="list-style-type: none"> <li>■ 파이프라인을 실행합니다.</li> <li>■ 파이프라인 실행을 일시 중지, 재개, 취소 및 삭제합니다.</li> <li>■ 파이프라인 실행을 다시 실행합니다.</li> <li>■ 실행 중인 파이프라인 실행을 동기화합니다.</li> <li>■ 실행 중인 파이프라인 실행을 강제 삭제합니다.</li> <li>■ Gerrit 트리거 이벤트를 재개하고 다시 실행하고 삭제하고 수동으로 트리거합니다.</li> <li>■ 제한된 항목을 해결하고 파이프라인 실행을 계속합니다.</li> <li>■ 사용자 컨텍스트를 전환하고 사용자 작업 승인 후에 파이프라인 실행을 계속합니다.</li> <li>■ 파이프라인 실행을 통해 수행할 수 있는 모든 사용 권한.</li> </ul>
실행 관리	<ul style="list-style-type: none"> <li>■ 파이프라인을 실행합니다.</li> <li>■ 파이프라인 실행을 일시 중지, 재개, 취소 및 삭제합니다.</li> <li>■ 파이프라인 실행을 다시 실행합니다.</li> <li>■ Gerrit 트리거 이벤트를 재개하고 다시 실행하고 삭제하고 수동으로 트리거합니다.</li> <li>■ 파이프라인 실행을 통해 수행할 수 있는 모든 사용 권한.</li> </ul>

사용자 지정 역할에는 사용 권한의 조합이 포함될 수 있습니다. 이러한 사용 권한은 사용자가 제한된 리소스를 사용하거나 사용하지 않고 파이프라인을 관리하거나 실행할 수 있도록 하는 기능 그룹으로 구성됩니다. 이러한 사용 권한은 각 역할이 Code Stream에서 수행할 수 있는 모든 기능을 나타냅니다.

예를 들어 사용자 지정 역할을 생성하고 **제한된 파이프라인 관리**라는 사용 권한을 포함하는 경우 Code Stream 개발자 역할을 가진 사용자는 다음을 수행할 수 있습니다.

- 끝점을 생성, 업데이트 및 삭제합니다.
- 끝점을 제한된 리소스로 표시하고 제한된 끝점을 업데이트 및 삭제합니다.
- 일반 변수와 비밀 변수를 생성, 업데이트 및 삭제합니다.
- 제한된 변수를 생성, 업데이트 및 삭제합니다.

표 2-5. 사용자 지정 역할의 파이프라인 사용 권한 조합 예

사용자 지정 역할에 할당된 사용 권한 수	결합된 사용 권한의 예	이 조합을 사용하는 방법
사용 권한 1개	파이프라인 실행	
사용 권한 2개	파이프라인 관리 및 파이프라인 실행	
사용 권한 3개	파이프라인 관리, 파이프라인 실행 및 제한된 파이프라인 실행	이 조합은 Code Stream 개발자 역할에 적용될 수 있지만 사용자가 멤버인 프로젝트로 제한됩니다.

표 2-5. 사용자 지정 역할의 파이프라인 사용 권한 조합 예 (계속)

사용자 지정 역할 에 할당된 사용 권 한 수	결합된 사용 권한의 예	이 조합을 사용하는 방법
	파이프라인 관리, 사용자 지정 통합 관리 및 실행 관리	이 조합은 Code Stream 관리자 역할에 적용될 수 있지만 사용자가 멤버인 프로젝트로 제한됩니다.
	파이프라인 관리, 제한된 파이프라인 관리 및 사용자 지정 통합 관리	이 조합을 사용하는 경우 사용자가 모든 사용 권한을 가지며 Code Stream에서 모든 항목을 생성하고 삭제할 수 있습니다.

## 관리자 역할이 있는 경우

관리자는 사용자 지정 통합, 끝점, 변수, 트리거, 파이프라인 및 대시보드를 생성할 수 있습니다.

프로젝트를 사용하면 파이프라인이 인프라 리소스에 액세스할 수 있습니다. 관리자는 사용자가 파이프라인, 끝점 및 대시보드를 함께 그룹화할 수 있도록 프로젝트를 생성합니다. 그런 다음 사용자가 해당 파이프라인에서 프로젝트를 선택합니다. 각 프로젝트에는 역할을 할당받은 사용자와 관리자가 포함됩니다.

관리자 역할이 있다면 끝점과 변수를 제한된 리소스로 표시할 수 있으며 제한된 리소스를 사용하는 파이프라인을 실행할 수 있습니다. 관리자가 아닌 사용자가 제한된 끝점 또는 변수를 포함하는 파이프라인을 실행하는 경우, 제한된 변수가 사용되는 작업에서 파이프라인이 중지되고 관리자가 파이프라인을 재개해야 합니다.

또한 관리자는 파이프라인이 vRealize Automation Service Broker에 게시되도록 요청할 수도 있습니다.

## 개발자 역할이 있는 경우

제한된 끝점 또는 변수를 사용할 수 없다는 점을 제외하면 관리자가 수행하는 것과 동일한 파이프라인 작업을 수행할 수 있습니다.

제한된 끝점이나 변수를 사용하는 파이프라인을 실행하면 파이프라인은 제한된 리소스를 사용하는 작업까지만 실행됩니다. 그런 다음 파이프라인이 중지되며 Code Stream 관리자 또는 프로젝트 관리자가 파이프라인을 재개해야 합니다.

## 사용자 역할이 있는 경우

Code Stream에 액세스할 수 있지만 다른 역할이 제공하는 권한은 없습니다.

## 뷰어 역할이 있는 경우

파이프라인, 끝점, 파이프라인 실행, 대시보드, 사용자 지정 통합, 트리거와 같이 관리자가 볼 수 있는 리소스를 동일하게 볼 수 있지만 해당 리소스를 생성, 업데이트 또는 삭제할 수 없습니다. 작업을 수행하려면 뷰어 역할에 프로젝트 관리자 또는 프로젝트 멤버 역할도 지정되어야 합니다.

뷰어 역할이 있는 사용자는 프로젝트를 볼 수 있습니다. 제한된 끝점과 제한된 변수도 볼 수 있지만 관련 세부 정보는 볼 수 없습니다.

## Executor 역할이 있는 경우

파이프라인을 실행하고 사용자 작업에 대한 조치를 취할 수 있습니다. 파이프라인 실행을 재개, 일시 중지 및 취소할 수도 있습니다. 단, 파이프라인을 수정할 수는 없습니다.

## 역할 할당 및 업데이트 방법

관리자만 다른 사용자에게 역할 할당하고 업데이트할 수 있습니다.

- 1 활성 사용자와 해당 역할을 보려면 vRealize Automation의 오른쪽 상단에서 9개의 점을 클릭합니다.
- 2 ID 및 액세스 관리를 클릭합니다.



- 3 사용자 이름 및 역할을 표시하려면 **활성 사용자**를 클릭합니다.

ID 및 액세스 관리				
활성 사용자    엔터프라이즈 그룹				
역할 편집    액세스 제거				
<input type="checkbox"/>	이름	사용자 작업	조직 역할	서비스 역할
<input type="checkbox"/>	Local Admin	admin		

- 4 사용자에게 역할 추가하거나 역할을 변경하려면 사용자 이름 옆에 있는 확인란을 클릭하고 **역할 편집**을 클릭합니다.
- 5 사용자 역할을 추가하거나 변경할 때 서비스에 대한 액세스 권한을 추가할 수도 있습니다.
- 6 변경 내용을 저장하려면 **저장**을 클릭합니다.

## Code Stream에서 사용자 작업 및 승인이란?

사용자 작업 영역에는 승인이 필요한 파이프라인 실행이 표시됩니다. 필수 승인자는 파이프라인 실행을 승인하거나 거부할 수 있습니다.

파이프라인 생성 시 다음과 같은 경우에는 파이프라인에 승인 단계를 추가해야 할 수 있습니다.

- 팀 멤버가 코드를 검토해야 하는 경우

- 다른 사용자가 빌드 아티팩트를 확인해야 하는 경우
- 모든 테스트가 완료되었는지 확인해야 하는 경우
- 작업은 관리자가 제한됨으로 표시한 리소스를 사용하며 작업에는 승인이 필요합니다.
- 파이프라인은 소프트웨어를 운영 환경으로 릴리스합니다.

파이프라인 작업을 승인할지 여부를 결정하려면 필수 승인자에 권한 및 전문 지식이 있어야 합니다.

사용자 작업을 추가할 때 만료 시간 제한을 일, 시간 또는 분 단위로 설정할 수 있습니다. 예를 들어, 필수 사용자가 파이프라인을 30분 내에 승인해야 할 수 있습니다. 30분 내에 승인하지 않으면 파이프라인이 예상대로 실패합니다.

이메일 알림 보내기를 사용하도록 설정하면 [사용자 작업]은 전체 이메일 주소가 있는 승인자에게만 알림을 보내고 이메일 형식이 아닌 승인자 이름으로는 보내지 않습니다.

필수 사용자가 작업을 승인한 후에는:

- 보류 중인 파이프라인 실행을 계속할 수 있습니다.
- 파이프라인이 계속되면 동일한 사용자 작업의 승인을 위해 보류 중인 이전 요청이 취소됩니다.

## User Operations

GUIDED SETUP

Active Items
Inactive Items

✓ APPROVE
✗ REJECT

<input type="checkbox"/>	Index#	Execution	Summary	Requested By	Request Date	Approvers
<input type="checkbox"/>	> c07b12	Demo2-Jenkins-K8s#7	Testing	fritz	Nov 13, 2019, 11:32:31 AM	f...om
<input type="checkbox"/>	> a0a990	Demo2-Jenkins-K8s#6	Testing	fritz	Nov 11, 2019, 1:34:11 PM	k...om, f...m
<input checked="" type="checkbox"/>	▼ User Operation #8f1728	<div>Request Details</div> <div> <div>Execution</div> <div>Demo-Jenkins-K8s #5</div> </div> <div> <div>Summary</div> <div>Testing</div> </div> <div> <div>Approvers</div> <div>k...om, f...com</div> </div> <div> <div>Requested By</div> <div>fritz</div> </div> <div> <div>Requested On</div> <div>Nov 11, 2019, 1:22:21 PM</div> </div> <div> <div>Expires On</div> <div>Nov 14, 2019, 1:22:21 PM</div> </div>				

1
Items per page 20
1 - 7 of 7 items

[사용자 작업] 영역에서 승인하거나 거부할 항목이 활성 또는 비활성 항목으로 표시됩니다. 각 항목은 파이프라인의 사용자 작업에 매핑됩니다.

- **활성 항목**은 작업을 검토하고 승인 또는 거부해야 하는 승인자를 기다립니다. 승인자 목록에 있는 사용자인 경우 사용자 작업 행을 확장하고 **수락** 또는 **거부**를 클릭할 수 있습니다.
- **비활성 항목**은 승인되거나 거부된 항목입니다. 사용자가 사용자 작업을 거부했거나 작업에 대한 승인이 시간 초과되면 더 이상 승인할 수 없습니다.

[인덱스#]은 특정 승인을 검색하기 위해 필터로 사용할 수 있는 고유한 6자의 영숫자 문자열입니다.

파이프라인 승인은 **실행** 영역에도 나타납니다.

- 승인 대기 중인 파이프라인의 상태는 대기 중으로 표시됩니다.
- 다른 상태로는 대기열에 있음, 완료됨 및 실패함이 포함됩니다.
- 파이프라인이 대기 상태인 경우에는 필수 승인자가 파이프라인 작업을 승인해야 합니다.

# Code Stream에서 파이프라인 생성 및 사용

## 3

vRealize Automation Code Stream을 사용하여 구축, 테스트 및 배포 프로세스를 모델링할 수 있습니다. vRealize Automation Code Stream을 사용하여 릴리스 주기를 지원하는 인프라를 설정하고 소프트웨어 릴리스 작업을 모델링하는 파이프라인을 생성합니다. vRealize Automation Code Stream은 개발 코드부터 테스트를 거쳐 운영 인스턴스에 배포하는 방식으로 소프트웨어를 제공합니다.

각 파이프라인에는 단계와 작업이 포함되어 있습니다. 단계는 개발 단계를 나타내며 작업은 단계를 통해 소프트웨어 애플리케이션을 전달하는 필수 작업을 수행합니다.

## vRealize Automation Code Stream의 파이프라인 소개

파이프라인은 소프트웨어 릴리스 프로세스의 지속적인 통합 및 지속적 전달 모델입니다. 소스 코드에서 테스트를 거쳐 운영 환경으로 소프트웨어를 릴리스합니다. 여기에는 소프트웨어 릴리스 주기의 활동을 나타내는 작업이 포함된 일련의 단계가 포함됩니다. 소프트웨어 애플리케이션은 파이프라인을 통해 한 단계에서 다음 단계로 흐릅니다.

파이프라인의 작업이 데이터 소스, 저장소 또는 알림 시스템으로 연결될 수 있도록 끝점을 추가합니다.

## 파이프라인 생성

빈 캔버스로 시작하거나 스마트 파이프라인 템플릿을 사용하거나 YAML 코드를 가져오는 방법으로 파이프라인을 생성할 수 있습니다.

- 빈 캔버스를 사용합니다. 예를 보려면 작업을 수동으로 추가하기 전에 [Code Stream에서 CICD 네이티브 빌드 계획](#) 항목을 참조하십시오.
- 스마트 파이프라인 템플릿을 사용합니다. 예를 보려면 [장 4 Code Stream에서 기본적인 코드 구축, 통합 및 제공 계획](#) 항목을 참조하십시오.
- YAML 코드를 가져옵니다. **파이프라인 > 가져오기**를 클릭합니다. **가져오기** 대화 상자에서 YAML 파일을 선택하거나 YAML 코드를 입력하고 **가져오기**를 클릭합니다.

빈 캔버스를 사용하여 파이프라인을 생성하는 경우에는 단계, 작업 및 승인을 추가합니다. 파이프라인은 애플리케이션을 구축, 테스트, 배포 및 릴리스하는 프로세스를 자동화합니다. 각 단계의 작업은 각 단계를 통해 코드를 구축, 테스트 및 릴리스하는 작업을 실행합니다.



### 표 3-1. 파이프라인 단계 및 사용의 예

예제 단계	수행할 수 있는 작업의 예
개발	<p>개발 단계에서는 시스템을 프로비저닝하고 아티팩트를 검색하고 코드의 지속적 통합을 위해 Docker 호스트를 생성하는 구축 작업을 추가하는 등의 작업을 수행할 수 있습니다.</p> <p>예를 들면 다음과 같습니다.</p> <ul style="list-style-type: none"> <li>■ vRealize Automation Code Stream의 기본 구축 기능을 사용하여 코드를 전달하는 CI(지속적 통합) 빌드를 계획하고 생성하려면 <b>스마트 파이프라인</b> 템플릿을 사용하기 전에 <b>Code Stream</b>에서 <b>지속적 통합 네이티브 빌드 계획</b> 항목을 참조하십시오.</li> </ul>
테스트	<p>테스트 단계에서는 Jenkins 작업을 추가하여 소프트웨어 애플리케이션을 테스트하고 JUnit 및 JaCoCo 등과 같은 처리 후 테스트 툴을 포함할 수 있습니다.</p> <p>예를 들면 다음과 같습니다.</p> <ul style="list-style-type: none"> <li>■ vRealize Automation Code Stream을 Jenkins와 통합하고 파이프라인에서 Jenkins 작업을 실행하여 소스 코드를 구축하고 테스트합니다. <b>Code Stream</b>과 <b>Jenkins</b>를 통합하는 방법의 내용을 참조하십시오.</li> <li>■ vRealize Automation Code Stream의 기능을 확장하여 자체 구축, 테스트 및 배포 도구와 통합할 수 있는 사용자 지정 스크립트를 생성합니다. <b>고유한 구축, 테스트 및 배포 툴과 Code Stream</b>을 통합하는 방법의 내용을 참조하십시오.</li> <li>■ CI(지속적 통합) 파이프라인에 대한 처리 후 추세를 추적합니다. <b>Code Stream</b>에서 <b>사용자 지정 대시보드를 사용하여 파이프라인에 대한 주요 성능 지표를 추적하는 방법</b>의 내용을 참조하십시오.</li> </ul>
운영	<p>운영 단계에서는 인프라를 프로비저닝하고, 소프트웨어를 Kubernetes 클러스터에 배포하는 등의 작업을 수행하는 클라우드 템플릿을 Cloud Assembly에 통합할 수 있습니다.</p> <p>예를 들면 다음과 같습니다.</p> <ul style="list-style-type: none"> <li>■ 자체 블루-그린 배포 모델에서 소프트웨어 애플리케이션을 배포할 수 있는 개발 및 운영 단계의 예를 보려면 <b>Code Stream</b>에서 애플리케이션을 블루-그린 배포에 배포하는 방법 항목을 참조하십시오.</li> <li>■ 클라우드 템플릿을 파이프라인에 통합하려면 <b>Code Stream</b>의 <b>YAML 클라우드 템플릿에서 배포하는 애플리케이션의 릴리스를 자동화하는 방법</b> 항목을 참조하십시오. 스크립트를 실행하여 애플리케이션을 배포하는 배포 작업을 추가할 수도 있습니다.</li> <li>■ Kubernetes 클러스터에 소프트웨어 애플리케이션의 배포를 자동화하려면 <b>Code Stream</b>에서 <b>Kubernetes 클러스터에 애플리케이션의 릴리스를 자동화하는 방법</b> 항목을 참조하십시오.</li> <li>■ 코드를 파이프라인에 통합하고 빌드 이미지를 배포하려면 <b>Code Stream</b>에서 <b>GitHub 또는 GitLab 저장소의 코드를 파이프라인에 지속적으로 통합하는 방법</b> 항목을 참조하십시오.</li> </ul>

파이프라인을 YAML 파일로 내보낼 수 있습니다. **파이프라인**을 클릭하고 파이프라인 카드를 클릭한 다음 **작업 > 내보내기**를 클릭합니다.

## 파이프라인 승인

파이프라인의 특정 지점에서 다른 팀 멤버의 승인을 받을 수 있습니다.

- 파이프라인에 사용자 작업을 포함하여 파이프라인에 대한 승인을 요구하려면 **파이프라인을 실행하고 결과를 보는 방법** 항목을 참조하십시오. 이 작업은 검토가 필요한 사용자에게 이메일 알림을 보냅니다. 파이프라인이 계속 실행되려면 먼저 검토자가 승인을 승인하거나 거부해야 합니다. [사용자 작업]에 만료 시간 제한이 일, 시간 또는 분 단위로 설정되어 있으면, 필수 사용자는 작업이 만료되기 전에 파이프라인을 승인해야 합니다. 그렇지 않으면 파이프라인이 예상대로 실패합니다.

- 파이프라인의 단계에서 작업 또는 단계가 실패하면 vRealize Automation Code Stream에서 Jira 티켓이 생성되도록 설정할 수 있습니다. 파이프라인 작업이 실패한 경우 Code Stream에서 Jira 티켓을 생성하는 방법의 내용을 참조하십시오.

## 파이프라인 트리거

파이프라인은 개발자가 코드를 저장소에 체크인하거나 코드를 검토하거나 새 빌드 아티팩트 또는 업데이트된 빌드 아티팩트를 식별할 때 트리거할 수 있습니다.

- vRealize Automation Code Stream을 Git 수명 주기와 통합하고 개발자가 코드를 업데이트할 때 파이프라인을 트리거하려면 Git 트리거를 사용합니다. Code Stream에서 Git 트리거를 사용하여 파이프라인을 실행하는 방법의 내용을 참조하십시오.
- vRealize Automation Code Stream을 Gerrit 코드 검토 수명 주기와 통합하고 코드 검토 시 파이프라인을 트리거하려면 Gerrit 트리거를 사용합니다. Code Stream에서 Gerrit 트리거를 사용하여 파이프라인을 실행하는 방법의 내용을 참조하십시오.
- Docker 빌드 아티팩트가 생성되거나 업데이트될 때 파이프라인을 트리거하려면 Docker 트리거를 사용합니다. Code Stream에서 Docker 트리거를 사용하여 지속적 전달 파이프라인을 실행하는 방법의 내용을 참조하십시오.

vRealize Automation Code Stream이 지원하는 트리거에 대한 자세한 내용은 [장 7 Code Stream에서 파이프라인 트리거](#) 항목을 참조하십시오.

본 장은 다음 항목을 포함합니다.

- 파이프라인을 실행하고 결과를 보는 방법
- Code Stream에서 사용할 수 있는 작업 유형
- Code Stream 파이프라인에서 변수 바인딩을 사용하는 방법
- Code Stream에서 파이프라인의 실행 또는 중지를 위해 조건 작업에서 변수 바인딩을 사용하는 방법
- Code Stream에서 파이프라인 작업을 바인딩할 때 사용할 수 있는 변수 및 표현식
- Code Stream에서 내 파이프라인에 대한 알림을 보내는 방법
- 파이프라인 작업이 실패한 경우 Code Stream에서 Jira 티켓을 생성하는 방법
- Code Stream에서 배포를 롤백하는 방법

## 파이프라인을 실행하고 결과를 보는 방법

파이프라인 카드에서, 파이프라인 편집 모드에서 그리고 파이프라인 실행에서 파이프라인을 실행할 수 있습니다. 특정 이벤트가 발생할 때 사용 가능한 트리거를 사용하여 Code Stream이 파이프라인을 실행하도록 할 수도 있습니다.

파이프라인의 모든 단계와 작업이 유효하면 해당 파이프라인을 릴리스, 실행 또는 트리거할 수 있습니다.

Code Stream을 사용하여 파이프라인을 실행하거나 트리거하려면 파이프라인 카드에서 또는 파이프라인에 있는 동안 파이프라인을 사용하도록 설정하고 실행할 수 있습니다. 그런 다음 파이프라인 실행을 보고 파이프라인이 코드를 구축, 테스트 및 배포했는지 확인할 수 있습니다.

파이프라인 실행이 진행 중인 경우 관리자이거나 관리자가 아닌 사용자는 실행을 삭제할 수 있습니다.

- 관리자: 실행 중인 파이프라인 실행을 삭제하려면 **실행**을 클릭합니다. 삭제할 실행에서 **작업 > 삭제**를 클릭합니다.
- 관리자가 아닌 사용자: 실행 중인 파이프라인 실행을 삭제하려면 **실행**을 클릭하고 **Alt Shift d**를 클릭합니다.

파이프라인 실행이 진행 중이고 중단된 것으로 나타나는 경우 관리자는 [실행] 페이지 또는 [실행 세부 정보] 페이지에서 실행을 새로 고칠 수 있습니다.

- [실행] 페이지: **실행**을 클릭합니다. 새로 고칠 실행에서 **작업 > 동기화**를 클릭합니다.
- [실행 세부 정보] 페이지: **실행**을 클릭하고 실행 세부 정보 링크를 클릭한 다음 **작업 > 동기화**를 클릭합니다.

특정 이벤트가 발생할 때 파이프라인을 실행하려면 트리거를 사용합니다.

- Git 트리거는 개발자가 코드를 업데이트할 때 파이프라인을 실행할 수 있습니다.
- Gerrit 트리거는 코드 검토가 발생할 때 파이프라인을 실행할 수 있습니다.
- Docker 트리거는 Docker 레지스트리에 아티팩트가 생성될 때 파이프라인을 실행할 수 있습니다.
- curl 명령이나 wget 명령은 Jenkins 빌드가 완료된 후 Jenkins에서 파이프라인을 실행하도록 할 수 있습니다.

트리거 사용에 대한 자세한 내용은 [장 7 Code Stream에서 파이프라인 트리거](#) 항목을 참조하십시오.

다음 절차에서는 파이프라인 카드에서 파이프라인을 실행하고, 실행을 보고, 실행 세부 정보를 참조하고, 작업을 사용하는 방법을 보여줍니다. 또한 vRealize Automation Service Broker에 추가할 수 있도록 파이프라인을 해제하는 방법도 보여줍니다.

### 사전 요구 사항

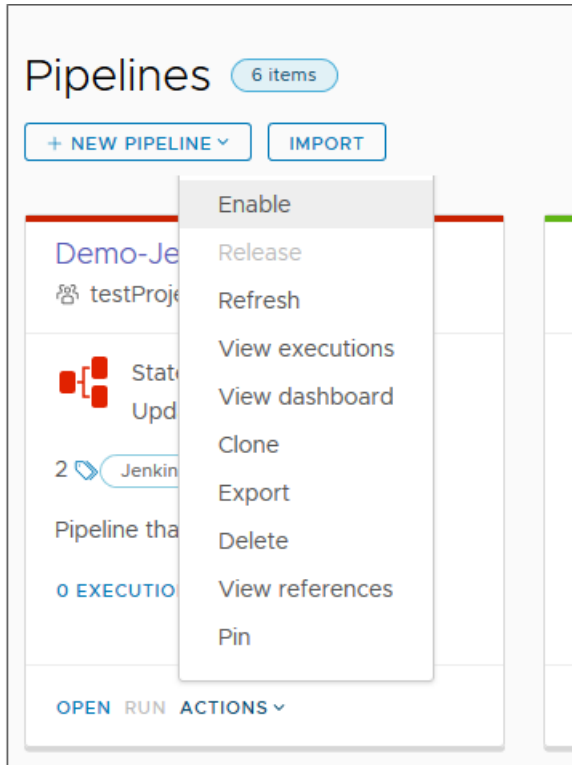
- 파이프라인을 하나 이상 생성했는지 확인합니다. [장 5 Code Stream 사용](#)을 위한 자습서 항목의 예시를 참조하십시오.

## 절차

### 1 파이프라인을 사용하도록 설정합니다.

파이프라인을 실행하거나 릴리스하려면 먼저 파이프라인을 사용하도록 설정해야 합니다.

- a 파이프라인을 클릭합니다.
- b 파이프라인 카드에서 **작업 > 사용**을 클릭합니다.



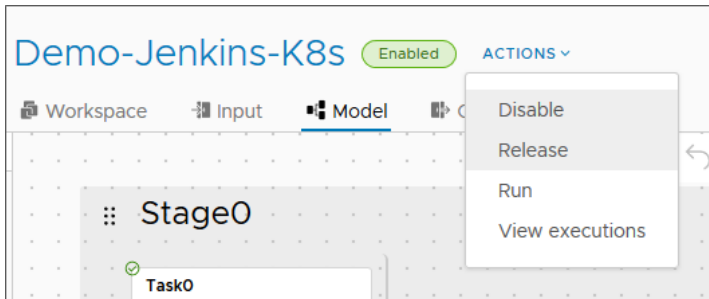
파이프라인에 있는 동안 파이프라인을 사용하도록 설정할 수도 있습니다. 파이프라인을 사용하도록 이미 설정된 경우 **실행**이 활성화 상태이고 **작업** 메뉴에 **사용 안 함**이 표시됩니다.

## 2 (선택 사항) 파이프라인을 릴리스합니다.

파이프라인을 vRealize Automation Service Broker에서 카탈로그 항목으로 사용할 수 있도록 하려면 Code Stream에서 파이프라인을 릴리스해야 합니다.

- a 파이프라인을 클릭합니다.
- b 파이프라인 카드에서 **작업 > 릴리스**를 클릭합니다.

파이프라인에 있는 동안 파이프라인을 릴리스할 수도 있습니다.



파이프라인을 릴리스한 후 Service Broker를 열어 파이프라인을 카탈로그 항목으로 추가하고 실행합니다. [Service Broker 카탈로그에 Code Stream 파이프라인 추가](#)를 참조하십시오.

**참고** 파이프라인을 120분 이상 실행해야 하는 경우 대략적인 실행 시간을 요청 시간 초과 값으로 제공합니다. 프로젝트에 대한 요청 시간 초과를 설정하거나 검토하려면 Service Broker를 관리자 로 열고 **인프라 > 프로젝트**를 선택합니다. 프로젝트를 클릭한 다음 **프로비저닝** 을 클릭합니다.

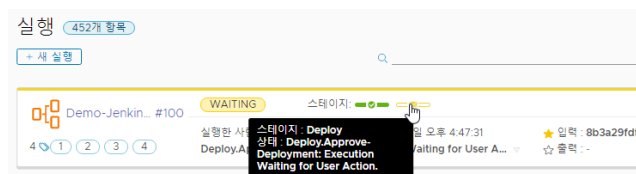
요청 시간 초과 값이 설정되지 않은 경우 120분 이상 실행해야 하는 실행은 콜백 시간 초과 요청 오류와 함께 실패한 것으로 나타납니다. 그러나 파이프라인 실행은 영향을 받지 않습니다.

- 3 파이프라인 카드에서 **실행**을 클릭합니다.
- 4 실행 중인 파이프라인을 보려면 **실행**을 클릭합니다.

파이프라인은 각 단계를 차례로 실행하고 파이프라인 실행은 각 단계에 대한 상태 아이콘을 표시합니다. 파이프라인에 사용자 작업이 포함된 경우 사용자가 작업을 승인해야 파이프라인이 계속 실행될 수 있습니다. 사용자 작업이 사용되면 파이프라인 실행이 중지되고 필요한 사용자가 작업을 승인할 때까지 기다립니다.

예를 들어 사용자 작업을 사용하여 운영 환경에 대한 코드 배포를 승인할 수 있습니다.

[사용자 작업]에 만료 시간 제한이 일, 시간 또는 분 단위로 설정되어 있으면, 필수 사용자는 작업이 만료되기 전에 파이프라인을 승인해야 합니다. 그렇지 않으면 파이프라인이 예상대로 실패합니다.



- 5 사용자 승인을 기다리는 파이프라인 단계를 보려면 해당 단계의 상태 아이콘을 클릭합니다.



- 6 작업에 대한 세부 정보를 보려면 작업을 클릭합니다.

해당하는 사용자가 작업을 승인한 후에는 적절한 역할을 가진 사용자가 파이프라인을 재개해야 합니다. 필수 역할에 대해서는 [Code Stream에서 사용자 액세스 및 승인을 관리하는 방법](#)의 내용을 참조하십시오.

실행이 실패하면 실패의 원인을 심사하고 해결해야 합니다. 그런 다음 실행으로 이동하고 **작업 > 다시 실행**을 클릭합니다.

기본 파이프라인 실행 및 중첩된 실행을 재개할 수 있습니다.



- 7 파이프라인 실행에서 **작업**을 클릭하여 파이프라인을 보고, **일시 중지**, **취소** 등의 작업을 선택할 수 있습니다. 파이프라인 실행이 진행 중인 경우 관리자는 파이프라인 실행을 삭제하거나 동기화할 수 있습니다. 관리자가 아닌 사용자는 실행 중인 파이프라인을 삭제할 수 있습니다.

- 8 실행 사이에서 쉽게 이동하고 작업에 대한 세부 정보를 보려면 **실행**을 클릭하고 파이프라인 실행을 클릭합니다. 그런 다음 맨 위에 있는 탭을 클릭하고 파이프라인 실행을 선택합니다.



## 결과

축하합니다! 파이프라인을 실행하고 파이프라인 실행을 검사한 후 파이프라인을 계속 실행하려면 승인이 필요한 사용자 작업을 검토했습니다. 파이프라인 실행에서 **작업** 메뉴를 사용하여 파이프라인 모델로 돌아가서 필요한 변경 작업을 수행할 수도 있습니다.

## 다음에 수행할 작업

Code Stream을 사용하여 소프트웨어 릴리스 주기를 자동화하는 데 대한 자세한 내용은 [장 5 Code Stream 사용을 위한 자습서](#) 항목을 참조하십시오.

# Code Stream에서 사용할 수 있는 작업 유형

파이프라인을 구성할 때는 필요한 작업을 위해 파이프라인이 실행하는 특정 유형의 작업을 추가합니다. 각 작업 유형은 다른 애플리케이션과 통합되며 애플리케이션을 구축, 테스트 및 제공하면서 파이프라인을 사용하도록 설정합니다.

배포를 위해 저장소에서 아티팩트를 끌어오거나 원격 스크립트를 실행해야 하는 경우 또는 사용자 작업에 대해 팀 멤버의 승인이 필요한 경우, 파이프라인을 실행할 수 있도록 Code Stream에 사용자를 위한 작업 유형이 준비되어 있습니다!

Code Stream은 다양한 유형의 작업에 대한 파이프라인 실행 취소를 지원합니다. 파이프라인 실행에 대해 **취소**를 클릭하면 작업, 단계 또는 전체 파이프라인이 취소 중 상태가 되고 파이프라인 실행이 취소됩니다.

Code Stream을 사용하면 다음 작업을 사용할 때 작업, 단계 또는 전체 파이프라인에서 파이프라인 실행을 취소할 수 있습니다.

- Jenkins
- SSH
- PowerShell
- 사용자 작업
- 파이프라인
- 클라우드 템플릿
- vRO

## ■ 폴링

Code Stream은 CI, 사용자 지정 통합 또는 Kubernetes와 같은 작업에 대한 취소 동작을 타사 시스템에 전파하지 않습니다. Code Stream은 작업이 취소된 것으로 표시하고 작업이 완료될 때까지 기다리지 않고 상태 가져오기를 즉시 중지합니다. 타사 시스템에서는 작업이 완료되거나 실패할 수 있지만 **취소**를 클릭하는 즉시 Code Stream에서 실행이 중지됩니다.

파이프라인에서 작업을 사용하기 전에 해당하는 끝점을 사용할 수 있는지 확인하십시오.

**표 3-2. 승인 획득 또는 결정 시점 설정**

작업 유형	수행 작업	예제 및 세부 정보
사용자 작업	[사용자 작업]을 통해 파이프라인을 실행하고 승인을 위해 중지해야 하는 시기를 제어하는 필수 승인을 사용할 수 있습니다.	파이프라인을 실행하고 결과를 보는 방법 및 Code Stream에서 사용자 액세스 및 승인을 관리하는 방법 항목을 참조하십시오.
조건	조건 표현식에 따라 파이프라인을 계속 실행할지 아니면 중지할지 결정하는 결정 시점을 추가합니다. 조건이 true이면 파이프라인이 후속 작업을 실행합니다. false 이면 파이프라인이 중지됩니다.	Code Stream에서 파이프라인의 실행 또는 중지를 위해 조건 작업에서 변수 바인딩을 사용하는 방법 항목을 참조하십시오.



표 3-3. 지속적 통합 및 배포 자동화

작업 유형	수행 작업	예제 및 세부 정보
클라우드 템플릿	GitHub에서 자동화 클라우드 템플릿을 배포하고, 애플리케이션을 프로비저닝하고, 배포를 위한 해당 클라우드 템플릿의 CICD(지속적 통합 및 전달)를 자동화합니다.	<p>Code Stream의 YAML 클라우드 템플릿에서 배포하는 애플리케이션의 릴리스를 자동화하는 방법 항목을 참조하십시오.</p> <p>클라우드 템플릿 매개 변수는 <b>생성</b> 또는 <b>업데이트</b>를 먼저 선택한 다음 <b>클라우드 템플릿 및 버전</b>을 선택하면 나타납니다. 변수 바인딩을 수용하는 다음과 같은 요소를 클라우드 템플릿 작업의 입력 텍스트 영역에 추가할 수 있습니다.</p> <ul style="list-style-type: none"> <li>■ 정수</li> <li>■ 열거형 문자열</li> <li>■ 부울</li> <li>■ 어레이 변수</li> </ul> <p>입력에서 변수 바인딩을 사용하는 경우 이러한 예외 사항에 유의해야 합니다. 열거형의 경우 고정 집합에서 열거형 값을 선택해야 합니다. 부울 값의 경우 입력 텍스트 영역에 값을 입력해야 합니다.</p> <p>클라우드 템플릿 매개 변수는 Cloud Assembly의 클라우드 템플릿에 입력 변수가 포함된 경우 클라우드 템플릿 작업에 나타납니다. 예를 들어 클라우드 템플릿에 Integer 입력 유형이 있는 경우 정수를 직접 입력하거나 변수 바인딩을 사용하여 변수로 입력할 수 있습니다.</p>
CI	<p>CI 작업을 통해 레지스트리 끝점에서 Docker 구축 이미지를 가져와 Kubernetes 클러스터에 배포함으로써 코드를 파이프라인에 지속적으로 통합할 수 있습니다.</p> <p>CI 작업은 로그 100줄을 출력으로 표시하고 로그를 다운로드할 경우 500줄을 표시합니다.</p> <p>CI 작업에는 사용 후 삭제 포트 32768~61000이 필요합니다.</p>	스마트 파이프라인 템플릿을 사용하기 전에 Code Stream에서 CICD 기본 구축 계획 항목을 참조하십시오.
사용자 지정	사용자 지정 작업은 Code Stream을 사용자 소유의 구축, 테스트 및 배포 도구와 통합합니다.	고유한 구축, 테스트 및 배포 톨과 Code Stream을 통합하는 방법 항목을 참조하십시오.
Kubernetes	AWS의 Kubernetes 클러스터의 소프트웨어 애플리케이션 배포를 자동화합니다.	Code Stream에서 Kubernetes 클러스터에 애플리케이션의 릴리스를 자동화하는 방법 항목을 참조하십시오.
파이프라인	<p>기본 파이프라인에 파이프라인을 중첩합니다. 중첩된 파이프라인은 기본 파이프라인에서 작업으로 작동합니다.</p> <p>기본 파이프라인의 [작업] 탭에서 해당 링크를 클릭하여 중첩된 파이프라인으로 쉽게 이동할 수 있습니다. 중첩된 파이프라인이 새 브라우저 탭에서 열립니다.</p>	실행에서 중첩된 파이프라인을 찾으려면 검색 영역에 <b>nested</b> 를 입력합니다.

표 3-4. 개발, 테스트 및 배포 애플리케이션 통합

작업 유형...	수행하는 작업...	예 및 세부 정보...
<b>Bamboo</b>	Bamboo CI(지속적 통합) 서버와 상호 작용하여 배포 준비를 위해 소프트웨어를 지속적으로 구축, 테스트 및 통합하고 개발자가 변경 내용을 커밋하면 코드 구축을 트리거합니다. 한 작업에서 출력한 매개 변수를 구축 및 배포를 위해 다른 작업에서 사용할 수 있도록 Bamboo 구축에서 생성하는 아티팩트 위치를 노출합니다.	Bamboo 서버 끝점에 연결하고 파이프라인에서 Bamboo 구축 계획을 시작합니다.
<b>Jenkins</b>	Jenkins 작업을 트리거하여 소스 코드를 구축 및 테스트하고, 테스트 사례를 실행하며, 사용자 지정 스크립트를 사용할 수 있습니다.	<a href="#">Code Stream과 Jenkins를 통합하는 방법</a> 항목을 참조하십시오.
<b>TFS</b>	코드를 구축하고 테스트하는 구성된 작업을 포함하여 구축 프로젝트를 관리하고 호출하도록 파이프라인을 Team Foundation Server에 연결할 수 있습니다.	Code Stream에서 지원하는 Team Foundation Server 버전은 <a href="#">Code Stream의 끝점 소개</a> 항목을 참조하십시오.
<b>vRO</b>	vRealize Orchestrator에서 미리 정의된 워크플로 또는 사용자 지정 워크플로를 실행하여 Code Stream의 기능을 확장합니다.  Code Stream은 vRealize Orchestrator에 대해 기본 인증 및 토큰 기반 인증을 지원합니다. Code Stream은 API 토큰을 사용하여 vRealize Orchestrator 클러스터를 인증하고 검증합니다. 토큰 기반 인증을 통해, Code Stream은 클라우드 확장성 프록시를 사용하는 vRealize Orchestrator 끝점을 지원합니다. 따라서 Code Stream에서 사용자는 클라우드 확장성 프록시를 사용하는 vRealize Orchestrator 끝점을 사용하여 워크플로를 트리거할 수 있습니다.	<a href="#">Code Stream과 vRealize Orchestrator를 통합하는 방법</a> 항목을 참조하십시오.

표 3-5. API를 통해 다른 애플리케이션 통합

작업 유형...	수행하는 작업...	예 및 세부 정보...
<b>REST</b>	Code Stream을 REST API를 사용하는 다른 애플리케이션과 통합함으로써 서로 상호 작용하는 소프트웨어 애플리케이션을 지속적으로 개발하고 제공할 수 있도록 합니다.	<a href="#">Code Stream을 다른 애플리케이션과 통합하기 위해 REST API를 사용하는 방법</a> 항목을 참조하십시오.
<b>폴링</b>	REST API를 호출한 다음 파이프라인 작업이 종료 조건을 충족하고 완료될 때까지 REST API를 폴링합니다.  Code Stream 관리자는 폴링 수를 최대 10000으로 설정할 수 있습니다. 폴링 간격은 60초보다 크거나 같아야 합니다.  <b>실패 시 계속</b> 확인란에 표시하는 경우 개수 또는 간격이 이러한 값을 초과하면 폴링 작업이 계속 실행됩니다.  POLL Iteration Count: 파이프라인 실행에 표시되어 POLL 작업이 URL에서 응답을 요청한 횟수를 나타냅니다. 예를 들어 POLL 입력이 65이고 POLL 요청이 실행된 실제 횟수가 4인 경우 파이프라인 실행 출력의 반복수에 4(총 65회)가 표시됩니다.	<a href="#">Code Stream을 다른 애플리케이션과 통합하기 위해 REST API를 사용하는 방법</a> 항목을 참조하십시오.

표 3-6. 원격 및 사용자 정의 스크립트 실행

작업 유형	수행 작업	예제 및 세부 정보
PowerShell	<p>PowerShell 작업을 통해 Code Stream은 원격 호스트에서 스크립트 명령을 실행할 수 있습니다. 예를 들어 스크립트는 테스트 작업을 자동화하고 관리 명령 유형을 실행할 수 있습니다.</p> <p>스크립트는 원격 또는 사용자 정의일 수 있습니다. HTTP 또는 HTTPS를 통해 연결할 수 있으며 TLS를 사용할 수 있습니다.</p> <p>Windows 호스트에는 winrm 서비스가 구성되어 있어야 하며 winrm에는 MaxShellsPerUser 및 MaxMemoryPerShellMB가 구성되어 있어야 합니다.</p> <p>PowerShell 작업을 실행하려면 원격 Windows 호스트에 대한 활성 세션이 있어야 합니다.</p> <p><b>PowerShell 명령줄 길이</b></p> <p>base64 PowerShell 명령을 입력하는 경우 전체 명령 길이를 계산해야 합니다.</p> <p>Code Stream 파이프라인은 base64 PowerShell 명령을 다른 명령으로 인코딩하고 래핑하므로 명령의 전체 길이가 늘어납니다.</p> <p>PowerShell winrm 명령에 허용되는 최대 길이는 8192 바이트입니다. 인코딩 및 래핑되었을 때 PowerShell 작업에 대한 명령 길이 제한이 더 낮습니다. 따라서 PowerShell 명령을 입력하기 전에 명령 길이를 계산해야 합니다.</p> <p>Code Stream PowerShell 작업의 명령 길이 제한은 원래 명령의 base64 인코딩 길이에 따라 달라집니다. 명령 길이는 다음과 같이 계산됩니다.</p> $3 * (\text{length of original command} / 4) - (\text{numberOfPaddingCharacters}) + 77 (\text{Length of Write-output command})$ <p>Code Stream의 명령 길이는 최대 제한인 8192보다 작아야 합니다.</p>	<p>MaxShellsPerUser 및 MaxMemoryPerShellMB를 구성하는 경우:</p> <ul style="list-style-type: none"> <li>MaxShellsPerUser에 대해 허용 가능한 값은 500이며, 이는 50개의 동시 파이프라인 및 각 파이프라인에 대해 5개의 PowerShell 작업을 나타냅니다. 값을 설정하려면 다음을 실행합니다. winrm set winrm/config/winrs '@{MaxShellsPerUser="500"}'</li> <li>MaxMemoryPerShellMB에 허용 가능한 메모리 값은 2048입니다. 값을 설정하려면 다음을 실행합니다. winrm set winrm/config/winrs '@{MaxMemoryPerShellMB="2048"}'</li> </ul> <p>이 스크립트는 출력을 응답 파일로 작성하여 다른 파이프라인이 사용할 수 있도록 합니다.</p>
SSH	<p>SSH 작업을 통해 Bash shell script 작업으로 원격 호스트에서 스크립트 명령을 실행할 수 있습니다. 예를 들어 스크립트는 테스트 작업을 자동화하고 관리 명령 유형을 실행할 수 있습니다.</p> <p>스크립트는 원격 또는 사용자 정의일 수 있습니다. HTTP 또는 HTTPS를 통해 연결할 수 있으며 개인 키 또는 암호가 필요합니다.</p> <p>SSH 서비스는 Linux 호스트에서 구성되어야 하며 SSHD의 MaxSessions 구성은 50으로 설정되어야 합니다.</p>	<p>스크립트는 원격 또는 사용자 정의일 수 있습니다. 예를 들어 스크립트는 다음과 유사할 수 있습니다.</p> <pre>message="Hello World" echo \$message</pre> <p>이 스크립트는 출력을 응답 파일로 작성하여 다른 파이프라인이 사용할 수 있도록 합니다.</p>

표 3-6. 원격 및 사용자 정의 스크립트 실행 (계속)

작업 유형	수행 작업	예제 및 세부 정보
	<p>여러 SSH 작업을 동시에 실행하는 경우 SSH 호스트에서 <code>MaxSessions</code> 및 <code>MaxOpenSessions</code>를 늘립니다.</p> <p><code>MaxSessions</code> 및 <code>MaxOpenSessions</code> 구성 설정을 수정해야 하는 경우 vRealize Automation 인스턴스를 SSH 호스트로 사용하지 마십시오.</p>	

## Code Stream 파이프라인에서 변수 바인딩을 사용하는 방법

파이프라인 작업을 바인딩한다는 것은 파이프라인이 실행될 때 작업에 대한 종속성을 생성한다는 의미입니다. 여러 가지 방법으로 파이프라인 작업에 대한 바인딩을 생성할 수 있습니다. 작업을 다른 작업에 바인딩하거나 변수 및 표현식에 바인딩하거나 조건에 바인딩할 수 있습니다.

### 클라우드 템플릿 작업의 클라우드 템플릿 변수에 달러 바인딩을 적용하는 방법

Code Stream 파이프라인 클라우드 템플릿 작업에서 클라우드 템플릿 변수에 달러 바인딩을 적용할 수 있습니다. Code Stream에서 변수를 수정하는 방법은 클라우드 템플릿의 변수 속성 코딩에 따라 다릅니다.

클라우드 템플릿 작업에서 달러 바인딩을 사용해야 하지만 클라우드 템플릿 작업에서 사용 중인 현재 버전의 클라우드 템플릿이 이를 허용하지 않는 경우 Cloud Assembly에서 클라우드 템플릿을 수정하고 새 버전을 배포합니다. 그런 다음, 클라우드 템플릿 작업에서 새 클라우드 템플릿 버전을 사용하고 필요한 위치에 달러 바인딩을 추가합니다.

Cloud Assembly 클라우드 템플릿이 제공하는 속성 유형에 달러 바인딩을 적용하려면 올바른 사용 권한이 있어야 합니다.

- Cloud Assembly에서 클라우드 템플릿 배포를 생성한 사용자와 동일한 역할이 있어야 합니다.
- 파이프라인을 모델링하는 사용자와 파이프라인을 실행하는 사용자는 서로 다른 두 사용자이며 역할이 다를 수 있습니다.
- 개발자에게 Code Stream Executor 역할이 있고 개발자가 파이프라인을 모델링하는 경우 개발자는 클라우드 템플릿을 배포한 사용자와 동일한 Cloud Assembly 역할을 가지고 있어야 합니다. 예를 들어 필수 역할은 Cloud Assembly 관리자일 수 있습니다.
- 파이프라인을 모델링하는 사용자만 권한이 있기 때문에 파이프라인을 생성하고 배포를 생성할 수 있습니다.

클라우드 템플릿 작업에서 API 토큰을 사용하려면:

- 파이프라인을 모델링하는 사용자는 Code Stream Executor 역할을 가진 다른 사용자에게 API 토큰을 제공할 수 있습니다. 그런 다음 Executor가 파이프라인을 실행하면 API 토큰과 API 토큰이 생성하는 자격 증명을 사용합니다.
- 사용자가 클라우드 템플릿 작업에 API 토큰을 입력하면 파이프라인에 필요한 자격 증명이 생성됩니다.

- API 토큰 값을 암호화하려면 **변수 생성**을 클릭합니다.
- API 토큰에 대한 변수를 생성하지 않고 이를 클라우드 템플릿 작업에서 사용하는 경우 API 토큰 값이 일반 텍스트로 표시됩니다.

클라우드 템플릿 작업의 클라우드 템플릿 변수에 달려 바인딩을 적용하려면 다음 단계를 따릅니다.

integerVar, stringVar, flavorVar, BooleanVar, objectVar 및 arrayVar와 같은 입력 변수 속성이 정의된 클라우드 템플릿으로 시작합니다. resources 섹션에 정의된 이미지 속성을 찾을 수 있습니다. 클라우드 템플릿 코드의 속성은 다음과 유사할 수 있습니다.

```
formatVersion: 1
inputs:
  integerVar:
    type: integer
    encrypted: false
    default: 1
  stringVar:
    type: string
    encrypted: false
    default: bkix
  flavorVar:
    type: string
    encrypted: false
    default: medium
  BooleanVar:
    type: boolean
    encrypted: false
    default: true
  objectVar:
    type: object
    encrypted: false
    default:
      bkix2: bkix2
  arrayVar:
    type: array
    encrypted: false
    default:
      - '1'
      - '2'
resources:
  Cloud_Machine_1:
    type: Cloud.Machine
    properties:
      image: ubuntu
      flavor: micro
      count: '${input.integerVar}'
```

image 및 flavor에 대해 달러 기호 변수(\$)를 사용할 수 있습니다. 예를 들면 다음과 같습니다.

```
resources:
  Cloud_Machine_1:
    type: Cloud.Machine
    properties:
      input: '${input.image}'
      flavor: '${input.flavor}'
```

Code Stream 파이프라인에서 클라우드 템플릿을 사용하고 여기에 달러 바인딩을 추가하려면 다음 단계를 수행합니다.

- 1 Code Stream에서 **파이프라인 > 빈 캔버스**를 클릭합니다.
- 2 **클라우드 템플릿** 작업을 파이프라인에 추가합니다.
- 3 클라우드 템플릿 작업에서 **클라우드 템플릿 소스**에 대해 **Cloud Assembly 클라우드 템플릿**을 선택하고 클라우드 템플릿 이름을 입력한 다음, 클라우드 템플릿 버전을 선택합니다.
- 4 파이프라인에 대한 자격 증명을 제공하는 API 토큰을 입력할 수 있습니다. 클라우드 템플릿 작업에서 API 토큰을 암호화하는 변수를 생성하려면 **변수 생성**을 클릭합니다.
- 5 표시되는 **매개 변수 및 값** 테이블에서 매개 변수 값을 확인합니다. flavor의 기본값은 small이고 image의 기본값은 ubuntu입니다.
- 6 Cloud Assembly에서 클라우드 템플릿을 변경해야 한다고 가정합니다. 예를 들어,
  - a array 유형의 속성을 사용하도록 flavor를 설정합니다. Cloud Assembly는 유형이 **array**인 경우 Flavor에 대해 쉼표로 구분된 값을 허용합니다.
  - b **배포**를 클릭합니다.
  - c [배포 유형] 페이지에서 배포 이름을 입력하고 클라우드 템플릿 버전을 선택합니다.
  - d [배포 입력] 페이지에서 Flavor에 대해 하나 이상의 값을 정의할 수 있습니다.
  - e 배포 입력에는 클라우드 템플릿 코드에 정의된 모든 변수가 포함되며, 클라우드 템플릿 코드에 정의된 대로 표시됩니다. 예: Integer Var, String Var, Flavor Var, Boolean Var, Object Var 및 Array Var. String Var과 Flavor Var은 문자열 값이고 Boolean Var은 확인란입니다.
  - f **배포**를 클릭합니다.
- 7 Code Stream에서 새 클라우드 템플릿 버전을 선택하고 **매개 변수 및 값** 테이블에 값을 입력합니다. 클라우드 템플릿은 달러 기호 변수를 사용하여 Code Stream 바인딩을 사용하도록 설정하는 다음 유형의 매개 변수를 지원합니다. Code Stream 클라우드 템플릿 작업의 사용자 인터페이스와 Cloud Assembly 클라우드 템플릿의 사용자 인터페이스 사이에는 약간의 차이가 있습니다. Cloud Assembly의 클라우드 템플릿 코딩에 따라서는, Code Stream의 클라우드 템플릿 작업에 값을 입력하는 것이 허용되지 않을 수 있습니다.
  - a **flavorVar**의 경우 클라우드 템플릿이 문자열 또는 어레이로 유형을 정의한 경우에는 문자열 또는 쉼표로 구분된 값 어레이를 입력합니다. 어레이 예는 **test, test**와 유사합니다.

- b **BooleanVar**의 경우 드롭다운 메뉴에서 **true** 또는 **false**를 선택합니다. 또는 변수 바인딩을 사용하려면 **\$**를 입력하고 목록에서 변수 바인딩을 선택합니다.

Parameter	Value
stringVar	raj
integerVar	1
flavorVar	medium
BooleanVar	\$
objectVar	var
arrayVar	input

comments  
 requestBy  
 executionIndex  
 executionId  
 executionUrl  
 name  
 description  
 Stage0

Output Parameter

status
deploymentCriteria
deploymentId
deploymentName

- c **objectVar**의 경우 **{"bkix": "bkix":}**와 같이 중괄호 및 따옴표와 함께 값을 입력합니다.
- d **objectVar**은 클라우드 템플릿으로 전달되며, 클라우드 템플릿에 따라 다양한 방식으로 사용될 수 있습니다. 이는 JSON 개체에 대한 문자열 형식을 허용하며 키-값 쌍을 키-값 테이블에 쉼표로 구분된 값으로 추가할 수 있습니다. JSON 개체에 일반 텍스트를 입력하거나 키-값 쌍을 JSON의 일반 stringified 형식으로 입력할 수 있습니다.
- e **arrayVar**의 경우 쉼표로 구분된 입력 값을 **["1", "2"]** 형식의 어레이로 입력합니다.
- 8 파이프라인에서 입력 매개 변수를 어레이에 바인딩할 수 있습니다.
- a **입력** 탭을 클릭합니다.
- b 입력의 이름을 입력합니다. 예를 들어 **arrayInput**이라고 입력합니다.
- c **매개 변수 및 값** 테이블에서 **arrayVar**을 클릭하고 **\${input.arrayInput}**을 입력합니다.
- d 파이프라인을 저장 및 사용하도록 설정한 후 파이프라인이 실행되면 어레이 입력 값을 제공해야 합니다. 예를 들어 **["1", "2"]**를 입력하고 **실행**을 클릭합니다.

이제 Code Stream 파이프라인 클라우드 템플릿 작업의 클라우드 템플릿에서 달러 기호(\$) 변수 바인딩을 사용하는 방법을 알아보았습니다.

## 파이프라인이 실행될 때 파이프라인에 매개 변수를 전달하는 방법

Code Stream에서 입력 매개 변수를 파이프라인에 전달하도록 파이프라인에 입력 매개 변수를 추가할 수 있습니다. 그런 다음 파이프라인이 실행될 때 사용자가 입력 매개 변수의 값을 입력해야 합니다. 파이프라인에 출력 매개 변수를 추가하면 파이프라인 작업에서 작업의 출력 값을 사용할 수 있습니다. Code Stream은 고유한 파이프라인 요구 사항을 지원하는 여러 가지 방법으로 매개 변수 사용을 지원합니다.

예를 들어 REST 작업이 포함된 파이프라인이 실행될 때 사용자에게 Git 서버의 URL을 묻는 메시지를 표시하려는 경우 REST 작업을 Git 서버 URL에 바인딩할 수 있습니다.

변수 바인딩을 생성하려면 URL 바인딩 변수를 REST 작업에 추가합니다. 파이프라인이 실행되고 REST 작업에 도달하면 사용자는 Git 서버에 대한 해당 URL을 입력해야 합니다. 다음은 바인딩을 생성하는 방법입니다.

- 1 파이프라인에서 **입력** 탭을 클릭합니다.
- 2 매개 변수를 설정하려면 **매개 변수 자동 삽입**에 대해 **Git**를 클릭합니다.  
Git 매개 변수 목록이 표시되고 **GIT\_SERVER\_URL**이 포함됩니다. Git 서버 URL에 대해 기본값을 사용해야 하는 경우 이 매개 변수를 편집합니다.
- 3 **모델**을 클릭하고 REST 작업을 클릭합니다.
- 4 **작업** 탭의 URL 영역에서 **\$**를 입력한 다음 **입력** 및 **GIT\_SERVER\_URL**을 선택합니다.

The screenshot shows the configuration window for a task named 'Task3'. The 'Type' is set to 'REST'. Under 'REST Request', the 'Action' is 'GET' and the 'URL' is '\$(input|'. A dropdown menu is open, showing a list of variables including 'GIT\_SERVER\_URL', which is highlighted. The 'Output Parameters' section at the bottom shows 'status', 'responseHeaders', 'responseBody', 'responseJson', and 'responseCode'.

항목은 **\$(input.GIT\_SERVER\_URL)**과 유사합니다.

- 5 작업에 대한 변수 바인딩의 무결성을 확인하려면 **작업 검증**을 클릭합니다.  
작업이 성공적으로 검증되었다고 Code Stream에 표시됩니다.
- 6 파이프라인이 REST 작업을 실행할 때 사용자가 Git 서버의 URL을 입력해야 합니다. 그렇지 않으면 작업 실행이 완료되지 않습니다.



## 입력 및 출력 매개 변수를 생성하여 두 파이프라인 작업을 바인딩하는 방법

작업을 함께 바인딩할 때 수신하는 작업의 입력 구성에 바인딩 변수를 추가합니다. 그런 다음 파이프라인이 실행될 때 사용자가 바인딩 변수를 필요한 입력으로 바꿉니다.

파이프라인 작업을 함께 바인딩하려면 입력 매개 변수 및 출력 매개 변수에서 달러 기호 변수(\$)를 사용합니다. 이 예는 방법을 보여줍니다.

파이프라인이 REST 작업에서 URL을 호출하고 응답을 출력하도록 설정해야 한다고 가정합니다. URL을 호출하고 응답을 출력하려면 REST 작업에 입력 및 출력 매개 변수를 모두 포함합니다. 또한 작업을 승인할 수 있는 사용자가 필요하므로 파이프라인이 실행될 때 작업을 승인할 수 있는 다른 사용자에게 대한 [사용자 작업] 작업을 포함합니다. 이 예에서는 입력 및 출력 매개 변수에서 포현식을 사용하는 방법과 파이프라인이 작업에 대한 승인을 대기하는 방법을 보여 줍니다.

- 1 파이프라인에서 **입력** 탭을 클릭합니다.

rest-ix-1 Enabled ACTIONS ▾

Workspace **Input** Model Output

Input Parameters ⓘ

Auto inject parameters ☐ Gerrit ☐ Git ☐ Docker ☒ None

ADD ADD/REMOVE INJECTED PARAMETERS

Starred ⓘ	Name ▾	Value ▾	Description ▾
⋮ ☆	URL	{Stage0.Task3.input.http://www.docs.vmware.com}	Docs URL

- 2 매개 변수 자동 삽입을 **없음**으로 둡니다.
- 3 **추가**를 클릭하고 매개 변수 이름, 값 및 설명을 입력한 후 **확인**을 클릭합니다. 예를 들면 다음과 같습니다.
  - a URL 이름을 입력합니다.
  - b 값을 입력합니다. {Stage0.Task3.input.http://www.docs.vmware.com}
  - c 설명을 입력합니다.
- 4 **출력** 탭을 클릭하고 **추가**를 클릭한 다음 출력 매개 변수와 매핑을 입력합니다.

**Add Pipeline Output Parameter**

Name \*

Reference \$ \* 

- responseHeaders
- responseBody
- responseJson
- responseCode

- 고유한 출력 매개 변수 이름을 입력합니다.
- 참조 영역을 클릭하고 \$를 입력합니다.
- 팝업되는 옵션을 선택하여 작업 출력 매핑을 입력합니다. **Stage0, Task3, 출력**을 차례로 선택하고 **responseCode**를 선택합니다. 그런 다음 **확인**을 클릭합니다.

**rest-ix-1** Enabled ACTIONS ▾

Workspace Input Model **Output**

**Output Parameters** ⓘ

ADD

Starred ⓘ	Name ▾	Reference
⋮ ☆	RESTResponse	\${Stage0.Task3.output.responseCode}

- 파이프라인을 저장합니다.
- 작업** 메뉴에서 **실행**을 클릭합니다.
- 작업 > 실행 보기**를 클릭합니다.
- 파이프라인 실행을 클릭하고 정의한 입력 매개 변수 및 출력 매개 변수를 검토합니다.

**rest-ix-1 #2** WAITING 0 ACTIONS ▾

Stage0

Task2 Task3

Project chim

Execution rest-ix-1 #2

Status WAITING Stage0.Task2: Execution Waiting for User Action.

Updated By

Executed By kim@vmware.com

Comments Test Vars Expressions

Duration 37 seconds (Feb 4, 2020, 3:17:31 PM - Feb 4, 2020, 3:17:42 PM)

Input Parameters ▾

URL {Stage0.Task3.input.http://www.docs.vmware.com}

Workspace

No details available

Output Parameters ▾

Response tasks['Stage0.Task3']['output.responseCode']

- 9 파이프라인을 승인하려면 **사용자 작업**을 클릭하고 **활성 항목** 탭에서 승인 목록을 봅니다. 또는 실행 상태를 유지하고 작업을 클릭한 후 **승인**을 클릭합니다.
- 10 **승인** 및 **거부** 버튼을 사용하도록 설정하려면 실행 옆에 있는 확인란을 클릭합니다.
- 11 세부 정보를 보려면 드롭다운 화살표를 확장합니다.
- 12 작업을 승인하려면 **승인**을 클릭하고 이유를 입력한 후 **확인**을 클릭합니다.

The screenshot shows the 'User Operations' page with a 'GUIDED SETUP' link in the top right. Below the title are tabs for 'Active Items' and 'Inactive Items'. There are two buttons: a green 'APPROVE' button and a red 'REJECT' button. A refresh icon is in the top right corner. Below these is a table with columns for 'Index#' and 'Execution'. The first row is selected, showing 'User Operation #f0d252'. Below the table, the 'Request Details' are displayed in a light blue box. At the bottom of this box are three buttons: 'APPROVE' (green), 'REJECT' (red), and 'VIEW DASHBOARD' (blue).

**User Operations** GUIDED SETUP

Active Items Inactive Items

✓ APPROVE ✗ REJECT

Index# Execution

✓ User Operation #f0d252

**Request Details**

Execution	rest-ix-1 #2
Summary	hello
Approvers	kern@vmware.com, f0d2@vmware.com
Requested By	kern@vmware.com
Requested On	Feb 4, 2020, 3:17:40 PM
Expires On	Feb 7, 2020, 3:17:40 PM

APPROVE REJECT VIEW DASHBOARD

13 실행을 클릭하고 파이프라인이 계속되는지 확인합니다.

The screenshot shows the 'Executions' page with a 'GUIDED SETUP' link in the top right. Below the title is a badge showing '3,347 items'. There is a '+ NEW EXECUTION' button and a search bar. Below these is a table with columns for 'rest-i... #3', 'RUNNING', 'Stages', and 'ACTIONS'. The first row is selected, showing 'rest-i... #3' with a 'RUNNING' status. Below the table, the execution details are displayed in a light blue box. At the bottom of this box are three buttons: 'APPROVE' (green), 'REJECT' (red), and 'VIEW DASHBOARD' (blue).

**Executions** 3,347 items GUIDED SETUP

+ NEW EXECUTION

rest-i... #3 RUNNING Stages: 0 ACTIONS

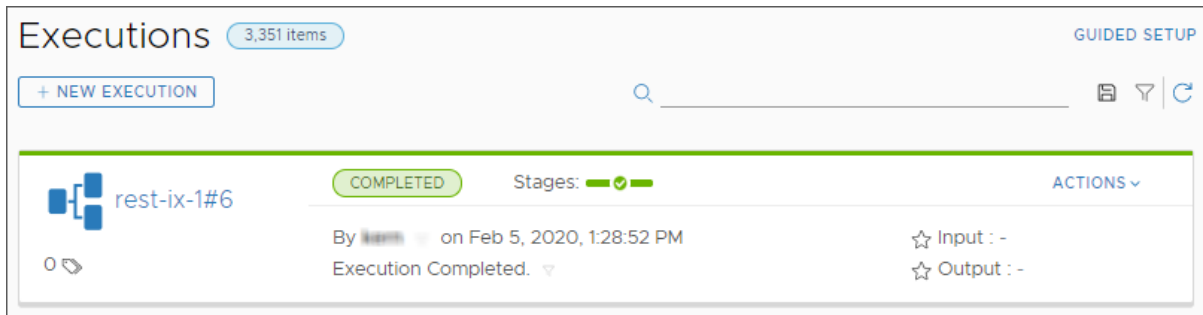
By kern on Feb 4, 2020, 3:41:05 PM

Input : -

Output : -

Comments: Testing

14 파이프라인이 실패하면 오류를 수정한 다음 파이프라인을 저장하고 다시 실행합니다.



## 변수 및 표현식에 대해 자세히 알아보는 방법

파이프라인 작업을 바인딩할 때 변수 및 표현식을 사용하는 방법에 대한 자세한 내용을 보려면 [Code Stream](#)에서 파이프라인 작업을 바인딩할 때 사용할 수 있는 변수 및 표현식 항목을 참조하십시오.

조건 변수 바인딩과 함께 파이프라인 작업 출력을 사용하는 방법을 알아보려면 [Code Stream](#)에서 파이프라인의 실행 또는 중지를 위해 조건 작업에서 변수 바인딩을 사용하는 방법 항목을 참조하십시오.

## Code Stream에서 파이프라인의 실행 또는 중지를 위해 조건 작업에서 변수 바인딩을 사용하는 방법

사용자가 제공하는 조건에 따라 파이프라인의 작업 출력으로 파이프라인을 실행할지 아니면 중지하도록 할지 결정할 수 있습니다. 작업 출력을 기반으로 파이프라인 성공 또는 실패를 지정하려면 [조건] 작업을 사용합니다.

**조건** 작업을 파이프라인의 결정 시점으로 사용합니다. 제공하는 조건 표현식이 있는 조건 작업을 사용하여 파이프라인, 단계 및 작업의 모든 속성을 평가할 수 있습니다.

조건 작업의 결과에 따라 파이프라인의 다음 작업이 실행되는지 여부가 결정됩니다.

- **true** 조건은 파이프라인 실행이 계속되도록 허용합니다.
- **false** 조건은 파이프라인을 중지합니다.

작업을 조건 작업과 함께 바인딩하여 한 작업의 출력 값을 다음 작업에 대한 입력으로 사용하는 방법에 대한 예는 [Code Stream](#) 파이프라인에서 변수 바인딩을 사용하는 방법의 내용을 참조하십시오.

표 3-7. 조건 작업 및 해당 조건 표현식이 파이프라인과 관련되는 방식

조건 작업	영향받는 대상	수행 작업
조건 작업	파이프라인	조건 작업은 작업 출력이 true인지 false인지에 따라 파이프라인을 해당 지점에서 실행할지 아니면 중지할지 결정합니다.
조건 표현식	조건 작업 출력	파이프라인이 실행되면 조건 작업에 포함하는 조건 표현식이 true 또는 false 출력 상태를 생성합니다. 예를 들어 조건식에서 조건 작업 출력 상태를 완료됨으로 요구하거나 빌드 번호 74를 사용할 수 있습니다. 조건식은 [조건] 작업의 [작업] 탭에 나타납니다. 

조건 작업은 다른 작업 유형의 조건부 설정과 기능 및 동작이 서로 다릅니다.



다른 작업 유형에서 조건부는 true 또는 false의 전제 조건 표현식 평가에 따라 후속 작업이 아닌 현재 작업이 실행되는지 여부를 결정합니다. 조건부 설정의 조건식은 파이프라인이 실행될 때 현재 작업에 대한 true 또는 false 출력 상태를 생성합니다. 조건부 설정은 [작업] 탭에 자체 조건식과 함께 나타납니다.

이 예에서는 조건 작업을 사용합니다.

#### 사전 요구 사항

- 파이프라인이 있고 파이프라인에 단계 및 작업이 포함되어 있는지 확인합니다.

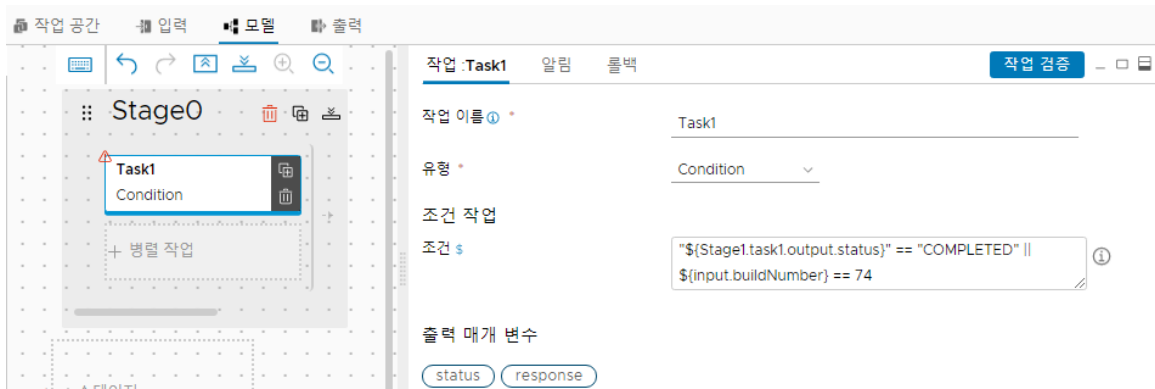
#### 절차

- 파이프라인에서 조건 작업이 나타나야 하는 결정 시점을 결정합니다.

- 성공 또는 실패 상태에 의존하는 작업 앞에 조건 작업을 추가합니다.
- 조건 작업에 조건 표현식을 추가합니다.

예:

```
"${Stage1.task1.output.status}" == "COMPLETED" || ${input.buildNumber} == 74
```



- 작업을 검증합니다.
- 파이프라인을 저장한 다음 사용하도록 설정하고 실행합니다.

## 결과

파이프라인 실행을 보고 파이프라인이 계속 실행되는지 아니면 조건 작업에서 중지되는지 확인합니다.

## 다음에 수행할 작업

파이프라인 배포를 롤백하는 경우 조건 작업을 사용할 수도 있습니다. 예를 들어 롤백 파이프라인에서 조건 작업은 Code Stream이 조건 표현식을 기반으로 파이프라인 실패를 표시하고 다양한 실패 유형에 대해 단일 롤백 흐름을 트리거할 수 있도록 도와 줍니다.

배포를 롤백하려면 [Code Stream](#)에서 배포를 롤백하는 방법 항목을 참조하십시오.

## Code Stream에서 파이프라인 작업을 바인딩할 때 사용할 수 있는 변수 및 표현식

변수와 표현식을 사용하면 파이프라인 작업에서 입력 매개 변수 및 출력 매개 변수를 사용할 수 있습니다. 입력한 매개 변수는 파이프라인 작업을 하나 이상의 변수, 표현식 또는 조건에 바인딩하고 실행 시 파이프라인 동작을 결정합니다.

## 파이프라인은 간단하거나 복잡한 소프트웨어 제공 솔루션을 실행할 수 있습니다.

파이프라인 작업을 함께 바인딩하는 경우 기본 및 복합 표현식을 포함할 수 있습니다. 그 결과, 파이프라인은 간단하거나 복잡한 소프트웨어 제공 솔루션을 실행할 수 있습니다.

파이프라인에서 매개 변수를 생성하려면 **입력** 또는 **출력** 탭을 클릭하고 달러 기호(\$) 및 표현식을 입력하여 변수를 추가합니다. 예를 들어 이 매개 변수는 \${Stage0.Task3.input.URL} URL을 호출하는 작업 입력으로 사용됩니다.

변수 바인딩 형식은 범위 및 키라는 구문 구성 요소를 사용합니다. SCOPE는 컨텍스트를 입력 또는 출력으로 정의하고 KEY는 세부 정보를 정의합니다. 매개 변수 예 \${Stage0.Task3.input.URL}에서 input은 SCOPE이고 URL은 KEY입니다.

모든 작업의 출력 속성은 변수 바인딩의 여러 중첩 수준으로 확인할 수 있습니다.

파이프라인에서 변수 바인딩 사용에 대한 자세한 내용은 [Code Stream 파이프라인에서 변수 바인딩을 사용하는 방법](#) 항목을 참조하십시오.

## 범위 및 키에 달러 표현식을 사용하여 파이프라인 작업 바인딩

달러 기호 변수에 표현식을 사용하여 파이프라인 작업을 바인딩할 수 있습니다. 표현식을 \${SCOPE.KEY.<PATH>}로 입력합니다.

파이프라인 작업의 동작을 확인하기 위해 각 표현식에서 SCOPE은 Code Stream에 사용되는 컨텍스트입니다. 범위는 작업이 수행하는 작업에 대한 세부 정보를 정의하는 KEY를 찾습니다. KEY의 값이 중첩된 개체인 경우 선택적 PATH를 제공할 수 있습니다.

제공된 예에서는 SCOPE 및 KEY에 대해 설명하고 파이프라인에서 사용하는 방법을 보여줍니다.



표 3-8. SCOPE 및 KEY 사용

SCOPE	표현식 및 예제의 목적	KEY	파이프라인에서 SCOPE 및 KEY를 사용하는 방법
input	파이프라인의 입력 속성: <code>\${input.input1}</code>	입력 속성의 이름	<p>작업에서 파이프라인의 입력 속성을 참조하려면 다음 형식을 사용합니다.</p> <pre>tasks:   mytask:     type: REST     input:       url: \$       {input.url}     action: get</pre> <pre>input:   url: https://   www.vmware.com</pre>
output	파이프라인의 출력 속성: <code>\${output.output1}</code>	출력 속성의 이름	<p>알림을 보내기 위한 출력 속성을 참조하려면 다음 형식을 사용합니다.</p> <pre>notifications:   email:     - endpoint:       MyEmailEndpoint       subject:       "Deployment       Successful"       event: COMPLETED       to:       -       user@example.org       body:           Pipeline         deployed the         service         successfully.         Refer \$         {output.serviceURL}</pre>

표 3-8. SCOPE 및 KEY 사용 (계속)

SCOPE	표현식 및 예제의 목적	KEY	파이프라인에서 SCOPE 및 KEY를 사용하는 방법
<b>task input</b>	작업에 대한 입력: \$ {MY_STAGE.MY_TASK.input. SOMETHING}	알림에서 작업의 입력을 나타냅니다.	Jenkins 작업이 시작되면 작업 입력에서 트리거된 작업의 이름을 참조할 수 있습니다. 이 경우 다음 형식을 사용하여 알림을 전송합니다.  <pre> notifications:   email:     - endpoint:       MyEmailEndpoint         stage: MY_STAGE         task: MY_TASK         subject:           "Build Started"           event: STARTED           to:             -               user@example.org               body:                   Jenkins job \$                 {MY_STAGE.MY_TASK.i                 nput.job} started                 for commit id \$                 {input.COMMITID}). </pre>
<b>task output</b>	작업의 출력: \$ {MY_STAGE.MY_TASK.output. .SOMETHING}	후속 작업에서의 작업 출력을 나타냅니다.	작업 2에서 파이프라인 작업 1의 출력을 참조하려면 다음 형식을 사용합니다.  <pre> taskOrder:   - task1   - task2 tasks:   task1:     type: REST     input:       action: get       url: https://       www.example.org/api       /status   task2:     type: REST     input:       action: post       url: https://       status.internal.exa       mple.org/api/       activity       payload: \$       {MY_STAGE.task1.out       put.responseBody} </pre>

표 3-8. SCOPE 및 KEY 사용 (계속)

SCOPE	표현식 및 예제의 목적	KEY	파이프라인에서 SCOPE 및 KEY를 사용하는 방법
<b>var</b>	변수: <code>\${var.myVariable}</code>	끝점의 변수를 참조합니다.	암호의 끝점에서 비밀 변수를 참조하려면 다음 형식을 사용합니다.  <pre>--- project: MyProject kind: ENDPOINT name: MyJenkinsServer type: jenkins properties:   url: https:// jenkins.example.com username: jenkinsUser password: \$ {var.jenkinsPassword}</pre>
<b>var</b>	변수: <code>\${var.myVariable}</code>	파이프라인에서 변수를 참조합니다.	파이프라인 URL에서 변수를 참조하려면 다음 형식을 사용합니다.  <pre>tasks:   task1:     type: REST     input:       action: get       url: \$ {var.MY_SERVER_URL}</pre>
<b>task status</b>	작업의 상태: <code>\$</code> <code>{MY_STAGE.MY_TASK.status}</code> <code>}</code> <code>\$</code> <code>{MY_STAGE.MY_TASK.status</code> <code>Message}</code>		
<b>stage status</b>	단계의 상태: <code>\${MY_STAGE.status}</code> <code>\$</code> <code>{MY_STAGE.statusMessage}</code>		

## 기본 표현식

파이프라인에서 표현식과 함께 변수를 사용할 수 있습니다. 이 요약에는 사용할 수 있는 기본 표현식이 포함되어 있습니다.

표현식	설명
<code>\${comments}</code>	파이프라인 실행 요청 시 제공되는 주석입니다.
<code>\${duration}</code>	파이프라인 실행의 기간입니다.
<code>\${endTime}</code>	완료된 경우 파이프라인 실행 종료 시간(UTC)입니다.
<code>\${executedOn}</code>	시작 시간과 마찬가지로, 파이프라인 실행 시작 시간(UTC)입니다.
<code>\${executionId}</code>	파이프라인 실행의 ID입니다.
<code>\${executionUrl}</code>	사용자 인터페이스에서 파이프라인 실행으로 이동하는 URL입니다.
<code>\${name}</code>	파이프라인의 이름입니다.
<code>\${requestBy}</code>	실행을 요청한 사용자의 이름입니다.
<code>\${stageName}</code>	단계 범위에서 사용되는 경우 현재 단계의 이름입니다.
<code>\${startTime}</code>	파이프라인 실행 시작 시간(UTC)입니다.
<code>\${status}</code>	실행의 상태입니다.
<code>\${statusMessage}</code>	파이프라인 실행의 상태 메시지입니다.
<code>\${taskName}</code>	작업 입력 또는 알림에서 사용되는 경우 현재 작업의 이름입니다.

## 파이프라인 작업에 SCOPE 및 KEY 사용

지원되는 파이프라인 작업에 표현식을 사용할 수 있습니다. 다음 예에서는 SCOPE 및 KEY를 정의하고 구문을 확인하는 방법을 보여줍니다. 코드 예제에서는 MY\_STAGE 및 MY\_TASK를 파이프라인 단계 및 작업 이름으로 사용합니다.

사용 가능한 작업에 대한 자세한 내용을 보려면 [Code Stream](#)에서 사용할 수 있는 작업 유형 항목을 참조하십시오.

표 3-9. 게이팅 작업

작업	Scope	Key	작업에서 SCOPE 및 KEY를 사용하는 방법
<b>사용자 작업</b>			
	Input	<p>summary: 사용자 작업에 대한 요청 요약</p> <p>description: 사용자 작업에 대한 요청 설명</p> <p>approvers: 승인자 이메일 주소 목록. 여기서 각 항목은 선택 가능한 변수이거나, 별도의 이메일에 대해 세미콜론을 사용할 수 있음</p> <p>approverGroups: 플랫폼 및 ID에 대한 승인자 그룹 주소 목록</p> <p>sendemail: <b>True</b>로 설정하면 필요에 따라 요청 또는 응답 시 이메일 알림 전송</p> <p>expirationInDays: 요청의 만료 시간을 나타내는 일 수</p>	<p><code>\${MY_STAGE.MY_TASK.input.summary}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.description}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.approvers}</code></p> <p><code>\$</code></p> <p><code>{MY_STAGE.MY_TASK.input.approverGroups}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.sendemail}</code></p> <p><code>\$</code></p> <p><code>{MY_STAGE.MY_TASK.input.expirationInDays}</code></p>
	Output	<p>index: 요청을 나타내는 6자리 16진수 문자열</p> <p>respondedBy: 사용자 작업을 승인/거부한 사람의 계정 이름</p> <p>respondedByEmail: 응답한 사람의 이메일 주소</p> <p>comments: 응답 중에 제공된 설명</p>	<p><code>\${MY_STAGE.MY_TASK.output.index}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.respondedBy}</code></p> <p><code>\$</code></p> <p><code>{MY_STAGE.MY_TASK.output.respondedByEmail}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.comments}</code></p>
<b>조건</b>			
	Input	<p>condition: 평가할 조건. 조건이 <b>True</b>로 평가되면 작업이 완료로 표시되며 다른 응답은 작업이 실패함</p>	<p><code>\${MY_STAGE.MY_TASK.input.condition}</code></p>
	Output	<p>result: 평가 시 결과</p>	<p><code>\${MY_STAGE.MY_TASK.output.response}</code></p>

표 3-10. 파이프라인 작업

작업	Scope	Key	작업에서 SCOPE 및 KEY를 사용하는 방법
파이프라인			
	Input	name: 실행할 파이프라인의 이름 inputProperties: 중첩된 파이프라인 실행에 전달할 입력 속성	<pre> \${MY_STAGE.MY_TASK.input.name}  \${MY_STAGE.MY_TASK.input.inputProperties}# 모든 속성 참조  \$ {MY_STAGE.MY_TASK.input.inputProperties.inpu t1} # 입력1의 값 참조 </pre>
	Output	executionStatus: 파이프라인 실행 상태 executionIndex: 파이프라인 실행의 인덱스 outputProperties: 파이프라인 실행의 출력 속성:	<pre> \${MY_STAGE.MY_TASK.output.executionStatus}  \${MY_STAGE.MY_TASK.output.executionIndex}  \$ {MY_STAGE.MY_TASK.output.outputProperties}# 모든 속성 참조  \$ {MY_STAGE.MY_TASK.output.outputProperties.ou tput1} # 출력1의 값 참조 </pre>

표 3-11. 지속적 통합 작업 자동화

작업	Scope	Key	작업에서 SCOPE 및 KEY를 사용하는 방법
CI			
	Input	steps: 실행할 명령을 나타내는 문자열 집합 export: 단계를 실행한 후 보존할 환경 변수 artifacts: 공유 경로에서 보존할 아티팩트의 경로 process: JUnit, JaCoCo, Checkstyle, FindBugs 처리를 위한 구성 요소의 집합	<pre> \${MY_STAGE.MY_TASK.input.steps}  \${MY_STAGE.MY_TASK.input.export}  \${MY_STAGE.MY_TASK.input.artifacts}  \${MY_STAGE.MY_TASK.input.process}  \$ {MY_STAGE.MY_TASK.input.process[0].path } # 첫 번째 구성 경로 참조 </pre>
	Output	exports: 입력 export에서 내보낸 환경 변수를 나타내는 키-값 쌍 artifacts: 성공적으로 보존된 아티팩트 경로 processResponse: 입력 process에 대한 처리 결과 집합	<pre> \${MY_STAGE.MY_TASK.output.exports}# 모든 내보내기 참조  \$ {MY_STAGE.MY_TASK.output.exports.myvar} # myvar 값 참조  \${MY_STAGE.MY_TASK.output.artifacts}  \$ {MY_STAGE.MY_TASK.output.processRespons e}  \$ {MY_STAGE.MY_TASK.output.processRespons e[0].result} # 첫 번째 프로세스 구성의 결과 </pre>
사용자 지정			

표 3-11. 지속적 통합 작업 자동화 (계속)

작업	Scope	Key	작업에서 SCOPE 및 KEY를 사용하는 방법
	Input	name: 사용자 지정 통합의 이름 version: 릴리스 또는 폐기된 사용자 지정 통합의 버전 properties: 사용자 지정 통합에 보낼 속성	<pre> \${MY_STAGE.MY_TASK.input.name} \${MY_STAGE.MY_TASK.input.version} \${MY_STAGE.MY_TASK.input.properties} # 모든 속성 참조 \$ {MY_STAGE.MY_TASK.input.properties.property1} #속성1의 값 참조 </pre>
	Output	properties: 사용자 지정 통합 응답의 출력 속성	<pre> \${MY_STAGE.MY_TASK.output.properties} # 모든 속성 참조 \$ {MY_STAGE.MY_TASK.output.properties.property1} #속성1의 값 참조 </pre>

표 3-12. 지속적 배포 작업 자동화: 클라우드 템플릿

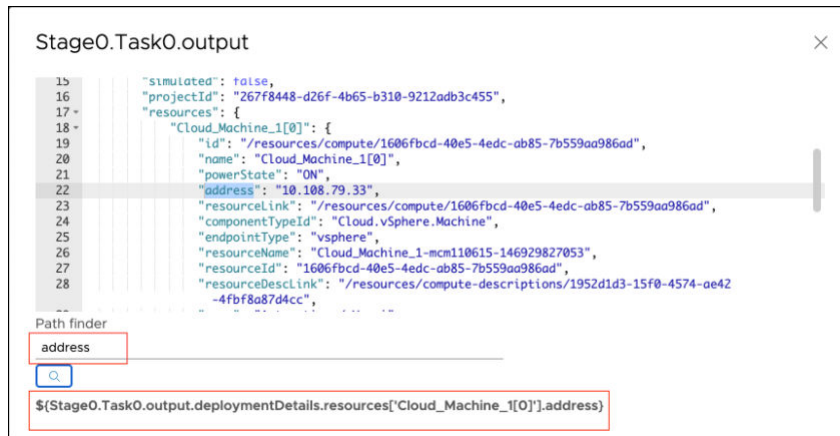
작업	Scope	Key	작업에서 SCOPE 및 KEY를 사용하는 방법
클라우드 템플릿	Input	<p>action: <b>createDeployment</b>,  <b>updateDeployment</b>,  <b>deleteDeployment</b>,  <b>rollbackDeployment</b> 중 하나</p> <p>blueprintInputParams: 배포  생성 및 배포 업데이트 작업에 사  용</p> <p>allowDestroy: 배포 업데이트  프로세스에서 시스템이 제거될  수 있습니다.</p> <p><b>CREATE_DEPLOYMENT</b></p> <ul style="list-style-type: none"> <li>■ blueprintName: 클라우드  템플릿의 이름</li> <li>■ blueprintVersion: 클라우  드 템플릿의 버전</li> </ul> <p>또는</p> <ul style="list-style-type: none"> <li>■ fileUrl: GIT 서버를 선택한  후 원격 클라우드 템플릿  YAML의 URL입니다.</li> </ul> <p><b>UPDATE_DEPLOYMENT</b></p> <p>다음 조합 중 하나입니다.</p> <ul style="list-style-type: none"> <li>■ blueprintName: 클라우드  템플릿의 이름</li> <li>■ blueprintVersion: 클라우  드 템플릿의 버전</li> </ul> <p>또는</p> <ul style="list-style-type: none"> <li>■ fileUrl: GIT 서버를 선택한  후 원격 클라우드 템플릿  YAML의 URL입니다.</li> </ul> <p>-----</p> <ul style="list-style-type: none"> <li>■ deploymentId: 배포의 ID</li> </ul> <p>또는</p> <ul style="list-style-type: none"> <li>■ deploymentName: 배포의 이  름</li> </ul> <p>-----</p> <p><b>DELETE_DEPLOYMENT</b></p> <ul style="list-style-type: none"> <li>■ deploymentId: 배포의 ID</li> </ul> <p>또는</p> <ul style="list-style-type: none"> <li>■ deploymentName: 배포의 이  름</li> </ul> <p><b>ROLLBACK_DEPLOYMENT</b></p> <p>다음 조합 중 하나입니다.</p> <ul style="list-style-type: none"> <li>■ deploymentId: 배포의 ID</li> </ul>	



표 3-12. 지속적 배포 작업 자동화: 클라우드 템플릿 (계속)

작업	Scope	Key	작업에서 SCOPE 및 KEY를 사용하는 방법
		<p>또는</p> <ul style="list-style-type: none"> <li>■ deploymentName: 배포의 이름</li> </ul> <p>-----</p> <ul style="list-style-type: none"> <li>■ blueprintName: 클라우드 템플릿의 이름</li> <li>■ rollbackVersion: 롤백할 버전</li> </ul>	
	Output		<p>다른 작업 또는 파이프라인의 출력에 바인딩할 수 있는 매개 변수:</p> <ul style="list-style-type: none"> <li>■ 배포 이름은 <code>\${Stage0.Task0.output.deploymentName}</code> 으로 액세스할 수 있습니다.</li> <li>■ 배포 ID는 <code>\${Stage0.Task0.output.deploymentId}</code> 로 액세스할 수 있습니다.</li> <li>■ 배포 세부 정보는 복합 개체이며, JSON 결과를 사용하여 내부 세부 정보에 액세스할 수 있습니다.</li> </ul> <p>속성에 액세스하려면 점 연산자를 사용하여 JSON 계층을 따릅니다. 예를 들어 리소스 <code>Cloud_Machine_1[0]</code> 의 주소에 액세스하려는 경우 <code>\$</code> 바인딩은 다음과 같습니다.</p> <pre>\$ {Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].address}</pre> <p>마찬가지로 버전의 경우 <code>\$</code> 바인딩은 다음과 같습니다.</p> <pre>\$ {Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].flavor}</pre> <p>Code Stream 사용자 인터페이스에서 모든 속성에 대한 <code>\$</code> 바인딩을 얻을 수 있습니다.</p> <ol style="list-style-type: none"> <li>1 작업 출력 속성 영역에서 <b>View Output JSON</b>을 클릭합니다.</li> <li>2 <code>\$</code> 바인딩을 찾으려면 속성을 입력합니다.</li> <li>3 해당하는 <code>\$</code> 바인딩을 표시하는 검색 아이콘을 클릭합니다.</li> </ol>

JSON 출력 예:



샘플 배포 세부 정보 개체:

```

{
  "id": "6a031f92-d0fa-42c8-bc9e-3b260ee2f65b",
  "name": "deployment_6a031f92-d0fa-42c8-bc9e-3b260ee2f65b",
  "description": "Pipeline Service triggered operation",
  "orgId": "434f6917-4e34-4537-b6c0-3bf3638a71bc",
  "blueprintId": "8d1dd801-3a32-4f3b-adde-27f8163dfe6f",
  "blueprintVersion": "1",
  "createdAt": "2020-08-27T13:50:24.546215Z",
  "createdBy": "user@vmware.com",
  "lastUpdatedAt": "2020-08-27T13:52:50.674957Z",
  "lastUpdatedBy": "user@vmware.com",
  "inputs": {},
  "simulated": false,
  "projectId": "267f8448-d26f-4b65-b310-9212adb3c455",
  "resources": {
    "Cloud_Machine_1[0]": {
      "id": "/resources/compute/1606fbcd-40e5-4edc-ab85-7b559aa986ad",
      "name": "Cloud_Machine_1[0]",
      "powerState": "ON",
      "address": "10.108.79.33",
      "resourceLink": "/resources/compute/1606fbcd-40e5-4edc-ab85-7b559aa986ad",
      "componentTypeId": "Cloud.vSphere.Machine",
      "endpointType": "vsphere",
      "resourceName": "Cloud_Machine_1-mcm110615-146929827053",
      "resourceId": "1606fbcd-40e5-4edc-ab85-7b559aa986ad",
      "resourceDescLink": "/resources/compute-descriptions/1952d1d3-15f0-4574-ae42-4fbf8a87d4cc",
      "zone": "Automation / Vms",
      "countIndex": "0",
      "image": "ubuntu",
      "count": "1",
      "flavor": "small",
      "region": "MYBU",
      "_clusterAllocationSize": "1",
      "osType": "LINUX",
      "componentType": "Cloud.vSphere.Machine",
      "account": "bha"
    }
  }
}

```

```

    },
    "status": "CREATE_SUCCESSFUL",
    "deploymentURI": "https://api.yourenv.com/automation-ui/#/deployment-ui;ash=/deployment/6a031f92-d0fa-42c8-bc9e-3b260ee2f65b"
  }
}

```

표 3-13. 지속적인 배포 작업 자동화: Kubernetes

작업	Scope	Key	작업에서 SCOPE 및 KEY를 사용하는 방법
Kubernetes			
	Input	<p>action: <b>GET, CREATE, APPLY, DELETE, ROLLBACK</b> 중 하나</p> <ul style="list-style-type: none"> <li>■ timeout: 모든 작업에 대한 전체 시간 초과</li> <li>■ filterByLabel: K8S labelSelector를 사용하여 <b>GET</b> 작업에 대해 필터링할 추가 레이블</li> </ul> <p><b>GET, CREATE, DELETE, APPLY</b></p> <ul style="list-style-type: none"> <li>■ yaml: 처리하여 Kubernetes에 보낼 인라인 YAML</li> <li>■ parameters: KEY, VALUE 쌍 - <b>\$\$KEY</b>를 인라인 YAML 입력 영역의 <b>VALUE</b>로 바꿈</li> <li>■ filePath: YAML을 가져올 SCM Git 끝점의 상대 경로(제공된 경우)</li> <li>■ scmConstants: KEY, VALUE 쌍 - <b>\$\$KEY</b>를 SCM을 통해 가져온 YAML의 <b>VALUE</b>로 바꿈</li> <li>■ continueOnConflict: <b>True</b>로 설정하면 리소스가 이미 있는 경우 작업이 계속됨</li> </ul> <p><b>ROLLBACK</b></p> <ul style="list-style-type: none"> <li>■ resourceType: 롤백할 리소스 유형</li> <li>■ resourceName: 롤백할 리소스 이름</li> <li>■ namespace: 롤백을 수행해야 하는 네임스페이스</li> <li>■ revision: 롤백할 개정</li> </ul>	<p><code>\${MY_STAGE.MY_TASK.input.action}</code> #수행할 작업을 결정합니다.</p> <p><code>\${MY_STAGE.MY_TASK.input.timeout}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.filterByLabel}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.yaml}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.parameters}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.filePath}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.scmConstants}</code></p> <p><code>\$</code></p> <p><code>{MY_STAGE.MY_TASK.input.continueOnConflict}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.resourceType}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.resourceName}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.namespace}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.revision}</code></p>
	Output	<p>response: 전체 응답 캡처</p> <p>response.&lt;RESOURCE&gt;: configMaps, deployments, endpoints, ingresses, jobs, namespaces, pods, replicaSets, replicationControllers, secrets, services, statefulSets, nodes, loadBalancers에 해당하는 리소스</p> <p>response.&lt;RESOURCE&gt;.&lt;KEY&gt;: apiVersion, kind, metadata, spec 중 하나에 해당하는 키</p>	<p><code>\${MY_STAGE.MY_TASK.output.response}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.response.}</code></p>

표 3-14. 개발, 테스트 및 배포 애플리케이션 통합

작업	Scope	Key	작업에서 SCOPE 및 KEY를 사용하는 방법
Bamboo			
	Input	plan: 계획의 이름	<code>\${MY_STAGE.MY_TASK.input.plan}</code>
		planKey: 계획 키	<code>\${MY_STAGE.MY_TASK.input.planKey}</code>
		variables: 계획에 전달할 변수	<code>\${MY_STAGE.MY_TASK.input.variables}</code>
		parameters: 계획에 전달할 매개 변수	<code>\${MY_STAGE.MY_TASK.input.parameters}</code> # 모든 매개 변수 참조
			<code>\${MY_STAGE.MY_TASK.input.parameters.param1}</code> # param1의 값 참조
	Output	resultUrl: 결과 빌드의 URL	<code>\${MY_STAGE.MY_TASK.output.resultUrl}</code>
		buildResultKey: 결과 빌드의 키	<code>\${MY_STAGE.MY_TASK.output.buildResultKey}</code>
		buildNumber: 빌드 번호	<code>\${MY_STAGE.MY_TASK.output.buildNumber}</code>
		buildTestSummary: 실행된 테스트의 요약	<code>\${MY_STAGE.MY_TASK.output.buildTestSummary}</code> # 모든 결과 참조
		successfulTestCount: 테스트 결과 통과	<code>\${MY_STAGE.MY_TASK.output.successfulTestCount}</code> # 특정 테스트 수 참조
		failedTestCount: 테스트 결과 실패	<code>\${MY_STAGE.MY_TASK.output.buildNumber}</code>
		skippedTestCount: 테스트 결과 건너뛴	
		artifacts: 빌드의 아티팩트	
Jenkins			
	Input	job: Jenkins 작업의 이름	<code>\${MY_STAGE.MY_TASK.input.job}</code>
		parameters: 작업에 전달할 매개 변수	<code>\${MY_STAGE.MY_TASK.input.parameters}</code> # 모든 매개 변수 참조
			<code>\${MY_STAGE.MY_TASK.input.parameters.param1}</code> # 매개 변수의 값 참조
	Output	job: Jenkins 작업의 이름	<code>\${MY_STAGE.MY_TASK.output.job}</code>
		jobId: 결과 작업의 ID(예: 1234)	<code>\${MY_STAGE.MY_TASK.output.jobId}</code>
		jobStatus: Jenkins의 상태	<code>\${MY_STAGE.MY_TASK.output.jobStatus}</code>
		jobResults: 테스트/코드 범위 결과 수집	<code>\${MY_STAGE.MY_TASK.output.jobResults}</code> # 모든 결과 참조
			<code>\${MY_STAGE.MY_TASK.output.jobResults.junitResponse}</code> # JUnit 결과 참조
		jobUrl: 결과 작업 실행의 URL	<code>\${MY_STAGE.MY_TASK.output.jobResults.jacocoResponse}</code> # JaCoCo 결과 참조
		<code>\${MY_STAGE.MY_TASK.output.jobUrl}</code>	
TFS			

표 3-14. 개발, 테스트 및 배포 애플리케이션 통합 (계속)

작업	Scope	Key	작업에서 SCOPE 및 KEY를 사용하는 방법
	Input	projectCollection: TFS에서 프로젝트 수집 teamProject: 사용 가능한 수집에서 선택한 프로젝트 buildDefinitionId: 실행할 빌드 정의 ID	<code>\${MY_STAGE.MY_TASK.input.projectCollection}</code> <code>\${MY_STAGE.MY_TASK.input.teamProject}</code> <code>\${MY_STAGE.MY_TASK.input.buildDefinitionId}</code>
	Output	buildId: 결과 빌드 ID buildUrl: 빌드 요약물 방문하기 위한 URL logUrl: 로그 방문을 위한 URL dropLocation: 아티팩트의 삭제 위치(있는 경우)	<code>\${MY_STAGE.MY_TASK.output.buildId}</code> <code>\${MY_STAGE.MY_TASK.output.buildUrl}</code> <code>\${MY_STAGE.MY_TASK.output.logUrl}</code> <code>\${MY_STAGE.MY_TASK.output.dropLocation}</code>
<b>vRO</b>			
	Input	workflowId: 실행할 워크플로의 ID parameters: 워크플로에 전달할 매개 변수	<code>\${MY_STAGE.MY_TASK.input.workflowId}</code> <code>\${MY_STAGE.MY_TASK.input.parameters}</code>
	Output	workflowExecutionId: 워크플로 실행 ID properties: 워크플로 실행의 출력 속성	<code>\${MY_STAGE.MY_TASK.output.workflowExecutionId}</code> <code>\${MY_STAGE.MY_TASK.output.properties}</code>

표 3-15. API를 통해 다른 애플리케이션 통합

작업	Scope	Key	작업에서 SCOPE 및 KEY를 사용하는 방법
<b>REST</b>			
	Input	url: 호출할 URL action: 사용할 HTTP 메서드 headers: 전달할 HTTP 헤더 payload: 요청 페이로드 fingerprint: https URL과 일치시킬 지문 allowAllCerts: true로 설정하면 URL이 https인 인증서일 수 있음	<code>\${MY_STAGE.MY_TASK.input.url}</code> <code>\${MY_STAGE.MY_TASK.input.action}</code> <code>\${MY_STAGE.MY_TASK.input.headers}</code> <code>\${MY_STAGE.MY_TASK.input.payload}</code> <code>\${MY_STAGE.MY_TASK.input.fingerprint}</code> <code>\${MY_STAGE.MY_TASK.input.allowAllCerts}</code>

표 3-15. API를 통해 다른 애플리케이션 통합 (계속)

작업	Scope	Key	작업에서 SCOPE 및 KEY를 사용하는 방법
	Output	<p>responseCode: HTTP 응답 코드</p> <p>responseHeaders: HTTP 응답 헤더</p> <p>responseBody: 수신한 응답의 문자열 형식</p> <p>responseJson: 콘텐츠 유형이 <b>application/json</b>인 경우 통과 가능한 응답</p>	<p><code>\${MY_STAGE.MY_TASK.output.responseCode}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseHeaders}</code></p> <p><code>\$</code></p> <p><code>{MY_STAGE.MY_TASK.output.responseHeaders.header1} # 응답 헤더 'header1' 참조</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseBody}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseJson} # JSON으로 응답 참조</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseJson.a.b.c} # 응답에서 a.b.c JSON 경로 다음에 나오는 중첩된 개체 참조</code></p>
폴링			
	Input	<p>url: 호출할 URL</p> <p>headers: 전달할 HTTP 헤더</p> <p>exitCriteria: 작업이 성공하거나 실패하기 위해 충족해야 하는 조건. '성공'→표현식, '실패'→표현식의 키-값 쌍</p> <p>pollCount: 수행할 반복 횟수. <b>Code Stream</b> 관리자는 폴링 수를 최대 10000으로 설정할 수 있습니다.</p> <p>pollIntervalSeconds: 각 반복 사이의 대기 시간(초). 폴링 간격은 60초보다 크거나 같아야 합니다.</p> <p>ignoreFailure: <b>True</b>로 설정하면 중간 응답 실패를 무시함</p> <p>fingerprint: https URL과 일치시킬 지문</p> <p>allowAllCerts: <b>true</b>로 설정하면 URL이 https인 인증서일 수 있음</p>	<p><code>\${MY_STAGE.MY_TASK.input.url}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.headers}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.exitCriteria}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.pollCount}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.pollIntervalSeconds}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.ignoreFailure}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.fingerprint}</code></p> <p><code>\${MY_STAGE.MY_TASK.input.allowAllCerts}</code></p>
	Output	<p>responseCode: HTTP 응답 코드</p> <p>responseBody: 수신한 응답의 문자열 형식</p> <p>responseJson: 콘텐츠 유형이 <b>application/json</b>인 경우 통과 가능한 응답</p>	<p><code>\${MY_STAGE.MY_TASK.output.responseCode}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseBody}</code></p> <p><code>\${MY_STAGE.MY_TASK.output.responseJson} # Refer to response as JSON</code></p>

표 3-16. 원격 및 사용자 정의 스크립트 실행

작업	Scope	Key	작업에서 SCOPE 및 KEY를 사용하는 방법
<b>PowerShell</b> PowerShell 작업을 실행하려면 다음을 수행해야 합니다. <ul style="list-style-type: none"> <li>■ 원격 Windows 호스트에 대한 활성 세션이 있어야 합니다.</li> <li>■ base64 PowerShell 명령을 입력하려면 먼저 전체 명령 길이를 계산합니다. 자세한 내용은 <a href="#">Code Stream에서 사용할 수 있는 작업 유형</a> 항목을 참조하십시오.</li> </ul>			
	Input	host: 시스템의 IP 주소 또는 호스트 이름 username: 연결에 사용할 사용자 이름 password: 연결에 사용할 암호 useTLS: <b>https</b> 연결 시도 trustCert: <b>True</b> 로 설정하면 자체 서명된 인증서를 신뢰함 script: 실행할 스크립트 workingDirectory: 스크립트를 실행하기 전에 전환할 디렉토리 경로 environmentVariables: 설정할 환경 변수의 키-값 쌍 arguments: 스크립트에 전달할 인수	<pre> \${MY_STAGE.MY_TASK.input.host} \${MY_STAGE.MY_TASK.input.username} \${MY_STAGE.MY_TASK.input.password} \${MY_STAGE.MY_TASK.input.useTLS} \${MY_STAGE.MY_TASK.input.trustCert} \${MY_STAGE.MY_TASK.input.script} \$ {MY_STAGE.MY_TASK.input.workingDirectory} \$ {MY_STAGE.MY_TASK.input.environmentVariables} \${MY_STAGE.MY_TASK.input.arguments} </pre>
	Output	response: \$SCRIPT_RESPONSE_FILE 파일의 콘텐츠 responseFilePath: \$SCRIPT_RESPONSE_FILE의 값 exitCode: 프로세스 종료 코드 logFilePath: <b>stdout</b> 를 포함하는 파일 경로 errorFilePath: <b>stderr</b> 를 포함하는 파일 경로	<pre> \${MY_STAGE.MY_TASK.output.response} \$ {MY_STAGE.MY_TASK.output.responseFilePath} \${MY_STAGE.MY_TASK.output.exitCode} \${MY_STAGE.MY_TASK.output.logFilePath} \${MY_STAGE.MY_TASK.output.errorFilePath} </pre>
<b>SSH</b>			

표 3-16. 원격 및 사용자 정의 스크립트 실행 (계속)

작업	Scope	Key	작업에서 SCOPE 및 KEY를 사용하는 방법
	Input	host: 시스템의 IP 주소 또는 호스트 이름 username: 연결에 사용할 사용자 이름 password: 연결에 사용할 암호 (필요한 경우 <b>privateKey</b> 를 사용할 수 있음) privateKey: 연결에 사용할 <b>PrivateKey</b> passphrase: <b>privateKey</b> 잠금 해제를 위한 선택적 암호 script: 실행할 스크립트 workingDirectory: 스크립트를 실행하기 전에 전환할 디렉토리 경로 environmentVariables: 설정할 환경 변수의 키-값 쌍	<pre> \${MY_STAGE.MY_TASK.input.host} \${MY_STAGE.MY_TASK.input.username} \${MY_STAGE.MY_TASK.input.password} \${MY_STAGE.MY_TASK.input.privateKey} \${MY_STAGE.MY_TASK.input.passphrase} \${MY_STAGE.MY_TASK.input.script} \$ {MY_STAGE.MY_TASK.input.workingDirectory} \$ {MY_STAGE.MY_TASK.input.environmentVariables} </pre>
	Output	response: <b>SCRIPT_RESPONSE_FILE</b> 파일의 콘텐츠 responseFilePath: <b>SCRIPT_RESPONSE_FILE</b> 의 값 exitCode: 프로세스 종료 코드 logFilePath: <b>stdout</b> 를 포함하는 파일 경로 errorFilePath: <b>stderr</b> 를 포함하는 파일 경로	<pre> \${MY_STAGE.MY_TASK.output.response} \$ {MY_STAGE.MY_TASK.output.responseFilePath} \${MY_STAGE.MY_TASK.output.exitCode} \${MY_STAGE.MY_TASK.output.logFilePath} \${MY_STAGE.MY_TASK.output.errorFilePath} </pre>

## 작업 간에 변수 바인딩을 사용하는 방법

다음 예에서는 파이프라인 작업에서 변수 바인딩을 사용하는 방법을 보여 줍니다.

표 3-17. 샘플 구문 형식

예	구문
파이프라인 알림 및 파이프라인 출력 속성에 대해 작업 출력 값을 사용하려는 경우	<code>\${&lt;Stage Key&gt;.&lt;Task Key&gt;.output.&lt;Task output key&gt;}</code>
이전 작업 출력 값을 현재 작업에 대한 입력으로 참조하려는 경우	<code>\${&lt;Previous/Current Stage key&gt;.&lt;Previous task key not in current Task group&gt;.output.&lt;task output key&gt;}</code>



## 자세한 내용

작업의 바인딩 변수에 대한 자세한 내용은 다음을 참조하십시오.

- [Code Stream 파이프라인에서 변수 바인딩을 사용하는 방법](#)
- [Code Stream에서 파이프라인의 실행 또는 중지를 위해 조건 작업에서 변수 바인딩을 사용하는 방법](#)
- [Code Stream에서 사용할 수 있는 작업 유형](#)

## Code Stream에서 내 파이프라인에 대한 알림을 보내는 방법

Code Stream에서 알림을 통해 팀과 통신하고 파이프라인의 상태를 알릴 수 있습니다.

파이프라인이 실행될 때 알림을 보내려면 전체 파이프라인, 단계 또는 작업의 상태를 기반으로 Code Stream 알림을 구성하면 됩니다.

- 이메일 알림은 다음과 같은 경우 이메일을 보냅니다.
  - 파이프라인 완료, 대기, 실패, 취소 또는 시작
  - 단계 완료, 실패 또는 시작
  - 작업 완료, 대기, 실패 또는 시작
- 티켓 알림은 다음과 같은 경우 티켓을 생성하고 이를 팀 멤버에게 할당합니다.
  - 파이프라인 실패 또는 완료
  - 단계 실패
  - 작업 실패
- Webhook 알림은 다음과 같은 경우 다른 애플리케이션에 요청을 보냅니다.
  - 파이프라인 실패, 완료, 대기, 취소 또는 시작
  - 단계 실패, 완료 또는 시작
  - 작업 실패, 완료, 대기 또는 시작

예를 들어 사용자 작업에 대한 이메일 알림을 구성하여 파이프라인의 특정 지점에서 승인을 받을 수 있습니다. 파이프라인이 실행되면 이 작업은 작업을 승인해야 하는 사람에게 이메일을 보냅니다. [사용자 작업]에 만료 시간 제한이 일, 시간 또는 분 단위로 설정되어 있으면, 필수 사용자는 작업이 만료되기 전에 파이프라인을 승인해야 합니다. 그렇지 않으면 파이프라인이 예상대로 실패합니다.

파이프라인 작업이 실패할 때 Jira 티켓을 생성하기 위해 알림을 구성할 수 있습니다. 또는 파이프라인 이벤트를 기반으로 파이프라인 상태에 대한 요청을 Slack 채널에 보내기 위해 Webhook 알림을 구성할 수 있습니다.

모든 유형의 알림에서 변수를 사용할 수 있습니다. 예를 들어 Webhook 알림의 URL에서 `${var}`을 사용할 수 있습니다.

## 사전 요구 사항

- 하나 이상의 파이프라인이 생성되었는지 확인합니다. [장 5 Code Stream](#) 사용을 위한 자습서의 사용 사례를 참조하십시오.
- 이메일 알림을 보내기 위해 이메일 서버에 액세스할 수 있는지 확인합니다. 도움이 필요한 경우 관리자에게 문의하십시오.
- Jira 티켓과 같은 티켓을 생성하려면 끝점이 있어야 합니다. [Code Stream의 끝점 소개](#) 항목을 참조하십시오.
- 통합을 기반으로 알림을 보내려면 Webhook 알림을 생성합니다. 그런 다음 Webhook가 추가되고 작동하는지 확인합니다. 알림은 Slack, GitHub 또는 GitLab과 같은 애플리케이션에서 사용할 수 있습니다.

## 절차

- 1 파이프라인을 엽니다.
- 2 전반적인 파이프라인 상태나 단계 또는 작업의 상태에 대한 알림을 생성하려면 다음을 수행합니다.

생성할 알림:	수행할 작업:
파이프라인 상태	파이프라인 캔버스에서 빈 영역을 클릭합니다.
단계의 상태	파이프라인 단계에서 빈 영역을 클릭합니다.
작업의 상태	파이프라인 단계의 작업을 클릭합니다.

- 3 **알림** 탭을 클릭합니다.
- 4 **추가**를 클릭하고 알림 유형을 선택한 다음 알림 세부 정보를 구성합니다.
- 5 파이프라인이 성공할 경우 **Slack** 알림을 생성하려면 **Webhook** 알림을 생성합니다.
  - a **Webhook**를 선택합니다.
  - b Slack 알림을 구성하기 위해 정보를 입력합니다.
  - c **저장**을 클릭합니다.
  - d 파이프라인이 실행되면 **Slack** 채널에서 파이프라인 상태에 대한 알림을 받습니다. 예를 들어, 사용자가 Slack 채널에서 다음 정보를 볼 수 있습니다.

```
Codestream APP [12:01 AM]
Tested by User1 - Staging Pipeline 'User1-Pipeline', Pipeline ID
'e9b5884d809ce2755728177f70f8a' succeeded
```

- 6 Jira 티켓을 생성하려면 티켓 정보를 구성합니다.
  - a 티켓을 선택합니다.
  - b Jira 알림을 구성하기 위해 정보를 입력합니다.
  - c **저장**을 클릭합니다.

### Notification

**Send notification type**

☐ Email
 ☒ Ticket
 ☐ Webhook

**When pipeline \***

☒ Fails
 ☐ Completes

**Jira endpoint \***

Jira-Notification

**Create Ticket**

**Jira project \***

YourProject

**Issue type \***

Bug

**Assignee \***

username@yourcompany.com

**Summary \$ \***

Pipeline failed

**Description \$**

Research and correct

CANCEL

SAVE

## 결과

축하합니다! Code Stream에서 파이프라인의 여러 영역에 대한 다양한 유형의 알림을 생성할 수 있다는 것을 알아보았습니다.

## 다음에 수행할 작업

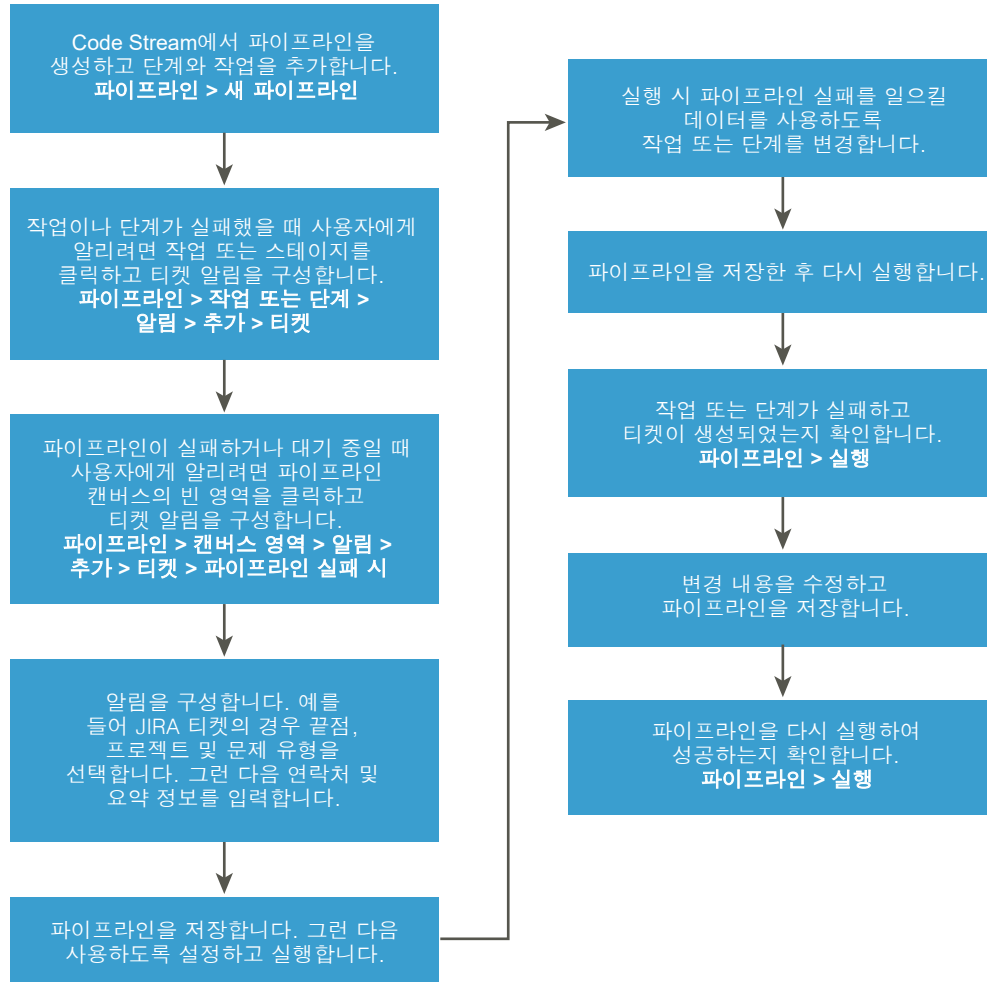
알림을 생성하는 방법에 대한 자세한 예는 [파이프라인 작업이 실패한 경우 Code Stream에서 Jira 티켓을 생성하는 방법](#) 항목을 참조하십시오.

## 파이프라인 작업이 실패한 경우 Code Stream에서 Jira 티켓을 생성하는 방법

파이프라인의 단계 또는 작업이 실패하면 Code Stream에서 Jira 티켓이 생성되도록 설정할 수 있습니다. 이 티켓은 문제를 해결해야 하는 사람에게 할당할 수 있습니다. 파이프라인이 대기 중일 때나 완료된 경우에도 티켓을 생성할 수 있습니다.

작업, 단계 또는 파이프라인에 대한 알림을 추가하고 구성할 수 있습니다. Code Stream은 사용자가 알림을 추가하는 작업, 단계 또는 파이프라인의 상태에 기반하여 티켓을 생성합니다. 예를 들어, 사용할 수 있는 끝점이 없는 경우, 끝점에 연결할 수 없어서 실패한 작업에 대해 Code Stream에서 Jira 티켓을 생성할 수 있습니다.

파이프라인이 완료된 경우에 알림을 생성할 수도 있습니다. 예를 들어, QA 팀이 빌드를 확인하고 다른 테스트 파이프라인을 실행할 수 있게 성공한 파이프라인에 대한 정보를 QA 팀에 알릴 수 있습니다. 또는 성능 팀에 알려서 성능 팀이 파이프라인의 성능을 측정하고 스테이징 또는 운영에 대한 업데이트를 준비하도록 할 수 있습니다.



이 예제에서는 파이프라인 작업이 실패할 때 Jira 티켓을 생성합니다.

#### 사전 요구 사항

- 유효한 Jira 계정이 있는지 그리고 Jira 인스턴스에 로그인할 수 있는지 확인합니다.
- Jira 끝점이 있고 현재 작동 중인지 확인합니다.

#### 절차

- 1 파이프라인에서 작업을 클릭합니다.

- 2 작업 구성 영역에서 **알림**을 클릭합니다.
- 3 **추가**를 클릭하고 티켓 정보를 구성합니다.
  - a **티켓**을 클릭합니다.
  - b Jira 끝점을 선택합니다.
  - c Jira 프로젝트 및 문제 유형을 입력합니다.
  - d 티켓을 수신할 담당자의 이메일 주소를 입력합니다.
  - e 티켓에 대한 요약 및 설명을 입력한 후 **저장**을 클릭합니다.

### Notification

Send notification type ☐ Email ☒ Ticket ☐ Webhook

When task \* ☒ Fails

Jira endpoint \* TestJira ▾

Create Ticket

Jira project \* YourProject

Issue type \* Bug

Assignee \* username@yourcompany.com

Summary \$ \* CI task failed

Description\$ 

Research and correct

CANCEL

SAVE

- 4 파이프라인을 저장한 다음 사용하도록 설정하고 실행합니다.
- 5 티켓을 테스트합니다.
  - a 작업 실패를 일으키는 데이터를 포함하도록 작업 정보를 변경합니다.
  - b 파이프라인을 저장한 후 다시 실행합니다.
  - c **실행**을 클릭하여 파이프라인이 실패했는지 확인합니다.
  - d 실행 시 Code Stream이 티켓을 생성한 후 전송했는지 확인합니다.
  - e 작업 정보를 변경하여 수정한 후 파이프라인을 다시 실행하여 파이프라인이 완료되는지 확인합니다.

## 결과

축하합니다! 파이프라인 작업이 실패할 때 Code Stream에서 Jira 티켓을 생성하도록 하여 문제를 해결해야 하는 담당자에게 할당했습니다.

## 다음에 수행할 작업

계속해서 팀에게 파이프라인에 대해 경고하는 알림을 추가합니다.

# Code Stream에서 배포를 롤백하는 방법

배포 파이프라인 실패 후 이전의 안정된 상태로 배포를 되돌리는 작업이 포함된 파이프라인으로 롤백을 구성합니다. 실패가 발생하는 경우 롤백하려면 롤백 파이프라인을 작업 또는 단계에 연결합니다.

사용자의 역할에 따라 롤백 이유는 다를 수 있습니다.

- 릴리스 엔지니어는 릴리스하는 동안 Code Stream을 통해 릴리스의 성공 여부를 확인하여 릴리스를 계속할지 아니면 롤백할지 알 수 있습니다. 가능한 실패에는 작업 실패, UserOps의 거부, 메트릭 임계값 초과 등이 포함됩니다.
- 환경 소유자는 이전 릴리스를 다시 배포하여 환경을 알려진 정상 상태로 빠르게 되돌릴 수 있습니다.
- 환경 소유자는 블루-그린 배포의 롤백을 지원하여 릴리스 실패에 따른 다운타임을 최소화할 수 있습니다.

롤백 옵션을 클릭한 상태로 스마트 파이프라인 템플릿을 사용하여 CD 파이프라인을 생성하면 롤백이 자동으로 파이프라인의 작업에 추가됩니다. 이 사용 사례에서는 스마트 파이프라인 템플릿을 사용하여 롤링 업 그레이드 배포 모델을 사용하는 Kubernetes 클러스터에 애플리케이션을 배포하기 위한 롤백을 정의합니다. 스마트 파이프라인 템플릿은 배포 파이프라인과 하나 이상의 롤백 파이프라인을 생성합니다.

- 배포 파이프라인에서 [배포 업데이트] 또는 [배포 확인] 작업이 실패하면 롤백이 필요합니다.
- 롤백 파이프라인에서 배포는 이전 이미지로 업데이트됩니다.

빈 템플릿을 사용하여 롤백 파이프라인을 수동으로 생성할 수도 있습니다. 롤백 파이프라인을 생성하기 전에 롤백 흐름을 계획할 수도 있습니다. 롤백에 대한 자세한 배경 정보는 [Code Stream에서 롤백 계획](#) 항목을 참조하십시오.

## 사전 요구 사항

- Code Stream에서 프로젝트의 멤버인지 확인합니다. 그렇지 않으면 프로젝트 멤버로 추가해 달라고 Code Stream 관리자에게 요청합니다. [Code Stream에서 프로젝트를 추가하는 방법](#)의 내용을 참조하십시오.
- 파이프라인이 애플리케이션을 배포할 Kubernetes 클러스터를 설정합니다. 개발 클러스터와 운영 클러스터를 각각 하나씩 설정합니다.
- Docker 레지스트리 설정이 있는지 확인합니다.
- 파이프라인, 끝점 및 대시보드를 비롯한 모든 작업을 그룹화할 프로젝트를 식별합니다.

- 스마트 파이프라인 템플릿을 사용하기 전에 **Code Stream**에서 **CICD 기본 구축 계획**의 **CD** 부분에 설명된 대로 **CD** 스마트 템플릿에 관한 내용을 숙지합니다. 예를 들면 다음과 같습니다.
  - 애플리케이션 이미지를 **Kubernetes** 클러스터에 배포하는 **Kubernetes** 개발 및 운영 끝점을 생성합니다.
  - 네임스페이스, 서비스 및 배포를 생성하는 **Kubernetes** **YAML** 파일을 준비합니다. 개인 소유 저장소에서 이미지를 다운로드해야 하는 경우 **YAML** 파일에 **Docker** 구성 비밀이 있는 섹션이 포함되어야 합니다.

## 절차

- 1 **파이프라인 > 새 파이프라인 > 스마트 템플릿 > 지속적 전달**을 클릭합니다.
- 2 스마트 파이프라인 템플릿에 정보를 입력합니다.
  - a 프로젝트를 선택합니다.
  - b 파이프라인 이름을 입력합니다(예: **RollingUpgrade-Example**).
  - c 애플리케이션에 대한 환경을 선택합니다. 배포에 롤백을 추가하려면 **운영**을 선택해야 합니다.
  - d **선택**을 클릭하고 **Kubernetes** **YAML** 파일을 선택한 후 **처리**를 클릭합니다.  
사용 가능한 서비스와 배포 환경이 스마트 파이프라인 템플릿에 표시됩니다.
  - e 배포를 위해 파이프라인에서 사용할 서비스를 선택합니다.
  - f 개발 환경 및 운영 환경에 대한 클러스터 끝점을 선택합니다.
  - g [이미지 소스]로는 **파이프라인 런타임 입력**을 선택합니다.
  - h [배포 모델]로는 **롤링 업그레이드**를 선택합니다.

- i **롤백**을 클릭합니다.
- j **상태 점검 URL**을 제공합니다.

스마트 템플릿: 지속적 전달

플랫폼 사전 요구 사항 [?](#) Kubernetes Docker 레지스트리

프로젝트 \* test1

파이프라인 이름 \* RollbackUpgrade-Example

환경 [?](#) \* ☒ 개발 ☒ 운영

Kubernetes YAML 파일 \* 선택 표로세스  
저장된 파일 Kubernetesbgreen1.yaml

서비스 선택

배포 이름	서비스	네임스페이스	이미지
codestream-demo	codestream-demo	bgreen1	symphony-tango-beta2.jfrog.io/codestream-demo

1개 서비스

배포

환경	클러스터 클러스터	네임스페이스
개발	Kubernetes-Endpoint-Staging	bgreen1-542132
운영	Kubernetes-Endpoint-Staging	bgreen1

이미지 소스 \* ☐ Docker 트리거 ☒ 파이프라인 런타임 입력

배포 모델 \* ☐ Canary ☒ 롤링 업그레이드 ☐ 블루-그린

롤백 ☒

상태 점검 URL \* /health-check.json

생성 취소

- 3 RollbackUpgrade-Example이라는 이름의 파이프라인을 생성하려면 **생성**을 클릭합니다.

RollbackUpgrade-Example이라는 파이프라인이 나타나고, 개발 단계 및 운영 단계에서 롤백할 수 있는 작업에 롤백 아이콘이 표시됩니다.

RollbackUpgrade-Example 사용 안 함

작업 공간 | 입력 | **모형** | 출력

작업 'Create Secret'

유형 \* Kubernetes

실패 시 계속 ☐

작업 실행 ☒ 항상 ☐ 조건부

Kubernetes 작업 속성

Kubernetes 클러스터 \* Dev-VKE-Cluster

시간 초과(분) \* 5

작업 \* ☐ 가져오기 ☒ 생성 ☐ 적용 ☐ 삭제 ☐ 롤백

종료 시 계속 ☐

소스 유형 \* ☐ 소스 제어 ☒ 로컬 정의

로컬 YAML 정의 [파일에서 읽기](#)

```

1 apiVersion: v1
2 data:
3   .dockercfg: eyJ2LWw6G9ueS10YV5nby1iZXRhMiSsd1afjka12sdefxrg2hsc2hsh
4   2fsh5zxdg2dfh5ss13h8dfs5453hdfsfnf3as15ghh1fs315h3f1ds5h5s3df15
5   h315sdf15h53108f0s45h04fsd54h56h4In19
6 Kind: Secret
7 metadata:
8   name: jfrog-beta2
9   namespace: bgreen-549930
10 type: kubernetes.io/dockercfg

```

출력 매개 변수

status k8sRollbackTaskFields endpoint response yamls operation config

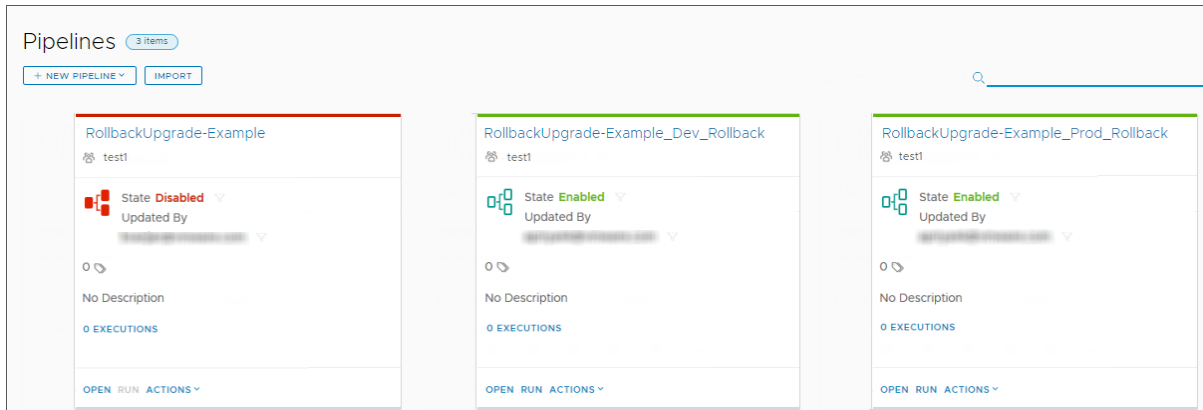
저장 실행 닫기 마지막 저장 시간: 10분 전



#### 4 파이프라인을 닫습니다.

파이프라인 페이지에 생성한 파이프라인이 나타나고 파이프라인의 각 단계에 대한 새 파이프라인이 나타납니다.

- **RollingUpgrade-Example.** 생성한 파이프라인은 Code Stream에서 기본적으로 비활성화되기 때문에 파이프라인을 실행하기 전에 검토할 수 있습니다.
- **RollingUpgrade-Example\_Dev\_Rollback.** 개발 단계(예: **서비스 생성, 비밀 생성, 배포 생성, 배포 확인**)에서 작업이 실패하면 개발 롤백 파이프라인이 호출됩니다. 개발 작업의 롤백을 보장하기 위해 Code Stream은 기본적으로 개발 롤백 파이프라인을 사용하도록 설정합니다.
- **RollingUpgrade-Example\_Prod\_Rollback.** 운영 단계(예: **배포 단계 1, 확인 단계 1, 롤아웃 배포 단계, 롤아웃 완료 단계, 롤아웃 확인 단계**)에서 작업이 실패하면 운영 롤백 파이프라인이 호출됩니다. 운영 작업의 롤백을 보장하기 위해 Code Stream은 기본적으로 운영 롤백 파이프라인을 사용하도록 설정합니다.



#### 5 생성한 파이프라인을 사용하도록 설정하고 실행합니다.

실행을 시작할 때 Code Stream은 입력 매개 변수를 묻는 메시지를 표시합니다. 사용 중인 Docker 저장소의 끝점에 대한 이미지 및 태그를 제공합니다.

#### 6 [실행] 페이지에서 **작업 > 실행 보기**를 선택하고 파이프라인 실행을 봅니다.

파이프라인이 **RUNNING**을 시작하고 개발 단계 작업을 통해 이동합니다. 개발 단계에서 파이프라인이 작업 실행에 실패하면 RollingUpgrade-Example\_Dev\_Rollback이라는 파이프라인이 트리거되어 배포를 롤백하고 파이프라인 상태가 **ROLLING\_BACK**으로 변경됩니다.

< BACK

## RollbackUpgrade-Example #1 ROLLING\_BACK 0 ACTIONS ▾

● Development

✓ Create Namespace
✓ Create Secret
✓ Create Service
● Create Deployment
⌂
● Verify Deployment

---

Project: test1

Execution: RollbackUpgrade-Example #1

Status: ROLLING\_BACK RUNNING

Updated by:

Executed by:

Duration: 12m 9s 186ms (01/11/2019 1:24 PM - )

Input Parameters ▾

image: demo-image-cs

tag: latest

Workspace

Details not available

Output Parameters ▾

The Execution did not output any properties

롤백 후에는 [실행] 페이지에 두 개의 RollingUpgrade-Example 파이프라인 실행이 나열됩니다.

- 생성한 파이프라인이 롤백되고 **ROLLBACK\_COMPLETED**가 표시됩니다.
- 롤백을 트리거하고 수행한 개발 롤백 파이프라인에 **COMPLETED**가 표시됩니다.

Executions 604 items

[+ NEW EXECUTION](#) 🔍

---

RollbackUpgrade-Example\_Dev... #1

1 Rollback for RollbackUpgrade-Example#1

COMPLETED

Stages: ● ● ●

By Cloud on 01/11/2019 1:36 PM

Execution Completed.

Comments: Triggered to rollback Development. Create Deployment of RollbackUpgrade-Example#1

---

RollbackUpgrade-Example#1

0

ROLLBACK\_COMPLETED

Stages: ● ● ● ● ●

By Cloud on 01/11/2019 1:24 PM

Create Deployment ROLLBACK\_COMPLETED

## 결과

축하합니다! 롤백이 포함된 파이프라인을 정의했고 Code Stream이 실패 지점에서 파이프라인을 롤백하는 것을 확인했습니다.

# Code Stream에서 기본적인 코드 구축, 통합 및 제공 계획

## 4

Code Stream을 통해 CICD, CI 또는 CD 파이프라인을 생성하는 네이티브 기능을 사용하여 코드를 구축, 통합 및 제공하려면 먼저 네이티브 빌드를 계획합니다. 그런 다음 스마트 파이프라인 템플릿 중 하나를 사용하거나 수동으로 단계 및 작업을 추가하여 파이프라인을 생성할 수 있습니다.

지속적 통합 및 지속적 전달 구축을 계획하기 위해 방법을 보여주는 몇 가지 예제가 포함되어 있습니다. 이러한 계획에서는 파이프라인을 구축할 때 기본 구축 기능을 효과적으로 준비하고 사용하는 데 유용한 사전 요구 사항 및 개요를 설명합니다.

본 장은 다음 항목을 포함합니다.

- 파이프라인 작업 공간 구성
- 스마트 파이프라인 템플릿을 사용하기 전에 Code Stream에서 CICD 기본 구축 계획
- 스마트 파이프라인 템플릿을 사용하기 전에 Code Stream에서 지속적 통합 네이티브 빌드 계획
- 스마트 파이프라인 템플릿을 사용하기 전에 Code Stream에서 지속적 전달 네이티브 빌드 계획
- 작업을 수동으로 추가하기 전에 Code Stream에서 CICD 네이티브 빌드 계획
- Code Stream에서 롤백 계획

## 파이프라인 작업 공간 구성

지속적 통합 작업 및 사용자 지정 작업을 실행하려면 Code Stream 파이프라인에 대한 업무 공간을 구성해야 합니다.

파이프라인 업무 공간에서 Docker 또는 Kubernetes로 **유형**을 선택하고 해당 끝점을 제공합니다. Docker 및 Kubernetes 플랫폼은 CI(지속적 통합) 작업 또는 사용자 지정 작업을 실행하기 위해 Code Stream에서 배포하는 컨테이너의 전체 수명 주기를 관리합니다.

- Docker 업무 공간에는 Docker 호스트 끝점, 빌더 이미지 URL, 이미지 레지스트리, 작업 디렉토리, 캐시, 환경 변수, CPU 제한 및 메모리 제한이 필요합니다. Git 저장소의 복제본을 생성할 수도 있습니다.
- Kubernetes 업무 공간에는 Kubernetes API 끝점, 빌더 이미지 URL, 이미지 레지스트리, 네임스페이스, NodePort, PVC(영구 볼륨 할당), 작업 디렉토리, 환경 변수, CPU 제한 및 메모리 제한이 필요합니다. Git 저장소의 복제본을 생성할 수도 있습니다.

파이프라인 업무 공간 구성에는 다음 표에 설명된 것처럼 많은 공통 매개 변수와 업무 공간 유형과 관련된 기타 매개 변수가 있습니다.

표 4-1. 작업 공간 영역, 세부 정보 및 가용성

선택	설명	세부 정보 및 가용성
유형	업무 공간의 유형입니다.	Docker 또는 Kubernetes와 함께 사용할 수 있습니다.
호스트 끝점	CI(지속적 통합) 및 사용자 지정 작업이 실행되는 호스트 끝점입니다.	Docker 호스트 끝점을 선택하면 Docker 업무 공간에서 사용할 수 있습니다. Kubernetes API 끝점을 선택하면 Kubernetes 업무 공간에서 사용할 수 있습니다.
빌더 이미지 URL	빌더 이미지의 이름 및 위치입니다. 컨테이너는 Docker 호스트 및 Kubernetes 클러스터에서 이 이미지를 사용하여 생성됩니다. CI(지속적 통합) 작업 및 사용자 지정 작업은 이 컨테이너 내에서 실행됩니다.	예: <b>fedora:latest</b> 빌더 이미지에는 curl 또는 wget이 있어야 합니다.
이미지 레지스트리	빌더 이미지를 레지스트리에서 사용할 수 있고 레지스트리에 자격 증명이 필요한 경우에는 레지스트리에서 이미지를 가져올 수 있도록 이미지 레지스트리 끝점을 생성한 다음 여기에서 선택해야 합니다.	Docker 및 Kubernetes 업무 공간에서 사용할 수 있습니다.
작업 디렉토리	작업 디렉토리는 CI(지속적 통합) 작업의 단계가 실행되는 컨테이너 내부의 위치이며 Git Webhook가 파이프라인 실행을 트리거할 때 코드가 복제되는 위치입니다.	Docker 또는 Kubernetes와 함께 사용할 수 있습니다.
네임스페이스	네임스페이스를 입력하지 않으면 Code Stream은 사용자가 제공한 Kubernetes 클러스터에 고유한 이름을 생성합니다.	Kubernetes 업무 공간에만 해당됩니다.
프록시	Kubernetes 클러스터의 업무 공간 포트와 통신하기 위해 Code Stream은 각 Kubernetes 클러스터의 네임스페이스 <b>codestream-proxy</b> 에 단일 프록시 인스턴스를 배포합니다. 클러스터 구성에 따라 <b>NodePort</b> 또는 <b>LoadBalancer</b> 유형을 선택할 수 있습니다. 선택하는 옵션은 배포된 Kubernetes 클러스터의 특성에 따라 달라집니다. <ul style="list-style-type: none"> <li>■ 일반적으로 끝점에 지정된 Kubernetes API 서버 URL이 기본 노드 중 하나를 통해 노출되는 경우 <b>NodePort</b>를 선택합니다.</li> <li>■ Kubernetes API 서버 URL이 Amazon EKS(Elastic Kubernetes Service)의 경우처럼 로드 밸런서에 의해 노출되는 경우에는 <b>LoadBalancer</b>를 선택합니다.</li> </ul>	
NodePort	Code Stream은 NodePort를 사용하여 Kubernetes 클러스터 내에서 실행되는 컨테이너와 통신합니다. 포트를 선택하지 않으면 Code Stream은 Kubernetes가 할당하는 사용 후 삭제 포트를 사용합니다. 방화벽 규칙의 구성이 사용 후 삭제 포트 범위(30000-32767)에 대한 수신을 허용하는지 확인해야 합니다. 포트를 입력하는 경우 클러스터의 다른 서비스가 해당 포트를 이미 사용하고 있지 않으며 방화벽 규칙이 해당 포트를 허용하는지 확인해야 합니다.	Kubernetes 업무 공간에만 해당됩니다.

표 4-1. 작업 공간 영역, 세부 정보 및 가용성 (계속)

선택	설명	세부 정보 및 가용성
영구 볼륨 할당	<p>Kubernetes 업무 공간이 파이프라인 실행 간에 파일을 유지하는 방법을 제공합니다. 영구 볼륨 할당 이름을 제공하면 로그, 아티팩트 및 캐시를 저장할 수 있습니다.</p> <p>영구 볼륨 할당 생성에 대한 자세한 내용은 Kubernetes 설명서(<a href="https://kubernetes.io/docs/concepts/storage/persistent-volumes/">https://kubernetes.io/docs/concepts/storage/persistent-volumes/</a>)를 참조하십시오.</p>	Kubernetes 업무 공간에만 해당됩니다.
환경 변수	여기에 전달된 키-값 쌍은 파이프라인이 실행될 때 파이프라인의 모든 CI(지속적 통합) 작업 및 사용자 지정 작업에서 사용할 수 있습니다.	<p>Docker 또는 Kubernetes와 함께 사용할 수 있습니다.</p> <p>변수에 대한 참조를 여기에 전달할 수 있습니다.</p> <p>업무 공간에 제공된 환경 변수는 파이프라인의 모든 CI(지속적 통합) 작업 및 사용자 지정 작업에 전달됩니다.</p> <p>환경 변수가 여기에 전달되지 않으면 해당 변수는 파이프라인의 각 CI(지속적 통합) 작업 및 사용자 지정 작업에 명시적으로 전달되어야 합니다.</p>
CPU 제한	CI(지속적 통합) 컨테이너 또는 사용자 지정 작업 컨테이너에 대한 CPU 리소스 제한입니다.	기본값은 1입니다.
메모리 제한	CI(지속적 통합) 컨테이너 또는 사용자 지정 작업 컨테이너에 대한 메모리 제한입니다.	단위는 MB입니다.

표 4-1. 작업 공간 영역, 세부 정보 및 가용성 (계속)

선택	설명	세부 정보 및 가용성
Git 클론	Git 클론을 선택하고 Git Webhook가 파이프라인을 호출하면 코드가 업무 공간(컨테이너)에 복제됩니다.	Git 클론을 사용하도록 설정하지 않은 경우 먼저 코드를 복제한 다음 빌드 및 테스트와 같은 다른 단계를 수행하도록 파이프라인에서 또 다른 명시적 CI(지속적 통합)작업을 구성해야 합니다.
캐시	<p>Code Stream 업무 공간을 사용하면 일련의 디렉토리 또는 파일을 캐시하여 후속 파이프라인 실행 속도를 높일 수 있습니다. 이러한 디렉토리의 예로는 .m2 및 npm_modules가 있습니다. 파이프라인 실행 간에 데이터 캐싱이 필요하지 않은 경우 영구 볼륨 할당이 필요하지 않습니다.</p> <p>컨테이너의 파일 또는 디렉토리나 같은 아티팩트는 파이프라인 실행 전반에서 재사용할 수 있도록 캐시됩니다. 예: node_modules 또는 .m2 폴더를 캐시할 수 있습니다. 캐시는 경로 목록을 허용합니다.</p> <p>예를 들면 다음과 같습니다.</p> <pre>workspace:   type: K8S   endpoint: K8S-Micro   image: fedora:latest   registry: Docker Registry   path: ''   cache:     - /path/to/m2     - /path/to/node_modules</pre>	<p>업무 공간 유형에 따라 다릅니다.</p> <p>Docker 업무 공간에서 캐시된 데이터, 아티팩트 및 로그를 유지하기 위해 Docker 호스트의 공유 경로를 사용하여 캐시가 수행됩니다.</p> <p>Kubernetes 업무 공간에서 캐시를 사용할 수 있도록 설정하려면 영구 볼륨 할당을 제공해야 합니다. 그렇지 않으면 캐시를 사용할 수 없습니다.</p>

파이프라인 업무 공간에서 Kubernetes API 끝점을 사용하는 경우 Code Stream은 CI(지속적 통합) 작업 또는 사용자 지정 작업을 실행하는 데 필요한 Kubernetes 리소스(예: ConfigMap, Secret 및 포트)를 생성합니다. Code Stream은 NodePort를 사용하여 컨테이너와 통신합니다.

파이프라인 실행 간에 데이터를 공유하려면 영구 볼륨 할당을 제공해야 합니다. 그러면 Code Stream은 영구 볼륨 할당을 컨테이너에 마운트하여 데이터를 저장하고 후속 파이프라인 실행에 사용합니다.

## 스마트 파이프라인 템플릿을 사용하기 전에 Code Stream에서 CICD 기본 구축 계획

Code Stream에서 CICD(지속적 통합 및 지속적 전달) 파이프라인을 생성하려는 경우 CICD 스마트 파이프라인 템플릿을 사용할 수 있습니다. CICD 네이티브 빌드를 계획하려면 이 예시 계획에서 파이프라인을 생성하기 전에 스마트 파이프라인 템플릿에 대한 정보를 수집합니다.

CICD 파이프라인을 생성하려면 파이프라인의 CI(지속적 통합) 단계와 CD(지속적 전달) 단계 둘 모두를 계획해야 합니다.

스마트 파이프라인 템플릿에 정보를 입력하고 저장하면 템플릿은 단계와 작업이 포함된 파이프라인을 생성합니다. 또한 선택한 환경 유형(예: **Dev** 및 **Prod**)에 기반하여 이미지의 배포 대상이 표시됩니다. 파이프라인은 컨테이너 이미지를 게시하고, 이것을 실행하는 데 필요한 작업을 수행합니다. 파이프라인 실행 후에는 파이프라인 실행 전반의 추세를 모니터링할 수 있습니다.

파이프라인에 **Docker Hub**의 이미지가 포함된 경우 파이프라인을 실행하기 전에 이미지에 **cURL** 또는 **wget**이 내장되어 있는지 확인해야 합니다. 파이프라인이 실행되면 **Code Stream**은 **cURL** 또는 **wget**을 사용하여 명령을 실행하는 이진 파일을 다운로드합니다.

업무 공간 구성에 대한 자세한 내용은 **파이프라인 작업 공간 구성**에서 참조하십시오.

## CI(지속적 통합) 단계 계획

파이프라인의 CI 단계를 계획하려면 외부 및 내부 요구 사항을 설정하고, 스마트 파이프라인 템플릿의 CI 부분에 필요한 정보를 결정합니다. 다음은 요약 정보입니다.

이 예에서는 **Docker** 업무 공간을 사용합니다.

필요한 끝점 및 저장소:

- 개발자가 코드를 체크인하는 **Git** 소스 코드 저장소. 개발자가 변경 내용을 커밋하면 **Code Stream**은 가장 최신 코드를 파이프라인으로 풀합니다.
- 개발자 소스 코드가 상주하는 저장소에 대한 **Git** 끝점
- 컨테이너 내부의 구축 명령을 실행할 **Docker** 구축 호스트에 대한 **Docker** 끝점
- **Code Stream**이 이미지를 **Kubernetes** 클러스터에 배포하는 데 필요한 **Kubernetes** 끝점
- 지속적 통합 테스트가 실행되는 컨테이너를 생성하는 빌더 이미지
- **Docker** 구축 호스트가 빌더 이미지를 풀하는 데 필요한 이미지 레지스트리 끝점

프로젝트에 대한 액세스 권한이 필요합니다. 프로젝트는 파이프라인, 끝점, 대시보드를 포함하여 모든 작업을 그룹화합니다. **Code Stream**에서 프로젝트의 멤버인지 확인합니다. 그렇지 않으면 프로젝트 멤버로 추가해 달라고 **Code Stream** 관리자에게 요청합니다. **Code Stream**에서 프로젝트를 추가하는 방법의 내용을 참조하십시오.

개발자가 코드 변경 내용을 커밋할 때 **Code Stream**이 **Git** 트리거를 사용하여 파이프라인을 트리거할 수 있도록 하는 **Git Webhook**가 필요합니다. **Code Stream**에서 **Git** 트리거를 사용하여 파이프라인을 실행하는 방법의 내용을 참조하십시오.

구축 도구 집합

- 구축 유형(예: **Maven**)
- 사용하는 모든 처리 후 구축 도구(예: **JUnit**, **JaCoCo**, **Checkstyle**, **FindBugs** 등)

게시 도구

- 구축 컨테이너를 배포할 도구(예: **Docker**)
- 이미지 태그(커밋 ID 또는 빌드 번호)



## 구축 작업 공간

### ■ Docker 끝점인 Docker 구축 호스트

- 이미지 레지스트리. 파이프라인의 CI 부분은 선택된 레지스트리 끝점에서 이미지를 풀합니다. 컨테이너는 CI 작업을 실행하고 이미지를 배포합니다. 레지스트리에 자격 증명이 필요한 경우에는 레지스트리 끝점을 생성한 후, 호스트가 레지스트리에서 이미지를 끌어올 수 있도록 여기에서 선택해야 합니다.

- 지속적 통합 작업이 실행되는 컨테이너를 생성하는 빌더 이미지의 URL

## CD(지속적 전달) 단계 계획

파이프라인의 CD 단계를 계획하려면 외부 및 내부 요구 사항을 설정하고, 스마트 파이프라인 템플릿의 CD 부분에 입력할 정보를 결정합니다.

필요한 끝점:

- Code Stream이 이미지를 Kubernetes 클러스터에 배포하는 데 필요한 Kubernetes 끝점

환경 유형 및 파일

- Code Stream이 애플리케이션을 배포할 모든 환경 유형(예: Dev 및 Prod). 스마트 파이프라인 템플릿은 사용자가 선택하는 환경 유형에 기반하여 파이프라인에 단계와 작업을 생성합니다.

**표 4-2. CICD 스마트 파이프라인 템플릿이 생성하는 파이프라인 단계**

파이프라인 콘텐츠	수행 작업
구축-게시 단계	코드를 구축 및 테스트하고, 빌더 이미지를 생성하고, Docker 호스트에 이미지를 게시합니다.
개발 단계	개발 AWS(Amazon Web Services) 클러스터를 사용하여 이미지를 생성하고 배포합니다. 이 단계에서는 클러스터에 네임스페이스를 생성하고 비밀 키를 생성할 수 있습니다.
운영 단계	VMware Tanzu Kubernetes Grid Integrated Edition(이전 이름: VMware Enterprise PKS)의 운영 버전을 사용하여 이미지를 운영 Kubernetes 클러스터에 배포합니다.

- CICD 스마트 파이프라인 템플릿의 CD 섹션에서 선택하는 Kubernetes YAML 파일.

Kubernetes YAML 파일에는 네임스페이스, 서비스 및 배포에 대한 필수 섹션 3개와 비밀에 대한 선택적 섹션 1개가 포함되어 있습니다. 개인 소유 저장소에서 이미지를 다운로드하여 파이프라인을 생성하려는 경우 Docker 구성 비밀이 있는 섹션이 포함되어야 합니다. 생성하는 파이프라인이 공개적으로 사용 가능한 이미지만 사용하는 경우에는 비밀이 필요하지 않습니다. 다음 샘플 YAML 파일에는 4개의 섹션이 포함되어 있습니다.

```
apiVersion: v1
kind: Namespace
metadata:
  name: codestream
  namespace: codestream
---
apiVersion: v1
data:
  .dockerconfigjson:
    eyJhdXRob3RocmVhbnQ6eyJ0dHRwciovL2luZ1234567890lci5pbj92MS8iOmsidXNlcm5hbWUiOiJhdXRvbWF0aW9uYmV0YSI
```

```

sInBhc3N3b3JkIjoiVk13YXJlQDEyMyIsImVtYWlsIjoiYXV0b21hdGlvbmJldGF1c2VyQGdtYWlsLmNvbSIsImF1dG
giOiJZWfYwYjIxaGRHbHZibUpsZEdFNlZrMTNZWEpsUURFeU13PT0ifX19
kind: Secret
metadata:
  name: dockerhub-secret
  namespace: codestream
type: kubernetes.io/dockerconfigjson
---
apiVersion: v1
kind: Service
metadata:
  name: codestream-demo
  namespace: codestream
  labels:
    app: codestream-demo
spec:
  ports:
    - port: 80
  selector:
    app: codestream-demo
    tier: frontend
  type: LoadBalancer
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: codestream-demo
  namespace: codestream
  labels:
    app: codestream-demo
spec:
  replicas: 10
  selector:
    matchLabels:
      app: codestream-demo
      tier: frontend
  template:
    metadata:
      labels:
        app: codestream-demo
        tier: frontend
    spec:
      containers:
        - name: codestream-demo
          image: automationbeta/codestream-demo:01
          ports:

```

```
- containerPort: 80
  name: codestream-demo
imagePullSecrets:
- name: dockerhub-secret
```

**참고** Kubernetes YAML 파일은 다음 사용 사례 예와 같이 CD 스마트 파이프라인 템플릿에서도 사용 됩니다.

- Code Stream에서 애플리케이션을 블루-그린 배포에 배포하는 방법
- Code Stream에서 배포를 롤백하는 방법
- Code Stream에서 Docker 트리거를 사용하여 지속적 전달 파이프라인을 실행하는 방법

스마트 템플릿에 파일을 적용하려면 **선택**을 클릭하고 Kubernetes YAML 파일을 선택합니다. 그런 다음 **프로세스**를 클릭합니다. 사용 가능한 서비스와 배포 환경이 스마트 파이프라인 템플릿에 표시됩니다. 서비스, 클러스터 끝점 그리고 배포 전략을 선택합니다. 예를 들어 카나리아 배포 모델을 사용하려면 **카나리아**를 선택하고 배포 단계의 백분율을 입력합니다.

## 스마트 템플릿: CI/CD

2/2 단계

환경 ① \* ☒ 개발 ☒ 운영

Kubernetes YAML 파일 \* 선택 프로세스

저러된 파일: codestream.yaml

### 서비스 선택

배포 이름	서비스	네임스페이스	이미지
<span>+</span> codestream-demo	codestream-demo	bgreen1	6

17개 서비스

### 배포

환경	클러스터 끝점	네임스페이스
개발	1030Endpoint-Kubernetes 駱家表术あA中ㄱ궡停ㄱ궡U8aU*n	bgreen1-828706
운영	1030Endpoint-Kubernetes 駱家表术あA中ㄱ궡停ㄱ궡U8aU*n	bgreen1

배포 모델 \* ☒ Canary ☐ 롤링 업그레이드 ☐ 블루-그린

1단계 \* 20 %

롤백 ☐

상태 점검 URL \* /health-check.json

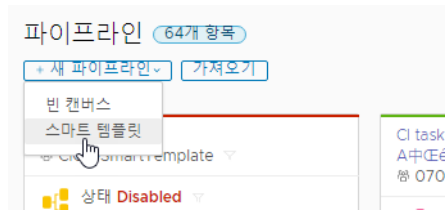
생성
뒤로
취소

스마트 파이프라인 템플릿을 사용하여 블루-그린 배포를 위한 파이프라인을 생성하는 예를 보려면 [Code Stream](#)에서 애플리케이션을 블루-그린 배포에 배포하는 방법의 내용을 참조하십시오.

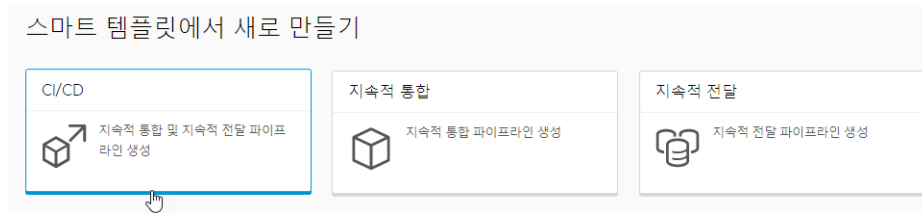
## 스마트 파이프라인 템플릿을 사용하여 CI/CD 파이프라인을 생성하는 방법

모든 정보를 수집하고 필요한 설정을 완료한 후에는 다음과 같은 방법으로 CICD 스마트 파이프라인 템플릿에서 파이프라인을 생성합니다.

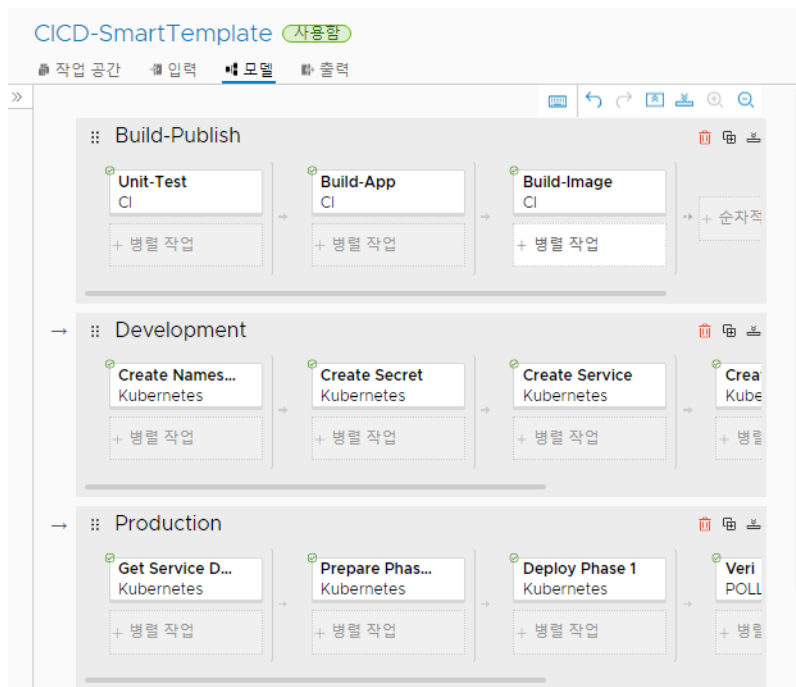
[파이프라인]에서 **새 파이프라인 > 스마트 템플릿**을 선택합니다.



CICD 스마트 파이프라인 템플릿을 선택합니다.



템플릿을 작성하고, 생성되는 단계와 함께 파이프라인을 저장합니다. 최종 변경 사항을 적용해야 하는 경우 파이프라인을 편집한 후 저장할 수 있습니다.



그런 다음 파이프라인을 사용하도록 설정하고 실행합니다. 파이프라인이 실행된 이후 다음의 몇 가지 사항을 확인해야 합니다.

- 파이프라인이 성공했는지 확인합니다. **실행**을 클릭하고 파이프라인을 검색합니다. 실패했다면 오류를 수정하고 다시 실행합니다.
- Git Webhook이 올바르게 작동 중인지 확인합니다. Git **작업** 탭은 이벤트를 표시합니다. **트리거 > Git > 작업**을 클릭합니다.

- 파이프라인 대시보드를 살펴보고 추세를 검토합니다. **대시보드**를 클릭하고 파이프라인 대시보드를 검색합니다. 사용자 지정 대시보드를 생성하여 추가 KPI에 대해 보고할 수도 있습니다.

세부적인 예는 [Code Stream에서 GitHub 또는 GitLab 저장소의 코드를 파이프라인에 지속적으로 통합하는 방법](#) 항목을 참조하십시오.

## 스마트 파이프라인 템플릿을 사용하기 전에 Code Stream에서 지속적 통합 네이티브 빌드 계획

VMware Code Stream에서 CI(지속적인 통합) 파이프라인을 생성하려면 지속적 통합 스마트 파이프라인 템플릿을 사용하면 됩니다. 지속적 통합 네이티브 빌드를 계획하려면 이 예시 계획에서 파이프라인을 생성하기 전에 스마트 파이프라인 템플릿에 대한 정보를 수집합니다.

스마트 파이프라인 템플릿을 채우면 저장소에 지속적 통합 파이프라인이 생성되고 파이프라인이 실행될 수 있도록 작업이 수행됩니다. 파이프라인 실행 후에는 파이프라인 실행 전반의 추세를 모니터링할 수 있습니다.

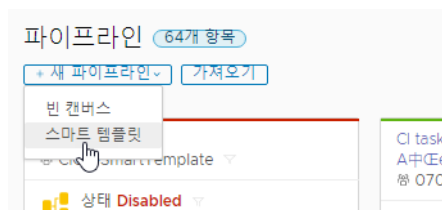
지속적 통합 스마트 파이프라인 템플릿을 사용하기 전에 빌드를 계획하려면:

- 파이프라인, 끝점 및 대시보드를 비롯한 모든 작업을 그룹화할 프로젝트를 식별합니다.
- **스마트 파이프라인 템플릿을 사용하기 전에** [Code Stream에서 CICD 기본 구축 계획](#)의 지속적 전달 부분에 설명된 대로 빌드에 대한 정보를 수집합니다.

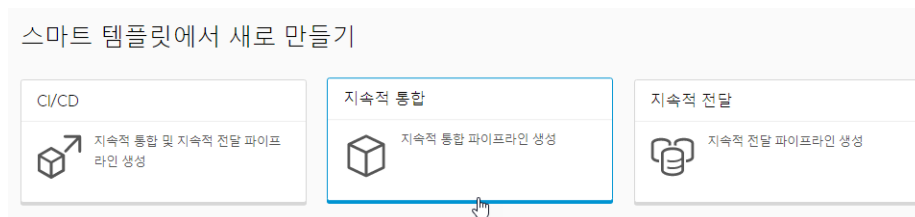
예를 들어, Code Stream에서 컨테이너를 배포할 Kubernetes 끝점을 추가합니다.

그런 다음 지속적 통합 스마트 파이프라인 템플릿을 사용하여 파이프라인을 생성합니다.

[파이프라인]에서 **스마트 템플릿**을 선택합니다.



지속적 통합 스마트 파이프라인 템플릿을 선택합니다.



파이프라인이 생성하는 단계와 함께 파이프라인을 저장하려면 템플릿을 작성하고 파이프라인의 이름을 입력합니다. 파이프라인이 생성하는 단계와 함께 파이프라인을 저장하려면 **생성**을 클릭합니다.

Code Stream 파이프라인 업무 공간은 지속적 통합 작업 및 사용자 지정 작업을 위해 Docker 및 Kubernetes를 지원합니다.

업무 공간 구성에 대한 자세한 내용은 [파이프라인 작업 공간 구성](#)에서 참조하십시오.

최종 변경을 위해 파이프라인을 편집할 수 있습니다. 이제 파이프라인을 사용하도록 설정하고 실행할 수 있습니다. 파이프라인이 실행된 후에는:

- 파이프라인이 성공했는지 확인합니다. **실행**을 클릭하고 파이프라인을 검색합니다. 파이프라인이 실패한 경우 오류를 해결하고 다시 실행합니다.
- Git Webhook가 올바르게 작동하는지 확인합니다. **Git 작업** 탭에 이벤트가 표시됩니다. **트리거 > Git > 작업**을 클릭합니다.
- 파이프라인 대시보드를 살펴보고 추세를 검토합니다. **대시보드**를 클릭하고 파이프라인 대시보드를 검색합니다. 더 많은 주요 성능 지표에 대해 보고하기 위해 사용자 지정 대시보드를 생성할 수 있습니다.

세부적인 예는 [Code Stream](#)에서 [GitHub](#) 또는 [GitLab](#) 저장소의 코드를 파이프라인에 지속적으로 통합하는 방법 항목을 참조하십시오.

## 스마트 파이프라인 템플릿을 사용하기 전에 Code Stream에서 지속적 전달 네이티브 빌드 계획

Code Stream에서 CD(지속적 전달) 파이프라인을 생성하려면 지속적 전달 스마트 파이프라인 템플릿을 사용하면 됩니다. 지속적 전달 네이티브 빌드를 계획하려면 이 예시 계획에서 파이프라인을 생성하기 전에 스마트 파이프라인 템플릿에 대한 정보를 수집합니다.

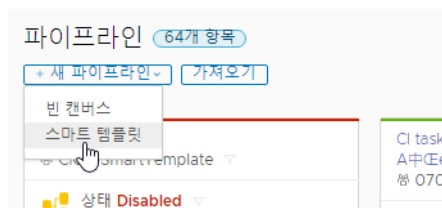
스마트 파이프라인 템플릿을 채우면 저장소에 지속적 전달 파이프라인이 생성되고 파이프라인이 실행될 수 있도록 작업이 수행됩니다. 파이프라인 실행 후에는 파이프라인 실행 전반의 추세를 모니터링할 수 있습니다.

지속적 전달 스마트 파이프라인 템플릿을 사용하기 전에 빌드를 계획하려면:

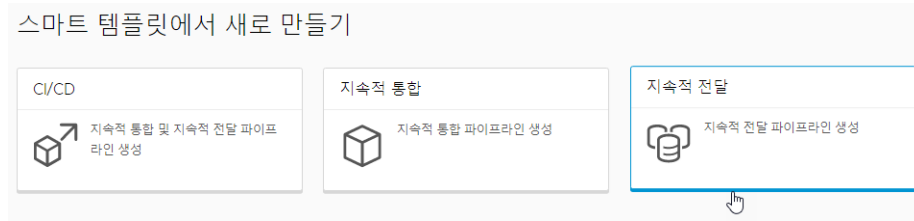
- 파이프라인, 끝점 및 대시보드를 비롯한 모든 작업을 그룹화할 프로젝트를 식별합니다.
- **스마트 파이프라인 템플릿을 사용하기 전에** [Code Stream](#)에서 [CICD 기본 구축 계획](#)의 지속적 전달 부분에 설명된 대로 빌드에 대한 정보를 수집합니다. 예를 들면 다음과 같습니다.
  - Code Stream에서 컨테이너를 배포할 **Kubernetes** 끝점을 추가합니다.
  - 네임스페이스, 서비스 및 배포를 생성하는 **Kubernetes YAML** 파일을 준비합니다. 개인 소유 저장소에서 이미지를 다운로드하려면 **YAML** 파일에 **Docker** 구성 비밀이 있는 섹션이 포함되어야 합니다.

그런 다음 지속적 전달 스마트 파이프라인 템플릿을 사용하여 파이프라인을 생성합니다.

[파이프라인]에서 **스마트 템플릿**을 선택합니다.



지속적 전달 스마트 파이프라인 템플릿을 선택합니다.



템플릿을 채우고 파이프라인의 이름을 입력합니다. 파이프라인이 생성하는 단계와 함께 파이프라인을 저장하려면 **생성**을 클릭합니다.

Code Stream 파이프라인 업무 공간은 지속적 통합 작업 및 사용자 지정 작업을 위해 Docker 및 Kubernetes를 지원합니다.

업무 공간 구성에 대한 자세한 내용은 [파이프라인 작업 공간 구성](#)에서 참조하십시오.

최종 변경을 위해 파이프라인을 편집할 수 있습니다. 이제 파이프라인을 사용하도록 설정하고 실행할 수 있습니다. 파이프라인이 실행된 후에는:

- 파이프라인이 성공했는지 확인합니다. **실행**을 클릭하고 파이프라인을 검색합니다. 실패했다면 오류를 수정하고 다시 실행합니다.
- Git Webhook이 올바르게 작동 중인지 확인합니다. **Git 작업** 탭은 이벤트를 표시합니다. **트리거 > Git > 작업**을 클릭합니다.
- 파이프라인 대시보드를 살펴보고 추세를 검토합니다. **대시보드**를 클릭하고 파이프라인 대시보드를 검색합니다. 더 많은 주요 성능 지표에 대해 보고하기 위해 사용자 지정 대시보드를 생성할 수 있습니다.

세부적인 예는 [Code Stream에서 GitHub 또는 GitLab 저장소의 코드를 파이프라인에 지속적으로 통합하는 방법](#) 항목을 참조하십시오.

## 작업을 수동으로 추가하기 전에 Code Stream에서 CICD 네이티브 빌드 계획

Code Stream에서 CICD(지속적 통합 및 전달) 파이프라인을 생성하기 위해 단계와 작업을 수동으로 추가할 수 있습니다. CICD 기본 구축을 계획하기 위해 필요한 정보를 수집하고 파이프라인을 생성한 다음 여기에 단계와 작업을 수동으로 추가합니다.

파이프라인의 CI(지속적 통합) 단계와 CD(지속적 전달) 단계를 모두 계획해야 합니다. 파이프라인을 생성하고 실행한 후에는 파이프라인 실행 전반의 추세를 모니터링할 수 있습니다.

파이프라인에 Docker Hub의 이미지가 포함된 경우 파이프라인을 실행하기 전에 이미지에 curl 또는 wget이 내장되어 있는지 확인해야 합니다. 파이프라인이 실행되면 Code Stream은 curl 또는 wget을 사용하여 명령을 실행하는 이전 파일을 다운로드합니다.

Code Stream 파이프라인 업무 공간은 지속적 통합 작업 및 사용자 지정 작업을 위해 Docker 및 Kubernetes를 지원합니다.

업무 공간 구성에 대한 자세한 내용은 [파이프라인 작업 공간 구성](#)에서 참조하십시오.

## 외부 및 내부 요구 사항 계획

파이프라인의 CI 및 CD 단계를 계획하기 위해 다음 요구 사항은 파이프라인을 생성하기 전에 수행해야 하는 작업을 나타냅니다.

이 예에서는 Docker 업무 공간을 사용합니다.

이 계획 예에서는 파이프라인을 생성하기 위해 Docker 호스트, Git 저장소, Maven 및 몇 가지 처리 후 구축 툴을 사용합니다.

필요한 끝점 및 저장소:

- 개발자가 코드를 체크인하는 Git 소스 코드 저장소. 개발자가 변경 내용을 커밋하면 Code Stream은 가장 최신 코드를 파이프라인으로 풀합니다.
- 컨테이너 내부의 구축 명령을 실행할 Docker 구축 호스트에 대한 Docker 끝점
- 지속적 통합 테스트가 실행되는 컨테이너를 생성하는 빌더 이미지
- Docker 구축 호스트가 빌더 이미지를 풀하는 데 필요한 이미지 레지스트리 끝점

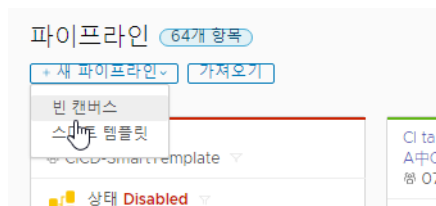
프로젝트에 대한 액세스 권한이 필요합니다. 프로젝트는 파이프라인, 끝점, 대시보드를 포함하여 모든 작업을 그룹화합니다. Code Stream에서 프로젝트의 멤버인지 확인합니다. 그렇지 않으면 프로젝트 멤버로 추가해 달라고 Code Stream 관리자에게 요청합니다. [Code Stream에서 프로젝트를 추가하는 방법](#)의 내용을 참조하십시오.

개발자가 코드 변경 내용을 커밋할 때 Code Stream이 Git 트리거를 사용하여 파이프라인을 트리거할 수 있도록 하는 Git Webhook가 필요합니다. [Code Stream에서 Git 트리거를 사용하여 파이프라인을 실행하는 방법](#)의 내용을 참조하십시오.

## CICD 파이프라인 생성 및 업무 공간 구성 방법

파이프라인을 생성한 다음 업무 공간, 파이프라인 입력 매개 변수 및 작업을 구성해야 합니다.

파이프라인을 생성하려면 **파이프라인 > 새 파이프라인 > 빈 캔버스**를 클릭합니다.



[업무 공간] 탭에서 지속적 통합 정보를 입력합니다.

- Docker 구축 호스트를 포함합니다.
- 빌더 이미지의 URL을 입력합니다.
- 파이프라인에서 이미지를 가져올 수 있도록 이미지 레지스트리 끝점을 선택합니다. 컨테이너에서 CI 작업을 실행하고 이미지를 배포합니다. 레지스트리에 자격 증명이 필요한 경우 먼저 이미지 레지스트리 끝점을 생성한 다음 여기에서 해당 끝점을 선택해야 호스트가 레지스트리에서 이미지를 가져올 수 있습니다.



- 캐시해야 할 아티팩트를 추가합니다. 빌드의 성공을 위해 디렉토리와 같은 아티팩트가 종속적으로 다운로드됩니다. 캐시는 이러한 아티팩트가 상주하는 위치입니다. 예를 들어 종속 아티팩트에는 Maven용 .m2 디렉토리와 Node.js용 node\_modules 디렉토리가 포함될 수 있습니다. 이러한 디렉토리는 파이프라인 실행 전반에 걸쳐 캐시되어 구축 동안 시간을 절약합니다.

Provide details about the container and host for running continuous integration tasks.

Type \*

☒ Docker ☐ Kubernetes

Host endpoint \*

codestream-ci-test

Host endpoint for build location in CI task or for running Custom Integration Task code.

Builder image URL \$ \*

automationbeta/cs-builder:latest

Name and location of the builder image. The CI tasks run on the container that the image creates.

Image registry

Docker Registry

The pipeline pulls the image from the selected registry endpoint. The container runs the CI tasks, and deploys your image. If the host can pull the image from the registry.

Working directory \$

CI pipeline tasks run steps for continuous integration. The working directory is the location where the steps run, and is where ci

Cache \$

[입력] 탭에서 파이프라인 입력 매개 변수를 구성합니다.

- 파이프라인이 Git, Gerrit 또는 Docker 트리거 이벤트의 입력 매개 변수를 사용하는 경우, 자동 삽입 매개 변수에 대한 트리거 유형을 선택합니다. 이벤트에는 Gerrit 또는 Git의 제목 변경 또는 Docker에 대한 이벤트 소유자 이름이 포함될 수 있습니다. 파이프라인이 이벤트에서 전달된 입력 매개 변수를 사용하지 않는 경우 매개 변수 자동 삽입을 **없음**으로 설정해둡니다.
- 값과 설명을 파이프라인 입력 매개 변수에 적용하려면 3개의 세로 점을 클릭하고 **편집**을 클릭합니다. 입력하는 값은 작업, 단계 또는 알림에 대한 입력으로 사용됩니다.
- 파이프라인 입력 매개 변수를 추가하려면 **추가**를 클릭합니다. 예를 들어 모든 실행에 대해 기본값을 표시하기 위해 approvers를 추가할 수 있지만 런타임 시 다른 승인자를 표시하도록 재정의할 수 있습니다.
- 삽입된 매개 변수를 추가하거나 제거하려면 **삽입된 매개 변수 추가/제거**를 클릭합니다. 예를 들어 사용되지 않는 매개 변수를 제거하면 결과 페이지의 복잡성을 줄이고 사용된 입력 매개 변수만 표시할 수 있습니다.

☒ Input
 ☐ Workspace
 ☐ Model
 ☐ Output

### Input Parameters

The input parameters for this pipeline are passed to the pipeline before it runs.

When you add input parameters, and star the most useful or unique input parameter for each pipeline, the parameter appears in locations like the pipeline execution cards. For example, if you include the committer ID (GIT\_COMMIT\_ID) as an input parameter, you can select it as the starred input parameter to identify which developer commits trigger a pipeline execution before the pipeline runs.

**Auto inject parameters**

☐ Gerrit
 ☐ Git
 ☐ Docker
 ☒ None

	Starred	Name	Value	Description
⋮	☆	GIT_BRANCH_NAME		
⋮	☆	GIT_CHANGE_SUBJECT		
⋮	☆	GIT_COMMIT_ID		
⋮	☆	GIT_EVENT_DESCRIPTION		
⋮	☆	GIT_EVENT_OWNER_NAME		
⋮	☆	GIT_EVENT_TIMESTAMP		
⋮	☆	GIT_REPO_NAME		
⋮	☆	GIT_SERVER_URL		

8 items

코드를 테스트하도록 파이프라인을 구성합니다.

- CI 작업을 추가하고 구성합니다.
- 코드에 대해 `mvn test`를 실행하기 위한 단계를 추가합니다.
- 작업이 실행된 후 문제를 식별하려면 JUnit 및 JaCoCo, FindBugs, Checkstyle과 같은 처리 후 구축 도구를 실행합니다.

Task: Unit-Test
 Notifications
Rollback
VALIDATE TASK

Task name \* Unit-Test  
Can contain alphanumeric (a-z, A-Z, 0-9), whitespace, hyphen(-), and underscore(\_) characters. Dot(.) is not allowed.

Type \* CI

Precondition §

Continue on failure ☐

**Continuous Integration**  
A Docker host must be set up to use a CI task in a pipeline. Configure the workspace section.

Steps §

```

1 cd demo-project
2 mvn test
  
```

Preserve artifacts  
Specify the paths of artifact to preserve.

Export  
Enter comma separated values

JUnit

JaCoCo

FindBugs

Checkstyle

코드를 구축하도록 파이프라인을 구성합니다.

- CI 작업을 추가하고 구성합니다.
- 코드에서 `mvn clean install`을 실행하는 단계를 포함합니다.
- 위치 및 JAR 파일 이름을 포함하여 아티팩트가 보존되도록 합니다.

Task Build-App

Task name \* Build-App  
Can contain alphanumeric (a-z, A-Z, 0-9), whitespace, hyphen(-), and underscore(\_) characters. Dot(.) is not allowed.

Type \* CI

Precondition \$

Continue on failure ☐

Continuous Integration  
A Docker host must be set up to use a CI task in a pipeline. Configure the workspace section.

Steps \$

```
1 cd demo-project
2 mvn clean install -DskipTests
```

Preserve artifacts  
Specify the paths of artifact to preserve.

Export  
Enter comma separated values

JUnit  
/demo-project

JaCoCo  
/demo-project

FindBugs  
/demo-project

Checkstyle  
/demo-project

이미지를 Docker 호스트에 게시하도록 파이프라인을 구성합니다.

- CI 작업을 추가하고 구성합니다.
- 이미지 커밋, 내보내기, 구축 및 푸시를 위한 단계를 추가합니다.
- 다음 작업에서 사용하도록 IMAGE의 내보내기 키를 추가합니다.

**Task: Build-Image** Notifications Rollback VALIDATE TASK

**Task name \*** Build-Image  
Can contain alphanumeric (a-z, A-Z, 0-9), whitespace, hyphen(-), and underscore(\_) characters. Dot(.) is not allowed.

**Type \*** CI

**Precondition §**  SYNTAX GUIDE

**Continue on failure** ☐

**Continuous Integration**  
A Docker host must be set up to use a CI task in a pipeline. Configure the workspace section.

**Steps §**

```

1 cd demo-project
2 export IMAGE=automationbeta/demo-cicd-smart-template:{{executionIndex}}
3 export DOCKER_HOST=http://10.211.211.27:4243
4 docker login --username=automationbeta --password=
5 docker build -t $IMAGE --file ./docker/Dockerfile .
6 docker push $IMAGE

```

**Preserve artifacts** +  
Specify the paths of artifact to preserve.

**Export** IMAGE X  
Enter comma separated values

**JUnit** Label Path +

**JaCoCo** Label Path +

**FindBugs** Label Path +

**Checkstyle** Label

업무 공간, 입력 매개 변수, 테스트 작업 및 구축 작업을 구성했다면 파이프라인을 저장합니다.

## 파이프라인을 사용하도록 설정하고 실행하는 방법

단계 및 작업으로 파이프라인을 구성한 후에는 파이프라인을 저장하고 사용하도록 설정할 수 있습니다.

그런 다음 파이프라인이 실행되어 완료될 때까지 기다린 후 파이프라인이 성공했는지 확인합니다. 실패했다면 오류를 수정하고 다시 실행합니다.

파이프라인이 성공했다면 다음과 같은 내용을 확인해야 합니다.

- 파이프라인 실행을 검토하고 작업 단계의 결과를 봅니다.
- 파이프라인 실행의 업무 공간에서, 컨테이너 및 복제된 Git 저장소에 대한 세부 정보를 찾습니다.
- 업무 공간에서, 처리 후 도구의 결과를 살펴보고 오류, 코드 범위, 버그 및 스타일 문제를 확인합니다.
- 아티팩트가 보존되었는지 확인합니다. 또한 이미지가 **IMAGE** 이름 및 값으로 내보내졌는지 확인합니다.
- Docker 저장소로 이동하여 파이프라인에서 컨테이너를 게시했는지 확인합니다.

Code Stream이 코드를 지속적으로 통합하는 방법을 보여주는 자세한 예를 보려면 [Code Stream에서 GitHub 또는 GitLab 저장소의 코드를 파이프라인에 지속적으로 통합하는 방법](#) 항목을 참조하십시오.

## Code Stream에서 롤백 계획

파이프라인 실행이 실패하면 롤백을 사용하여 이전의 안정적인 상태로 환경을 되돌릴 수 있습니다. 롤백을 사용하려면 롤백 흐름을 계획하고 구현 방법을 이해해야 합니다.

롤백 흐름은 배포 실패를 되돌리는 데 필요한 단계를 규정합니다. 흐름은 실행 및 실패한 배포 유형에 따라 달라지는 하나 이상의 순차적 작업이 포함된 롤백 파이프라인의 형식으로 지정됩니다. 예를 들어 기존 애플리케이션의 배포 및 롤백은 컨테이너 애플리케이션의 배포 및 롤백과 다릅니다.

배포를 양호한 상태로 되돌리기 위해, 롤백 파이프라인에는 보통 다음을 위한 작업이 포함되어 있습니다.

- 상태 또는 환경을 정리합니다.
- 사용자 지정 스크립트를 실행하여 변경 내용을 되돌립니다.
- 배포의 이전 개정을 배포합니다.

기존 배포 파이프라인에 롤백을 추가하려면 배포 파이프라인을 실행하기 전에 롤백하려는 배포 파이프라인의 작업 또는 단계에 롤백 파이프라인을 연결합니다.

### 롤백을 구성하는 방법

배포에서 롤백을 구성하려면 다음을 수행해야 합니다.

- 배포 파이프라인을 생성합니다.
- 롤백 파이프라인을 연결하기 위해, 배포 파이프라인에서 롤백을 트리거하는 잠재적 실패 지점을 식별합니다. 예를 들어 배포 파이프라인에서 이전 작업이 성공적으로 완료되었는지 확인하는 조건 또는 폴링 작업 유형에 롤백 파이프라인을 연결할 수 있습니다. 조건 작업에 대한 자세한 내용은 [Code Stream에서 파이프라인의 실행 또는 중지를 위해 조건 작업에서 변수 바인딩을 사용하는 방법](#) 항목을 참조하십시오.
- 작업 또는 단계 실패와 같이 롤백 파이프라인을 트리거할 실패의 범위를 결정합니다. 롤백을 단계에 연결할 수도 있습니다.
- 실패 시 실행할 롤백 작업을 결정합니다. 이러한 작업으로 롤백 파이프라인을 생성합니다.

롤백 파이프라인을 수동으로 생성하거나 **Code Stream**에서 생성할 수 있습니다.

- 빈 캔버스를 사용하면 기존 배포 파이프라인에 평행한 흐름을 따르는 롤백 파이프라인을 수동으로 생성할 수 있습니다. 그런 다음 배포 파이프라인에서 실패 시 롤백을 트리거하는 하나 이상의 작업에 롤백 파이프라인을 연결합니다.
- 스마트 파이프라인 템플릿을 사용하여 롤백 작업으로 배포 파이프라인을 구성할 수 있습니다. 그러면 **Code Stream**에서 실패 시 배포를 롤백하는 미리 정의된 작업으로 하나 이상의 기본 롤백 파이프라인이 자동으로 생성됩니다.

스마트 파이프라인 템플릿을 사용하여 롤백이 포함된 CD 파이프라인을 구성하는 방법에 대한 자세한 예는 [Code Stream에서 배포를 롤백하는 방법](#)의 내용을 참조하십시오.

## 배포 파이프라인에 롤백이 포함된 작업 또는 단계가 여러 개 있을 때 발생하는 문제

롤백이 추가된 작업이나 단계가 여러 개 있는 경우 롤백 순서가 달라질 수 있습니다.

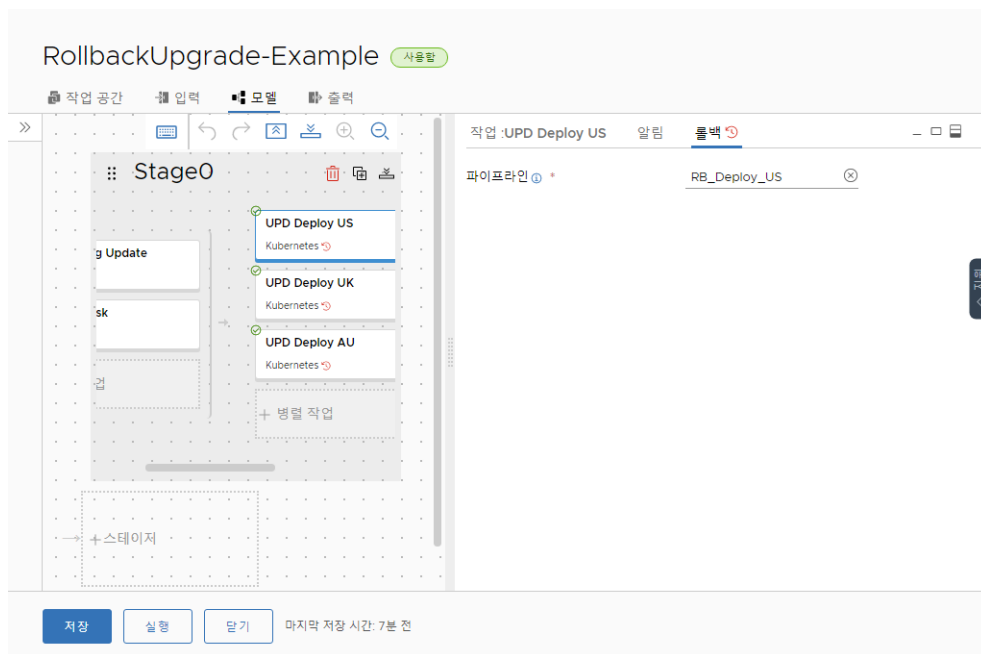
표 4-3. 롤백 순서 결정

롤백 추가 위치...	롤백 발생 시기...
병렬 작업	병렬 작업 중 하나가 실패하면 모든 병렬 작업이 완료 또는 실패한 후 해당 작업에 대한 롤백이 실행됩니다. 롤백은 작업 실패 직후에 실행되지 않습니다.
단계 및 단계 내의 작업	작업이 실패하면 작업 롤백이 실행됩니다. 작업이 병렬 작업 그룹에 있는 경우 모든 병렬 작업이 완료 또는 실패한 후 작업 롤백이 실행됩니다. 작업 롤백이 완료되거나 완료에 실패한 후에 단계 롤백이 실행됩니다.

다음은 포함하는 파이프라인을 고려합니다.

- 롤백이 있는 운영 단계.
- 각 작업에 고유한 롤백이 있는 병렬 작업 그룹.

이름이 **UPD Deploy US**인 작업에 롤백 파이프라인 **RB\_Deploy\_US**가 있습니다. **UPD Deploy US**가 실패하면 롤백이 **RB\_Deploy\_US** 파이프라인에 정의된 흐름을 따릅니다.



**UPD Deploy US**가 실패하면 **UPD Deploy UK** 및 **UPD Deploy AU**가 모두 완료 또는 실패한 후 **RB\_Deploy\_US** 파이프라인이 실행됩니다. 롤백은 **UPD Deploy US** 실패 직후에 실행되지 않습니다. 또한 운영 단계에도 롤백이 있으므로 **RB\_Deploy\_US** 파이프라인 실행 후에 단계 롤백 파이프라인이 실행됩니다.

# Code Stream 사용을 위한 자습서

# 5

Code Stream은 DevOps 릴리스 수명 주기를 모델링 및 지원하고 애플리케이션을 지속적으로 테스트하고 개발 환경 및 운영 환경에 릴리스합니다.

필요한 모든 항목을 이미 설정했으면 Code Stream을 사용할 수 있습니다. [장 2 Code Stream을 설정하여 릴리스 프로세스 모델링](#) 항목을 참조하십시오.

이제 운영 환경에 릴리스하기 전에 개발자 코드의 빌드 및 테스트를 자동화하는 파이프라인을 생성할 수 있습니다. Code Stream에서는 컨테이너 기반 또는 기존 애플리케이션을 배포할 수 있습니다.

**표 5-1. DevOps 수명주기에서 Code Stream 사용**

기능	수행할 수 있는 작업의 예
Code Stream에서 기본 구축 기능을 사용합니다.	코드를 지속적으로 통합하고, 컨테이너화하고, 전달하는 CICD(지속적 통합 및 전달), CI(지속적 통합) 및 CD(지속적 전달) 파이프라인을 생성합니다. <ul style="list-style-type: none"><li>■ 파이프라인을 생성해주는 스마트 파이프라인 템플릿을 사용합니다.</li><li>■ 파이프라인에 단계 및 작업을 수동으로 추가합니다.</li></ul>
애플리케이션을 릴리스하고 릴리스를 자동화합니다.	다양한 방식으로 애플리케이션을 통합하고 릴리스합니다. <ul style="list-style-type: none"><li>■ GitHub 또는 GitLab 저장소의 코드를 파이프라인에 지속적으로 통합합니다.</li><li>■ 블로그 문서 <a href="#">vRealize Automation Code Stream용 Docker 호스트</a> 생성의 설명대로 Docker 호스트를 통합하여 지속적 통합 작업을 실행합니다.</li><li>■ YAML 클라우드 템플릿을 사용하여 애플리케이션 배포를 자동화합니다.</li><li>■ Kubernetes 클러스터에 대한 애플리케이션 배포를 자동화합니다.</li><li>■ 애플리케이션을 블루-그린 배포에 릴리스합니다.</li><li>■ Code Stream을 고유한 구축, 테스트 및 배포 톨과 통합합니다.</li><li>■ Code Stream을 다른 애플리케이션과 통합하는 REST API를 사용합니다.</li></ul>
추세, 메트릭 및 KPI(주요 성능 지표)를 추적합니다.	사용자 지정 대시보드를 생성하고 파이프라인의 성능에 대한 인사이트를 얻습니다.
문제를 해결합니다.	파이프라인 실행이 실패하면 Code Stream에서 Jira 티켓을 생성하도록 합니다.

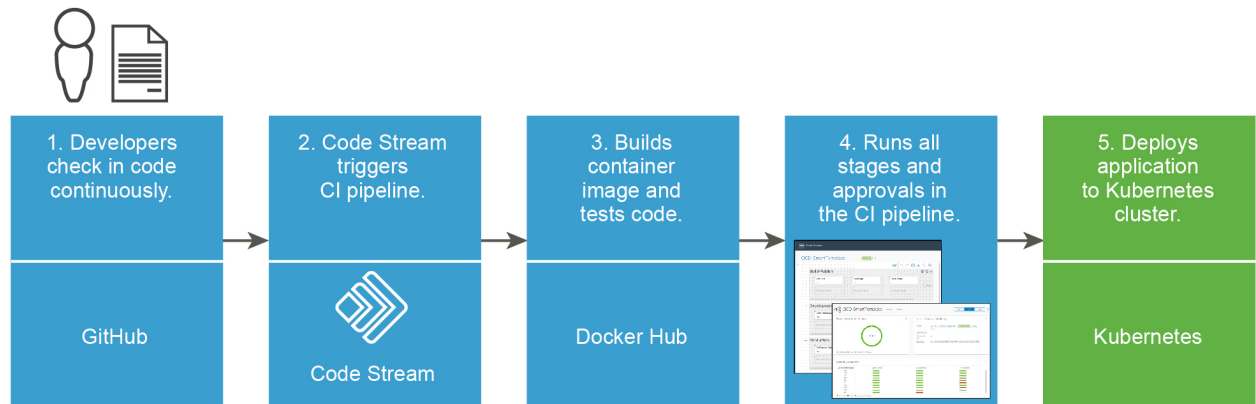
본 장은 다음 항목을 포함합니다.

- Code Stream에서 GitHub 또는 GitLab 저장소의 코드를 파이프라인에 지속적으로 통합하는 방법
- Code Stream의 YAML 클라우드 템플릿에서 배포하는 애플리케이션의 릴리스를 자동화하는 방법
- Code Stream에서 Kubernetes 클러스터에 애플리케이션의 릴리스를 자동화하는 방법

- Code Stream에서 애플리케이션을 블루-그린 배포에 배포하는 방법
- 고유한 구축, 테스트 및 배포 톨과 Code Stream을 통합하는 방법
- 다음 작업에서 클라우드 템플릿 작업의 리소스 속성을 사용하는 방법
- Code Stream을 다른 애플리케이션과 통합하기 위해 REST API를 사용하는 방법
- Code Stream에서 파이프라인을 코드로 활용하는 방법

## Code Stream에서 GitHub 또는 GitLab 저장소의 코드를 파이프라인에 지속적으로 통합하는 방법

개발자는 GitHub 저장소 또는 GitLab Enterprise 저장소의 코드를 지속적으로 통합할 수 있습니다. 개발자가 코드를 업데이트하고 저장소 변경 내용을 커밋할 때마다 Code Stream은 그러한 변경 내용을 수신하고 파이프라인을 트리거할 수 있습니다.



Code Stream이 코드 변경 시 파이프라인을 트리거하도록 하려면 **Git** 트리거를 사용합니다. 그러면 Code Stream은 코드에 대한 변경 내용을 커밋할 때마다 파이프라인을 트리거합니다.

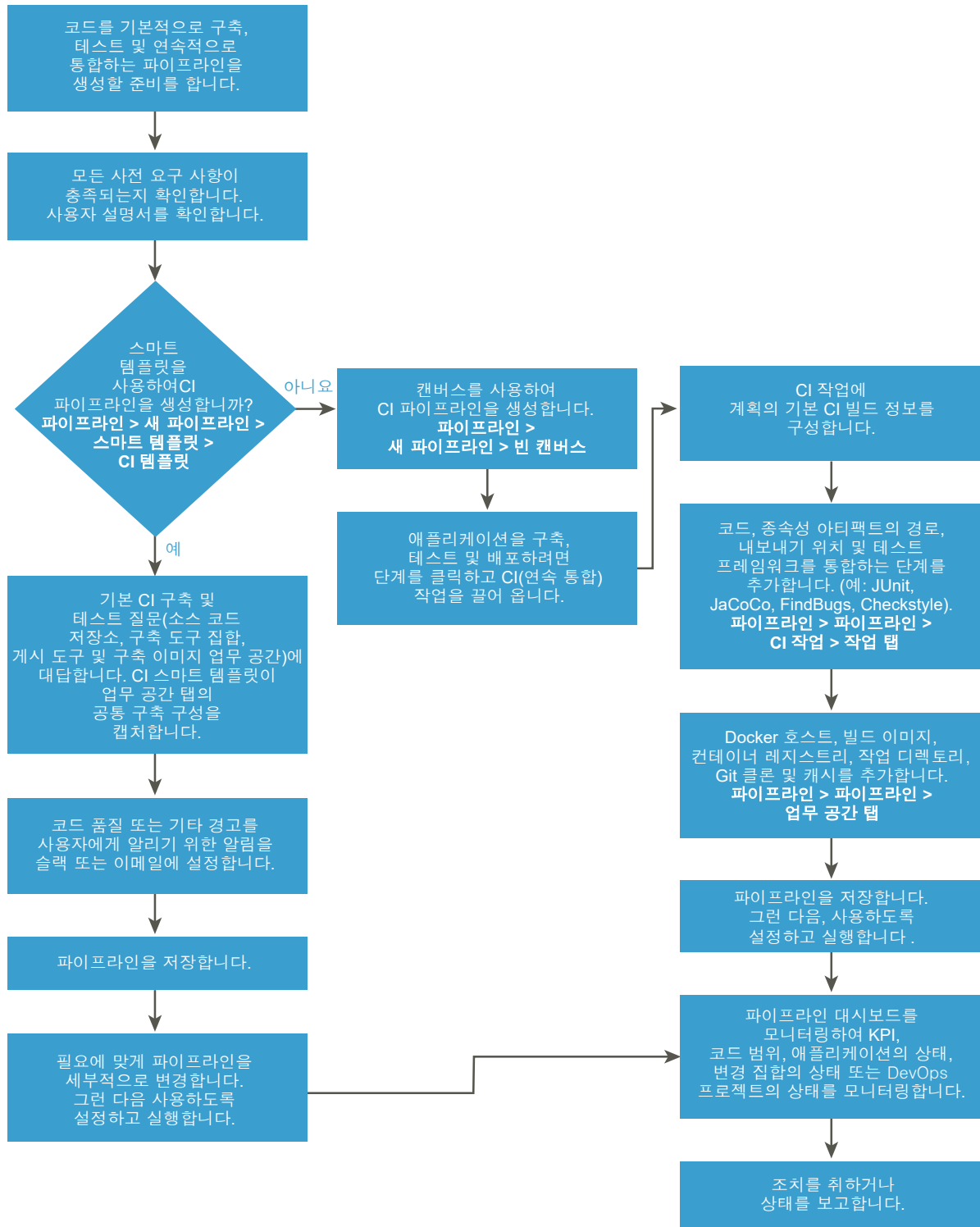
Code Stream 파이프라인 업무 공간은 지속적 통합 작업 및 사용자 지정 작업을 위해 **Docker** 및 **Kubernetes**를 지원합니다.

업무 공간 구성에 대한 자세한 내용은 [파이프라인 작업 공간 구성](#)에서 참조하십시오.

다음 순서도는 스마트 파이프라인 템플릿을 사용하여 파이프라인을 생성하거나 수동으로 파이프라인을 구축하는 경우 수행할 수 있는 워크플로를 보여줍니다.



그림 5-1. 스마트 파이프라인 템플릿을 사용하거나 수동으로 파이프라인을 생성하는 워크플로



다음 예에서는 Docker 업무 공간을 사용합니다.

코드를 구축하기 위해 **Docker** 호스트를 사용합니다. 단위 테스트 및 코드 검사를 실행하는 테스트 프레임워크 툴로 **JUnit** 및 **JaCoCo**를 사용합니다. 이러한 툴을 파이프라인에 포함합니다.

그러면 코드를 구축 및 테스트하고 **AWS**에 있는 프로젝트 팀 **Kubernetes** 클러스터에 배포하는 지속적 통합 파이프라인을 생성하는 지속적 통합 스마트 파이프라인 템플릿을 사용할 수 있습니다. 지속적 통합 작업에 대한 코드 종속성 아티팩트를 저장하여 코드 구축 시 시간을 절약하려면 캐시를 사용하면 됩니다.

코드를 구축하고 테스트하는 파이프라인 작업에서, 몇 가지의 지속적 통합 단계를 포함할 수 있습니다. 이러한 단계는 파이프라인이 트리거될 때 **Code Stream**이 소스 코드를 복제하는 동일한 작업 디렉토리에 상주할 수 있습니다.

코드를 **Kubernetes** 클러스터에 배포하기 위해 파이프라인에서 **Kubernetes** 작업을 사용할 수 있습니다. 그런 다음 파이프라인을 사용하도록 설정하고 실행해야 합니다. 그런 다음 저장소에서 코드를 변경하고 파이프라인 트리거를 확인합니다. 파이프라인을 실행한 후 파이프라인 추세를 모니터링하고 보고하려면 대시보드를 사용합니다.

다음 예에서는 코드를 파이프라인에 지속적으로 통합하는 지속적 통합 파이프라인을 생성하기 위해 지속적 통합 스마트 파이프라인 템플릿을 사용합니다. 이 예에서는 **Docker** 업무 공간을 사용합니다.

필요한 경우 파이프라인을 수동으로 생성하고 여기에 단계 및 작업을 추가할 수 있습니다. 지속적인 통합 구축을 계획하고 파이프라인을 수동으로 생성하는 방법에 대한 자세한 내용은 [작업을 수동으로 추가하기 전에 Code Stream에서 CI/CD 네이티브 빌드 계획](#) 항목을 참조하십시오.

#### 사전 요구 사항

- 지속적인 통합 구축을 계획합니다. [스마트 파이프라인 템플릿을 사용하기 전에 Code Stream에서 지속적 통합 네이티브 빌드 계획](#) 항목을 참조하십시오.
- **GitLab** 소스 코드 저장소가 있는지 확인합니다. 도움이 필요하면 **Code Stream** 관리자에게 문의하십시오.
- **Git** 끝점을 추가합니다. 예를 보려면 [Code Stream에서 Git 트리거를 사용하여 파이프라인을 실행하는 방법](#) 항목을 참조하십시오.
- **Code Stream**이 **GitHub** 저장소나 **GitLab** 저장소의 변경 내용을 수신 대기하다가 변경이 발생할 때 파이프라인을 트리거하도록 하려면 **Webhook**을 추가합니다. 예를 보려면 [Code Stream에서 Git 트리거를 사용하여 파이프라인을 실행하는 방법](#) 항목을 참조하십시오.
- 여러 지속적 통합 작업에서 사용할 수 있는 지속적 통합 작업에 대한 컨테이너가 생성되는 **Docker** 호스트 끝점을 추가합니다. 끝점에 대한 자세한 내용은 [Code Stream의 끝점 소개](#) 항목을 참조하십시오.
- 이미지 URL, 빌드 호스트 및 빌드 이미지의 URL을 확보합니다. 도움이 필요하면 **Code Stream** 관리자에게 문의하십시오.
- 테스트 프레임워크 툴로 **JUnit** 및 **JaCoCo**를 사용 중인지 확인합니다.
- 지속적 통합 구축을 위한 외부 인스턴스(**Jenkins**, **TFS** 또는 **Bamboo**)를 설정합니다. **Kubernetes** 플러그인이 코드를 배포합니다. 도움이 필요하면 **Code Stream** 관리자에게 문의하십시오.

## 절차

- 1 사전 요구 사항을 따릅니다.
- 2 스마트 파이프라인 템플릿을 사용하여 파이프라인을 생성하려면 지속적인 통합 스마트 파이프라인 템플릿을 열고 양식을 채웁니다.
  - a **파이프라인 > 새 파이프라인 > 스마트 템플릿 > 지속적 통합**을 클릭합니다.
  - b 소스 코드 저장소, 구축 툴셋, 게시 툴 및 구축 이미지 업무 공간에 대한 템플릿의 질문에 답합니다.
  - c 팀에 대한 Slack 알림 또는 이메일 알림을 추가합니다.
  - d 스마트 파이프라인 템플릿이 파이프라인을 생성하도록 하려면 **생성**을 클릭합니다.
  - e 파이프라인을 추가로 변경하려면 **편집**을 클릭하고 내용을 변경한 다음 **저장**을 클릭합니다.
  - f 파이프라인을 사용하도록 설정하고 실행합니다.
- 3 파이프라인을 수동으로 생성하려면 캔버스에 단계 및 작업을 추가하고 지속적 통합 작업에 지속적 통합 기본 빌드 정보를 포함합니다.
  - a **파이프라인 > 새 파이프라인 > 빈 캔버스**를 클릭합니다.
  - b 단계를 클릭한 다음 몇 가지 지속적 통합 작업을 탐색 창에서 단계로 끌어옵니다.
  - c 지속적 통합 작업을 구성하려면 클릭하고 **작업** 탭을 클릭합니다.
  - d 코드를 지속적으로 통합하는 단계를 추가합니다.
  - e 종속성 아티팩트 경로를 포함합니다.
  - f 내보내기 위치를 추가합니다.
  - g 사용할 테스트 프레임워크 툴을 추가합니다.
  - h Docker 호스트 및 구축 이미지를 추가합니다.
  - i 컨테이너 레지스트리, 작업 디렉토리 및 캐시를 추가합니다.
  - j 파이프라인을 저장한 다음 사용하도록 설정합니다.
- 4 GitHub 저장소 또는 GitLab 저장소에서 코드를 변경합니다.  
Git 트리거는 실행을 시작하는 파이프라인을 활성화합니다.
- 5 코드 변경으로 인한 파이프라인 트리거를 확인하려면 **트리거 > Git > 작업**을 클릭합니다.

- 6 파이프라인의 실행을 보려면 **실행**을 클릭하고 단계에서 빌드 이미지를 생성하고 내보냈는지 확인합니다.

The screenshot shows the vRealize Automation Code Stream interface. On the left is a sidebar with navigation options: Dashboards, Executions, User Operations, Pipelines, Manage (with sub-options: Endpoints, Variables, Triggers, Gerrit, Git), and a search bar. The main area displays the execution details for a pipeline named 'CICD-SmartTemplate #51'. The pipeline is in a 'COMPLETED' state. The 'Build-Publish' stage is active, showing a 'Build-Image' task. The task status is 'COMPLETED' with a message 'Execution Completed.' and a duration of '5s (09/11/2018 7:16 AM - 09/11/2018 7:16 AM)'. The 'Result' section shows the steps executed successfully, including setting environment variables, logging into Docker, and building the image. The 'Preserved Artifacts' section shows the path to the built image: '/sharedPath/pipelines/CICD-SmartTemplate/51/Build-Publish Build-Image/artifacts/'.

- 7 KPI 및 추세를 추적할 수 있도록 파이프라인 대시보드를 모니터링하려면 **대시보드 > 파이프라인 대시보드**를 클릭합니다.

## 결과

축하합니다! GitHub 저장소 또는 GitLab 저장소의 코드를 파이프라인에 지속적으로 통합하고 구축 이미지를 배포하는 파이프라인을 생성했습니다.

## 다음에 수행할 작업

자세한 내용을 보려면 [Code Stream 관리자 및 개발자를 위한 추가 리소스](#) 항목을 참조하십시오.

# Code Stream의 YAML 클라우드 템플릿에서 배포하는 애플리케이션의 릴리스를 자동화하는 방법

개발자는 변경을 커밋할 때마다 온-프레미스 GitHub 인스턴스에서 자동화 클라우드 템플릿을 가져오는 파이프라인이 필요합니다. WordPress 애플리케이션을 AWS(Amazon Web Services) EC2 또는 데이터

센터에 배포하는 파이프라인이 필요합니다. Code Stream은 파이프라인에서 클라우드 템플릿을 호출하고 이 클라우드 템플릿의 CICD(지속적 통합 및 전달)를 자동화하여 애플리케이션을 배포합니다.

파이프라인을 생성하고 트리거하려면 VMware 클라우드 템플릿이 필요합니다.

Code Stream 클라우드 템플릿 작업의 **클라우드 템플릿 소스**에 대해 다음을 선택할 수 있습니다.

- **Cloud Assembly** 템플릿을 소스 제어로 선택할 수 있습니다. 이 경우 GitLab 또는 GitHub 저장소는 필요하지 않습니다.
- 소스 제어로 GitLab 또는 GitHub를 사용하는 경우 **소스 제어**를 선택합니다. 이 경우 Git Webhook이 있어야 하며 Webhook을 통해 파이프라인을 트리거해야 합니다.

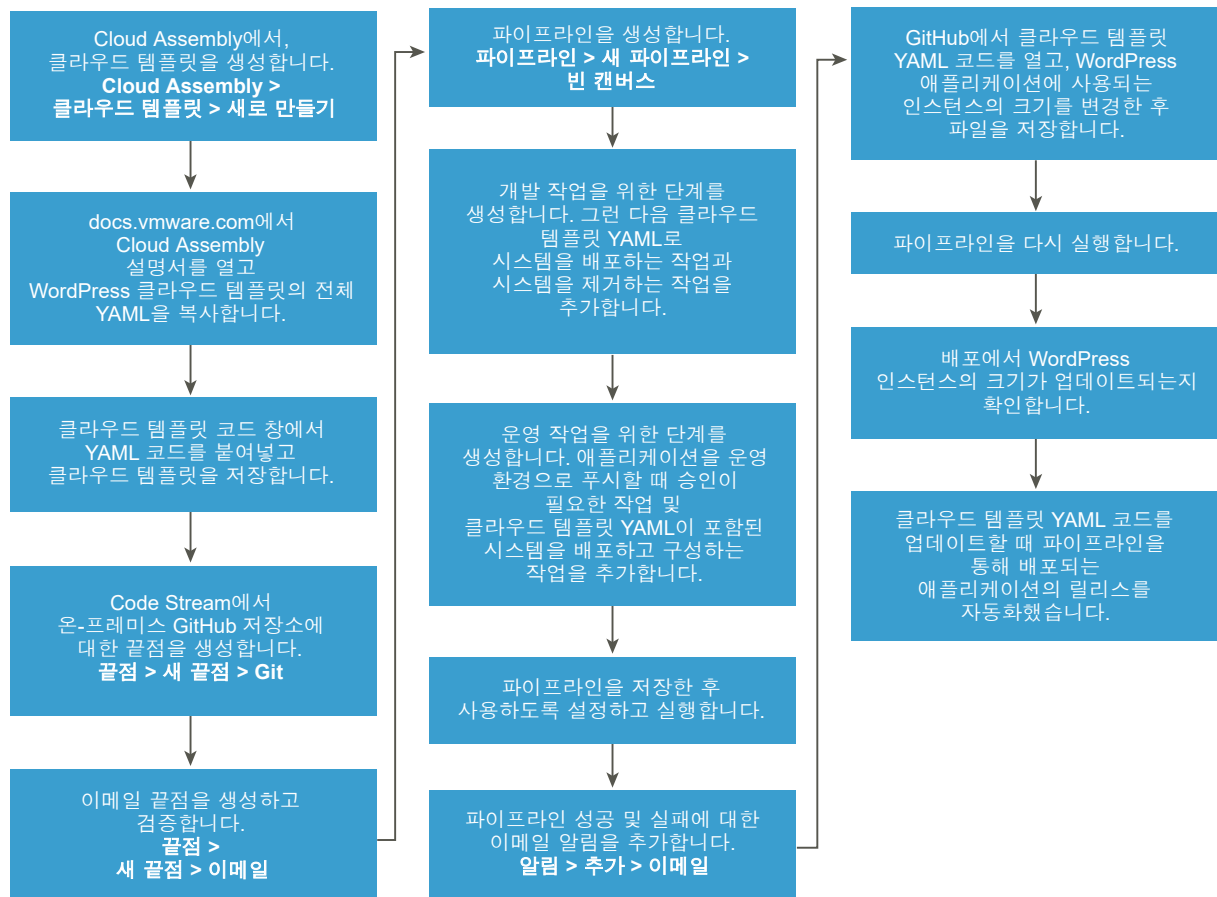
GitHub 저장소에 YAML 클라우드 템플릿이 있고 파이프라인에서 클라우드 템플릿을 사용하려는 경우 다음 작업을 수행해야 합니다.

- 1 Cloud Assembly에서 클라우드 템플릿을 GitHub 저장소로 푸시합니다.
- 2 Code Stream에서 Git 끝점을 생성합니다. 그런 다음 Git 끝점과 파이프라인을 사용하는 Git Webhook을 생성합니다.
- 3 파이프라인을 트리거하려면 GitHub 저장소에 있는 모든 파일을 업데이트하고 변경을 커밋합니다.

GitHub 저장소에 YAML 클라우드 템플릿이 없고 소스 제어의 클라우드 템플릿을 사용하려는 경우 이 절차를 사용하면 그 방법을 알 수 있습니다. 이 절차는 WordPress 애플리케이션에 대한 클라우드 템플릿을 생성하고 온-프레미스 GitHub 저장소에서 이 클라우드 템플릿을 트리거하는 방법을 보여줍니다. YAML 클라우드 템플릿을 변경할 때마다 파이프라인이 애플리케이션의 릴리스를 트리거하고 자동화합니다.

- Cloud Assembly에서 클라우드 계정을 추가하고, 클라우드 영역을 추가하고, 클라우드 템플릿을 생성합니다.
- Code Stream에서 클라우드 템플릿을 호스팅하는 온-프레미스 GitHub 저장소에 대한 끝점을 추가합니다. 그런 다음 클라우드 템플릿을 파이프라인에 추가합니다.

이 사용 사례 예는 온-프레미스 GitHub 저장소의 클라우드 템플릿을 사용하는 방법을 보여줍니다.

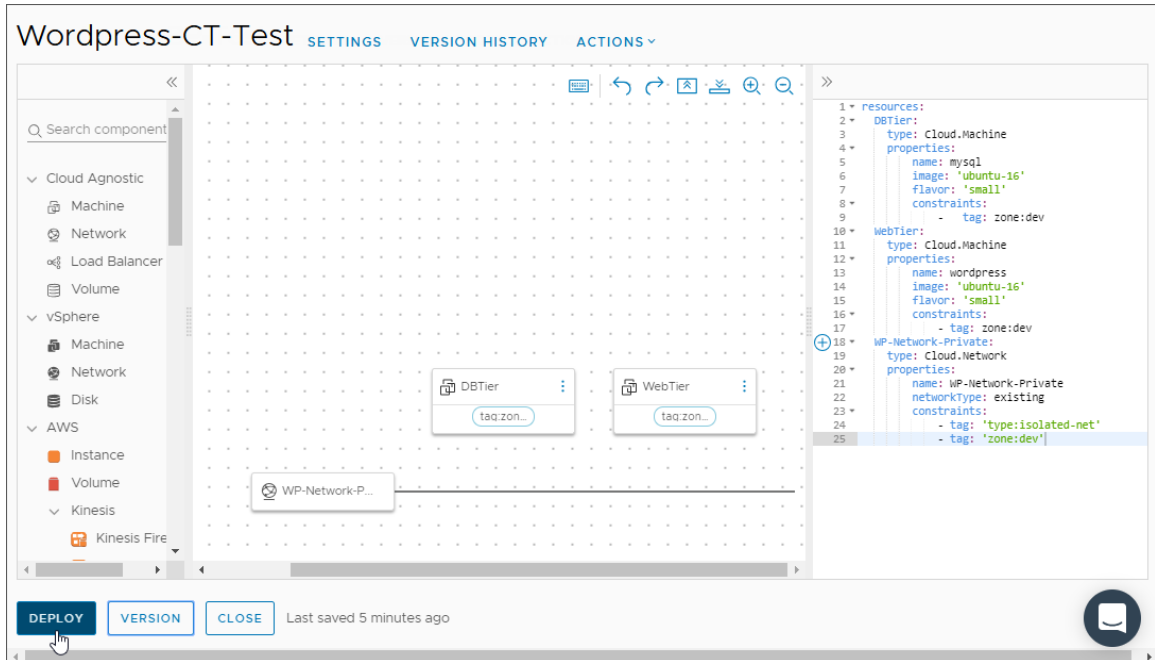


### 사전 요구 사항

- vRealize Automation Cloud Assembly 인프라에서 클라우드 계정 및 클라우드 영역을 추가합니다. vRealize Automation Cloud Assembly 설명서를 참조하십시오.
- 다음 절차에서 클라우드 템플릿을 생성하려면 WordPress YAML 코드를 클립보드에 복사합니다. vRealize Automation Cloud Assembly 설명서의 WordPress 사용 사례에서 클라우드 템플릿 YAML 코드를 참조하십시오.
- WordPress 애플리케이션의 YAML 코드를 GitHub 인스턴스에 추가합니다.
- 변경 내용을 커밋할 때마다 파이프라인에서 YAML 코드를 끌어올 수 있도록 Git 트리거에 대한 Webhook를 추가합니다. Code Stream에서 **트리거 > Git > Git에 대한 Webhook**를 클릭합니다.
- 클라우드 템플릿 작업을 사용하려면 Cloud Assembly 역할 중 하나가 있어야 합니다.

## 절차

- 1 Cloud Assembly에서 다음 단계를 따릅니다.
  - a **VMware Cloud Templates**을 클릭한 다음, 클라우드 템플릿 및 WordPress 애플리케이션에 대한 배포를 생성합니다.
  - b 클립보드에 복사한 WordPress YAML 코드를 클라우드 템플릿에 붙여 넣고 배포합니다.



## 2 Code Stream에서 끝점을 생성합니다.

- a YAML 파일이 상주하는 온 프레미스 GitHub 저장소에 대한 Git 끝점을 생성합니다.
- b 파이프라인이 실행되면 파이프라인 상태를 사용자에게 알릴 수 있는 이메일 끝점을 추가합니다.

## 3 파이프라인을 생성하고 파이프라인 성공 및 실패에 대한 알림을 추가합니다.



#### 4 개발에 대한 단계를 추가하고 클라우드 템플릿 작업을 추가합니다.

- a 시스템을 배포하는 클라우드 템플릿 작업을 추가하고, WordPress 애플리케이션에 클라우드 템플릿 YAML을 사용하도록 작업을 구성합니다.

```
resources:
  DBTier:
    type: Cloud.Machine
    properties:
      name: mysql
      image: 'ubuntu-16'
      flavor: 'small'
      constraints:
        - tag: zone:dev
  WebTier:
    type: Cloud.Machine
    properties:
      name: wordpress
      image: 'ubuntu-16'
      flavor: 'small'
      constraints:
        - tag: zone:dev
  WP-Network-Private:
    type: Cloud.Network
    properties:
      name: WP-Network-Private
      networkType: existing
      constraints:
        - tag: 'type:isolated-net'
        - tag: 'zone:dev'
```

- b 리소스를 확보하기 위해 시스템을 제거하는 클라우드 템플릿 작업을 추가합니다.

## 5 운영을 위한 단계를 추가하고 승인 및 배포 작업을 추가합니다.

- a WordPress 애플리케이션을 운영 환경에 푸시하기 위해 승인을 요구하는 [사용자 작업] 작업을 추가합니다.
- b 시스템을 배포하는 클라우드 템플릿 작업을 추가하고, WordPress 애플리케이션용 클라우드 템플릿 YAML을 사용하여 작업을 구성합니다.

**생성**을 선택할 때 배포 이름이 고유해야 합니다. 이름을 비워 두면 Code Stream은 고유한 임의의 이름을 할당합니다.

자체 사용 사례에서 **롤백**을 선택하는 경우 알아야 할 내용은 다음과 같습니다. **롤백** 작업을 선택하고 **롤백 버전**을 선택하는 경우 버전은 **n-x** 형식이어야 합니다. 예를 들어 **n-1**, **n-2**, **n-3** 등을 사용할 수 있습니다. Code Stream 이외의 위치에서 배포를 생성하고 업데이트하는 경우 롤백이 허용됩니다.

Code Stream에 로그인하면 30분 동안 유효한 사용자 토큰을 가져옵니다. 실행 기간이 긴 파이프라인의 경우 클라우드 템플릿 작업 이전의 작업을 실행하는 데 30분 이상이 소요되면 사용자 토큰이 만료됩니다. 따라서 클라우드 템플릿 작업이 실패합니다.

파이프라인이 30분 이상 실행될 수 있도록 선택적 API 토큰을 입력할 수 있습니다. Code Stream에서 클라우드 템플릿을 호출하면 API 토큰이 유지되고 클라우드 템플릿 작업은 API 토큰을 계속 사용합니다.

API 토큰을 변수로 사용하는 경우 토큰이 암호화됩니다. 그렇지 않으면 일반 텍스트로 사용됩니다.

The screenshot displays the configuration page for a task named 'Deploy CT'. The top navigation bar includes 'Task:Deploy CT', 'Notifications', and 'Rollback', along with a 'VALIDATE TASK' button. The configuration fields are as follows:

- Task name:** Deploy CT
- Type:** VMware cloud template
- Continue on failure:** ☐
- Execute task:** ☒ Always ☐ On condition
- Deployment Task:**
  - Action:** ☒ Create ☐ Update ☐ Delete ☐ Rollback
  - API token:** API token (with a red error icon) and a 'CREATE VARIABLE' button.
  - Deployment Name:** Enter deployment name
  - Cloud template source:** ☒ VMware cloud templates ☐ Source Control
  - Cloud template:** --Select template--
  - Version:** --Select template Version--
  - Output Parameters:** (Section header)

## 6 파이프라인을 실행합니다.

각 작업이 성공적으로 완료되었는지 확인하려면 실행에서 작업을 클릭하고 배포 세부 정보에서 상태를 검사하여 자세한 리소스 정보를 살펴봅니다.

## 7 GitHub에서 WordPress 서버 인스턴스의 버전을 small에서 medium으로 수정합니다.

변경 내용을 커밋하면 파이프라인이 트리거됩니다. 업데이트된 코드를 GitHub 저장소에서 끌어와서 애플리케이션을 구축합니다.

```
WebTier:
  type: Cloud.Machine
  properties:
    name: wordpress
    image: 'ubuntu-16'
    flavor: 'medium'
    constraints:
      - tag: zone:dev
```

## 8 파이프라인을 다시 실행한 후 파이프라인이 성공했는지 그리고 WordPress 인스턴스의 버전이 small에서 medium으로 변경되었는지 확인합니다.

### 결과

축하합니다! YAML 클라우드 템플릿에서 배포한 애플리케이션의 릴리스를 자동화했습니다.

### 다음에 수행할 작업

Code Stream 사용 방법에 대한 자세한 내용을 보려면 [장 5 Code Stream 사용을 위한 자습서](#) 항목을 참조하십시오.

추가로 참조하려면 [Code Stream 관리자 및 개발자를 위한 추가 리소스](#) 항목을 참조하십시오.

## Code Stream에서 Kubernetes 클러스터에 애플리케이션의 릴리스를 자동화하는 방법

Code Stream 관리자 또는 개발자는 Code Stream 및 VMware Tanzu Kubernetes Grid Integrated Edition(이전 이름: VMware Enterprise PKS)을 사용하여 소프트웨어 애플리케이션 배포를 Kubernetes 클러스터로 자동화할 수 있습니다. 이 사용 사례에서는 애플리케이션의 릴리스를 자동화하는 데 사용할 수 있는 다른 방법을 설명합니다.

이 사용 사례에서는 두 단계가 포함된 파이프라인을 생성하고 Jenkins를 사용하여 애플리케이션을 구축하고 배포합니다.

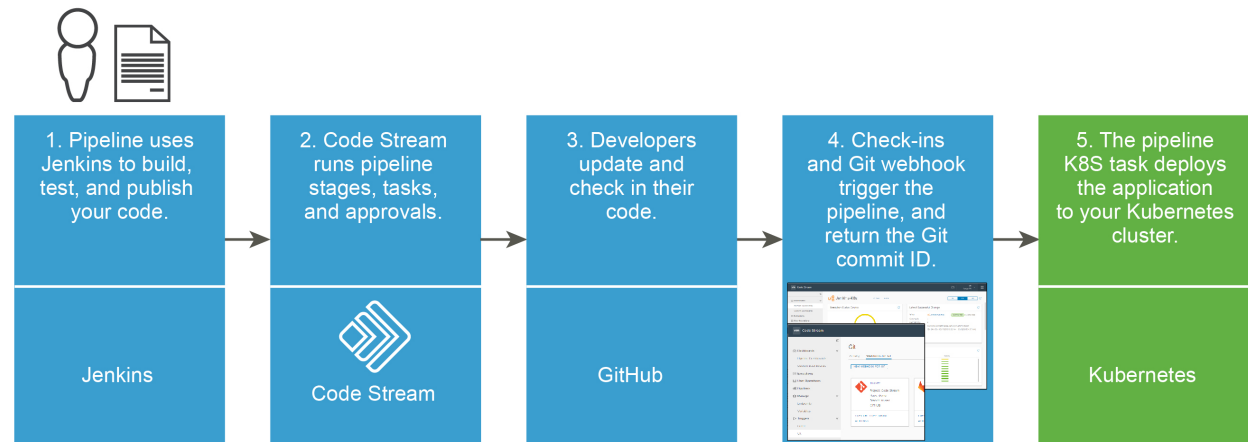
- 첫 번째 단계는 개발을 위한 단계입니다. 이 단계에서는 Jenkins를 사용하여 코드를 GitHub 저장소의 분기에서 풀한 후 구축하고 테스트하고 게시합니다.
- 두 번째 단계는 배포를 위한 단계입니다. 이 단계에서는 파이프라인이 애플리케이션을 Kubernetes 클러스터에 배포할 수 있도록 주요 사용자의 승인이 필요한 사용자 작업을 실행합니다.

파이프라인 업무 공간에서 Kubernetes API 끝점을 사용하는 경우 Code Stream은 CI(지속적 통합) 작업 또는 사용자 지정 작업을 실행하는 데 필요한 Kubernetes 리소스(예: ConfigMap, Secret 및 포트)를 생성합니다. Code Stream은 NodePort를 사용하여 컨테이너와 통신합니다.

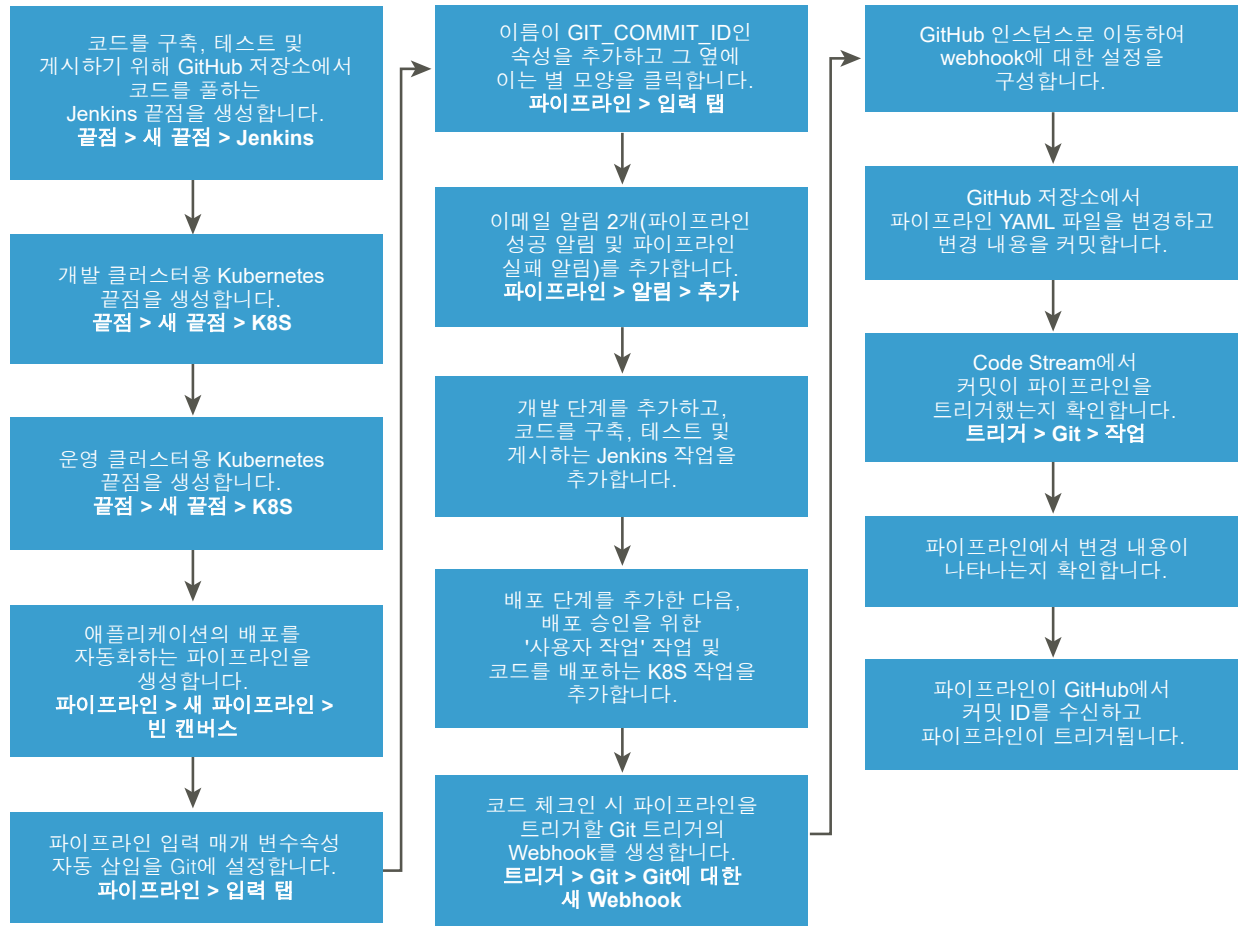
파이프라인 실행 간에 데이터를 공유하려면 영구 볼륨 할당을 제공해야 합니다. 그러면 Code Stream은 영구 볼륨 할당을 컨테이너에 마운트하여 데이터를 저장하고 후속 파이프라인 실행에 사용합니다.

Code Stream 파이프라인 업무 공간은 지속적 통합 작업 및 사용자 지정 작업을 위해 Docker 및 Kubernetes를 지원합니다.

업무 공간 구성에 대한 자세한 내용은 [파이프라인 작업 공간 구성](#)에서 참조하십시오.



파이프라인이 애플리케이션 구축, 테스트, 게시 및 배포를 수행할 수 있도록 개발 도구, 배포 인스턴스 및 파이프라인 YAML 파일이 준비되어 있어야 합니다. 파이프라인은 AWS에 있는 Kubernetes 클러스터의 개발 및 운영 인스턴스에 애플리케이션을 배포합니다.



애플리케이션 릴리스를 자동화하는 기타 방법:

- Jenkins를 사용하여 애플리케이션을 구축하는 대신 Code Stream 기본 빌드 기능 및 Docker 빌드 호스트를 사용할 수 있습니다.
- 애플리케이션을 Kubernetes 클러스터에 배포하는 대신 AWS(Amazon Web Services) 클러스터에 배포할 수 있습니다.

Code Stream 기본 구축 기능 및 Docker 호스트 사용에 대한 자세한 내용은 다음을 참조하십시오.

- 스마트 파이프라인 템플릿을 사용하기 전에 Code Stream에서 CICD 기본 구축 계획
- 작업을 수동으로 추가하기 전에 Code Stream에서 CICD 네이티브 빌드 계획

#### 사전 요구 사항

- 배포할 애플리케이션 코드가 작동하는 GitHub 저장소에 있는지 확인합니다.
- 작동하는 Jenkins 인스턴스가 있는지 확인합니다.
- 작동하는 이메일 서버가 있는지 확인합니다.
- 이메일 서버에 연결하는 이메일 끝점을 Code Stream에서 생성합니다.

- 파이프라인이 애플리케이션을 배포할 Kubernetes 클러스터 2개(개발 및 운영)를 AWS(Amazon Web Services)에 설정합니다.
- GitHub 저장소에 파이프라인의 YAML 코드가 포함되어 있는지, 또는 환경의 메타데이터와 규격을 정의하는 YAML 파일이 포함되어 있는지 확인합니다.

## 절차

- 1 Code Stream에서 **끝점 > 새 끝점**을 클릭하여, GitHub 저장소에서 코드를 풀하기 위해 파이프라인에서 사용할 Jenkins 끝점을 생성합니다.
- 2 Kubernetes 끝점을 생성하려면 **새 끝점**을 클릭합니다.
  - a 개발 Kubernetes 클러스터용 끝점을 생성합니다.
  - b 운영 Kubernetes 클러스터용 끝점을 생성합니다.

Kubernetes 클러스터의 URL은 포트 번호를 포함하거나 포함하지 않을 수 있습니다.

예를 들면 다음과 같습니다.

```
https://10.111.222.333:6443
https://api.kubernetesserver.fa2c1d78-9f00-4e30-8268-4ab81862080d.k8s-user.com
```
- 3 애플리케이션의 컨테이너(예: Wordpress)를 개발 Kubernetes 클러스터에 배포하는 파이프라인을 생성하고 파이프라인의 입력 속성을 설정합니다.
  - a 파이프라인을 트리거할 커밋 코드를 파이프라인이 GitHub에서 인식할 수 있게 하려면 파이프라인에서 **입력** 탭을 클릭하고 **속성 자동 삽입**을 선택합니다.
  - b **GIT\_COMMIT\_ID**라는 속성을 추가하고 옆에 있는 별 모양을 클릭합니다.

파이프라인 실행 시 Git 트리거가 반환하는 커밋 ID가 표시됩니다.

**Jenkins-K8s** Enabled

Pipeline **Input** Output Notifications CI Workspace

Pipeline Input Parameters

Auto inject properties ☐ Gerrit ☒ Git ☐ None

**ADD**

Starred	Name	Value	Description
<input type="checkbox"/>	GIT_BRANCH_NAME		
<input type="checkbox"/>	GIT_CHANGE_SUBJECT		
<input checked="" type="checkbox"/>	GIT_COMMIT_ID		
<input type="checkbox"/>	GIT_EVENT_DESCRIPTION		
<input type="checkbox"/>	GIT_EVENT_OWNER_NAME		
<input type="checkbox"/>	GIT_EVENT_TIMESTAMP		
<input type="checkbox"/>	GIT_REPO_NAME		
<input type="checkbox"/>	GIT_SERVER_URL		

8 input parameters

**SAVE** **RUN** **CLOSE** Last saved 5 days ago

#### 4 파이프라인 성공 또는 실패 시 이메일을 전송하는 알림을 추가합니다.

- a 파이프라인에서 **알림** 탭을 클릭하고 **추가**를 클릭합니다.
- b 파이프라인 실행이 완료되었을 때 이에 대한 이메일 알림을 추가하려면 **이메일**을 선택한 후 **완료**를 선택합니다. 그런 다음 이메일 서버를 선택하고 이메일 주소를 입력한 후 **저장**을 클릭합니다.
- c 파이프라인 실패에 대한 또 다른 이메일 알림을 추가하려면 **실패**를 선택하고 **저장**을 클릭합니다.

### Notification

**Send notification type** ☒ Email ☐ Ticket ☐ Webhook

**When pipeline** ☒ Completes ☐ Is Waiting ☐ Fails ☐ Is cancelled ☐ Starts to run

**Email server** ⓘ \* --Select Email server-- ▾

**Send Email**

**To** ⓘ \$ \* Email IDs of recipients

**Subject** \$ \* Email Subject

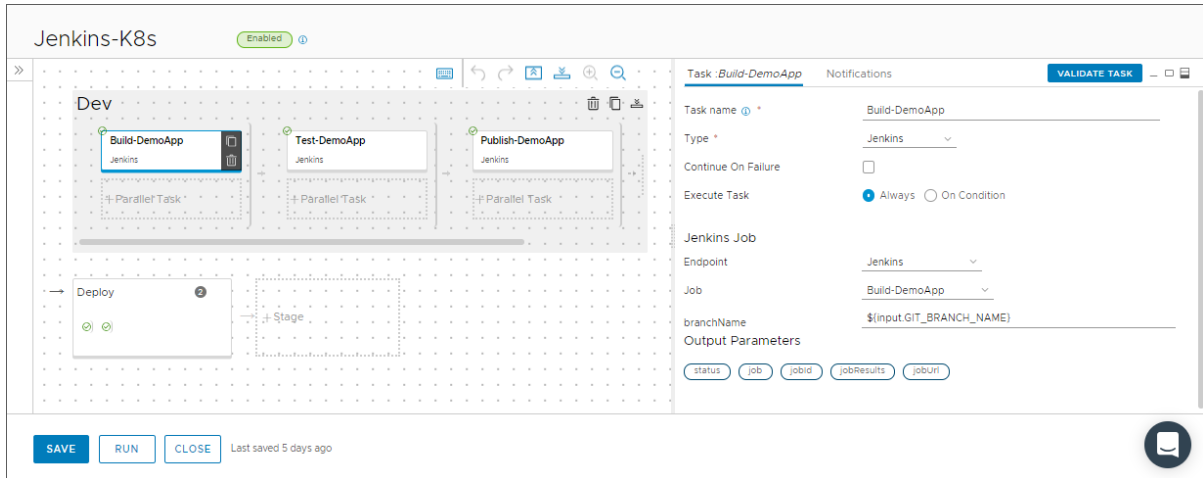
**Body** ⓘ \$ \* 

1

CANCEL
SAVE

#### 5 파이프라인에 개발 단계를 추가하고, 애플리케이션을 구축, 테스트 및 게시하는 작업을 추가합니다. 그런 다음 각 작업을 검증합니다.

- a 애플리케이션을 구축하려면 **Jenkins** 끝점을 사용하고 **Jenkins** 서버에서 구축 작업을 실행하는 **Jenkins** 작업을 추가합니다. 그런 다음 파이프라인이 코드를 풀할 수 있도록 **`${input.GIT_BRANCH_NAME}`** 형식으로 Git 분기를 입력합니다.
- b 애플리케이션을 테스트하려면 동일한 **Jenkins** 끝점을 사용하고 **Jenkins** 서버에서 테스트 작업을 실행하는 **Jenkins** 작업을 추가합니다. 그런 다음 동일한 Git 분기를 입력합니다.
- c 애플리케이션을 게시하려면 동일한 **Jenkins** 끝점을 사용하고 **Jenkins** 서버에서 게시 작업을 실행하는 **Jenkins** 작업을 추가합니다. 그런 다음 동일한 Git 분기를 입력합니다.



- 6 파이프라인에 배포 단계를 추가한 다음 애플리케이션 배포에 대한 승인을 요청하는 작업과 Kubernetes 클러스터에 애플리케이션을 배포하는 또 다른 작업을 추가합니다. 그런 다음 각 작업을 검증합니다.
  - a 애플리케이션 배포에 대한 승인을 요청하려면 '사용자 작업' 작업을 추가하고, 승인을 해야 하는 사용자의 이메일 주소를 추가한 다음 메시지를 입력합니다. 그런 다음 **이메일 보내기**를 사용하도록 설정합니다.
  - b 애플리케이션을 배포하려면 Kubernetes 작업을 추가합니다. 그런 다음 Kubernetes 작업 속성에서 개발 Kubernetes 클러스터를 선택하고 **생성** 작업을 선택한 다음 **로컬 정의** 페이로드 소스를 선택합니다. 그런 다음 로컬 YAML 파일을 선택합니다.



- 7 Code Stream이 Git 트리거를 사용하도록 하는 Git Webhook를 추가합니다. Git Webhook는 개발자가 코드를 커밋하면 파이프라인을 트리거합니다.

The screenshot displays the 'Git Webhooks for Git' configuration page. It includes the following fields and controls:

- Webhook URL:** A text field containing a long URL.
- Project:** A dropdown menu set to 'Code Stream'.
- Name:** A text field containing 'muser-Demo-WH'.
- Description:** A text area with the placeholder 'Description'.
- Endpoint:** A text field containing 'tpm-GitHub'.
- Branch:** A text field containing 'master'.
- Secret token:** A text field with a masked token and a 'GENERATE' button.
- File:** A dropdown menu.
- Inclusions:** A dropdown menu with a 'Value' field and a '+' button.
- Exclusions:** A dropdown menu with a 'Value' field and a '+' button.
- Prioritize Exclusion:** A toggle switch.
- Trigger:** Radio buttons for 'PUSH' (selected) and 'PULL REQUEST'.
- API token:** A text field with a masked token, a red 'X' icon, and 'CREATE VARIABLE' and 'GENERATE TOKEN' buttons.
- Pipeline:** A dropdown menu set to 'Jenkins-K8s'.
- Comments:** A text area.

- 8 파이프라인을 테스트하려면 GitHub 저장소로 이동하여 애플리케이션 YAML 파일을 업데이트한 다음 변경 내용을 커밋합니다.
- Code Stream에 커밋이 표시되는지 확인합니다.
  - 트리거 > Git > 작업**을 클릭합니다.
  - 파이프라인 트리거를 찾아봅니다.
  - 대시보드 > 파이프라인 대시보드**를 클릭합니다.
  - 파이프라인 대시보드의 최근에 성공한 변경 내용 영역에서 GIT\_COMMIT\_ID를 찾습니다.
- 9 파이프라인 코드를 확인하고, 변경 내용이 표시되는지 확인합니다.

## 결과

축하합니다! Kubernetes 클러스터에 소프트웨어 애플리케이션의 배포를 자동화했습니다.

## 예제: Kubernetes 클러스터에 애플리케이션을 배포하는 예시 파이프라인 YAML

이 예시에 사용된 파이프라인 유형의 경우 YAML은 다음 코드와 유사합니다.

```
apiVersion: v1
kind: Namespace
metadata:
  name: ${input.GIT_BRANCH_NAME}
  namespace: ${input.GIT_BRANCH_NAME}
---
apiVersion: v1
data:
  .dockercfg:
eyJzeWlwag9ueS10YW5nby1iZXRhMi5qZnJvZy5pbyI6eyJlc2VybmFtZSI6InRhbmdvLWJldGEyIiwicGFzc3dvcmQI Oi
JhRGstcmVOLWlUQi1IejciLCJlbWFPbCI6InRhbmdvLWJldGEyQHZtd2FyZS5jb20iLCJhdXRoIjoizEdGdVoyOHRZbVYw
WVRJN1lVUnJMWEPsVGkxdFZFSXRTSG8zIn19
kind: Secret
metadata:
  name: jfrog
  namespace: ${input.GIT_BRANCH_NAME}
type: kubernetes.io/dockercfg
---
apiVersion: v1
kind: Service
metadata:
  name: codestream
  namespace: ${input.GIT_BRANCH_NAME}
  labels:
    app: codestream
spec:
  ports:
    - port: 80
  selector:
    app: codestream
    tier: frontend
  type: LoadBalancer
---
apiVersion: extensions/v1
kind: Deployment
metadata:
  name: codestream
  namespace: ${input.GIT_BRANCH_NAME}
  labels:
    app: codestream
spec:
  selector:
    matchLabels:
      app: codestream
      tier: frontend
```

```

strategy:
  type: Recreate
template:
  metadata:
    labels:
      app: codestream
      tier: frontend
  spec:
    containers:
      - name: codestream
        image: cas.jfrog.io/codestream:${input.GIT_BRANCH_NAME}-${Dev.PublishApp.output.jobId}
        ports:
          - containerPort: 80
            name: codestream
    imagePullSecrets:
      - name: jfrog

```

### 다음에 수행할 작업

소프트웨어 애플리케이션을 운영 Kubernetes 클러스터에 배포하려면 위의 단계를 다시 수행하면서 운영 클러스터를 선택하십시오.

Code Stream과 Jenkins의 통합에 대한 자세한 내용은 [Code Stream과 Jenkins를 통합하는 방법](#) 항목을 참조하십시오.

## Code Stream에서 애플리케이션을 블루-그린 배포에 배포하는 방법

블루-그린은 Kubernetes 클러스터에서 동일하게 배포하고 구성하는 두 개의 Docker 호스트를 사용하는 배포 모델입니다. 블루-그린 배포 모델을 사용하면 Code Stream의 파이프라인에서 애플리케이션을 배포할 때 환경에서 발생할 수 있는 다운타임을 줄일 수 있습니다.

배포 모델의 블루 및 그린 인스턴스는 각각 서로 다른 용도로 사용됩니다. 한 번에 하나의 인스턴스만 애플리케이션을 배포하는 활성 트래픽을 수락하며, 각 인스턴스는 특정 시간에 해당 트래픽을 수락합니다. 블루 인스턴스는 애플리케이션의 첫 번째 버전을 수신하고 그린 인스턴스는 두 번째 버전을 수신합니다.

블루-그린 환경의 로드 밸런서는 활성 트래픽이 애플리케이션을 배포할 때 사용할 경로를 결정합니다. 블루-그린 모델을 사용하는 경우 사용자 환경은 작동 상태를 유지하고, 사용자는 다운타임을 감지하지 못하며, 파이프라인은 애플리케이션을 지속적으로 통합하여 운영 환경에 배포합니다.

Code Stream에서 생성하는 파이프라인은 다음 두 단계의 블루-그린 배포 모델을 나타냅니다. 한 단계는 개발 단계이고, 다른 한 단계는 운영 단계입니다.

Code Stream 파이프라인 업무 공간은 지속적 통합 작업 및 사용자 지정 작업을 위해 Docker 및 Kubernetes를 지원합니다.

업무 공간 구성에 대한 자세한 내용은 [파이프라인 작업 공간 구성](#)에서 참조하십시오.

표 5-2. 블루-그린 배포의 개발 단계 작업

작업 유형	작업
Kubernetes	블루-그린 배포를 위한 네임스페이스를 생성합니다.
Kubernetes	Docker 허브에 대한 비밀 키를 생성합니다.
Kubernetes	애플리케이션 배포에 사용되는 서비스를 생성합니다.
Kubernetes	블루 배포를 생성합니다.
폴링	블루 배포를 확인합니다.
Kubernetes	네임스페이스를 제거합니다.

표 5-3. 블루-그린 배포의 운영 단계 작업

작업 유형	작업
Kubernetes	그린이 블루에서 서비스 세부 정보를 가져옵니다.
Kubernetes	그린 복제 집합에 대한 세부 정보를 가져옵니다.
Kubernetes	그린 배포를 생성하고 비밀 키를 사용하여 컨테이너 이미지를 가져옵니다.
Kubernetes	서비스를 업데이트합니다.
폴링	운영 URL에서 배포가 성공했는지 확인합니다.
Kubernetes	블루 배포를 완료합니다.
Kubernetes	블루 배포를 제거합니다.

사용자 고유의 블루-그린 배포 모델에서 애플리케이션을 배포하려면 **Code Stream**에서 다음 두 단계를 포함하는 파이프라인을 생성합니다. 첫 번째 단계는 애플리케이션을 블루 인스턴스에 배포하는 블루 작업을 포함하고, 두 번째 단계는 애플리케이션을 그린 인스턴스에 배포하는 그린 작업을 포함합니다.

CICD 스마트 파이프라인 템플릿을 사용하여 파이프라인을 생성할 수 있습니다. 이 경우 템플릿에서 파이프라인 단계와 작업을 생성하고 배포 선택 내용을 포함합니다.

파이프라인을 수동으로 생성하는 경우 파이프라인 단계를 직접 계획해야 합니다. 예를 보려면 [작업을 수동으로 추가하기 전에 Code Stream에서 CICD 네이티브 빌드 계획](#) 항목을 참조하십시오.

이 예에서는 CICD 스마트 파이프라인 템플릿을 사용하여 블루-그린 파이프라인을 생성합니다.

#### 사전 요구 사항

- AWS에서 작동 중인 Kubernetes 클러스터에 액세스할 수 있는지 확인합니다.
- 블루-그린 배포 환경을 설정했고 블루 및 그린 인스턴스를 동일하게 구성했는지 확인합니다.
- Code Stream에서 애플리케이션 이미지를 AWS의 Kubernetes 클러스터에 배포하는 Kubernetes 끝점을 생성합니다.

- CICD 스마트 파이프라인 템플릿 사용에 관한 내용을 숙지합니다. 스마트 파이프라인 템플릿을 사용하기 전에 [Code Stream](#)에서 [CICD 기본 구축 계획](#) 항목을 참조하십시오.

#### 절차

- 1 **파이프라인 > 새 파이프라인 > 스마트 템플릿 > CI/CD 템플릿**을 클릭합니다.
- 2 CICD 스마트 파이프라인 템플릿의 CI 부분에 대한 정보를 입력하고 **다음**을 클릭합니다.  
도움이 필요하면 [스마트 파이프라인 템플릿을 사용하기 전에](#) [Code Stream](#)에서 [CICD 기본 구축 계획](#) 항목을 참조하십시오.
- 3 스마트 파이프라인 템플릿의 CD 부분에 대한 입력을 완료합니다.
  - a 애플리케이션 배포를 위한 환경을 선택합니다. 예를 들어 **개발** 및 **운영**을 선택합니다.
  - b 배포를 위해 파이프라인에서 사용할 서비스를 선택합니다.

- d 운영 배포 모델의 경우 **블루-그린**을 선택하고 **생성**을 클릭합니다.

# 스마트 템플릿: CI/CD

2/2 단계

환경 ① \*

☒ 개발
 ☒ 운영

Kubernetes YAML 파일 \*

선택
프로세스

처리된 파일: codestream.yaml

**서비스 선택**

배포 이름	서비스	네임스페이스	이미지
<input checked="" type="radio"/> codestream-demo	codestream-demo	bgreen1	13

1개 서비스

**배포**

환경	클러스터 끝점	네임스페이스
개발	1030Endpoint-Kubernetes- <span>弱家表術あA中CF@圖停B道UE</span> ▾	bgreen1-392882
운영	1030Endpoint-Kubernetes- <span>弱家表術あA中CF@圖停B道UE</span> ▾	bgreen1

**배포 모델 \***

☐ Canary
 ☐ 롤링 업그레이드
 ☒ 블루-그린

**롤백**

☐

**상태 점검 URL \***

상태 점검 URL 입력

생성
뒤로
취소

## 결과

축하합니다! 스마트 파이프라인 템플릿을 사용하여 AWS의 Kubernetes 운영 클러스터에 있는 블루-그린 인스턴스에 애플리케이션을 배포하는 파이프라인을 생성했습니다.

### 예제: 일부 블루-그린 배포 작업을 위한 YAML 코드 예

블루-그린 배포의 Kubernetes 파이프라인 작업에 나타나는 YAML 코드는 네임스페이스, 서비스 및 배포를 생성하는 다음 예와 유사할 수 있습니다. 개인 소유 저장소에서 이미지를 다운로드해야 하는 경우 YAML 파일에 Docker 구성 비밀이 있는 섹션이 포함되어야 합니다. [스마트 파이프라인 템플릿을 사용하기 전에 Code Stream에서 CICD 기본 구축 계획의 CD 부분을 참조하십시오.](#)

스마트 파이프라인 템플릿으로 파이프라인을 생성한 후 필요에 따라 배포에 대한 작업을 수정할 수 있습니다.

네임스페이스를 생성하는 YAML 코드 예:

```
apiVersion: v1
kind: Namespace
metadata:
  name: codestream-82855
  namespace: codestream-82855
```

서비스를 생성하는 YAML 코드 예:

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: codestream-demo
  name: codestream-demo
  namespace: bluegreen-799584
spec:
  minReadySeconds: 0
  ports:
    - port: 80
  selector:
    app: codestream-demo
    tier: frontend
  type: LoadBalancer
```

배포를 생성하는 YAML 코드 예:

```
apiVersion: extensions/v1
kind: Deployment
metadata:
  labels:
    app: codestream-demo
  name: codestream-demo
  namespace: bluegreen-799584
spec:
  minReadySeconds: 0
  replicas: 1
  selector:
    matchLabels:
      app: codestream-demo
      tier: frontend
  template:
    metadata:
      labels:
        app: codestream-demo
        tier: frontend
    spec:
      containers:
        - image: ${input.image}:${input.tag}
          name: codestream-demo
          ports:
            - containerPort: 80
              name: codestream-demo
```

```
imagePullSecrets:
- name: jfrog-2
minReadySeconds: 0
```

### 다음에 수행할 작업

Code Stream 사용 방법에 대한 자세한 내용을 보려면 [장 5 Code Stream 사용을 위한 자습서](#) 항목을 참조하십시오.

배포를 롤백하려면 [Code Stream에서 배포를 롤백하는 방법](#) 항목을 참조하십시오.

추가로 참조하려면 [Code Stream 관리자 및 개발자를 위한 추가 리소스](#) 항목을 참조하십시오.

## 고유한 구축, 테스트 및 배포 톨과 Code Stream을 통합하는 방법

DevOps 관리자 또는 개발자는 Code Stream의 기능을 확장하는 사용자 지정 스크립트를 생성할 수 있습니다.

스크립트를 사용하여 애플리케이션을 구축, 테스트 및 배포하는 고유한 CI/CD(지속적 통합 및 전달) 톨 및 API와 Code Stream을 통합할 수 있습니다. 사용자 지정 스크립트는 특히 애플리케이션 API를 공개적으로 노출하지 않는 경우에 유용합니다.

사용자 지정 스크립트는 Code Stream과 통합되는 빌드, 테스트 및 배포 도구에 필요한 거의 모든 작업을 수행할 수 있습니다. 예를 들어 스크립트는 파이프라인 업무 공간에서 작동하여 애플리케이션을 구축 및 테스트하는 지속적 통합 작업과 애플리케이션을 배포하는 지속적 전달 작업을 지원할 수 있습니다. 또한 파이프라인이 완료되면 Slack에 메시지를 보내는 등의 작업을 수행할 수 있습니다.

Code Stream 파이프라인 업무 공간은 지속적 통합 작업 및 사용자 지정 작업을 위해 Docker 및 Kubernetes를 지원합니다.

업무 공간 구성에 대한 자세한 내용은 [파이프라인 작업 공간 구성](#)에서 참조하십시오.

지원되는 언어 중 하나로 사용자 지정 스크립트를 작성합니다. 스크립트에서 비즈니스 논리를 포함하고 입력과 출력을 정의합니다. 출력 유형에는 숫자, 문자열, 텍스트 및 암호가 포함될 수 있습니다. 다양한 비즈니스 논리, 입력 및 출력을 사용하여 여러 버전의 사용자 지정 스크립트를 생성할 수 있습니다.

파이프라인에서 사용자 지정 작업의 스크립트 버전을 실행합니다. 생성한 스크립트는 Code Stream 인스턴스에 상주합니다.

파이프라인에서 사용자 지정 통합을 사용하는 경우 사용자 지정 통합을 삭제하려고 하면 오류 메시지가 표시되고 삭제할 수 없음을 나타냅니다.

사용자 지정 통합을 삭제하면 사용자 지정 스크립트의 모든 버전이 제거됩니다. 임의의 스크립트의 버전을 사용하는 사용자 지정 작업이 있는 기존 파이프라인은 실패합니다. 기존 파이프라인이 실패하지 않도록 하려면 더 이상 사용하지 않으려는 스크립트 버전을 폐기하고 철회하면 됩니다. 해당 버전을 사용하는 파이프라인이 없는 경우 이를 삭제할 수 있습니다.



표 5-4. 사용자 지정 스크립트를 작성한 후 수행할 작업

수행할 작업...	이 작업에 대한 자세한 내용...
파이프라인에 사용자 지정 작업을 추가합니다.	<p>사용자 지정 작업:</p> <ul style="list-style-type: none"> <li>■ 파이프라인의 다른 CI 작업과 동일한 컨테이너에서 실행됩니다.</li> <li>■ 파이프라인이 사용자 지정 작업을 실행하기 전에 스크립트가 채우는 입력 및 출력 변수를 포함합니다.</li> <li>■ 스크립트에서 입력과 출력으로 정의된 여러 데이터 유형과 다양한 메타데이터 유형을 지원합니다.</li> </ul>
사용자 지정 작업에서 스크립트를 선택합니다.	스크립트에서 입력 및 출력 속성을 선언합니다.
파이프라인을 저장한 다음 사용하도록 설정하고 실행합니다.	파이프라인이 실행되면 사용자 지정 작업이 지정된 스크립트를 호출하고 스크립트 내의 비즈니스 논리를 실행합니다. 그러면 구축, 테스트 및 배포 도구가 Code Stream과 통합됩니다.
파이프라인이 실행되면 실행을 살펴봅니다.	파이프라인에서 예상한 결과를 전달했는지 확인합니다.

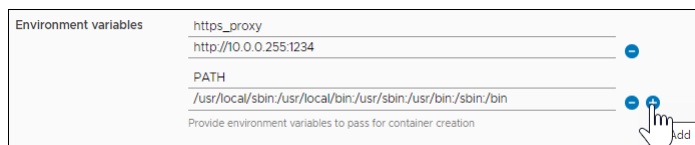
사용자 지정 통합 버전을 호출하는 사용자 지정 작업을 사용하는 경우 사용자 지정 환경 변수를 파이프라인 **작업 공간** 탭에 이름-값 쌍으로 포함할 수 있습니다. 빌더 이미지가 CI 작업을 실행하고 이미지를 배포하는 작업 공간 컨테이너를 생성하는 경우 Code Stream은 컨테이너에 환경 변수를 전달합니다.

예를 들어 Code Stream 인스턴스에 웹 프록시가 필요하고 Docker 호스트를 사용하여 사용자 지정 통합을 위한 컨테이너를 생성하는 경우 Code Stream은 파이프라인을 실행하고 이 컨테이너에 웹 프록시 설정 변수를 전달합니다.

표 5-5. 환경 변수 이름-값 쌍의 예

이름	값
HTTPS_PROXY	http://10.0.0.255:1234
https_proxy	http://10.0.0.255:1234
NO_PROXY	10.0.0.32, *.dept.vsphere.local
no_proxy	10.0.0.32, *.dept.vsphere.local
HTTP_PROXY	http://10.0.0.254:1234
http_proxy	http://10.0.0.254:1234
PATH	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

이름-값 쌍은 사용자 인터페이스에 다음과 같이 나타납니다.



이 예는 Code Stream을 Slack 인스턴스에 연결하고 메시지를 Slack 채널에 게시하는 사용자 지정 통합을 생성합니다.

### 사전 요구 사항

- 사용자 지정 스크립트를 작성하려면 Python 2, Python 3, Node.js와 같은 언어 중 하나 또는 Bash, sh 또는 zsh와 같은 셸 언어가 있는지 확인합니다.
- 설치된 Node.js 또는 Python 런타임을 사용하여 컨테이너 이미지를 생성합니다.

### 절차

1 사용자 지정 통합을 생성합니다.

- a **사용자 지정 통합 > 새로 만들기**를 클릭하고 적절한 이름을 입력합니다.
- b 기본 런타임 환경을 선택합니다.
- c **생성**을 클릭합니다.

스크립트가 열리면서 코드가 표시됩니다. 여기에는 필요한 런타임 환경이 포함되어 있습니다. 예를 들어 runtime: "nodejs"가 표시됩니다. 스크립트에는 파이프라인이 실행될 때 파이프라인에 추가하는 사용자 지정 작업이 성공할 수 있도록 빌더 이미지가 사용하는 런타임이 포함되어야 합니다. 그렇지 않으면 사용자 지정 작업이 실패합니다.

사용자 지정 통합 YAML의 주요 영역에는 런타임, 코드, 입력 속성 및 출력 속성이 포함되어 있습니다. 이 절차는 다양한 유형과 구문에 대해 설명합니다.

사용자 지정 통합 YAML 키	설명
runtime	Code Stream이 코드를 실행하는 작업 런타임 환경. 다음과 같은 문자열(대/소문자 구분 안 함) 중 하나일 수 있습니다. <ul style="list-style-type: none"> <li>■ nodejs</li> <li>■ python2</li> <li>■ python3</li> <li>■ shell</li> </ul> 아무 것도 제공하지 않으면 shell이 기본값으로 간주됩니다.
code	사용자 지정 작업의 일부로 실행할 사용자 지정 비즈니스 논리입니다.
inputProperties	사용자 지정 작업 구성의 일부로 캡처할 입력 속성 어레이입니다. 이러한 속성은 일반적으로 코드에서 사용됩니다.
outputProperties	사용자 지정 작업에서 내보내 파이프라인에 전파할 수 있는 출력 속성 어레이입니다.

2 사용 가능한 데이터 유형과 메타 데이터를 사용하여 스크립트에서 입력 속성을 선언합니다.

입력 속성은 스크립트에 대한 컨텍스트로 YAML의 code: 섹션에 전달됩니다.

사용자 지정 작업 YAML 입력 키	설명	필수
<b>type</b>	렌더링할 입력 유형: <ul style="list-style-type: none"> <li>■ <b>text</b></li> <li>■ <b>textarea</b></li> <li>■ <b>number</b></li> <li>■ <b>checkbox</b></li> <li>■ <b>password</b></li> <li>■ <b>select</b></li> </ul>	예
<b>name</b>	사용자 지정 작업에 대한 입력의 이름 또는 문자열. 사용자 지정 통합 YAML 코드에 삽입됩니다. 사용자 지정 통합에 대해 정의된 각 입력 속성에 대해 고유해야 합니다.	예
<b>title</b>	파이프라인 모델 캔버스에서 표시되는 사용자 지정 작업 입력 속성의 텍스트 문자열 레이블입니다. 비워 두면 <b>name</b> 이 기본적으로 사용됩니다.	아니요
<b>required</b>	사용자가 사용자 지정 작업을 구성할 때 입력 속성을 반드시 입력해야 하는지 여부를 결정합니다. <b>true</b> 또는 <b>false</b> 로 설정합니다. <b>true</b> 로 설정하면 사용자가 파이프라인 캔버스에서 사용자 지정 작업을 구성할 때 값을 제공하지 않을 경우 작업은 구성되지 않은 상태로 유지됩니다.	아니요
<b>placeholder</b>	값이 없는 경우 입력 속성의 입력 영역에 표시되는 기본 텍스트입니다. <b>html placeholder</b> 특성에 매핑됩니다. 특정 입력 속성 유형에 대해서만 지원됩니다.	아니요
<b>defaultValue</b>	파이프라인 모델 페이지에서 사용자 지정 작업을 렌더링할 때 입력 속성의 입력 영역을 채우는 기본값입니다.	아니요
<b>bindable</b>	파이프라인 캔버스에서 사용자 지정 작업을 모델링할 때 입력 속성이 달러 기호 변수를 수락하는지 여부를 결정합니다. 제목 옆에 <b>\$</b> 표시기를 추가합니다. 특정 입력 속성 유형에 대해서만 지원됩니다.	아니요
<b>labelMessage</b>	사용자를 위한 도움말 도구 설명으로 작동하는 문자열입니다. 입력 제목 옆에 도구 설명 아이콘 <b>i</b> 를 추가합니다.	아니요
<b>enum</b>	입력 속성 선택 옵션을 표시하는 값의 어레이를 취합니다. 특정 입력 속성 유형에 대해서만 지원됩니다.  사용자가 옵션을 선택하고 사용자 지정 작업에 대해 저장하면 <b>inputProperty</b> 값이 이 값에 해당하고 사용자 지정 작업 모델링에 표시됩니다.  예를 들어 값이 2015입니다. <ul style="list-style-type: none"> <li>■ 2015</li> <li>■ 2016</li> <li>■ 2017</li> <li>■ 2018</li> <li>■ 2019</li> <li>■ 2020</li> </ul>	아니요

사용자 지정 작업 YAML 입력 키	설명	필수
<b>options</b>	<p><b>optionKey</b> 및 <b>optionValue</b>을 사용하여 개체 어레이를 취합니다.</p> <ul style="list-style-type: none"> <li>■ <b>optionKey</b>. 작업의 코드 섹션에 전파되는 값입니다.</li> <li>■ <b>optionValue</b>. 사용자 인터페이스에 옵션을 표시하는 문자열입니다.</li> </ul> <p>특정 입력 속성 유형에 대해서만 지원됩니다.</p> <p>옵션:</p> <p><b>optionKey</b>: key1. 사용자 지정 작업을 위해 선택하고 저장하면 이 <b>inputProperty</b>의 값이 코드 섹션의 <b>key1</b>에 해당합니다.</p> <p><b>optionValue</b>: '1에 대한 레이블'. <b>key1</b>의 값이 사용자 인터페이스에 표시되고 사용자 인터페이스에 사용자 지정 작업의 다른 위치에는 표시되지 않습니다.</p> <p><b>optionKey</b>: key2</p> <p><b>optionValue</b>: '2에 대한 레이블'</p> <p><b>optionKey</b>: key3</p> <p><b>optionValue</b>: '3에 대한 레이블'</p>	아니요
<b>minimum</b>	이 입력 속성에 유효한 최소값으로 작동하는 숫자를 취합니다. 숫자 유형 입력 속성에 대해서만 지원됩니다.	아니요
<b>maximum</b>	이 입력 속성에 유효한 최대값으로 작동하는 숫자를 취합니다. 숫자 유형 입력 속성에 대해서만 지원됩니다.	아니요

표 5-6. 사용자 지정 스크립트에서 지원되는 데이터 유형 및 메타데이터

지원되는 데이터 유형	입력으로 지원되는 메타데이터
<ul style="list-style-type: none"> <li>■ String</li> <li>■ Text</li> <li>■ List: 임의 유형의 목록으로</li> <li>■ Map: map[string]any로</li> <li>■ Secure: 사용자 지정 작업을 저장할 때 암호화된 암호 텍스트 상자로 렌더링됨</li> <li>■ Number</li> <li>■ Boolean: 텍스트 상자로 나타남</li> <li>■ URL: 문자열과 동일, 추가 검증 포함</li> <li>■ 선택 항목, 라디오 버튼</li> </ul>	<ul style="list-style-type: none"> <li>■ type: String   Text 중 하나...</li> <li>■ default: 기본값</li> <li>■ options: 옵션 목록 또는 맵, 선택 항목 또는 라디오 버튼과 함께 사용됨</li> <li>■ min: 최소값 또는 최소 크기</li> <li>■ max: 최대값 또는 최대 크기</li> <li>■ title: 텍스트 상자의 세부적인 이름</li> <li>■ placeHolder: UI 자리 표시자</li> <li>■ description: 툴팁이 됨</li> </ul>

예를 들면 다음과 같습니다.

```
inputProperties:
  - name: message
    type: text
    title: Message
    placeHolder: Message for Slack Channel
    defaultValue: Hello Slack
    bindable: true
    labelInfo: true
    labelMessage: This message is posted to the Slack channel link provided in the
code
```

### 3 스크립트에서 출력 속성을 선언합니다.

스크립트는 출력에 대한 컨텍스트를 선언한 스크립트의 비즈니스 논리 **code:** 섹션에서 출력 속성을 캡처합니다.

파이프라인이 실행될 때 작업 출력에 대한 응답 코드를 입력할 수 있습니다. 예를 들어 **200**을 입력할 수 있습니다.

Code Stream이 각 **outputProperty**에 대해 지원하는 키.

키	설명
type	현재 <b>label</b> 의 단일 값이 포함되어 있습니다.
name	사용자 지정 통합 YAML의 코드 블록이 내보내는 키입니다.
title	<b>outputProperty</b> 를 표시하는 사용자 인터페이스의 레이블입니다.

예를 들면 다음과 같습니다.

```
outputProperties:
  - name: statusCode
    type: label
    title: Status Code
```

### 4 사용자 지정 스크립트의 입력 및 출력과 상호 작용하기 위해 **context**를 사용하여 입력 속성을 가져오거나 출력 속성을 설정합니다.

입력 속성의 경우: `(context.getInput("key"))`

출력 속성의 경우: `(context.setOutput("key", "value"))`

Node.js의 경우:

```
var context = require("../context.js")
var message = context.getInput("message");
//Your Business logic
context.setOutput("statusCode", 200);
```

Python의 경우:

```
from context import getInput, setOutput
message = getInput('message')
//Your Business logic
setOutput('statusCode', '200')
```

셸의 경우:

```
# Input, Output properties are environment variables
echo ${message} # Prints the input message
//Your Business logic
export statusCode=200 # Sets output property statusCode
```

## 5 code: 섹션에서 사용자 지정 통합을 위한 모든 비즈니스 논리를 선언합니다.

예를 들어 Node.js 런타임 환경에서 다음과 같이 합니다.

```
code: |
    var https = require('https');
    var context = require("./context.js")

    //Get the entered message from task config page and assign it to message var
    var message = context.getInput("message");
    var slackPayload = JSON.stringify(
        {
            text: message
        });

    const options = {
        hostname: 'hooks.slack.com',
        port: 443,
        path: '/YOUR_SLACK_WEBHOOK_PATH',
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
            'Content-Length': Buffer.byteLength(slackPayload)
        }
    };

    // Makes a https request and sets the output with statusCode which
    // will be displayed in task result page after execution
    const req = https.request(options, (res) => {
        context.setOutput("statusCode", res.statusCode);
    });

    req.on('error', (e) => {
        console.error(e);
    });
    req.write(slackPayload);
    req.end();
```

## 6 사용자 지정 통합 스크립트에 버전을 적용하고 릴리스하기 전에 Python 또는 Node.js의 컨텍스트 파일을 다운로드하고 스크립트에 포함된 비즈니스 논리를 테스트합니다.

- a 스크립트에 포인터를 놓은 다음, 캔버스 상단에서 컨텍스트 파일 버튼을 클릭합니다. 예를 들어 Python 스크립트의 경우 **CONTEXT.PY**를 클릭합니다.
- b 파일을 수정하고 저장합니다.
- c 개발 시스템에서 컨텍스트 파일을 사용하여 사용자 지정 스크립트를 실행하고 테스트합니다.

## 7 사용자 지정 통합 스크립트에 버전을 적용합니다.

- a **버전**을 클릭합니다.
- b 버전 정보를 입력합니다.

- c 사용자 지정 작업에서 스크립트를 선택할 수 있도록 **릴리스 버전**을 클릭합니다.
- d 버전을 생성하려면 **생성**을 클릭합니다.

## 버전 생성

버전 *	1.0
설명	New
변경 로그	New for 1.0
버전 릴리스	<input checked="" type="checkbox"/>

취소

생성

- 8 스크립트를 저장하려면 **저장**을 클릭합니다.

- 9 파이프라인에서 업무 공간을 구성합니다.

이 예에서는 Docker 업무 공간을 사용합니다.

- a **업무 공간** 탭을 클릭합니다.
- b Docker 호스트 및 빌더 이미지 URL을 선택합니다.

**Demo-customTask-nodejs** 사용함

**작업 공간** | 캐 입력 | 모델 | 출력

지속적 통합 작업을 실행할 컨테이너 및 호스트에 대한 세부 정보를 제공합니다.

호스트 *	Docker-saas
빌더 이미지 URL *	node:latest
이미지 레지스트리	--이미지 레지스트리 끝점 선택--
작업 디렉토리	
캐시	<input checked="" type="checkbox"/>
CPU 제한 *	
메모리 제한 *	
Git 클론	<input type="checkbox"/>

④ 이 파이프라인이 Webhook를 통해 Git에 연결되면 Git 이벤트에서 파이프라인이 트리거됩니다.

C 작업의 경우 연결된 Git 저장소(Git Webhook 매개 변수의 세부 정보)이(가) 업무 공간에 자동으로 복제됩니다.

- 10 파이프라인에 사용자 지정 작업을 추가하고 구성합니다.

- a **모델** 탭을 클릭합니다.
- b 작업을 추가하고, 유형을 **사용자 지정**으로 선택하고, 적절한 이름을 입력합니다.

- c 사용자 지정 통합 스크립트 및 버전을 선택합니다.
- d Slack에 사용자 지정 메시지를 표시하려면 메시지 텍스트를 입력합니다.

입력하는 텍스트는 사용자 지정 통합 스크립트의 defaultValue를 재정의합니다. 예를 들면 다음과 같습니다.



- 11 파이프라인을 저장하고 사용하도록 설정합니다.
  - a **저장**을 클릭합니다.
  - b [파이프라인] 탭에서 **파이프라인 사용**을 클릭하여 원을 오른쪽으로 이동합니다.
- 12 파이프라인을 실행합니다.
  - a **실행**을 클릭합니다.
  - b 파이프라인 실행을 살펴봅니다.



- c 출력에 예상한 상태 코드, 응답 코드, 상태 및 선언된 출력이 포함되어 있는지 확인합니다.

출력 속성으로 **statusCode**를 정의했습니다. 예를 들어 **statusCode** 200은 Slack 계시가 성공했음을 나타낼 수 있으며 **responseCode** 0은 스크립트가 오류 없이 성공했음을 나타낼 수 있습니다.

- d 실행 로그의 출력을 확인하려면 **실행**을 클릭하고, 파이프라인 링크를 클릭하고, 작업을 클릭한 다음, 로깅된 데이터를 살펴봅니다. 예를 들면 다음과 같습니다.

The screenshot displays the execution details for a task named 'Task1' within a pipeline 'custom-int-demo #5'. The task is in a 'COMPLETED' state. The output section shows 'statusCode' as 200 and 'Response code' as 0. The logs section shows a command: 'node -r ./context.js app.js'.

Task name	Task1	<a href="#">VIEW OUTPUT JSON</a>
Type	Custom	
Status	<b>COMPLETED</b> Execution Completed.	
Duration	6s (12/21/2018 3:04 AM - 12/21/2018 3:04 AM)	
Continue on failure	<input type="checkbox"/>	
Execute task	<input checked="" type="radio"/> Always <input type="radio"/> On condition	
Output		
statusCode	200	
Response code	0	
Logs	<pre> 1 + node -r ./context.js app.js 2 3 </pre>	

[View Full Log](#)

- 13 오류가 발생하면 문제를 해결한 후 파이프라인을 다시 실행합니다.

예를 들어 기본 이미지에 파일 또는 모듈이 누락되어 있는 경우 누락된 파일이 포함된 다른 기본 이미지를 생성해야 합니다. 그런 다음 Docker 파일을 제공하고 파이프라인을 통해 이미지를 푸시합니다.

## 결과

축하합니다! Code Stream을 Slack 인스턴스에 연결하고 메시지를 Slack 채널에 게시하는 사용자 지정 통합 스크립트를 생성했습니다.

## 다음에 수행할 작업

소프트웨어 릴리스 수명 주기 자동화에서 Code Stream의 기능을 확장할 수 있도록 파이프라인에서 사용자 지정 작업 사용을 지원하는 사용자 지정 통합을 계속 생성합니다.

## 다음 작업에서 클라우드 템플릿 작업의 리소스 속성을 사용하는 방법

Code Stream에서 클라우드 템플릿 작업을 사용하는 경우, 일반적인 질문은 파이프라인의 후속 작업에서 해당 작업의 출력을 사용하는 방법입니다. 클라우드 시스템과 같은 클라우드 템플릿 작업의 출력을 사용하려면 클라우드 템플릿 작업의 배포 세부 정보에서 리소스 속성과 클라우드 시스템의 IP 주소를 찾는 방법을 알아야 합니다.

예를 들어 VMware Cloud Templates의 배포 세부 정보에는 클라우드 시스템 리소스와 해당 IP 주소가 포함됩니다. 파이프라인에서 클라우드 시스템 및 IP 주소를 변수로 사용하여 클라우드 템플릿 작업을 REST 작업에 바인딩할 수 있습니다.

클라우드 시스템의 IP 주소를 찾는 데 사용하는 방법은 일반적이지 않습니다. VMware Cloud Templates의 배포가 완료되어야 배포 세부 정보를 사용할 수 있기 때문입니다. 그런 다음 VMware Cloud Templates 배포의 리소스를 사용하여 파이프라인 작업을 바인딩할 수 있습니다.

- 파이프라인의 클라우드 템플릿 작업에 나타나는 리소스 속성은 Cloud Assembly의 VMware Cloud Templates에 정의되어 있습니다.
- 해당 클라우드 템플릿의 배포가 종료된 시기를 모를 수 있습니다.
- Code Stream의 클라우드 템플릿 작업은 배포가 완료된 후에만 VMware Cloud Templates의 출력 속성을 표시할 수 있습니다.

이 예는 애플리케이션을 배포하고 다양한 API를 호출하는 경우 특히 유용할 수 있습니다. 예를 들어 REST API와 Wordpress 애플리케이션을 배포하는 VMware Cloud Templates을 호출하는 클라우드 템플릿 작업을 사용하는 경우, 배포 세부 정보에서 배포된 시스템의 IP 주소를 찾고 API를 사용하여 테스트할 수 있습니다.

클라우드 템플릿 작업은 미리 입력 자동 채우기 세부 정보를 표시하여 변수 바인딩을 사용하도록 지원합니다. 변수를 바인딩하는 방법은 사용자가 결정합니다.

이 예에서는 다음 방법을 보여줍니다.

- 실행하여 성공한 파이프라인에서 클라우드 템플릿 작업에 대한 배포 세부 정보 및 리소스 속성을 찾습니다.
- 배포 세부 정보의 리소스 섹션에서 클라우드 시스템 IP 주소를 찾습니다.
- 파이프라인의 클라우드 템플릿 작업 후에 REST 작업을 추가합니다.

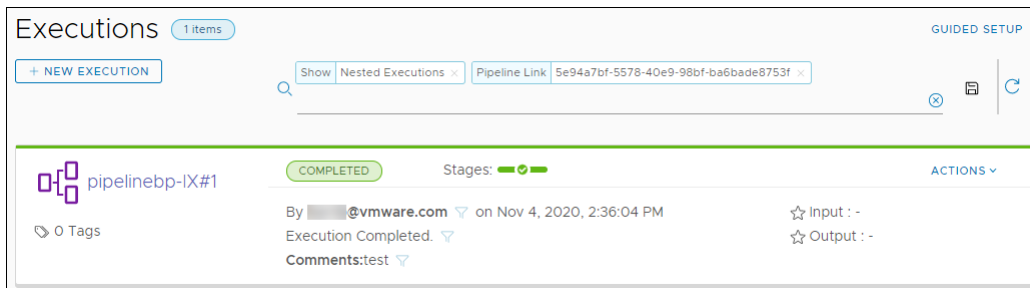
- REST 작업의 URL에 있는 클라우드 시스템 IP 주소를 사용하여 클라우드 템플릿 작업을 REST 작업에 바인딩합니다.
- 파이프라인을 실행하고 클라우드 템플릿 작업에서 REST 작업으로의 바인딩 작업을 살펴봅니다.

### 사전 요구 사항

- 버전이 지정된 작동 중인 VMware Cloud Templates이 있는지 확인합니다.
- VMware Cloud Templates 배포가 Cloud Assembly에서 성공했는지 확인합니다.
- 해당 VMware Cloud Templates을 사용하는 클라우드 템플릿 작업이 포함된 파이프라인이 있는지 확인합니다.
- 파이프라인이 실행되었고 성공했는지 확인합니다.

### 절차


- 1 파이프라인에 있는 클라우드 템플릿 작업 배포 세부 정보의 리소스 섹션에서 클라우드 시스템의 IP 주소를 찾습니다.
  - a **작업 > 실행 보기**를 클릭합니다.
  - b 성공한 파이프라인 실행에서 파이프라인 실행에 대한 링크를 클릭합니다.



- c 파이프라인 이름 아래에서 **작업**에 대한 링크를 클릭합니다.



d [출력] 영역에서 배포 세부 정보를 찾습니다.

 pipelinebp-IX #1 COMPLETED [ACTIONS](#)

Stage0

Task0

Task name	Task0 <a href="#">VIEW OUTPUT JSON</a>
Type	VMware cloud template
Status	COMPLETED
Message	Execution Completed.
Duration	0 milliseconds (Nov 4, 2020, 2:36:13 PM - Nov 4, 2020, 2:52:50 PM)
Precondition	-
Continue on failure	No

Output

Deployment

[deployment\\_c7185c47-1c12-40c5-9451-cbbbc4b16c89](#)

Deployment details

```

1 {
2   "id": "c7185c47-1c12-40c5-9451-cbbbc4b16c89",
3   "name": "deployment_c7185c47-1c12-40c5-9451-
4     -cbbbc4b16c89",
5   "description": "Pipeline Service triggered operation",
6   "orgId": "434f6917-4e34-4537-b6c8-3bf3638a71bc",
7   "blueprintId": "8d1dd801-3a32-4f3b-adde-27f8163dfe6f",
8   "blueprintVersion": "4",
9   "createdAt": "2020-11-04T21:36:14.500036Z",
10  "createdBy": "kernb@vmware.com",
11  "lastUpdatedAt": "2020-11-04T21:52:45.243028Z",
12  "lastUpdatedBy": "kernb@vmware.com",
13  "inputs": {},
14  "simulated": false,
15  "projectId": "267f8448-d26f-4b65-b310-9212adb3c455",
16  "resources": {
17    "Cloud_Machine_1[0]": {
18      "id": "/resources/compute/f5a846f3-c97c-4145-9e28
19        -951c36bd721c",
20      "name": "Cloud_Machine_1[0]",
21      "powerState": "ON".

```

Input

Action

Create Deployment

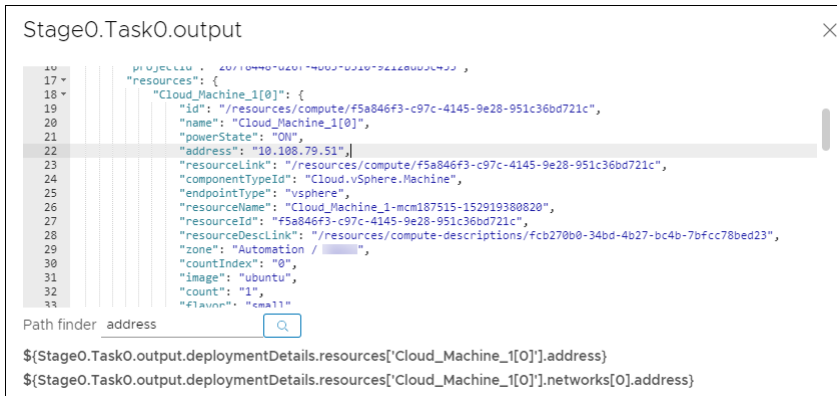
Cloud template

bhawesh

Cloud template version

4

- e 배포 세부 정보의 리소스 섹션에서 클라우드 시스템 이름을 찾습니다.  
REST 작업의 URL에 클라우드 시스템 이름에 대한 구문을 포함합니다.
- f 클라우드 템플릿 작업의 출력 속성에 대한 바인딩 표현식을 찾으려면 **VIEW OUTPUT JSON**을 클릭하고 주소 속성을 검색한 후 클라우드 시스템 IP 주소를 찾습니다.  
바인딩 표현식은 JSON 출력의 속성 및 검색 아이콘 아래에 표시됩니다.

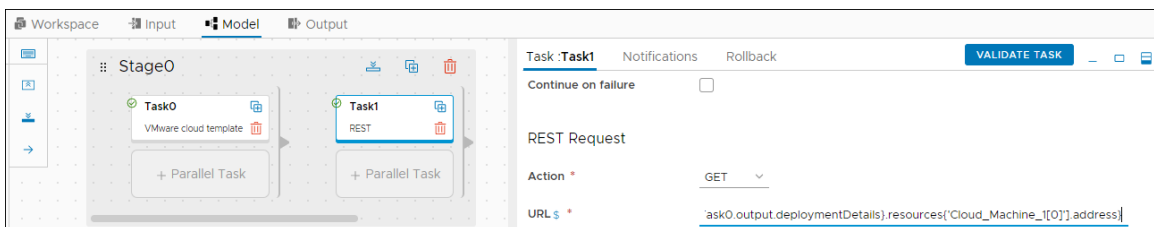


주소 리소스 속성은 클라우드 시스템 IP 주소를 표시합니다. 예를 들면 다음과 같습니다.

```
"resources": {
  "Cloud_Machine_1[0]": {
    "name": "Cloud_Machine_1[0]",
    "powerState": "ON",
    "address": "10.108.79.51",
    "resourceName": "Cloud_Machine_1-mcm187515-152919380820"
```

## 2 파이프라인 모델로 돌아가서 REST 작업에 URL을 입력합니다.

- a 작업 > 파이프라인 보기를 클릭합니다.
- b REST 작업을 클릭합니다.
- c REST 요청 URL 영역에서 \$를 입력하고 **스태이지, 작업, 출력, deploymentDetails**를 선택한 후 **리소스**를 입력합니다.  
자동 채우기를 사용하여 미리 입력할 수 있는 기능은 **리소스**를 입력해야 하는 시점까지 사용할 수 있습니다.
- d 배포 세부 정보에서 나머지 클라우드 시스템 리소스를 다음과 같이 입력합니다.  
**{'Cloud\_Machine\_1[0]'.address}**



클라우드 시스템 항목의 경우 표시된 대로 대괄호 표기법을 사용해야 합니다.

전체 URL 형식은 다음과 같습니다. \$

```
{Stage0.Task0.output.deploymentDetails.resources['Cloud_Machine_1[0]'].address}
```

- 3 파이프라인을 실행하고 REST 작업이 클라우드 템플릿 작업의 출력에서 클라우드 시스템 및 IP 주소를 테스트할 URL로 사용하는지 살펴봅니다.

## 결과

축하합니다! 클라우드 템플릿 작업의 JSON 출력 및 배포 세부 정보에서 클라우드 시스템 이름과 IP 주소를 찾은 후에 이를 사용하여 클라우드 템플릿 작업 출력을 파이프라인의 REST 작업 URL 입력에 바인딩했습니다.

## 다음에 수행할 작업

파이프라인의 다른 작업과 함께 클라우드 템플릿 작업의 리소스에서 바인딩 변수를 사용하여 계속 살펴봅니다.

# Code Stream을 다른 애플리케이션과 통합하기 위해 REST API를 사용하는 방법

Code Stream은 REST 플러그인을 제공합니다. 이 플러그인을 사용하면 Code Stream을 REST API를 사용하는 다른 애플리케이션과 통합하여 서로 상호 작용하는 소프트웨어 애플리케이션을 지속적으로 개발하고 전달할 수 있습니다. REST 플러그인은 API를 호출하고 API는 Code Stream과 다른 애플리케이션 간에 정보를 보내고 받습니다.

REST 플러그인을 사용하면 다음을 할 수 있습니다.

- 외부 REST API 기반 시스템을 Code Stream 파이프라인에 통합합니다.
- 외부 시스템 흐름의 일부로 Code Stream 파이프라인을 통합합니다.

REST 플러그인은 모든 REST API에서 작동하고 GET, POST, PUT, PATCH 및 DELETE 메서드를 지원하여 Code Stream과 다른 애플리케이션 간에 정보를 보내고 받습니다.

## 표 5-7. REST API를 통해 통신하기 위한 파이프라인 준비

수행할 작업	발생하는 결과
파이프라인에 REST 작업을 추가합니다.	REST 작업이 애플리케이션 간의 정보를 전달하고 파이프라인 단계에서 다음 작업에 대한 상태 정보를 제공할 수 있습니다.
REST 작업에서 REST 작업을 선택하고 URL을 포함합니다.	파이프라인이 실행될 때 파이프라인 작업이 URL을 호출합니다. POST, PUT 및 PATCH 작업의 경우 페이로드를 포함해야 합니다. 페이로드에서, 파이프라인이 실행될 때 파이프라인과 작업 속성을 바인딩할 수 있습니다.
이 예를 고려합니다.	REST 플러그인 사용 예: REST 작업을 추가하여 빌드의 Git 커밋에 대한 태그를 생성하고, 작업이 저장소에서 체크인 ID 가져오기 요청을 게시하도록 할 수 있습니다. 작업은 페이로드를 저장소로 보내고 빌드에 대한 태그를 생성할 수 있으며 저장소는 태그가 포함된 응답을 반환할 수 있습니다.

REST 플러그인을 사용하여 API를 호출하는 것과 유사하게, 파이프라인에 폴링 작업을 포함하여 REST API를 호출한 다음 REST API가 완료되고 파이프라인 작업이 종료 조건을 충족할 때까지 REST API를 폴링합니다.

또한 REST API를 사용하여 파이프라인을 가져오거나 내보낼 수 있으며, 예제 스크립트를 사용하여 파이프라인을 실행할 수도 있습니다.

이 절차는 간단한 URL을 가져옵니다.

### 절차

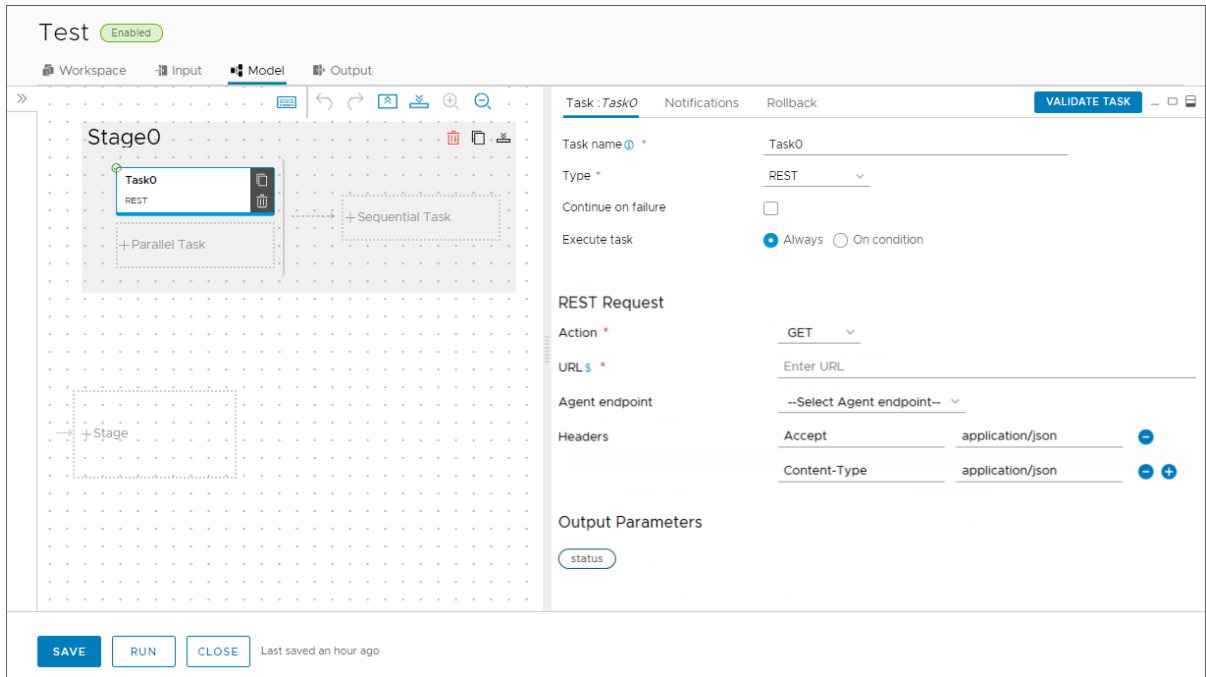
- 1 파이프라인을 생성하려면 **파이프라인 > 새 파이프라인 > 빈 캔버스**를 클릭합니다.
- 2 파이프라인 단계에서 **+ 순차적 작업**을 클릭합니다.
- 3 작업 창에서 REST 작업을 추가합니다.
  - a 작업의 이름을 입력합니다.
  - b [유형] 드롭다운 메뉴에서 **REST**를 선택합니다.
  - c [REST 요청] 영역에서 **GET**을 선택합니다.

REST 작업이 다른 애플리케이션의 데이터를 요청하도록 하려면 GET 메서드를 선택합니다. 데이터를 다른 애플리케이션으로 보내려면 POST 메서드를 선택합니다.

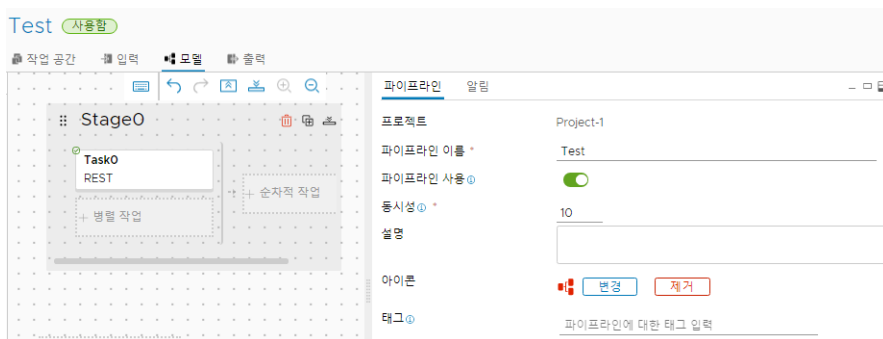
- d REST API 끝점을 식별하는 URL을 입력합니다. 예를 들어 `https://www.google.com`이 표시됩니다.

REST 작업이 다른 애플리케이션의 데이터를 가져오도록 페이로드 변수를 포함할 수 있습니다. 예를 들어 가져오기 작업에 대해 `${Stage0.export.responseBody}`를 입력할 수 있습니다. 응답 데이터 크기가 5MB를 초과하는 경우 REST 작업이 실패할 수 있습니다.

- e 작업에 대한 권한 부여를 제공하려면 **헤더 추가**를 클릭하고 헤더 키 및 값을 입력합니다.

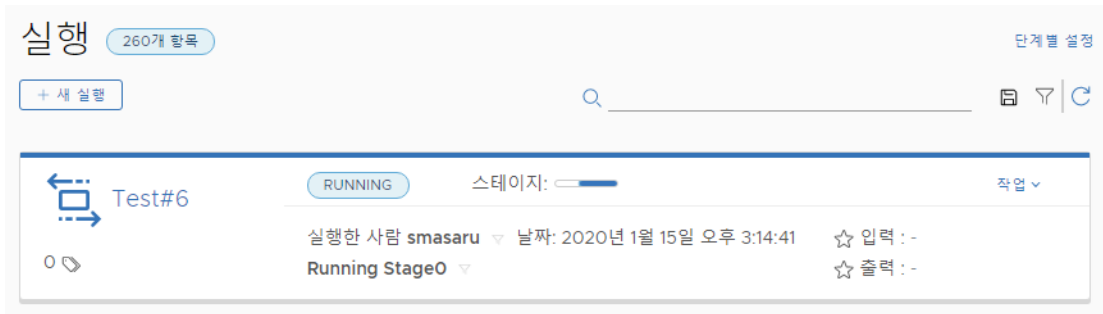


- 4 파이프라인을 저장하려면 **저장**을 클릭합니다.
- 5 [파이프라인] 탭에서 **파이프라인 사용**을 클릭합니다.



- 6 **저장**을 클릭한 다음 **닫기**를 클릭합니다.
- 7 **실행**을 클릭합니다.

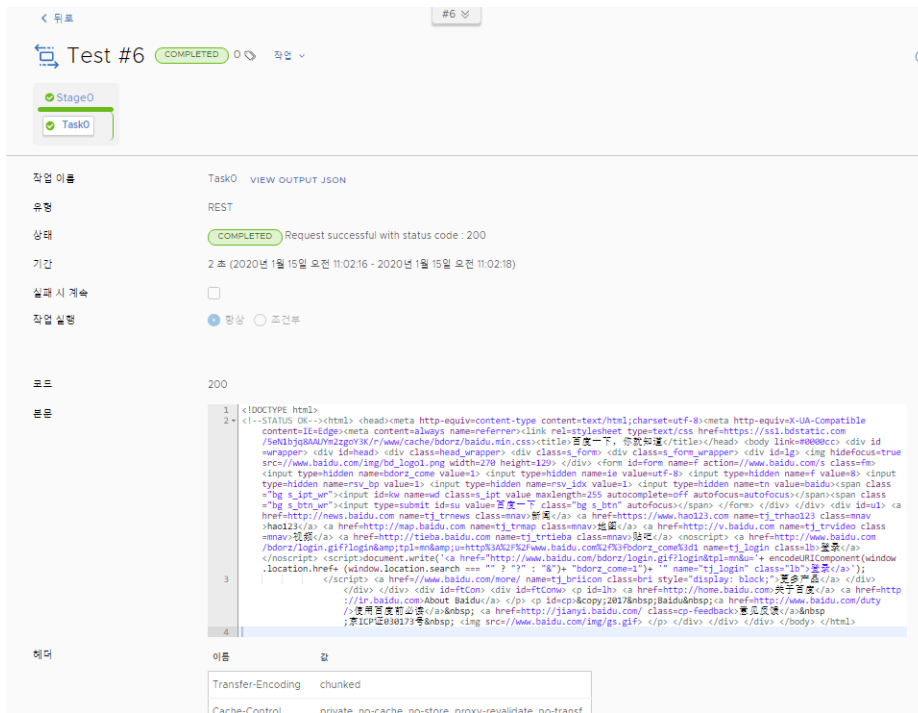


8 파이프라인 실행을 보려면 **실행**을 클릭합니다.

## 9 예상한 정보를 REST 작업이 반환하는지 확인하려면 파이프라인 실행과 작업 결과를 검토합니다.

- 파이프라인 완료 후 다른 애플리케이션이 사용자가 요청한 데이터를 반환했는지 확인하려면 파이프라인 실행에 대한 링크를 클릭합니다.
- 파이프라인에서 REST 작업을 클릭합니다.
- 파이프라인 실행에서 작업을 클릭하고, 작업 세부 정보를 살펴보고, REST 작업이 예상된 결과를 반환했는지 확인합니다.

작업 세부 정보에는 응답 코드, 본문, 헤더 키 및 값이 표시됩니다.



10 JSON 출력을 보려면 **VIEW OUTPUT JSON**을 클릭합니다.

```

1  {
2    "responseHeaders": {
3      "X-Frame-Options": "SAMEORIGIN",
4      "Transfer-Encoding": "chunked",
5      "Cache-Control": "private, max-age=0",
6      "Server": "gws",
7      "Alt-Svc": "quic=\":443\"; ma=2592000; v=\":44,43,39,35\"",
8      "Set-Cookie": "NID=148
9        =RTUkVjVhyg9KvAZR1S8yCCSEw8WosYfn9WMDfQ1N5fnd5DavrXUm5B3J8PyKMX1Z_zRNp3usXttMpd7YiqRUOSfMKTC7cTERbd
10       UmOnj3cTppHe3PHIXJPGHnTSZEweb3cXtjVIhVolS85ezVXaTSRYFcgo8_XIHZ8kqB8uwl1aE; expires=Tue, 28-May-2019
11       22:45:06 GMT; path=/; domain=.google.com; HttpOnly",
12      "Expires": "-1",
13      "P3P": "CP=\\\"This is not a P3P policy! See g.co/p3phelp for more info.\\\"",
14      "X-XSS-Protection": "1; mode=block",
15      "Date": "Mon, 26 Nov 2018 22:45:06 GMT",
16      "Content-Type": "text/html; charset=ISO-8859-1"
17    },
18    "responseBody": "<!doctype html><html itemscope=\\\"\\\" itemtype=\\\"http://schema.org/WebPage\\\" lang=\\\"en-IN\\\"
19      ><head><meta content=\\\"text/html; charset=UTF-8\\\" http-equiv=\\\"Content-Type\\\"><meta content=\\\"/images
20      /branding/googleg/1x/googleg_standard_color_128dp.png\\\" itemprop=\\\"image\\\"><title>Google</title><script
21      nonce=\\\"aNMw/ydugkGr9CHU6QQGzg=\\\">(function(){window.google={kEI:'cnf8W6KpJIEVkwXx-aLoDA',kEXPI:'0
22      ,1353747,57,50,1150,454,303,1017,1120,286,698,527,730,142,184,293,132,278,420,350,30,524,27,275,401,457
23      ,110,114,56,164,2336158,235,32,45,23,6,1,329219,1294,12383,4855,19577,13114,8163,7085,867,6056,636,2239
24      ,3232,5281,1100,3335,2,2,4605,2196,369,1212,2102,4133,1372,224,887,1331,260,1028,2714,1367,573,835,284
25      ,2,579,727,612,1820,58,2,2,189,1108,1712,28,2584,402,1693,664,630,8,300,1270,773,276,1230,609,134,978
26      ,430,2487,850,525,22,599,5,2,2,1963,528,3,1959,105,465,556,905,1378,966,942,108,334,130,1190,154,386,8
27      ,1003,81,7,3,25,463,620,29,989,406,458,1847,93,676,536,427,269,1456,1,2833,313,876,412,2,557,73,1483
28      ,698,59,318,273,108,167,323,744,101,1119,38,363,557,438,135,145,155,497,2,718,383,978,487,47,1080,901
29      ,387,422,659,359,8,59,32,416,283,9,1,211,2,460,25,60,386,282,528,307,2,67,30,13,1,255,122,143,217,37
30      ,628,255,1,1125,264,28,7,2,479,241,129,43,200,188,481,709,29,57,201,337,65,97,167,82,247,109,1049,14
31    }
32  }

```

## 결과

축하합니다! REST API를 호출하고 REST 플러그인을 사용하여 Code Stream과 다른 애플리케이션 간에 정보를 보낸 REST 작업을 구성했습니다.

## 다음에 수행할 작업

소프트웨어 애플리케이션을 개발하고 전달할 수 있도록 파이프라인에서 계속 REST 작업을 사용하여 명령을 실행하고 Code Stream과 다른 애플리케이션을 통합합니다. API가 완료되고 파이프라인 작업이 종료 조건을 충족할 때까지 API를 폴링하는 폴링 작업을 사용하는 것이 좋습니다.

## Code Stream에서 파이프라인을 코드로 활용하는 방법

DevOps 관리자 또는 개발자는 사용자 인터페이스를 사용하는 대신 YAML 코드를 사용하여 Code Stream에서 파이프라인을 생성하려고 할 수 있습니다. 파이프라인을 코드로 생성하면 코드 편집기를 사용하여 파이프라인 코드에 주석을 삽입할 수 있습니다.

파이프라인 코드에서 환경 변수 및 보안 자격 증명과 같은 외부 구성을 참조할 수 있습니다. 파이프라인 코드에서 사용하는 변수를 업데이트할 때 파이프라인 코드를 업데이트하지 않고 변수를 업데이트할 수 있습니다.

파이프라인 YAML 코드를 템플릿으로 사용하여 다른 파이프라인을 복제 및 생성하고 템플릿을 다른 사용자와 공유할 수 있습니다.

소스 제어 저장소에 파이프라인 코드 템플릿을 저장할 수 있고 이를 통해 버전을 관리하고 업데이트를 추적할 수 있습니다. 소스 제어 시스템을 사용하여 파이프라인 코드를 쉽게 백업하고 필요한 경우 복원할 수 있습니다.

### 사전 요구 사항

- 코드 편집기가 있는지 확인합니다.
- 소스 제어 저장소에 파이프라인 코드를 저장하려는 경우 작업 인스턴스에 액세스할 수 있는지 확인합니다.

### 절차

- 1 코드 편집기에서 파일을 생성합니다.
- 2 샘플 파이프라인 코드를 복사하여 붙여 넣고 특정 파이프라인 요구 사항을 반영하도록 업데이트합니다.
- 3 파이프라인 코드에 끝점을 포함하려면 예제 끝점 코드를 복사하여 붙여 넣은 후 끝점을 반영하도록 업데이트합니다.

파이프라인 업무 공간에서 Kubernetes API 끝점을 사용하는 경우 Code Stream은 CI(지속적 통합) 작업 또는 사용자 지정 작업을 실행하는 데 필요한 Kubernetes 리소스(예: ConfigMap, Secret 및 포트)를 생성합니다. Code Stream은 NodePort를 사용하여 컨테이너와 통신합니다.

Code Stream 파이프라인 업무 공간은 지속적 통합 작업 및 사용자 지정 작업을 위해 Docker 및 Kubernetes를 지원합니다.

업무 공간 구성에 대한 자세한 내용은 [파이프라인 작업 공간 구성](#)에서 참조하십시오.

- 4 코드를 저장합니다.
  - 5 파이프라인 코드를 저장하고 버전을 관리하려면 소스 제어 저장소에 해당 코드를 체크인합니다.
  - 6 지속적 통합 및 전달 파이프라인을 생성하는 경우 Kubernetes YAML 파일을 가져와야 합니다.
- Kubernetes YAML 파일을 가져오려면 스마트 파이프라인 템플릿의 지속적 전달 영역에서 파일을 선택하고 **프로세스**를 클릭합니다. 또는 API를 사용합니다.

### 결과

코드 예를 사용하여 파이프라인과 끝점을 나타내는 YAML 코드를 생성했습니다.

## 예제: 파이프라인 및 끝점에 대한 YAML 코드 예

이 YAML 코드 예에는 파이프라인의 Code Stream 기본 구축, 단계, 작업, 알림 등에 대한 업무 공간을 나타내는 섹션이 포함되어 있습니다.

지원되는 플러그인에 대한 코드 예는 [장 6 끝점에 Code Stream 연결](#) 항목을 참조하십시오.

```
---
kind: PIPELINE
name: myPipelineName
tags:
```

```

- tag1
- tag2

# Ready for execution
enabled: false

#Max number of concurrent executions
concurrency: 10

#Input Properties
input:
  input1: '30'
  input2: 'Hello'

#Output Properties
output:
  BuildNo: '${Dev.task1.buildNo}'
  Image: '${Dev.task1.image}'

#Workspace Definition
ciWorkspace:
  image: docker:maven-latest
  path: /var/tmp
  endpoint: my-k8s
  cache:
    - ~/.m2

# Starred Properties
starred:
  input: input1
  output: output1

# Stages in order of execution
stageOrder:
  - Dev
  - QA
  - Prod

# Task Definition Section
stages:
  Dev:
    taskOrder:
      - Task1, Task6
      - Task2 Long, Task Long Long
      - Task5
    tasks:
      Task1:
        type: jenkins
        ignoreFailure: false
        preCondition: ''
        endpoints:
          jenkinsServer: myJenkins
        input:
          job: Add Two Numbers

```

```

    parameters:
      number1: 10
      number2: 20
  Task2:
    type: blah
    # repeats like Task1 above
QA:
  taskOrder:
    - TaskA
    - TaskB
  tasks:
    TaskA:
      type: ssh
      ignoreFailure: false
      preCondition: ''
      input:
        host: x.y.z.w
        username: abcd
        password: ${var.mypassword}
        script: >
          echo "Hello, remote server"
    TaskB:
      type: blah
      # repeats like TaskA above

# Notificatons Section
notifications:
  email:
    - stage: Dev #optional ; if not found - use pipeline scope
      task: Task1 #optional; if not found use stage scope
      event: SUCCESS
      endpoint: default
      to:
        - user@yourcompany.com
        - abc@yourcompany.com
      subject: 'Pipeline ${name} has completed successfully'
      body: 'Pipeline ${name} has completed successfully'

  jira:
    - stage: QA #optional ; if not found - use pipeline scope
      task: TaskA #optional; if not found use stage scope
      event: FAILURE
      endpoint: myJiraServer
      issuetype: Bug
      project: Test
      assignee: abc
      summary: 'Pipeline ${name} has failed'
      description: |-
        Pipeline ${name} has failed
        Reason - ${resultsText}

  webhook:
    - stage: QA #optional ; if not found - use pipeline scope
      task: TaskB #optional; if not found use stage scope
      event: FAILURE
      agent: my-remote-agent

```

```

url: 'http://www.abc.com'
headers: #requestHeaders: '{"build_no":"123","header2":"456"}'
  Content-Type: application/json
  Accept: application/json
payload: |-
  Pipeline ${name} has failed
  Reason - ${resultsJson}
---
```

이 YAML 코드는 Jenkins 끝점 예를 나타냅니다.

```

---
name: My-Jenkins
tags:
- My-Jenkins
- Jenkins
kind: ENDPOINT
properties:
  offline: true
  pollInterval: 15.0
  retryWaitSeconds: 60.0
  retryCount: 5.0
  url: http://urlname.yourcompany.com:8080
description: Jenkins test server
type: your.jenkins:JenkinsServer
isLocked: false
---
```

이 YAML 코드는 Kubernetes 끝점 예를 나타냅니다.

```

---
name: my-k8s
tags: [
]
kind: ENDPOINT
properties:
  kubernetesURL: https://urlname.examplelocation.amazonaws.com
  userName: admin
  password: encryptedpassword
description: ''
type: kubernetes:KubernetesServer
isLocked: false
---
```

### 다음에 수행할 작업

파이프라인을 실행하고 필요에 따라 조정을 수행합니다. 파이프라인을 실행하고 결과를 보는 방법 항목을 참조하십시오.

# 끝점에 Code Stream 연결

# 6

Code Stream은 플러그인을 통해 개발 도구와 통합됩니다. 지원되는 플러그인으로는 Jenkins, Bamboo, vRealize Operations, Bugzilla, Team Foundation Server, Git 등이 있습니다.

Code Stream을 다른 개발 애플리케이션과 통합하는 자체 플러그인을 개발할 수도 있습니다.

Code Stream을 Jira와 통합하기 위해 외부 플러그인이 필요하지 않습니다. Code Stream에는 Jira 티켓 생성 기능이 알림 유형으로 포함되어 있기 때문입니다. 파이프라인 상태에 대한 Jira 티켓을 생성하려면 Jira 끝점을 추가해야 합니다.

본 장은 다음 항목을 포함합니다.

- Code Stream의 끝점 소개
- Code Stream과 Jenkins를 통합하는 방법
- Git와 Code Stream을 통합하는 방법
- Code Stream과 Gerrit을 통합하는 방법
- Code Stream과 vRealize Orchestrator를 통합하는 방법

## Code Stream의 끝점 소개

끝점은 Code Stream에 연결하고 데이터 소스, 저장소 또는 알림 시스템과 같이 파이프라인에서 실행할 데이터를 제공하는 DevOps 애플리케이션 인스턴스입니다.

Code Stream에서의 역할은 끝점을 사용하는 방법을 결정합니다.

- 관리자 및 개발자는 끝점을 생성하고, 업데이트하고, 삭제하고, 볼 수 있습니다.
- 관리자는 끝점을 제한된 리소스로 표시하고 제한된 끝점을 사용하는 파이프라인을 실행할 수 있습니다.
- 뷰어 역할이 있는 사용자는 끝점을 볼 수 있지만 생성, 업데이트하거나 삭제할 수는 없습니다.

자세한 내용은 [Code Stream에서 사용자 액세스 및 승인을 관리하는 방법](#) 항목을 참조하십시오.

Code Stream을 끝점에 연결하려면 다음 단계를 수행합니다.

- 1 파이프라인에 작업 추가
- 2 끝점과 통신하도록 작업을 구성합니다.

3 **검증**을 클릭하여 Code Stream이 끝점에 연결할 수 있는지 확인합니다.

4 그런 다음 파이프라인을 실행하면 작업이 끝점에 연결되어 작업을 실행할 수 있습니다.

이러한 끝점을 사용하는 작업 유형에 대한 자세한 내용은 [Code Stream에서 사용할 수 있는 작업 유형 항목](#)을 참조하십시오.

**표 6-1. Code Stream에서 지원하는 끝점**

끝점	기능	지원되는 버전	요구 사항
Bamboo	구축 계획을 생성합니다.	6.9.*	
Docker	기본 구축에서는 Docker 호스트를 사용할 수 있습니다.		파이프라인에 Docker Hub의 이미지가 포함된 경우 파이프라인을 실행하기 전에 이미지에 cURL 또는 wget이 내장되어 있는지 확인해야 합니다. 파이프라인이 실행되면 Code Stream은 cURL 또는 wget을 사용하여 명령을 실행하는 이전 파일을 다운로드합니다.
Docker 레지스트리	Docker 구축 호스트가 이미지를 가져올 수 있도록 컨테이너 이미지를 등록합니다.	2.7.1	
Gerrit	검토 및 트리거를 위해 Gerrit 서버에 연결합니다.	2.14.*	
Git	개발자가 코드를 업데이트하고 저장소에 체크인할 때 파이프라인을 트리거합니다.	Git Hub Enterprise 2.1.8 Git Lab Enterprise 11.9.12-ee	
Jenkins	코드 아티팩트를 구축합니다.	1.6.* 및 2.*	
Jira	파이프라인 작업이 실패할 때 Jira 티켓을 생성합니다.	8.3.*	
Kubernetes	컨테이너화된 애플리케이션을 배포, 확장 및 관리하는 단계를 자동화합니다.	Cloud Assembly 8.4 이상에서 지원되는 모든 버전 Cloud Assembly 8.3 이전의 경우 1.18	파이프라인 업무 공간에서 Kubernetes API 끝점을 사용하는 경우 Code Stream은 CI(지속적 통합) 작업 또는 사용자 지정 작업을 실행하는 데 필요한 Kubernetes 리소스(예: ConfigMap, Secret 및 포트)를 생성합니다. Code Stream은 NodePort를 사용하여 컨테이너와 통신합니다. 업무 공간 구성에 대한 자세한 내용은 <a href="#">파이프라인 작업 공간 구성</a> 에서 참조하십시오.
PowerShell	Windows 또는 Linux 시스템에서 PowerShell 스크립트를 실행하는 작업을 생성합니다.	4 및 5	
SSH	Windows 또는 Linux 시스템에서 SSH 스크립트를 실행하는 작업을 생성합니다.	7.0	



표 6-1. Code Stream에서 지원하는 끝점 (계속)

끝점	기능	지원되는 버전	요구 사항
TFS, Team Foundation Server	소스 코드, 자동화된 구축, 테스트 및 관련 작업을 관리합니다.	2015 및 2017	
vRealize Orchestrator	빌드 프로세스에서 워크플로를 정렬하고 자동화합니다.	7.* 및 8.*	

## GitHub 끝점에 대한 YAML 코드 예

이 YAML 코드 예는 Git 작업을 참조할 수 있는 GitHub 끝점을 정의합니다.

```
---
name: github-k8s
tags: [
]
kind: ENDPOINT
properties:
  serverType: GitHub
  repoURL: https://github.com/autouser/testrepok8s
  branch: master
  userName: autouser
  password: encryptedpassword
  privateToken: ''
description: ''
type: scm:git
isLocked: false
---
```

## Code Stream과 Jenkins를 통합하는 방법

Code Stream은 소스 코드를 구축하고 테스트하는 Jenkins 작업을 트리거하는 Jenkins 플러그인을 제공합니다. Jenkins 플러그인은 테스트 사례를 실행하며, 사용자 지정 스크립트를 사용할 수 있습니다.

파이프라인에서 Jenkins 작업을 실행하려면 Jenkins 서버를 사용하고 Code Stream에서 Jenkins 끝점을 추가합니다. 그런 다음 파이프라인을 생성하고 여기에 Jenkins 작업을 추가합니다.

Code Stream에서 Jenkins 작업 및 Jenkins 끝점을 사용하는 경우 Jenkins에서 다중 분기 작업을 지원하는 파이프라인을 생성할 수 있습니다. 다중 분기 작업은 Git 저장소의 각 분기에 있는 개별 작업을 포함합니다. 다중 분기 작업을 지원하는 파이프라인을 Code Stream에서 생성하는 경우:

- Jenkins 작업은 Jenkins 서버의 여러 폴더에 상주하는 Jenkins 작업을 실행할 수 있습니다.
- Jenkins 작업 구성의 폴더 경로를 재정의하여 다른 폴더 경로를 사용하게 할 수 있습니다. 이 경로는 Code Stream의 Jenkins 끝점에 정의된 기본 경로를 재정의합니다.
- Code Stream의 다중 분기 파이프라인은 Git 저장소 또는 GitHub 저장소에서 .groovy 유형의 Jenkins 작업 파일을 감지하고 저장소에서 검색하는 각 분기에 대한 작업 생성을 시작합니다.

- Jenkins 작업 구성에 제공된 경로로 Jenkins 끝점에 정의된 기본 경로를 재정의하고, 기본 Jenkins 작업 내 모든 분기와 연결된 작업 및 파이프라인을 실행할 수 있습니다.

#### 사전 요구 사항

- 버전 1.561 이상을 실행하는 Jenkins 서버를 설정합니다.
- Code Stream에서 프로젝트의 멤버인지 확인합니다. 멤버가 아니면 프로젝트 멤버로 추가해 달라고 Code Stream 관리자에게 요청합니다. [Code Stream에서 프로젝트를 추가하는 방법](#)의 내용을 참조하십시오.
- 파이프라인 작업에서 실행할 수 있도록 Jenkins 서버에 작업이 있는지 확인합니다.

#### 절차

- 1 Jenkins 끝점을 추가하고 검증합니다.
  - a **끝점 > 새 끝점**을 클릭합니다.
  - b 프로젝트를 선택하고 끝점의 유형으로 **Jenkins**를 선택합니다. 그런 다음 이름과 설명을 입력합니다.
  - c 이 끝점이 인프라에서 비즈니스에 중요한 구성 요소인 경우 **제한된 리소스로 표시**를 사용하도록 설정합니다.
  - d Jenkins 서버의 URL을 입력합니다.

- e Jenkins 서버 로그인을 위한 사용자 이름과 암호를 입력합니다. 그런 다음 나머지 정보를 입력합니다.

## 표 6-2. Jenkins 끝점에 대한 나머지 정보

끝점 항목	설명
폴더 경로	작업을 그룹화하는 폴더에 대한 경로입니다. Jenkins는 해당 폴더에 있는 모든 작업을 실행할 수 있습니다. 하위 폴더를 생성할 수 있습니다. 예를 들면 다음과 같습니다. <ul style="list-style-type: none"> <li>■ folder_1은 job_1을 포함할 수 있습니다.</li> <li>■ folder_1은 job_2를 포함할 수 있는 folder_2를 포함할 수 있습니다.</li> </ul> folder_1에 대한 끝점을 생성할 때 폴더 경로는 job/folder_1이고, 끝점은 job_1만 나열합니다. 하위 폴더 folder_2에 있는 작업의 목록을 얻으려면 폴더 경로를 /job/folder_1/job/folder_2/로 사용하는 다른 끝점을 생성해야 합니다.
다중 분기 Jenkins 작업의 폴더 경로	다중 분기 Jenkins 작업을 지원하려면 Jenkins 작업에서 Jenkins 서버 URL 및 전체 작업 경로를 포함하는 전체 경로를 입력합니다. Jenkins 작업에 폴더 경로를 포함하면 이 경로가 Jenkins 끝점에 나타나는 경로를 재정의합니다. Jenkins 작업의 사용자 지정 폴더 경로를 사용하면 Code Stream은 이 폴더에 있는 작업만 표시합니다. <ul style="list-style-type: none"> <li>■ 예: https://server.yourcompany.com/job/project</li> <li>■ 파이프라인에서 기본 Jenkins 작업도 트리거해야 하는 경우 https://server.yourcompany.com/job/project/job/main을 사용합니다.</li> </ul>
URL	Jenkins 서버의 호스트 URL입니다. URL을 protocol://host:port 형식으로 입력합니다. 예: http://192.10.121.13:8080
폴링 간격	Code Stream이 업데이트를 위해 Jenkins 서버를 폴링하는 간격입니다.
요청 재시도 횟수	Jenkins 서버에 대해 예약된 구축 요청을 재시도하는 횟수입니다.
재시도 대기 시간	Jenkins 서버에 대한 구축 요청을 재시도하기 전에 대기하는 시간(초)입니다.

- f **검증**을 클릭하고 끝점이 Code Stream에 연결되는지 확인합니다. 연결되지 않으면 오류를 수정한 다음 **저장**을 클릭합니다.

끝점 편집

프로젝트

test1

유형

Jenkins

이름 \*

aa

설명

제한 표시

☐ 제한되지 않음

URL \*

http(s)://<server\_url>:<port>

Username

username

Password

password

변수 생성

Folder Path

/job/DevFolder/

Poll Interval (sec) \*

15

Request Retries \*

5

Retry Wait Time ... \*

60

저장

검증

취소

2 코드를 구축하려면 파이프라인을 생성하고 Jenkins 끝점을 사용하는 작업을 추가합니다.

- a **파이프라인 > 새 파이프라인 > 빈 캔버스**를 클릭합니다.
- b 기본 단계를 클릭합니다.
- c [작업] 영역에서 작업의 이름을 입력합니다.
- d 작업 유형을 **Jenkins**로 선택합니다.
- e 생성한 Jenkins 끝점을 선택합니다.
- f 드롭다운 메뉴에서 파이프라인이 실행할 Jenkins 서버의 작업을 선택합니다.
- g 작업에 대한 매개 변수를 입력합니다.
- h Jenkins 작업에 대한 인증 토큰을 입력합니다.

**Build and Deploy** Enabled

Task: **Build** Notifications VALIDATE TASK

Task name: Build

Type: Jenkins

Continue On Failure: ☐

Execute Task: ☒ Always ☐ On Condition

Jenkins

Endpoint: aa

Job: add\_numbers

Num1: 22

Num2: 22


Token:

Output Parameters: status job jobId jobResults jobUri

**SAVE** **RUN** **CLOSE** Last saved a month ago

### 3 파이프라인을 사용하도록 설정하고 실행한 후 파이프라인 실행을 살펴봅니다.

[< BACK](#)


**Build and Deploy #28**
COMPLETED
0
ACTIONS ▾

Stage0

✓ Build

✓ Test

✓ Approval for Deployment

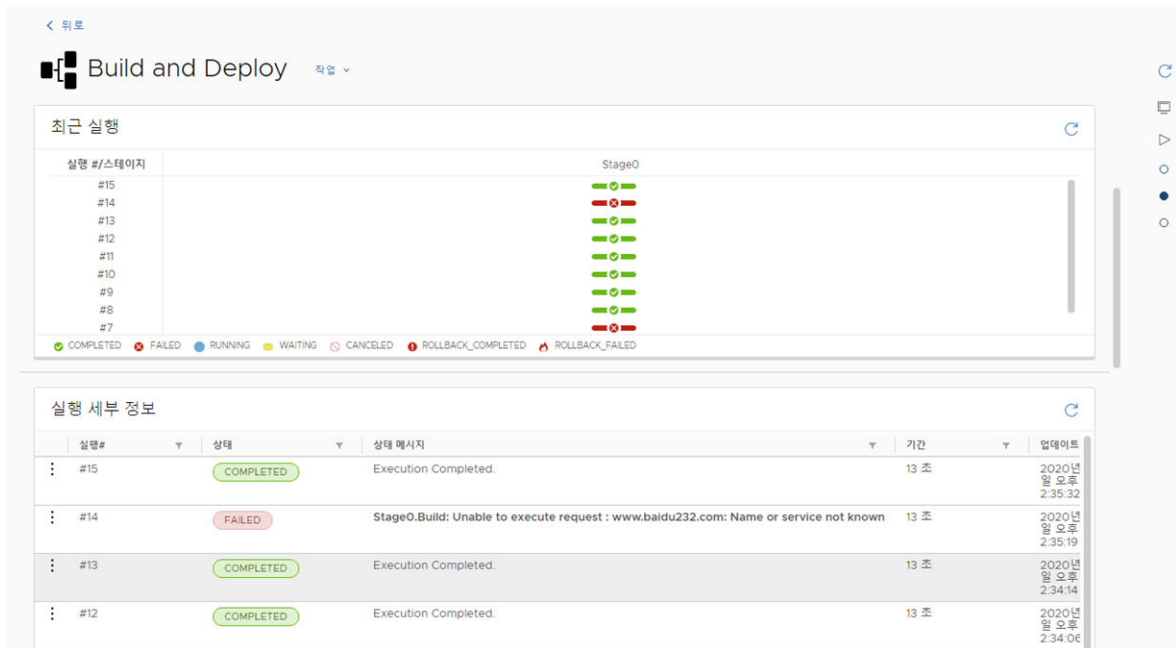
✓ Deployment

✓ Wait for application to start

Task name	Build <a href="#">VIEW OUTPUT JSON</a>														
Type	Jenkins														
Status	<span>COMPLETED</span> Execution Completed.														
Duration	11s (08/06/2018 12:27 AM - 08/06/2018 12:27 AM)														
Continue On Failure	<input type="checkbox"/>														
Execute Task	<input checked="" type="radio"/> Always <input type="radio"/> On Condition														
Jenkins Job															
Endpoint	aa														
Job Name	add_numbers														
Job ID	1428														
Job URL	<a href="http://.../job/add_numbers/1428/">http://.../job/add_numbers/1428/</a>														
Job Result	<table> <thead> <tr> <th>Key</th> <th>Value</th> </tr> </thead> <tbody> <tr> <td>junitResponse.failCount</td> <td>0</td> </tr> <tr> <td>junitResponse.skipCount</td> <td>0</td> </tr> <tr> <td>junitResponse.totalCount</td> <td>0</td> </tr> <tr> <td>junitResponse.successCount</td> <td>0</td> </tr> <tr> <td>jacocoResponse.lineCoverage</td> <td>0</td> </tr> <tr> <td>jacocoResponse.classCoverage</td> <td>0</td> </tr> </tbody> </table>	Key	Value	junitResponse.failCount	0	junitResponse.skipCount	0	junitResponse.totalCount	0	junitResponse.successCount	0	jacocoResponse.lineCoverage	0	jacocoResponse.classCoverage	0
Key	Value														
junitResponse.failCount	0														
junitResponse.skipCount	0														
junitResponse.totalCount	0														
junitResponse.successCount	0														
jacocoResponse.lineCoverage	0														
jacocoResponse.classCoverage	0														

#### 4 파이프라인 대시보드에서 실행 세부 정보 및 상태를 살펴봅니다.

모든 실패와 실패의 원인을 식별할 수 있습니다. 또한 파이프라인 실행 기간, 완료 및 실패에 대한 추세를 볼 수도 있습니다.



#### 결과

축하합니다! 끝점을 추가하고, 파이프라인을 생성하고, 코드를 구축하는 Jenkins 작업을 구성하여 Code Stream과 Jenkins를 통합했습니다.

### 예제: Jenkins 구축 작업을 위한 YAML 예

이 예에 사용된 Jenkins 구축 작업 유형의 경우 YAML은 다음 코드와 유사합니다(알림이 설정되어 있음).

```
test:
  type: Jenkins
  endpoints:
    jenkinsServer: jenkins
  input:
    job: Add two numbers
  parameters:
    Num1: '23'
    Num2: '23'
```

#### 다음에 수행할 작업

자세한 내용을 보려면 다른 섹션을 검토합니다. [장 6 끝점에 Code Stream 연결 항목](#)을 참조하십시오.

## Git와 Code Stream을 통합하는 방법

Code Stream에서는 GitHub, GitLab 또는 Bitbucket 저장소에서 코드 변경이 이루어질 경우 파이프라인을 트리거하는 방법을 제공합니다. Git 트리거에서 모니터링하려는 저장소의 분기에 Git 끝점을 사용해야 합니다. Code Stream은 Webhook를 통해 Git 끝점에 연결됩니다.

Code Stream에서 Git 끝점을 정의하려면 프로젝트를 선택하고 끝점이 위치할 Git 저장소의 분기를 입력합니다. 프로젝트를 통해 파이프라인에 끝점 및 다른 관련 개체가 함께 연결됩니다. Webhook 정의에서 프로젝트를 선택하면 트리거할 끝점 및 파이프라인이 선택됩니다.

---

**참고** 끝점을 사용하여 Webhook를 정의하고 나중에 끝점을 편집하는 경우에는 Webhook에서 끝점 세부 정보를 변경할 수 없습니다. 끝점 세부 정보를 변경하려면 끝점을 사용하여 Webhook를 삭제하고 재정의해야 합니다. [Code Stream에서 Git 트리거를 사용하여 파이프라인을 실행하는 방법](#)의 내용을 참조하십시오.

---

동일한 Git 끝점을 사용하고 Webhook 구성 페이지에서 분기 이름에 서로 다른 값을 제공하여 여러 분기에 대해 다수의 Webhook를 생성하고 할 수 있습니다. 동일한 Git 저장소의 다른 분기에 대해 또 다른 Webhook를 생성하기 위해, 여러 분기에 대해 Git 끝점을 여러 번 복제할 필요가 없습니다. 대신, Webhook에 분기 이름을 제공하면 Git 끝점을 재사용할 수 있습니다. Git Webhook의 분기가 끝점의 분기와 동일한 경우 Git Webhook 페이지에서 분기 이름을 제공하지 않아도 됩니다.

### 사전 요구 사항

- 연결하려는 GitHub, GitLab 또는 Bitbucket 저장소에 액세스할 수 있어야 합니다.
- Code Stream에서 프로젝트의 멤버인지 확인합니다. 그렇지 않으면 프로젝트 멤버로 추가해 달라고 Code Stream 관리자에게 요청합니다. [Code Stream에서 프로젝트를 추가하는 방법](#)의 내용을 참조하십시오.

### 절차

- 1 Git 끝점을 정의합니다.
  - a **끝점 > 새 끝점**을 클릭합니다.
  - b 프로젝트를 선택하고 끝점 유형으로 **Git**를 선택합니다. 그런 다음 이름과 설명을 입력합니다.

- c 이 끝점이 인프라에서 비즈니스에 중요한 구성 요소인 경우 **제한된 리소스로 표시**를 사용하도록 설정합니다.

파이프라인에서 제한된 끝점을 사용하는 경우 관리자는 파이프라인을 실행할 수 있으며 파이프라인 실행을 승인해야 합니다. 끝점 또는 변수가 제한된 것으로 표시되고 관리자가 아닌 사용자가 파이프라인을 트리거하면 해당 작업에서 파이프라인이 일시 중지되고 관리자가 재개할 때까지 기다립니다.

프로젝트 관리자는 사용자가 프로젝트 관리자인 프로젝트에 리소스가 있는 경우 제한된 끝점 또는 변수가 포함된 파이프라인을 시작할 수 있습니다.

관리자가 아닌 사용자가 제한된 리소스를 포함하는 파이프라인을 실행하려고 하면 제한된 리소스를 사용하는 작업에서 파이프라인이 중지됩니다. 그러면 관리자가 파이프라인을 재개해야 합니다.

제한된 리소스에 대한 자세한 내용 및 **제한된 파이프라인 관리**라는 사용 권한이 포함된 사용자 지정 역할에 대한 자세한 내용은 다음을 참조하십시오.

- [Code Stream에서 사용자 액세스 및 승인을 관리하는 방법](#)
- [장 2 Code Stream을 설정하여 릴리스 프로세스 모델링](#)

- d 지원되는 Git 서버 유형 중 하나를 선택합니다.

- e 경로에 서버의 API 게이트웨이가 있는 저장소의 URL을 입력합니다. 예를 들면 다음과 같습니다.

GitHub의 경우 `https://api.github.com/vmware-example/repo-example`을 입력합니다.

BitBucket의 경우 `https://api.bitbucket.org/{user}/{repo name}` 또는 `http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}`을 입력합니다.

- f 끝점이 위치한 저장소의 분기를 입력합니다.

- g 인증 유형을 선택하고 GitHub, GitLab 또는 BitBucket에 대한 사용자 이름을 입력합니다. 그런 다음 사용자 이름과 함께 사용되는 전용 토큰을 입력합니다.

- 암호. 나중에 Webhook을 생성하려면 암호에 대한 전용 토큰을 입력해야 합니다. Git에 대한 Webhook은 기본 인증을 사용하여 생성된 끝점을 지원하지 않습니다.

비밀 변수를 사용하여 중요한 정보를 숨기고 암호화합니다. 제한된 변수는 숨기고 암호화되어야 하는 문자열, 암호 및 URL에 사용하고, 실행에서 사용을 제한하기 위해 사용합니다. 예를 들어 암호 또는 URL에 대해 비밀 변수를 사용합니다. 파이프라인의 모든 작업 유형에서 비밀 변수와 제한된 변수를 사용할 수 있습니다.

- 전용 토큰. Git에만 해당되는 이 토큰을 사용하면 특정 작업에 액세스할 수 있습니다. [https://docs.gitlab.com/ee/user/profile/personal\\_access\\_tokens.html](https://docs.gitlab.com/ee/user/profile/personal_access_tokens.html) 항목을 참조하십시오. 전용 토큰에 대한 변수를 생성할 수도 있습니다.



## 2 검증

검증을 클릭하고 끝점이 Code Stream에 연결되는지 확인합니다.

연결되지 않을 경우 오류를 해결한 다음 **생성**을 클릭합니다.

**New endpoint**

Project \* test

Type \* GIT

Name \* DemoApp-Git

Description Git example branch

Mark restricted ☐ non-restricted

Git Server Type \* GitHub

Repo URL ⓘ \* https://api.github.com/vmware-example/repo-example

ACCEPT CERTIFICATE

Branch \* master

Authentication Type \* Password

Username \* ExampleUser

Password \* .....

CREATE VARIABLE

CREATE VALIDATE CANCEL

### 다음에 수행할 작업

자세한 내용을 보려면 다른 섹션을 검토합니다. [Code Stream에서 Git 트리거를 사용하여 파이프라인을 실행하는 방법](#) 항목을 참조하십시오.

## Code Stream과 Gerrit을 통합하는 방법

Code Stream을 사용하면 Gerrit 프로젝트에서 코드 검토가 발생할 때 파이프라인을 트리거 할 수 있습니다. Gerrit에 대한 트리거 정의에는 서로 다른 이벤트 유형에 대해 실행할 파이프라인과 Gerrit 프로젝트가 포함됩니다.

Gerrit에 대한 트리거는 모니터링하려는 Gerrit 서버에서 Gerrit 수신기를 사용합니다. Code Stream에서 Gerrit 끝점을 정의하려면 프로젝트를 선택하고 Gerrit 서버의 URL을 입력합니다. 그런 다음 이 서버에서 Gerrit 수신기를 생성할 때 끝점을 지정합니다.

FIPS를 사용하도록 설정된 vRealize Automation 인스턴스에서 Gerrit 서버를 Code Stream 끝점으로 사용하는 경우에는 Gerrit 구성 파일에 올바른 메시지 인증 키가 포함되어 있는지 확인해야 합니다. Gerrit 서버 구성 파일에 올바른 메시지 인증 키가 포함되어 있지 않으면 서버가 올바르게 시작되지 않고 다음 메시지가 표시됩니다. PrivateKey/PassPhrase is incorrect

### 사전 요구 사항

- 연결하려는 Gerrit 서버에 액세스할 수 있는지 확인합니다.
- Code Stream에서 프로젝트의 멤버인지 확인합니다. 멤버가 아니면 프로젝트 멤버로 추가해 달라고 Code Stream 관리자에게 요청합니다. [Code Stream에서 프로젝트를 추가하는 방법](#)의 내용을 참조하십시오.



### 절차

#### 1 Gerrit 끝점을 정의합니다.

- a **구성 > 끝점**을 클릭하고 **새 끝점**을 클릭합니다.
- b 프로젝트를 선택하고 끝점의 유형으로 **Gerrit**을 선택합니다. 그런 다음 이름과 설명을 입력합니다.
- c 이 끝점이 인프라에서 비즈니스에 중요한 구성 요소인 경우 **제한된 리소스로 표시**를 사용하도록 설정합니다.
- d Gerrit 서버의 URL을 입력합니다.  
기본 포트를 사용하려면 URL에 포트 번호를 제공하거나 값을 비워두면 됩니다.
- e Gerrit 서버의 사용자 이름과 암호를 입력합니다.  
암호를 암호화해야 하는 경우 **변수 생성**을 클릭하고 유형을 선택합니다.
  - 암호. 암호는 역할이 있는 사용자가 파이프라인을 실행할 때 확인됩니다.
  - 제한됨. 관리자 역할이 있는 사용자가 파이프라인을 실행할 때 암호가 확인됩니다.
 값에는 보안을 유지해야 하는 암호를 입력합니다(예: Jenkins 서버의 암호).
- f 개인 키의 경우 Gerrit 서버에 안전하게 액세스하는 데 사용되는 SSH 키를 입력합니다.  
이 키는 .ssh 디렉토리에 상주하는 RSA 개인 키입니다.
- g (선택 사항) 암호가 개인 키와 연결되어 있는 경우 암호를 입력합니다.  
암호를 암호화하려면 **변수 생성**을 클릭하고 유형을 선택합니다.
  - 암호. 암호는 역할이 있는 사용자가 파이프라인을 실행할 때 확인됩니다.
  - 제한됨. 관리자 역할이 있는 사용자가 파이프라인을 실행할 때 암호가 확인됩니다.
 값에는 보안을 유지해야 하는 암호를 입력합니다(예: SSH 서버의 암호).

- 2 **검증**을 클릭하고 Code Stream의 Gerrit 끝점이 Gerrit 서버에 연결되는지 확인합니다.  
연결되지 않으면 오류를 해결한 다음 **검증**을 다시 클릭합니다.

### 새 끝점

프로젝트 *	test
유형 *	Gerrit
이름 *	Gerrit-Demo-Endpoint
설명	<div></div>
제한 표시	<input type="checkbox"/> 제한되지 않음
클라우드 프로세서 *	기본값
URL *	http://example-gerrit.mycompany.com:8080
Username *	CS_user
Password *	.....  <a href="#">변수 생성</a>
Private Key *	<pre>-----BEGIN RSA PRIVATE KEY----- Proc-Type:4,ENCRYPTED DEK-Info:AES-128-CBC,FOOCEOB6526AF67DC77ADCD0962DBF92</pre>
Pass Phrase ⓘ	.....  <a href="#">변수 생성</a>

[생성](#)
[검증](#)
[취소](#)

- 3 **생성**을 클릭합니다.
- 4 vRealize Automation 환경에 FIPS가 사용되도록 설정되어 있는지 확인하거나, Jenkins 작업을 통해 Jenkins URL을 사용하여 FIPS를 사용하도록 설정된 환경을 생성하도록 합니다.
- 명령줄에서 명령을 실행하려면 SSH를 통해 vRealize Automation 8.x 장치에 연결하고 루트 사용자로 로그인합니다. 예를 들어 포트 22, 5480 또는 443에서 FQDN(정규화된 도메인 이름) URL(예: `https://cava-1-234-567.yourcompanyFQDN.com`)에 연결합니다.
  - vRealize Automation에서 FIPS를 확인하려면 `vracli security fips` 명령을 실행합니다.
  - 명령이 `FIPS mode: strict`를 반환하는지 확인합니다.

- 5 Gerrit 서버가 FIPS를 사용하도록 설정된 vRealize Automation 인스턴스의 끝점인 경우, Gerrit 구성 파일에 올바른 메시지 인증(MAC) 키가 포함되어 있는지 확인합니다.
  - a Gerrit를 열고 SSH 키 쌍을 생성합니다.
  - b '\$site\_path'/etc/gerrit.config에서 Gerrit 서버 구성 파일을 찾습니다.
  - c Gerrit 서버 구성 파일에 hmac-MD5를 제외하고 하나 이상의 MAC(메시지 인증 코드) 키가 포함되어 있는지 확인합니다.

---

**참고** FIPS 모드에서 hmac-MD5는 지원되는 MAC 알고리즘이 아닙니다. Gerrit 서버가 올바르게 시작되도록 하려면 Gerrit 서버 구성 파일에서 이 알고리즘을 제외해야 합니다. Gerrit 서버가 올바르게 시작되지 않으면 다음 메시지가 표시됩니다. PrivateKey/PassPhrase is incorrect

---

더하기 기호(+)로 시작하는 지원되는 MAC(메시지 인증 코드) 키 이름이 사용되도록 설정됩니다. 하이픈(-)으로 시작하는 MAC 키 이름은 기본 MAC 목록에서 제거됩니다. 기본적으로 이러한 지원되는 MAC은 Gerrit 서버의 Code Stream에서 사용할 수 있습니다.

- hmac-md5-96
- hmac-sha1
- hmac-sha1-96
- hmac-sha2-256
- hmac-sha2-512

#### 다음에 수행할 작업

자세한 내용을 보려면 다른 섹션을 검토합니다. [Code Stream](#)에서 [Gerrit 트리거](#)를 사용하여 [파이프라인](#)을 실행하는 방법 항목을 참조하십시오.

## Code Stream과 vRealize Orchestrator를 통합하는 방법

Code Stream은 vRO 워크플로를 실행하여 기능을 확장하기 위해 vRO(vRealize Orchestrator)와 통합할 수 있습니다. vRealize Orchestrator에는 타사 도구와 통합할 수 있는 미리 정의된 워크플로가 많이 있습니다. 이러한 워크플로는 DevOps 프로세스를 자동화 및 관리하고, 대량 작업을 자동화하는 등의 작업에 유용합니다.

예를 들어 파이프라인의 vRO 작업에서 워크플로를 사용하여 사용자를 사용하도록 설정하고, 사용자를 제거하고, VM을 이동하고, 테스트 프레임워크와 통합하여 파이프라인 실행 시 코드를 테스트하는 등의 작업을 할 수 있습니다. [code.vmware.com](http://code.vmware.com)에서 vRealize Orchestrator 워크플로에 대한 예시 코드를 찾아볼 수 있습니다.

vRealize Orchestrator 워크플로를 사용하면 애플리케이션을 구축, 테스트 및 배포할 때 파이프라인이 작업을 실행할 수 있습니다. 파이프라인에 사전 정의된 워크플로를 포함하거나 사용자 지정 워크플로를 생성하여 사용할 수 있습니다. 각 워크플로에는 입력, 작업 및 출력이 포함됩니다.

파이프라인에서 vRO 워크플로를 실행하려면 해당 워크플로가 파이프라인에 추가한 vRO 작업의 사용 가능한 워크플로 목록에 나타나야 합니다.

워크플로가 파이프라인의 vRO 작업에 나타나려면 먼저 관리자가 vRealize Orchestrator에서 다음 단계를 수행해야 합니다.

- 1 vRO 워크플로에 CODESTREAM 태그를 적용합니다.
- 2 vRO 워크플로를 글로벌로 표시합니다.

#### 사전 요구 사항

- vRealize Orchestrator의 온 프레미스 인스턴스에 관리자로 액세스할 수 있는지 확인합니다. 도움이 필요하면 해당 관리자에게 문의하거나 [vRealize Orchestrator 설명서](#)를 참조하십시오.
- Code Stream에서 프로젝트의 멤버인지 확인합니다. 그렇지 않으면 프로젝트 멤버로 추가해 달라고 Code Stream 관리자에게 요청합니다. [Code Stream에서 프로젝트를 추가하는 방법](#)의 내용을 참조하십시오.
- Code Stream에서 파이프라인을 생성하고 단계를 추가합니다.

#### 절차

- 1 관리자는 파이프라인이 실행할 vRealize Orchestrator 워크플로를 준비합니다.
  - a vRealize Orchestrator에서 파이프라인에서 사용해야 하는 워크플로(예: 사용자를 사용하도록 설정하는 워크플로)를 찾습니다.  
필요한 워크플로가 존재하지 않는다면 워크플로를 생성할 수 있습니다.
  - b 검색 창에 **Tag workflow**를 입력하여 Tag workflow라는 이름의 워크플로를 찾습니다.
  - c Tag workflow라는 이름의 카드에서 **실행**을 클릭합니다. 구성 영역이 표시됩니다.
  - d 태그가 지정된 워크플로 텍스트 영역에서 **Code Stream** 파이프라인에서 사용할 워크플로의 이름을 입력한 다음 목록에서 선택합니다.
  - e 태그 및 값 텍스트 영역에서 대문자로 CODESTREAM을 입력합니다.
  - f **전역 태그**라는 이름의 확인란을 클릭합니다.
  - g **실행**을 클릭합니다. 그러면 CODESTREAM 태그가 **Code Stream** 파이프라인에서 선택해야 하는 워크플로에 연결됩니다.
  - h 탐색 창에서 **워크플로**를 클릭하고 CODESTREAM 태그가 파이프라인에서 실행할 워크플로 카드에 나타나는지 확인합니다.

Code Stream에 로그인하고 파이프라인에 vRO 작업을 추가하면 태그가 지정된 워크플로가 워크플로 목록에 표시됩니다.

## 2 Code Stream에서 vRealize Orchestrator 인스턴스에 대한 끝점을 생성합니다.

- a **끝점 > 새 끝점**을 클릭합니다.
- b 프로젝트를 선택합니다.
- c 적절한 이름을 입력합니다.
- d vRealize Orchestrator 끝점의 URL을 입력합니다.

**https://vro-appliance.yourdomain.local:8281** 형식을 사용합니다.

https://vro-appliance.yourdomain.local:8281/vco/api 형식은 사용하지 마십시오.

vRealize Automation 장치에 내장된 vRealize Orchestrator 인스턴스의 URL은 포트가 없는 장치의 FQDN입니다. 예: **https://vro-appliance.yourdomain.local/vco**

vRealize Automation 8.x로 시작하는 외부 vRealize Orchestrator 장치의 경우 장치의 FQDN은 **https://vro-appliance.yourdomain.local**입니다.

vRealize Automation 7.x에 포함된 외부 vRealize Orchestrator 장치의 경우 장치의 FQDN은 **https://vro-appliance.yourdomain.local:8281/vco**입니다.

끝점을 추가할 때 문제가 발생하면 콜론이 제거된 SHA-256 인증서 지문이 포함된 YAML 구성을 가져와야 할 수 있습니다. 예를 들어 **B0:01:A2:72...**는 **B001A272...**가 됩니다. 샘플 YAML 코드는 다음과 유사합니다.

```
...
---
project: Demo
kind: ENDPOINT
name: external-vro
description: ''
type: vro
properties:
  url: https://yourVROhost.yourdomain.local
  username: yourusername
  password: yourpassword
  fingerprint: <your_fingerprint>
...
```

- e 입력한 URL에 인증서가 필요한 경우 **인증서 수락**을 클릭합니다.
- f vRealize Orchestrator 서버에 대한 사용자 이름과 암호를 입력합니다.

로컬이 아닌 사용자를 인증에 사용하는 경우 사용자 이름에서 도메인 부분을 생략해야 합니다. 예를 들어 **svc\_vro@yourdomain.local**을 인증하려면 **사용자 이름** 텍스트 영역에 **svc\_vro**를 입력해야 합니다.

## 3 vRO 작업을 실행하도록 파이프라인을 준비합니다.

- a 파이프라인 단계에 vRO 작업을 추가합니다.
- b 적절한 이름을 입력합니다.
- c [워크플로 속성] 영역에서 vRealize Orchestrator 끝점을 선택합니다.

- d vRealize Orchestrator에서 CODESTREAM으로 태그 지정된 워크플로를 선택합니다.

직접 생성한 사용자 지정 워크플로를 선택하는 경우에는 입력 매개 변수 값을 입력해야 할 수 있습니다.

- e **작업 실행**에 대해 **조건부**를 클릭합니다.

The screenshot displays the configuration interface for a vRO workflow. The top navigation bar includes '작업 :vRO workflow', '알림', '물백', and a '작업 검증' button. The main form contains the following elements:

- 작업 이름**: vRO workflow
- 유형**: vRO
- 실패 시 계속**: ☐
- 작업 실행**: ☐ 항상 ☒ 조건부
- 조건**: [Empty text box]
- 워크플로 속성**: vROEP
- 워크플로**: Test
- 출력 매개 변수**: status, workflowExecutionId, parameters, properties

- f 파이프라인 실행 시 적용할 조건을 입력합니다.

파이프라인 실행 시기...	조건 선택...
조건부	<p>정의된 조건이 <b>True</b>로 평가되는 경우에만 파이프라인 작업을 실행합니다. 조건이 <b>False</b>이면 작업을 건너뛵니다.</p> <p>vRO 작업을 사용하면 다음 피연산자와 연산자를 사용하는 부울 식을 포함할 수 있습니다.</p> <ul style="list-style-type: none"> <li>■ 파이프라인 변수(예: <code>\${pipeline.variableName}</code>). 변수를 입력할 때 중괄호만 사용합니다.</li> <li>■ 작업 출력 변수(예: <code>\${Stage1.task1.machines[0].value.hostIp[0]}</code>).</li> <li>■ 기본 파이프라인 바인딩 변수(예: <code>\${releasePipelineName}</code>).</li> <li>■ 대/소문자를 구분하지 않는 부울 값(예: <code>true</code>, <code>false</code>, <code>'true'</code>, <code>'false'</code>).</li> <li>■ 따옴표가 없는 정수 또는 10진수 값.</li> <li>■ 작은따옴표 또는 큰따옴표와 함께 사용된 문자열 값(예: <code>"test"</code>, <code>'test'</code>).</li> <li>■ 문자열 및 숫자 유형 값(예: <code>== Equals</code> 및 <code>!= Not Equals</code>).</li> <li>■ 관계 연산자(예: <code>&gt;</code>, <code>&gt;=</code>, <code>&lt;</code> 및 <code>&lt;=</code>).</li> <li>■ 부울 논리(예: <code>&amp;&amp;</code> 및 <code>  </code>).</li> <li>■ 산술 연산자(예: <code>+</code>, <code>-</code>, <code>*</code> 및 <code>/</code>).</li> <li>■ 소괄호를 사용하는 중첩된 식.</li> <li>■ 리터럴 값 ABCD를 포함하는 문자열은 <b>False</b>로 평가되고 작업을 건너뛵니다.</li> <li>■ 단항 연산자는 지원되지 않습니다.</li> </ul> <p>예를 들어 조건은 <code>\${Stage1.task1.output} == "Passed"    \${pipeline.variableName} == 39</code>와 같을 수 있습니다.</p>
항상	항상을 선택하는 경우 파이프라인은 조건 없이 작업을 실행합니다.

- g 인사말을 입력합니다.
- h **작업 검증**을 클릭하고 발생하는 모든 오류를 수정합니다.
- 4 파이프라인을 저장하고, 사용하도록 설정하고, 실행합니다.
- 5 파이프라인 실행 후에는 결과를 검토합니다.
- 실행을 클릭합니다.
  - 파이프라인을 클릭합니다.
  - 작업을 클릭합니다.
  - 결과, 입력 값 및 속성을 검토합니다.

워크플로 실행 ID, 작업에 응답한 사람과 시기, 이들이 추가한 설명을 식별할 수 있습니다.

## 결과

축하합니다! Code Stream에서 사용하도록 vRealize Orchestrator 워크플로에 태그를 지정했고, Code Stream 파이프라인에 vRO 작업을 추가했습니다. 이제 파이프라인이 DevOps 환경에서 작업을 자동화하는 워크플로를 실행할 수 있습니다.



## 예제: vRO 작업 출력 형식

vRO 작업의 출력 형식은 이 예와 유사합니다.

```
[{
    "name": "result",
    "type": "STRING",
    "description": "Result of workflow run.",
    "value": ""
},
{
    "name": "message",
    "type": "STRING",
    "description": "Message",
    "value": ""
}]
```

### 다음에 수행할 작업

개발, 테스트 및 운영 환경에서 작업을 자동화할 수 있도록 파이프라인에 vRO 워크플로 작업을 계속 추가합니다

# Code Stream에서 파이프라인 트리거

# 7

특정 이벤트가 발생했을 때 Code Stream에서 특정 파이프라인을 트리거하도록 할 수 있습니다.

예를 들면 다음과 같습니다.

- Docker 트리거는 새 아티팩트가 생성 또는 업데이트될 때 파이프라인을 실행할 수 있습니다.
- Git에 대한 트리거는 개발자가 코드를 업데이트할 때 파이프라인을 트리거할 수 있습니다.
- Gerrit에 대한 트리거는 개발자가 코드를 검토할 때 파이프라인을 트리거할 수 있습니다.

본 장은 다음 항목을 포함합니다.

- Code Stream에서 Docker 트리거를 사용하여 지속적 전달 파이프라인을 실행하는 방법
- Code Stream에서 Git 트리거를 사용하여 파이프라인을 실행하는 방법
- Code Stream에서 Gerrit 트리거를 사용하여 파이프라인을 실행하는 방법

## Code Stream에서 Docker 트리거를 사용하여 지속적 전달 파이프라인을 실행하는 방법

Code Stream 관리자나 개발자는 Code Stream에서 Docker 트리거를 사용할 수 있습니다. Docker 트리거는 빌드 아티팩트가 생성되거나 업데이트될 때마다 독립형 CD(지속적 전달) 파이프라인을 실행합니다. Docker 트리거는 새 아티팩트 또는 업데이트된 아티팩트를 Docker Hub 저장소에 컨테이너 이미지로 푸시하는 CD 파이프라인을 실행합니다. CD 파이프라인은 자동화된 구축의 일부로 실행될 수 있습니다.

예를 들어, 업데이트된 컨테이너 이미지를 CD 파이프라인을 통해 연속적으로 배포하려면 Docker 트리거를 사용합니다. 컨테이너 이미지가 Docker 레지스트리에 체크인되면 Docker Hub의 Webhook는 이미지가 변경된 것을 Code Stream에 알립니다. 이 알림은 CD 파이프라인에서 업데이트된 컨테이너 이미지를 사용하여 실행되도록 트리거하고 이미지를 Docker Hub 저장소에 업로드합니다.

Docker 트리거를 사용하려면 Code Stream에서 몇 가지 단계를 수행해야 합니다.

## 표 7-1. Docker 트리거를 사용하는 방법

수행할 작업...	이 작업에 대한 자세한 내용...
Docker 레지스트리 끝점을 생성합니다.	Code Stream이 파이프라인을 트리거할 수 있으려면 Docker 레지스트리 끝점이 있어야 합니다. 끝점이 없는 경우, Docker 트리거를 위한 Webhook를 추가할 때 끝점을 생성하는 옵션을 선택할 수 있습니다.  Docker 레지스트리 끝점에는 Docker Hub 저장소의 URL이 포함됩니다.
파이프라인이 실행될 때 Docker 매개 변수를 자동 삽입하는 입력 매개 변수를 파이프라인에 추가합니다.	Docker 매개 변수를 파이프라인에 삽입할 수 있습니다. 매개 변수에는 Docker 이벤트 소유자 이름, 이미지, 저장소 이름, 저장소 네임스페이스 및 태그가 포함될 수 있습니다.  CD 파이프라인에는 Docker Webhook가 파이프라인이 트리거되기 전에 파이프라인에 전달하는 입력 매개 변수를 포함합니다.
Docker Webhook를 생성합니다.	Code Stream에서 Docker Webhook를 생성하면 해당하는 Webhook가 Docker Hub에도 생성됩니다. Code Stream의 Docker Webhook는 Webhook에 포함된 URL을 통해 Docker Hub에 연결합니다.  Webhook는 서로 통신하고 Docker Hub에서 아티팩트가 생성되거나 업데이트되면 파이프라인을 트리거합니다.  Code Stream에서 Docker Webhook를 업데이트하거나 삭제하면 Docker Hub에 있는 Webhook도 업데이트되거나 삭제됩니다.
파이프라인에 Kubernetes 작업을 추가하고 구성합니다.	Docker Hub 저장소에서 아티팩트가 생성 또는 업데이트되면 파이프라인이 트리거됩니다. 그런 다음 파이프라인을 통해 Kubernetes 클러스터의 Docker 호스트에 아티팩트를 배포합니다.
작업에 로컬 YAML 정의를 추가합니다.	배포 작업에 적용하는 YAML 정의에는 Docker 컨테이너 이미지가 포함됩니다. 개인 소유 저장소에서 이미지를 다운로드해야 하는 경우 YAML 파일에 Docker 구성 비밀이 있는 섹션이 포함되어야 합니다. <a href="#">스마트 파이프라인 템플릿을 사용하기 전에</a> Code Stream에서 CICD 기본 구축 계획의 CD 부분을 참조하십시오.

Docker Hub 저장소에 아티팩트가 생성되거나 업데이트되면 Docker Hub의 Webhook가 Code Stream의 Webhook에 이 사실을 알리고, 이 Webhook가 파이프라인을 트리거합니다. 다음과 같은 작업이 발생합니다.

- 1 Docker Hub가 Webhook에 있는 URL에 POST 요청을 보냅니다.
- 2 Code Stream이 Docker 트리거를 실행합니다.
- 3 Docker 트리거가 CD 파이프라인을 시작합니다.
- 4 CD 파이프라인이 아티팩트를 Docker Hub 저장소에 푸시합니다.
- 5 Code Stream이 자체 Docker Webhook를 트리거하고, 이 Webhook는 아티팩트를 사용자의 Docker 호스트에 배포하는 CD 파이프라인을 실행합니다.

이 예시에서는 Code Stream에서 Docker 끝점 및 애플리케이션을 개발 Kubernetes 클러스터에 배포하는 Docker Webhook를 생성합니다. 이러한 단계에는 Docker가 Webhook의 URL에 게시하는 페이로드 코드 예시, 사용되는 API 코드 및 보안 토큰이 있는 인증 코드가 포함됩니다.

#### 사전 요구 사항

- Code Stream 인스턴스에 CD(지속적 전달) 파이프라인이 있는지 확인합니다. 애플리케이션을 배포하는 Kubernetes 작업이 하나 이상 포함되어 있는지도 확인합니다. [장 4 Code Stream에서 기본적인 코드 구축, 통합 및 제공 계획](#) 항목을 참조하십시오.
- CD 파이프라인을 통해 개발용 애플리케이션을 배포할 기존 Kubernetes 클러스터에 액세스할 수 있는지 확인합니다.
- Code Stream에서 프로젝트의 멤버인지 확인합니다. 그렇지 않으면 프로젝트 멤버로 추가해 달라고 Code Stream 관리자에게 요청합니다. [Code Stream에서 프로젝트를 추가하는 방법](#)의 내용을 참조하십시오.

#### 절차

- 1 Docker 레지스트리 끝점을 생성합니다.
  - a **끝점**을 클릭합니다.
  - b **새 끝점**을 클릭합니다.
  - c 기존 프로젝트의 이름을 입력하기 시작합니다.
  - d 유형은 **Docker 레지스트리**로 선택합니다.
  - e 적절한 이름을 입력합니다.
  - f 서버 유형은 **DockerHub**로 선택합니다.

- g Docker Hub 저장소의 URL을 입력합니다.
- h 저장소에 액세스할 수 있는 이름과 암호를 입력합니다.

## New endpoint

**Project \*** Q AWS\_PGProj

**Type \*** Docker Registry ▾

**Name \*** dockerhub-endpoint

**Description**

**Mark restricted** ☐ non-restricted

**Server type \*** DockerHub ▾

**Repo URL \*** https://hub.docker.com/repository/docker/automation/cs-builder

**ACCEPT CERTIFICATE**

**Username \*** admin

**Password \*** ●●●●●●●●●●

**CREATE VARIABLE**

**CREATE** **VALIDATE** **CANCEL**

- 2 파이프라인 실행 시 Docker 매개 변수를 자동 삽입하는 입력 속성을 CD 파이프라인에 설정합니다.

sm-1 사용함

작업 공간 **입력** 모델 출력

입력 매개 변수 ⓘ

매개 변수 자동 삽입 ☐ Gerrit ☐ Git ☒ Docker ☐ 없음

**추가** **삽입된 매개 변수 추가/제거**

별표 표시됨 ⓘ	이름
☆	DOCKER_REGISTRY_EVENT_OWNER_NAME
☆	DOCKER_REGISTRY_IMAGE
☆	DOCKER_REGISTRY_REPO_NAME
☆	DOCKER_REGISTRY_REPO_NAMESPACE
☆	DOCKER_REGISTRY_TAG

### 3 Docker Webhook를 생성합니다.

- a **트리거 > Docker**를 클릭합니다.
- b **Docker에 대한 새 Webhook**를 클릭합니다.
- c 프로젝트를 선택합니다.
- d 적절한 이름을 입력합니다.
- e Docker 레지스트리 끝점을 선택합니다.

끝점이 아직 없으면 **끝점 생성**을 클릭하여 끝점을 생성합니다.

- f Webhook가 트리거되도록 Docker 삽입 매개 변수가 있는 파이프라인을 선택합니다. [단계 2](#)의 내용을 참조하십시오.

사용자 지정 추가 입력 매개 변수를 사용하여 파이프라인이 구성된 경우 입력 매개 변수 목록에 매개 변수 및 값이 표시됩니다. 트리거 이벤트를 사용하여 파이프라인에 전달될 입력 매개 변수에 대한 값을 입력할 수 있습니다. 또는 값을 비워 두거나 기본값(정의된 경우)을 사용할 수도 있습니다.

입력 탭의 매개 변수에 대한 자세한 내용은 [작업을 수동으로 추가하기 전에 Code Stream에서 CICD 네이티브 빌드 계획](#) 항목을 참조하십시오.

- g API 토큰을 입력합니다.

CSP API 토큰은 Code Stream에 대한 외부 API 연결에 사용자를 인증합니다. API 토큰을 가져오려면:

- 1 **토큰 생성**을 클릭합니다.
- 2 사용자 이름과 연결된 이메일 주소와 암호를 입력하고 **생성**을 클릭합니다.

생성한 토큰은 6개월 동안 유효합니다. 새로 고침 토큰이라고도 합니다.

- 나중에 사용할 수 있도록 토큰을 변수로 유지하려면 **변수 생성**을 클릭하고 변수 이름을 입력한 후 **저장**을 클릭합니다.
- 나중에 사용할 수 있도록 토큰을 텍스트 값으로 유지하려면 **복사**를 클릭하고 토큰을 텍스트 파일에 붙여 넣어 로컬에 저장합니다.

나중에 사용할 수 있도록 변수를 생성하고 텍스트 파일에 토큰을 저장하도록 선택할 수 있습니다.

- 3 **닫기**를 클릭합니다.

- h 구축 이미지를 입력합니다.

- i 태그를 입력합니다.

- j **저장**을 클릭합니다.

Docker Webhook이 사용되도록 설정된 상태로 Webhook 카드가 나타납니다. Docker Webhook 트리거 및 파이프라인 실행 없이 Docker Hub 저장소에 대한 더미 푸시를 만들려면 **사용 안 함**을 클릭합니다.

4 CD 파이프라인에 Kubernetes 배포 작업을 구성합니다.

- a Kubernetes 작업 속성에서 개발 Kubernetes 클러스터를 선택합니다.
- b **생성** 작업을 선택합니다.

- c 페이로드 소스에 대한 **로컬 정의**를 선택합니다.
- d 그런 다음 로컬 YAML 파일을 선택합니다.

예를 들어 Docker Hub가 이 로컬 YAML 정의를 Webhook의 URL에 대한 페이로드로 게시할 수 있습니다.

```
{
  "callback_url": "https://registry.hub.docker.com/u/svendowideit/testhook/hook/2141b5bi5i5b02bec211i4eeih0242eg11000a/",
  "push_data": {
    "images": [
      "27d47432a69bca5f2700e4dff7de0388ed65f9d3fb1ec645e2bc24c223dc1cc3",
      "51a9c7c1f8bb2fa19bcd09789a34e63f35abb80044bc10196e304f6634cc582c",
      "...",
    ],
    "pushed_at": 1.417566161e+09,
    "pusher": "trustedbuilder",
    "tag": "latest"
  },
  "repository": {
    "comment_count": 0,
    "date_created": 1.417494799e+09,
    "description": "",
    "dockerfile": "#\n# BUILD\u0009\u0009docker build -t svendowideit/apt-cacher .\n#\nRUN\u0009\u0009\u0009docker run -d -p 3142:3142 -name apt-cacher-run apt-cacher\n#\n# and then you can run containers with:\n#\n\u0009\u0009\u0009docker run -t -i -rm -e http_proxy http://192.168.1.2:3142/ debian\nbash\n#\nFROM\u0009\u0009\u0009ubuntu\n#\nVOLUME\u0009\u0009\u0009[\/var/cache/apt-cacher-ng]\n#\nRUN\u0009\u0009\u0009apt-get update ; apt-get install -yq apt-cacher-ng\n#\nEXPOSE\n\u0009\u0009\u00093142\n#\nCMD\u0009\u0009\u0009chmod 777 /var/cache/apt-cacher-ng ; /etc/init.d/apt-cacher-ng start ; tail -f /var/log/apt-cacher-ng/*\n#",
    "full_description": "Docker Hub based automated build from a GitHub repo",
    "is_official": false,
    "is_private": true,
    "is_trusted": true,
    "name": "testhook",
    "namespace": "svendowideit",
    "owner": "svendowideit",
    "repo_name": "svendowideit/testhook",
    "repo_url": "https://registry.hub.docker.com/u/svendowideit/testhook/",
    "star_count": 0,
    "status": "Active"
  }
}
```

Docker Hub에 Webhook를 생성하는 API는 [https://cloud.docker.com/v2/repositories/%3CUSERNAME%3E/%3CREPOSITORY%3E/webhook\\_pipeline/](https://cloud.docker.com/v2/repositories/%3CUSERNAME%3E/%3CREPOSITORY%3E/webhook_pipeline/) 양식을 사용합니다.

JSON 코드 본문은 다음과 유사합니다.

```
{
  "name": "demo_webhook",
  "webhooks": [
```



```
{
  "name": "demo_webhook",
  "hook_url": "http://www.google.com"
}
```

Docker Hub 서버로부터 이벤트를 수신하기 위해, Code Stream에서 생성한 Docker Webhook의 인증 체계는 Webhook에 대한 임의의 문자열 토큰이 포함된 허용 목록 인증 메커니즘을 사용합니다. 이 인증 체계는 hook\_url에 추가한 보안 토큰에 기반하여 이벤트를 필터링합니다.

Code Stream은 구성된 보안 토큰을 사용하여 Docker Hub 서버의 모든 요청을 확인할 수 있습니다. 예: hook\_url = IP:Port/pipelines/api/docker-hub-webhooks?secureToken = ""

- 5 Docker Hub 저장소에 Docker 아티팩트를 생성합니다. 또는 기존 아티팩트를 업데이트합니다.
- 6 트리거가 발생했는지 확인하고 Docker Webhook의 작업을 보려면 **트리거 > Docker > 작업**을 클릭합니다.

**Docker** GUIDED SETUP

Activity Webhooks for Docker

Commit Time	Webhook	Image	Tag	Owner	Repository	Pipeline	Execution Status
01/09/2019 10:59 AM	dt11-Docker-WH	admin/repo:s1	s1	admin	repo		SKIPPED
01/09/2019 10:59 AM	fvxd-Docker-WH	admin/repo:s1	s1	admin	repo		SKIPPED
01/09/2019 10:59 AM	test-do-Docker-WH	admin/repo:s1	s1	admin	repo		SKIPPED
01/09/2019 10:59 AM	sm-Docker-WH	admin/repo:s1	s1	admin	repo		SKIPPED
01/09/2019 10:59 AM	t-token-Docker-WH	admin/repo:s1	s1	admin	repo		FAILED
01/09/2019 10:57 AM	dt11-Docker-WH	admin/repo:s01	s01	admin	repo		SKIPPED
01/09/2019 10:57 AM	sm-Docker-WH	admin/repo:s01	s01	admin	repo		SKIPPED
01/09/2019 10:57 AM	test-do-Docker-WH	admin/repo:s01	s01	admin	repo		SKIPPED
01/09/2019 10:57 AM	fvxd-Docker-WH	admin/repo:s01	s01	admin	repo		SKIPPED

- 7 실행을 클릭한 후 실행 중인 파이프라인을 관찰합니다.

**실행** 478개 항목 단계별 설정

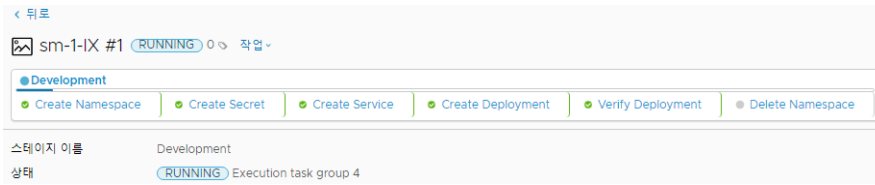
+ 새 실행 🔍 📄 🔍 🔄

sm-1-IX#15
RUNNING
스테이지: 
작업 ▾

실행한 사람 smasaru ▾ 날짜: 2020년 1월 22일 오후 3:35:39
☆ 입력: -

Running Stage0 ▾
☆ 출력: -

8 실행 중인 단계를 클릭하여 파이프라인이 실행 중일 때 작업을 봅니다.



## 결과

축하합니다! CD 파이프라인을 계속 실행하도록 Docker 트리거를 설정했습니다. 이제 파이프라인에서 신규 및 업데이트된 Docker 아티팩트를 Docker Hub 저장소에 업로드할 수 있습니다.

## 다음에 수행할 작업

새로 추가되거나 업데이트된 아티팩트가 개발 Kubernetes 클러스터의 Docker 호스트에 배포되었는지 확인합니다.

# Code Stream에서 Git 트리거를 사용하여 파이프라인을 실행하는 방법

Code Stream 관리자 또는 개발자는 Git 트리거를 사용하여 Code Stream을 Git 수명주기와 통합할 수 있습니다. GitHub, GitLab 또는 Bitbucket Enterprise에서 코드를 변경하면 이벤트가 Webhook을 통해 Code Stream과 통신하고 파이프라인을 트리거합니다. 동일한 네트워크에서 Cloud Assembly와 엔터프라이즈 버전을 모두 연결할 수 있는 경우, Webhook는 GitLab, GitHub 및 Bitbucket 온-프레미스 엔터프라이즈 버전에서 작동합니다.

Code Stream에 Git에 대한 Webhook를 추가하면 GitHub, GitLab 또는 Bitbucket 저장소에도 Webhook가 생성됩니다. 나중에 Webhook를 업데이트하거나 삭제하는 경우 해당 작업은 GitHub, GitLab 또는 Bitbucket의 Webhook도 업데이트하거나 삭제합니다.

Webhook 정의에는 모니터링할 저장소 분기의 Git 끝점이 포함되어야 합니다. Webhook를 생성하기 위해 Code Stream은 Git 끝점을 사용합니다. 끝점이 없는 경우 Webhook를 추가할 때 끝점을 생성할 수 있습니다. 이 예에서는 GitHub에 사전 정의된 Git 끝점이 있다고 가정합니다.

**참고** Webhook를 생성하려면 Git 끝점이 인증에 전용 토큰을 사용해야 하며 암호를 사용할 수 없습니다.

동일한 Git 끝점을 사용하고 Webhook 구성 페이지에서 분기 이름에 서로 다른 값을 제공하여 여러 분기에 대해 다수의 Webhook를 생성하고 할 수 있습니다. 동일한 Git 저장소의 다른 분기에 대해 또 다른 Webhook를 생성하기 위해, 여러 분기에 대해 Git 끝점을 여러 번 복제할 필요가 없습니다. 대신, Webhook에 분기 이름을 제공하면 Git 끝점을 재사용할 수 있습니다. Git Webhook의 분기가 끝점의 분기와 동일한 경우 Git Webhook 페이지에서 분기 이름을 제공하지 않아도 됩니다.

이 예에서는 Git 트리거를 GitHub 저장소와 사용하는 방법을 보여주지만 다른 Git 서버 유형을 사용하는 경우 준비가 필요하다는 내용이 사전 요구 사항에 포함되어 있습니다.

## 사전 요구 사항

- Code Stream에서 프로젝트의 멤버인지 확인합니다. 그렇지 않으면 프로젝트 멤버로 추가해 달라고 Code Stream 관리자에게 요청합니다. [Code Stream에서 프로젝트를 추가하는 방법](#)의 내용을 참조하십시오.
- 모니터링하려는 GitHub 분기에 Git 끝점이 있는지 확인합니다. [Git와 Code Stream을 통합하는 방법](#) 항목을 참조하십시오.
- Git 저장소에 Webhook을 생성할 수 있는 권한이 있는지 확인합니다.
- GitLab에서 Webhook를 구성하는 경우, GitLab Enterprise에서 기본 네트워크 설정을 변경하여 아웃바운드 요청을 사용하도록 설정하고 로컬 Webhook의 생성을 허용합니다.

**참고** 이 변경은 GitLab Enterprise에만 필요합니다. 이러한 설정은 GitHub 또는 Bitbucket에 적용되지 않습니다.

- GitLab Enterprise에 관리자로 로그인합니다.
- `http://{gitlab-server}/admin/application_settings/network`와 같은 URL을 사용하여 네트워크 설정으로 이동합니다.
- 아웃바운드 요청**을 확장하고 다음을 클릭합니다.
  - Webhook 및 서비스로부터의 로컬 네트워크에 대한 요청을 허용합니다.
  - 시스템 후크로부터의 로컬 네트워크에 대한 요청을 허용합니다.
- 트리거할 파이프라인에 대해, 파이프라인이 실행될 때 Git 매개 변수를 삽입하도록 입력 속성을 설정했는지 확인합니다.

**Build and Deploy** 사용함

작업 공간 **입력** 모델 출력

입력 매개 변수 ⓘ

매개 변수 자동 삽입 ☐ Gerrit ☒ Git ☐ Docker ☐ 없음

추가 삽입된 매개 변수 추가/제거

별표 표시됨 ⓘ	이름
⋮ ☆	GIT_BRANCH_NAME
⋮ ☆	GIT_CHANGE_SUBJECT
⋮ ☆	GIT_COMMIT_ID
⋮ ☆	GIT_EVENT_DESCRIPTION
⋮ ☆	GIT_EVENT_OWNER_NAME
⋮ ☆	GIT_EVENT_TIMESTAMP
⋮ ☆	GIT_REPO_NAME
⋮ ☆	GIT_SERVER_URL

입력 매개 변수에 대한 자세한 내용은 작업을 수동으로 추가하기 전에 [Code Stream에서 CI/CD 네이티브 빌드 계획](#) 항목을 참조하십시오.

## 절차

- 1 Code Stream에서 **트리거 > Git**을 클릭합니다.
- 2 **Git에 대한 Webhook** 탭을 클릭한 다음 **Git에 대한 새 Webhook**을 클릭합니다.
  - a 프로젝트를 선택합니다.
  - b Webhook에 대한 의미 있는 이름과 설명을 입력합니다.
  - c 모니터링하려는 분기에 대해 구성된 **Git** 끝점을 선택합니다.

Webhook을 생성할 때 Webhook 정의에는 현재 끝점 세부 정보가 포함됩니다.

- 나중에 끝점에서 **Git** 유형, **Git** 서버 유형 또는 **Git** 저장소 URL을 변경하는 경우 Webhook은 원래 끝점 세부 정보를 사용하여 **Git** 저장소에 액세스하려고 하기 때문에 더 이상 파이프라인을 트리거할 수 없습니다. Webhook을 삭제하고 해당 끝점으로 Webhook을 다시 생성해야 합니다.
- 나중에 끝점에서 인증 유형, 사용자 이름 또는 전용 토큰을 변경하는 경우 Webhook은 계속 작동합니다.
- BitBucket 저장소를 사용하는 경우 저장소의 URL 형식은 `https://api.bitbucket.org/{user}/{repo name}` 또는 `http(s)://{bitbucket-enterprise-server}/rest/api/1.0/users/{username}/repos/{repo name}` 중 하나여야 합니다.

---

**참고** 이전에 기본 인증에 암호를 사용하는 **Git** 끝점으로 Webhook을 생성한 경우 인증에 전용 토큰을 사용하는 **Git** 끝점이 있는 Webhook을 삭제하고 다시 정의해야 합니다.

---

**Git**과 **Code Stream**을 통합하는 방법의 내용을 참조하십시오.

- d (선택 사항) Webhook에서 모니터링할 분기를 입력합니다.  
분기를 지정하지 않은 상태로 두면 Webhook은 **Git** 끝점에 대해 구성한 분기를 모니터링합니다.
- e (선택 사항) Webhook에 대한 비밀 토큰을 생성합니다.

비밀 토큰을 사용하는 경우 **Code Stream**은 Webhook에 대한 임의의 문자열 토큰을 생성합니다. 그런 다음 Webhook이 **Git** 이벤트 데이터를 수신하면 Webhook은 비밀 토큰과 함께 데이터를 전송합니다. **Code Stream**은 이 정보를 사용하여 호출이 구성된 **GitHub** 인스턴스, 저장소, 분기와 같은 예상된 소스로부터 수신되었는지 여부를 판별합니다. 비밀 토큰은 **Git** 이벤트 데이터가 올바른 소스로부터 수신된 것인지 확인하는 데 사용되는 추가적인 보안 계층을 제공합니다.

f (선택 사항) 파일 포함 또는 제외를 트리거에 대한 조건으로 제공합니다.

- 파일 포함. 포함 경로 또는 정규식에 지정된 파일과 일치하는 파일이 커밋에 하나라도 있으면 커밋이 파이프라인을 트리거합니다. 정규식을 지정하는 경우 **Code Stream**은 변경 집합의 파일 이름이 제공된 표현식과 일치하는 경우에만 파이프라인을 트리거합니다. 정규식 필터는 단일 저장소의 여러 파이프라인에 대해 트리거를 구성할 때 유용합니다.
- 파일 제외. 커밋의 모든 파일이 제외 경로 또는 정규식에 지정된 파일과 일치하면 파이프라인이 트리거되지 않습니다.
- 제외 우선 순위 지정. [제외 우선 순위 지정]을 설정하면 커밋에서 임의의 파일이 제외 경로 또는 정규식에 지정된 파일과 일치하더라도 파이프라인이 트리거되지 않습니다. 기본 설정은 해제입니다.

조건이 파일 포함과 파일 제외를 모두 충족하면 파이프라인이 트리거되지 않습니다.

다음 예에서 파일 포함과 파일 제외 모두 트리거에 대한 조건입니다.

The screenshot shows the '파일' (File) configuration section. It has two main categories: '포함' (Include) and '제외' (Exclude). Under '포함', there are two rules: one with 'PLAIN' type and path 'runtime/src/main/a.java', and another with 'REGEX' type and path '([a-z A-Z]+)/[a-z A-Z]+'. Under '제외', there are two rules: one with 'PLAIN' type and path 'runtime/pom.xml', and another with 'PLAIN' type and path 'runtime/demo.yaml'. At the bottom, there is a toggle switch for '제외 우선 순위 지정' (Exclude priority) which is currently turned off.

- 파일 포함의 경우 runtime/src/main/a.java 또는 모든 **Java** 파일에 대한 변경을 커밋하면 이벤트 구성에서 구성된 파이프라인이 트리거됩니다.
- 파일 제외의 경우 두 파일에 대한 변경만 커밋하면 이벤트 구성에서 구성된 파이프라인이 트리거되지 않습니다.

g Git 이벤트의 경우 **푸시** 또는 **풀** 요청을 선택합니다.

## h API 토큰을 입력합니다.

CSP API 토큰은 Code Stream에 대한 외부 API 연결에 사용자를 인증합니다. API 토큰을 가져오려면:

## 1 토큰 생성을 클릭합니다.

## 2 사용자 이름과 연결된 이메일 주소와 암호를 입력하고 생성을 클릭합니다.

생성한 토큰은 6개월 동안 유효합니다. 새로 고침 토큰이라고도 합니다.

- 나중에 사용할 수 있도록 토큰을 변수로 유지하려면 **변수 생성**을 클릭하고 변수 이름을 입력한 후 **저장**을 클릭합니다.
- 나중에 사용할 수 있도록 토큰을 텍스트 값으로 유지하려면 **복사**를 클릭하고 토큰을 텍스트 파일에 붙여 넣어 로컬에 저장합니다.

나중에 사용할 수 있도록 변수를 생성하고 텍스트 파일에 토큰을 저장하도록 선택할 수 있습니다.

## 3 닫기를 클릭합니다.

## i Webhook가 트리거할 파이프라인을 선택합니다.

사용자 지정 추가 입력 매개 변수가 파이프라인에 포함된 경우 [입력 매개 변수] 목록에 매개 변수 및 값이 표시됩니다. 트리거 이벤트를 사용하여 파이프라인에 전달되는 입력 매개 변수에 대한 값을 입력할 수 있습니다. 또는 값을 비워 두거나 기본값(정의된 경우)을 사용할 수도 있습니다.

Git 트리거의 자동 삽입 입력 매개 변수에 대한 자세한 내용은 [사전 요구 사항](#)을 참조하십시오.

## j 생성을 클릭합니다.

Webhook이 새 카드로 나타납니다.

## 3 Webhook 카드를 클릭합니다.

Webhook 데이터 양식이 다시 나타나면 양식 상단에 Webhook URL이 추가된 것을 볼 수 있습니다. Git Webhook이 Webhook URL을 통해 GitHub 저장소에 연결됩니다.

# Git

Activity

Webhooks for Git

Webhook URL ⓘ

https://ca[REDACTED]om/codestream/api/git-webhook-listeners/963b2287-527f-4e9b[REDACTED]

Project

test

Name \*

test-webhook

Description

Description

Endpoint

DemoApp-Git

Branch ⓘ

master

Secret token ⓘ \*

GYH0cBWZx4dUn47Y/KA8H/B0kts=

GENERATE

File ⓘ

Inclusions

--Select-- ▾ Value +

Exclusions

--Select-- ▾ Value +

Prioritize Exclusion

☐

Trigger

For Git

☒ PUSH ☐ PULL REQUEST

API token \*

.....

⛔

CREATE VARIABLE

GENERATE TOKEN

Pipeline \*

CICD-2 ⓘ

Comments

Execution trigger delay ⓘ

1

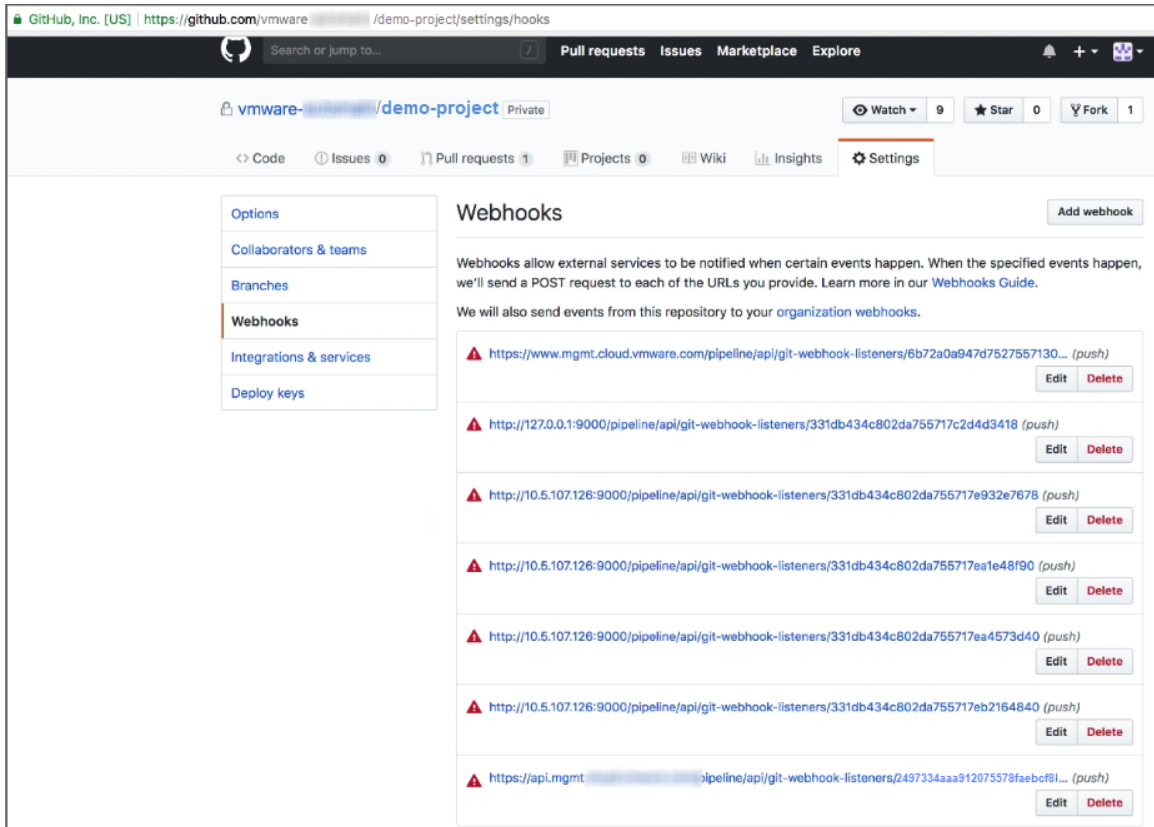
⌵

SAVE

CANCEL

4 새 브라우저 창에서 Webhook을 통해 연결되는 GitHub 저장소를 엽니다.

- a Code Stream에 추가한 Webhook을 보려면 **설정** 탭을 클릭하고 **Webhook**을 선택합니다.  
Webhook 목록의 맨 아래에 동일한 Webhook URL이 표시됩니다.



- b 코드를 변경하려면 **코드** 탭을 클릭하고 분기에서 파일을 선택합니다. 파일을 편집한 후 변경 내용을 커밋합니다.
- c Webhook URL이 작동하는지 확인하려면 **설정** 탭을 클릭하고 **Webhook**을 다시 선택합니다.  
Webhook 목록의 맨 아래에는 Webhook URL 옆에 녹색 확인 표시가 나타납니다.



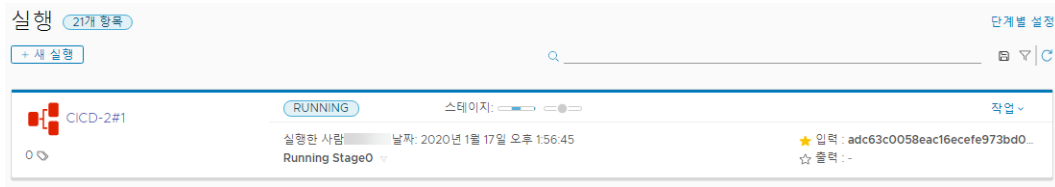
5 Git Webhook의 작업을 보려면 Code Stream으로 돌아갑니다. **트리거 > Git > 작업**을 클릭합니다.  
[실행 상태] 아래에서 파이프라인 실행이 시작되었는지 확인합니다.

Git									
Activity									GUIDED SETUP
Webhooks for Git									
Commit Time	Commit ID	Webhook	Change Subject	Owner	Branch	Repository	Events	Execution	Execution Status
Jan 15, 2019 9:42 PM	ack63c0058...	test-webhook	Update index.html	etauser	master	demo-project	PUSH	-	STARTED



## 6 실행을 클릭한 후 실행 중인 파이프라인을 추적합니다.

파이프라인 실행을 관찰하려면 새로 고침을 누르면 됩니다.



### 결과

축하합니다! Git에 대한 트리거 사용에 성공했습니다!

## Code Stream에서 Gerrit 트리거를 사용하여 파이프라인을 실행하는 방법

Code Stream 관리자 또는 개발자는 Gerrit 트리거를 사용하여 Code Stream을 Gerrit 코드 검토 수명 주기와 통합할 수 있습니다. 패치 집합을 생성하거나, 초안을 게시하거나, Gerrit 프로젝트에서 코드 변경 내용을 병합하거나, Git 분기에서 변경 내용을 직접 푸시하면 이벤트가 파이프라인이 실행되도록 트리거합니다.

Gerrit 트리거를 추가할 때 Gerrit 수신기, Gerrit 서버의 Gerrit 프로젝트를 선택하고 Gerrit 이벤트를 구성합니다. 이 예에서는 Gerrit 수신기를 먼저 구성한 다음, 세 가지 다른 파이프라인에서 두 개의 이벤트가 있는 Gerrit 트리거에서 해당 수신기를 사용합니다.

### 사전 요구 사항

- Code Stream에서 프로젝트의 멤버인지 확인합니다. 그렇지 않으면 프로젝트 멤버로 추가해 달라고 Code Stream 관리자에게 요청합니다. [Code Stream에서 프로젝트를 추가하는 방법](#)의 내용을 참조하십시오.
- Code Stream에서 Gerrit 끝점을 구성했는지 확인합니다. [Code Stream과 Gerrit을 통합하는 방법](#) 항목을 참조하십시오.
- Gerrit 릴리스 버전을 알고 있는지 확인합니다.
- 트리거할 파이프라인의 경우 파이프라인의 입력 속성을 **Gerrit**로 설정했는지 확인합니다. 그래야 파이프라인이 실행될 때 파이프라인이 Gerrit 매개 변수를 입력으로 받을 수 있습니다.

**Build and Deploy** 사용함

작업 공간    **입력**    모델    출력

입력 매개 변수 ⓘ

매개 변수 자동 삽입    ☒ Gerrit    ☐ Git    ☐ Docker    ☐ 없음

추가    삽입된 매개 변수 추가/제거

별표 표시됨 ⓘ	이름
⋮ ☆	GERRIT_BRANCH
⋮ ☆	GERRIT_CHANGE_COMMIT_MESSAGE
⋮ ☆	GERRIT_CHANGE_FILELIST
⋮ ☆	GERRIT_CHANGE_ID
⋮ ☆	GERRIT_CHANGE_NUMBER
⋮ ☆	GERRIT_CHANGE_OWNER
⋮ ☆	GERRIT_CHANGE_OWNER_EMAIL
⋮ ☆	GERRIT_CHANGE_OWNER_NAME
⋮ ☆	GERRIT_CHANGE_OWNER_USERNAME
⋮ ☆	GERRIT_CHANGE_SUBJECT

입력 매개 변수에 대한 자세한 내용은 작업을 수동으로 추가하기 전에 **Code Stream**에서 **CICD 네이티브 빌드 계획** 항목을 참조하십시오.

## 절차

- 1 **Code Stream**에서 **트리거 > Gerrit**을 클릭합니다.
- 2 (선택 사항) 수신기 탭을 클릭한 후 **새 수신기**를 클릭합니다.

---

**참고** Gerrit 트리거에 사용할 **Gerrit** 수신기가 이미 정의된 경우 이 단계를 건너뜁니다.

---

- a 프로젝트를 선택합니다.
- b Gerrit 수신기의 이름을 입력합니다.
- c Gerrit 끝점을 선택합니다.

## d API 토큰을 입력합니다.

CSP API 토큰은 Code Stream에 대한 외부 API 연결에 사용자를 인증합니다. API 토큰을 가져오려면:

## 1 토큰 생성을 클릭합니다.

2 사용자 이름과 연결된 이메일 주소와 암호를 입력하고 **생성**을 클릭합니다.

생성한 토큰은 6개월 동안 유효합니다. 새로 고침 토큰이라고도 합니다.

- 나중에 사용할 수 있도록 토큰을 변수로 유지하려면 **변수 생성**을 클릭하고 변수 이름을 입력한 후 **저장**을 클릭합니다.

- 나중에 사용할 수 있도록 토큰을 텍스트 값으로 유지하려면 **복사**를 클릭하고 토큰을 텍스트 파일에 붙여 넣어 로컬에 저장합니다.

나중에 사용할 수 있도록 변수를 생성하고 텍스트 파일에 토큰을 저장하도록 선택할 수 있습니다.

## 3 닫기를 클릭합니다.

변수를 생성한 경우 API 토큰은 달러 바인딩을 사용하여 입력한 변수 이름을 표시합니다. 토큰을 복사한 경우 API 토큰에 마스킹된 토큰이 표시됩니다.

The image shows the Gerrit configuration interface. At the top, there are tabs for '작업' (Jobs), '트리거' (Triggers), and '수신기' (Listeners), with '수신기' being the active tab. Below the tabs, there are four input fields: '프로젝트' (Project) with the value 'test1', '이름' (Name) with the value 'Gerrit-Demo-Listener', '끝점' (Endpoint) with the value 'corporate-gerrit', and 'API 토큰' (API Token) with the value '\$(var.CSuser API Token)'. To the right of the API Token field is a green checkmark icon and a button labeled '변수 생성' (Generate Variable). At the bottom, there are three buttons: '생성' (Generate), '검증' (Verify), and '취소' (Cancel).

e 토큰과 끝점 세부 정보를 검증하려면 **검증**을 클릭합니다.

토큰은 90일 후에 만료됩니다.

f **생성**을 클릭합니다.

g 수신기 카드에서 **연결**을 클릭합니다.

수신기가 Gerrit 서버에서 모든 작업을 모니터링하기 시작하고 해당 서버에서 사용되도록 설정된 모든 트리거를 수신합니다. 서버에서 트리거 수신을 중지하려면 트리거를 비활성화합니다.

**참고** 수신기에 연결된 Gerrit 끝점을 업데이트하려면 끝점을 업데이트하기 전에 수신기의 연결을 끊어야 합니다.

- **구성 > 트리거 > Gerrit**를 클릭합니다.
- **수신기** 탭을 클릭합니다.
- 업데이트하려는 끝점에 연결된 수신기에서 **연결 끊기**를 클릭합니다.

3 **트리거** 탭을 클릭하고 **새 트리거**를 클릭합니다.

4 Gerrit 서버에서 프로젝트를 선택합니다.

5 이름을 입력합니다.

Gerrit 트리거 이름은 고유해야 합니다.

6 구성된 Gerrit 수신기를 선택합니다.

Code Stream은 Gerrit 수신기를 사용하여 서버에서 사용할 수 있는 Gerrit 프로젝트 목록을 제공합니다.

7 Gerrit 서버에서 프로젝트를 선택합니다.

8 Gerrit 수신기가 모니터링할 저장소의 분기를 입력합니다.

9 (선택 사항) 파일 포함 또는 제외를 트리거에 대한 조건으로 제공합니다.

- **파이프라인을 트리거하는 파일 포함**을 제공합니다. 포함 경로 또는 정규식에 지정된 파일과 일치하는 파일이 하나라도 커밋에 있으면 파이프라인이 트리거됩니다. 정규식을 지정하는 경우 Code Stream은 변경 집합 중 제공된 식과 일치하는 파일 이름에 대해서만 파이프라인을 트리거합니다. 정규식 필터는 단일 저장소의 여러 파이프라인에 대해 트리거를 구성할 때 유용합니다.
- **파이프라인이 트리거되지 않도록 하는 파일 제외**를 제공합니다. 커밋의 모든 파일이 제외 경로 또는 정규식에 지정된 파일과 일치하면 파이프라인이 트리거되지 않습니다.
- **제외 우선 순위 지정**, 설정하면 파이프라인이 트리거되지 않도록 합니다. 커밋의 파일이 제외 경로 또는 정규식에 지정된 파일과 일치하더라도 파이프라인이 트리거되지 않습니다. **제외 우선 순위 지정**의 기본 설정은 꺼져 있습니다.

조건이 파일 포함과 파일 제외를 모두 충족하면 파이프라인이 트리거되지 않습니다.

다음 예에서 파일 포함과 파일 제외는 모두 트리거에 대한 조건입니다.

파일	포함	제외	제외 우선 순위 지정
runtime/src/main/a.java	PLAIN		
runtime/pom.xml	PLAIN		
runtime/demo.yaml	PLAIN		

- 파일 포함의 경우 runtime/src/main/a.java 또는 Java 파일에 대한 변경을 커밋하면 이벤트 구성에서 구성된 파이프라인이 트리거됩니다.
- 파일 제외의 경우 두 파일에 대한 변경만 커밋하면 이벤트 구성에 구성된 파이프라인이 트리거되지 않습니다.

## 10 새 구성을 클릭합니다.

- Gerrit 이벤트의 경우 **패치 집합 생성됨**, **초안 게시됨** 또는 **변경 사항 병합됨**을 선택합니다. 또는 Gerrit를 바이패스하는 Git로 직접 푸시하려면 **직접 Git 푸시**를 선택합니다.

**참고** Gerrit 릴리스 버전 2.15부터는 초안 변경 및 초안 변경 집합이 더 이상 지원되지 않습니다. 따라서 Gerrit 릴리스 버전 2.15 이상을 실행하는 경우 **초안 게시됨**은 지원되는 이벤트가 아닙니다.

- 트리거할 파이프라인을 선택합니다.

사용자 지정 추가 입력 매개 변수가 파이프라인에 포함된 경우 [입력 매개 변수] 목록에 매개 변수 및 값이 표시됩니다. 트리거 이벤트를 사용하여 파이프라인에 전달될 입력 매개 변수에 대한 값을 입력할 수 있습니다. 또는 값을 비워 두거나 기본값을 사용할 수 있습니다.

**참고** 기본값이 정의된 경우:

- 입력 매개 변수에 대해 입력하는 모든 값은 파이프라인 모델에서 정의된 기본값을 재정의합니다.
- 파이프라인 모델의 매개 변수 값이 변경되어도 트리거 구성의 기본값은 변경되지 않습니다.

Gerrit 트리거의 자동 삽입 입력 매개 변수에 대한 자세한 내용은 [사전 요구 사항](#)을 참조하십시오.

- 패치 집합 생성됨**, **초안 게시됨**, **변경 사항 병합됨**의 경우 일부 작업은 기본적으로 레이블과 함께 나타납니다. 레이블을 변경하거나 설명을 추가할 수 있습니다. 그런 다음 파이프라인이 실행될 때 **작업** 탭에서 파이프라인에 대한 레이블 또는 설명이 **작업 수행됨**으로 표시됩니다.

Gerrit 이벤트 구성을 사용하면 성공 주석 또는 실패 주석에 변수를 사용하여 주석을 입력할 수 있습니다. 예: \${var.success} 및 \${var.failure}.

- 저장**을 클릭합니다.

여러 파이프라인에 대해 여러 트리거 이벤트를 추가하려면 **새 구성**을 다시 클릭합니다.

다음 예에서는 파이프라인 3개에 대한 이벤트를 볼 수 있습니다.

- Gerrit 프로젝트에서 **변경 사항 병합됨** 이벤트가 발생하면 **Gerrit-Pipeline**이라는 파이프라인이 트리거됩니다.

- Gerrit 프로젝트에서 **패치 집합 생성됨** 이벤트가 발생하면 **Gerrit-Trigger-Pipeline** 및 **Gerrit-Demo-Pipeline**이라는 파이프라인이 트리거됩니다.

Gerrit 단계별 설정

작업 트리거 수신기

프로젝트 \* test1 ⓧ

이름 \* Gerrit-Demo-Trigger

Gerrit 수신기 \* Gerrit-Demo-Listener ▾

Gerrit 프로젝트 \* --Gerrit 프로젝트 선택-- ▾

분기 \* master

파일 ⓘ

포함 --선택-- ▾ 값 +

제외 --선택-- ▾ 값 +

제외 우선 순위 지정 ☐

+ 새 구성

이벤트 유형	파이프라인	레이블
⋮ Change Merged	Gerrit-Pipeline	
⋮ Patchset Created	Gerrit-Trigger-Pipeline	Verified
⋮ Patchset Created	Gerrit-Demo-Pipeline	Verified
3개 항목		

## 11 생성을 클릭합니다.

Gerrit 트리거는 **트리거** 탭에 새 카드로 표시되며 기본적으로 **사용 안 함**으로 설정됩니다.

## 12 트리거 카드에서 **사용**을 클릭합니다.

트리거를 사용하도록 설정한 후에는 **Gerrit** 수신기를 사용할 수 있습니다. 이것은 **Gerrit** 프로젝트의 분기에서 발생하는 이벤트 모니터링을 시작합니다.

파일 포함 조건 또는 파일 제외 조건이 동일하지만 트리거를 생성할 때 포함한 것과 다른 저장소를 사용하는 트리거를 생성하려면 트리거 카드에서 **작업 > 복제**를 클릭합니다. 그런 다음 복제된 트리거에서 **열기**를 클릭하고 매개 변수를 변경합니다.

## 결과

축하합니다! 서로 다른 3개의 파이프라인에 있는 2개의 이벤트가 포함된 **Gerrit** 트리거를 구성했습니다.

## 다음에 수행할 작업

Gerrit 프로젝트의 코드 변경을 커밋했으면 Code Stream에서 Gerrit 이벤트에 대한 **작업** 탭을 관찰합니다. 작업 목록에 트리거 구성의 모든 파이프라인 실행에 해당하는 항목이 포함되어 있는지 확인합니다.

이벤트가 발생하면 특정 유형의 이벤트와 관련된 Gerrit 트리거의 파이프라인만 실행될 수 있습니다. 이 예에서 패치 집합이 생성되면 **Gerrit-Trigger-Pipeline** 및 **Gerrit-Demo-Pipeline**만 실행됩니다.

**작업** 탭의 열에 있는 정보는 각 Gerrit 트리거 이벤트를 설명합니다. 테이블 아래에 나타나는 열 아이콘을 클릭하여 나타나는 열을 선택할 수 있습니다.

- 트리거가 직접 Git 푸시였다면 **제목 변경** 및 **실행** 열은 비어 있습니다.
- **Gerrit 트리거** 열에는 이벤트를 생성한 트리거가 표시됩니다.
- **수신기** 열은 기본적으로 꺼져 있습니다. 이 옵션을 선택하면 이벤트를 수신한 Gerrit 수신기가 열에 표시됩니다. 단일 수신기는 여러 트리거와 연결된 것으로 나타날 수 있습니다.
- **트리거 유형** 열은 기본적으로 꺼져 있습니다. 이 옵션을 선택하면 열에 트리거 유형이 자동 또는 수동으로 표시됩니다.
- 기타 열에는 **커밋 시간**, **변경#**, **상태**, **메시지**, **작업 수행됨**, **사용자**, **Gerrit 프로젝트**, **분기** 및 **이벤트**가 있습니다.

**Gerrit** GUIDED SETUP

Activity Triggers Listeners

TRIGGER MANUALLY ⓘ

Commit Time	Change#	Change Subject	Execution	Status	Message	Action taken	User	Gerrit project	Gerrit Trigger	Branch	Event
Nov 12, 2019, 12:47:53 PM	19570 /4	1111Dummy	Gerrit-Pipeline #1	COMPLETED	Execution Completed.	Verified +1	Drivank Upriyank@vm	test1	Gerrit-Demo-Trigger	master	Change Merged
Nov 12, 2019, 12:50:04 PM	19570 /6	11111Dummy	Gerrit-Pipeline #2	WAITING	Stage0.Task0: Execution Waiting for User Action.		Drivank Upriyank@vm	test1	Gerrit-Demo-Trigger	master	Change Merged
		11111Dummy	Gerrit-Demo-Pipeline #1	FAILED	Stage0.Task0: User Operation request has been rejected by Fritz.	Verified -1	Drivank Upriyank@vm	test1	Gerrit-Demo-Trigger	master	Patchset created
		11111Dummy	Gerrit-Trigger-Pipeline #1	WAITING	Stage0.Task0: Execution Waiting for User Action.		Drivank Upriyank@vm	test1	Gerrit-Demo-Trigger	master	Patchset created

Items per page: 20 1 - 4 of 4 items

완료된 또는 실패한 파이프라인 실행에 대한 작업을 제어하려면 [작업] 화면에서 항목 왼쪽에 있는 3개의 점을 클릭합니다.

- 파이프라인 모델에 실수가 있거나 다른 문제로 인해 파이프라인이 실행되지 못하면 실수를 수정하고 **다시 실행**을 선택하면 파이프라인이 다시 실행됩니다.

- 네트워크 연결 문제나 다른 문제로 인해 파이프라인이 실행되지 못할 때 **재개**를 선택하면 동일한 파이프라인 다시 시작되어 실행 시간이 절약됩니다.
- **실행 보기**를 사용하면 파이프라인 실행 보기가 열립니다. 파이프라인을 실행하고 결과를 보는 방법 항목을 참조하십시오.
- [작업] 화면에서 항목을 삭제하려면 **삭제**를 사용합니다.

Gerrit 이벤트가 파이프라인을 트리거하지 못하면 **수동으로 트리거**를 클릭한 다음 Gerrit 트리거를 선택하고 변경 ID를 입력한 후 **실행**을 클릭합니다.



# Code Stream에서 파이프라인 모니터링

## 8

Code Stream 관리자 또는 개발자는 Code Stream의 파이프라인 성능에 대한 인사이트가 필요합니다. 파이프라인이 개발에서 테스트를 거쳐 운영 환경으로 코드를 효율적으로 릴리스할 수 있는 방법을 알아야 할 필요가 있습니다.

인사이트를 얻으려면 Code Stream 대시보드를 사용하여 파이프라인 실행의 추세와 결과를 모니터링합니다. 기본 파이프라인 대시보드를 사용하여 단일 파이프라인을 모니터링하거나 사용자 지정 대시보드를 생성하여 여러 파이프라인을 모니터링할 수 있습니다.

- 파이프라인 메트릭에는 평균 시간과 같이 파이프라인 대시보드에서 사용할 수 있는 통계가 포함됩니다.
- 여러 파이프라인에 걸쳐 있는 메트릭을 보려면 사용자 지정 대시보드를 사용합니다.

본 장은 다음 항목을 포함합니다.

- [Code Stream에서 파이프라인 대시보드에 표시되는 내용](#)
- [Code Stream에서 사용자 지정 대시보드를 사용하여 파이프라인에 대한 주요 성능 지표를 추적하는 방법](#)

## Code Stream에서 파이프라인 대시보드에 표시되는 내용

파이프라인 대시보드는 추세, 상위 실패 및 성공한 변경 등 실행한 특정 파이프라인 결과에 대한 보기입니다. Code Stream은 사용자가 파이프라인을 생성할 때 파이프라인 대시보드를 생성합니다.

대시보드에는 파이프라인 실행 결과를 표시하는 위젯이 포함되어 있습니다.

### 파이프라인 실행 상태 수 위젯

상태(완료됨, 실패 또는 취소됨)별로 그룹화된 일정 기간 동안 파이프라인의 총 실행 수를 볼 수 있습니다. 길거나 짧은 기간 동안 파이프라인 실행 상태가 어떻게 변화했는지 확인하려면 표시에서 기간을 변경합니다.

### 파이프라인 실행 통계 위젯

파이프라인 실행 통계에는 시간 경과에 따른 파이프라인 복구, 전송 또는 실패에 소요된 평균 시간이 포함됩니다.

다음 상태는 모든 파이프라인 실행에 적용됩니다.

- 완료됨
- 실패
- 대기 중
- 실행 중
- 취소됨
- 대기열에 포함됨
- 시작되지 않음
- 롤백 중
- 롤백 완료됨
- 롤백 실패
- 일시 중지됨

표 8-1. 평균 시간 측정

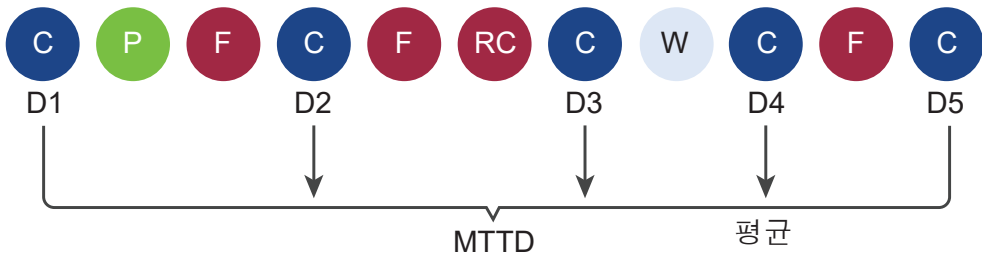
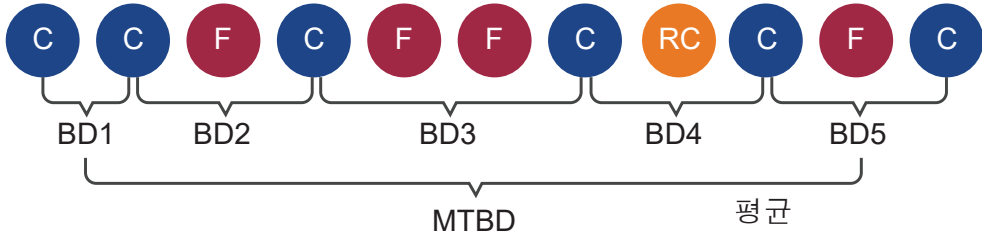
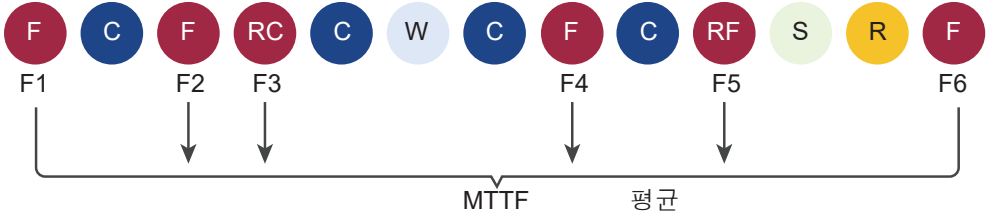
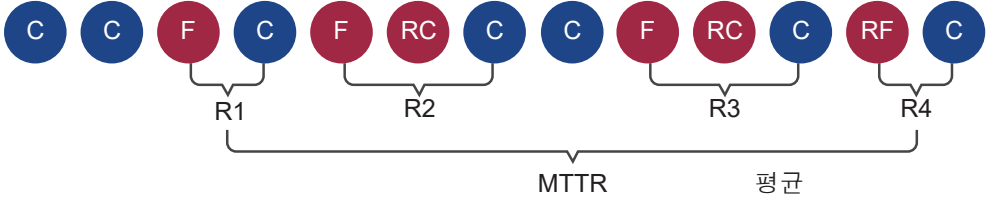
측정 항목...	의미...
평균 CI	CI 작업 유형에서 시간별로 측정된, 지속적 통합 단계에서 소요된 평균 시간입니다.
MTTD(평균 전송 시간)	<p>일정 기간 동안 모든 C(완료됨) 실행의 평균 소요 시간입니다. D1, D2 등은 각각의 C(완료됨) 실행을 전송하는 데 걸린 시간입니다.</p> 
MTBD(평균 전송 간격)	<p>일정 기간 동안 성공한 전송 사이에 경과된 평균 시간입니다. 성공한 전송 사이의 간격(예: BD1, BD2 등)이란 두 개의 연속 C(완료됨) 실행 사이에 경과된 시간입니다. MTBD는 운영 환경이 업데이트되는 빈도를 나타냅니다.</p> 

표 8-1. 평균 시간 측정 (계속)

측정 항목...	의미...
MTTF(평균 장애 시간)	<p>일정 기간 동안 F(실패), RC(물백 완료됨) 또는 RF(물백 실패) 상태로 끝난 실행의 평균 소요 시간입니다. F1, F2 등은 실행이 F(실패), RC(물백 완료됨) 또는 RF(물백 실패)로 끝나는 데 걸린 시간입니다.</p> 
MTTR(평균 복구 시간)	<p>일정 기간 동안 장애에서 복구하는 데 걸린 평균 시간입니다. 장애에서 복구하는 데 걸린 시간은 최종 상태가 F(실패), RC(물백 완료됨) 또는 RF(물백 실패)인 실행과 그 직후에 성공하여 C(완료됨) 상태인 실행 사이에 경과된 시간입니다. R1, R2 등은 F(실패) 또는 RF(물백 실패) 실행 후 복구하는 데 걸린 시간입니다.</p> 

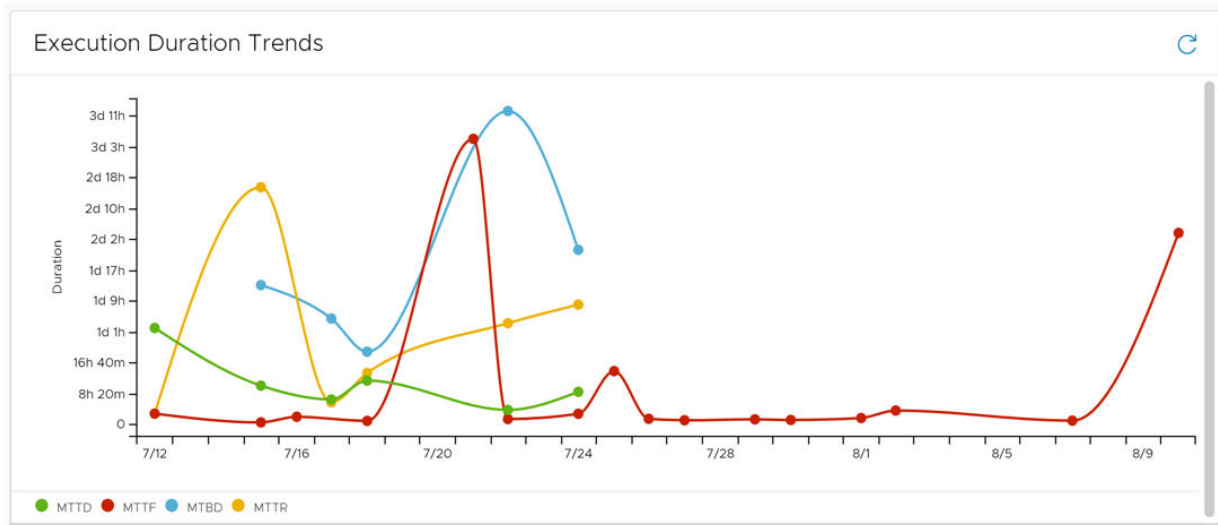
## 가장 많이 실패한 단계 및 작업 위젯

두 개의 위젯에는 파이프라인에서 가장 많이 실패한 단계와 작업이 표시됩니다. 각 측정은 각 파이프라인과 프로젝트에 대한 개발 및 개발 후 환경에 대한 매주 또는 매월 평균 실패 수와 비율을 보고합니다. 릴리스 자동화 프로세스에서 문제를 해결하려면 상위 오류를 확인합니다.

예를 들어 지난 7일과 같은 특정 기간 동안 디스플레이를 구성하고 해당 기간 동안 실패한 상위 작업을 기록해 볼 수 있습니다. 환경 또는 파이프라인에서 변경을 수행하고 파이프라인을 다시 실행한 후 더 긴 기간(지난 14일) 동안 실패한 상위 작업을 확인하면 실패한 상위 작업이 변경되었을 수 있습니다. 이러한 결과로 릴리스 자동화 프로세스의 변경이 파이프라인 실행의 성공률을 개선했음을 알 수 있습니다.

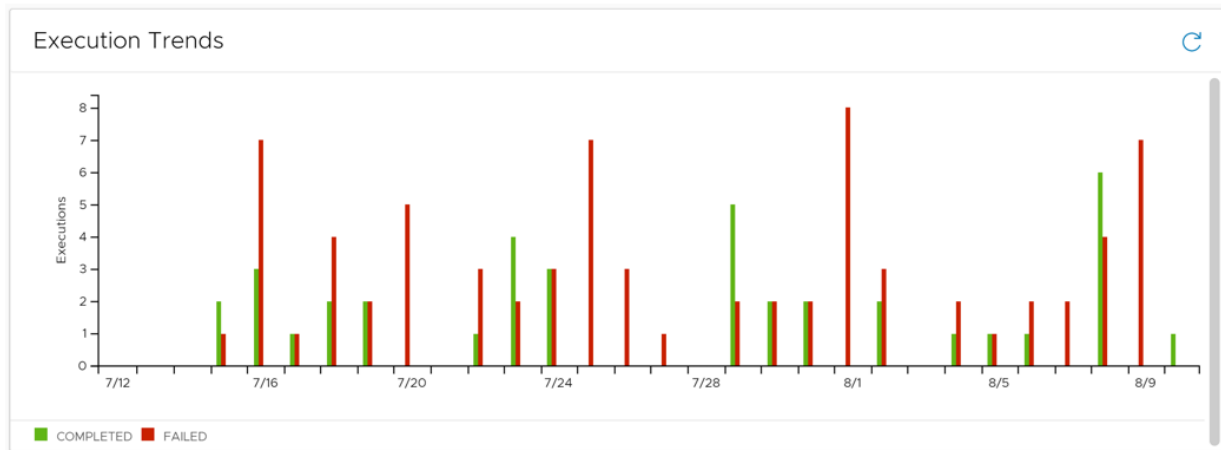
## 파이프라인 실행 기간 추세 위젯

파이프라인 실행 기간 추세에는 일정 기간 동안 MTTD, MTTF, MTBD 및 MTTR이 표시됩니다.



## 파이프라인 실행 추세 위젯

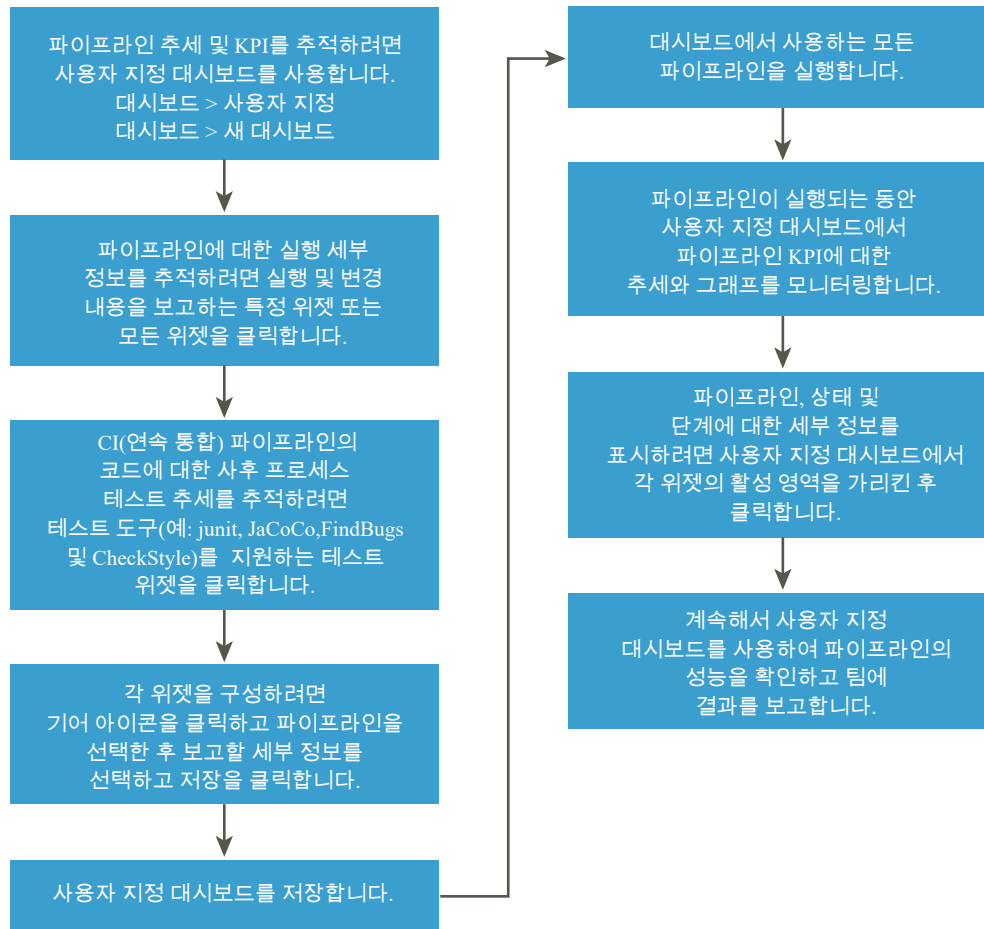
파이프라인 실행 추세에는 파이프라인의 총 일별 실행이 일정 기간 동안 상태별로 그룹화되어 표시됩니다. 현재 날짜를 제외하고 대부분의 일별 집계 수에는 C(완료됨) 및 F(실패) 실행만 표시됩니다.



## Code Stream에서 사용자 지정 대시보드를 사용하여 파이프라인에 대한 주요 성능 지표를 추적하는 방법

Code Stream 관리자 또는 개발자는 실행된 하나 이상의 파이프라인에 대해 확인하려는 결과를 표시하는 사용자 지정 대시보드를 생성합니다. 예를 들어 여러 파이프라인에서 수집한 KPI와 메트릭을 사용하여 프로젝트 범위의 대시보드를 생성할 수 있습니다. 실행 경고 또는 실패가 보고되면 대시보드를 사용하여 실패 문제를 해결할 수 있습니다.

사용자 지정 대시보드를 사용하여 파이프라인에 대한 추세 및 주요 성능 지표를 추적하려면 대시보드에 위젯을 추가하고 파이프라인에 대해 보고하도록 위젯을 구성합니다.



## 사전 요구 사항

- 하나 이상의 파이프라인이 있는지 확인합니다. 사용자 인터페이스에서 **파이프라인**을 클릭합니다.
- 모니터링하려는 파이프라인이 성공적으로 실행되었는지 확인합니다. **실행**을 클릭합니다.

## 절차

- 1 사용자 지정 대시보드를 생성하려면 **대시보드 > 사용자 지정 대시보드 > 새 대시보드**를 클릭합니다.
- 2 파이프라인에 대한 특정 추세 및 주요 성능 지표를 보고하도록 대시보드를 사용자 지정하려면 위젯을 클릭합니다.

예를 들어 파이프라인 상태, 단계, 작업, 실행 시간, 실행한 사람에 대한 세부 정보를 표시하려면 **실행 세부 정보** 위젯을 클릭합니다. 또한 CI(지속적 통합) 파이프라인의 경우 JUnit, JaCoCo, FindBugs 및 CheckStyle에 대한 위젯을 사용하여 처리 후 추세를 추적할 수 있습니다.

IX KPIS 작업						
실행 세부 정보						
실행#	상태	상태 메시지	모든 작업	TaskID (StageID)	기간	
#22	WAITING	StageID: Execution Waiting for User Action.			2 시간 37 분 55 초	
#21	COMPLETED	Execution Completed.			17 초	

**3** 추가한 각 위젯을 구성합니다.

- a 위젯에서 기어 아이콘을 클릭합니다.
- b 파이프라인을 선택하고, 사용 가능한 옵션을 설정하고, 표시할 열을 선택합니다.
- c 위젯 구성을 저장하려면 **저장**을 클릭합니다.
- d 사용자 지정 대시보드를 저장하려면 **저장**을 클릭하고 **닫기**를 클릭합니다.

**4** 파이프라인에 대한 세부 정보를 표시하려면 위젯에서 활성 영역을 클릭합니다.

예를 들어 **실행 세부 정보** 위젯에서 [상태] 열의 항목을 클릭하여 파이프라인 실행에 대한 세부 정보를 표시합니다. 또는 **성공한 최신 변경** 위젯에서 파이프라인 단계 및 작업에 대한 요약을 표시하려면 활성 링크를 클릭합니다.

**결과**

축하합니다! 파이프라인에 대한 추세와 KPI를 모니터링하는 사용자 지정 대시보드를 생성했습니다.

**다음에 수행할 작업**

Code Stream에서 파이프라인의 성능을 계속 모니터링하고 그 결과를 관리자 및 팀과 공유하여 애플리케이션 릴리스 프로세스를 지속적으로 개선합니다.

# Code Stream에 대해 알아보기

## 9

Code Stream 관리자 및 개발자가 Code Stream과 그 기능에 대해 알아볼 수 있는 방법에는 여러 가지가 있습니다.

이 설명서에서는 파이프라인과 파이프라인 실행, 끝점과 프로젝트를 추가하는 방법 등에 대해 자세히 알아볼 수 있습니다.

역할에 제공되는 사용 권한에 대해 설명합니다. 제한된 리소스를 사용하는 방법 및 파이프라인에 승인 요청을 설정하는 방법에 대해 설명합니다. [Code Stream에서 사용자 액세스 및 승인을 관리하는 방법](#)의 내용을 참조하십시오.

파이프라인, 실행 또는 끝점에서 특정 작업이나 구성 요소의 위치를 검색하여 유용한 검색 기능에 대해서도 배울 수 있습니다.

본 장은 다음 항목을 포함합니다.

- [Code Stream에서 검색이란?](#)
- [Code Stream 관리자 및 개발자를 위한 추가 리소스](#)

## Code Stream에서 검색이란?

검색은 특정 항목 또는 다른 구성 요소의 위치를 찾기 위해 사용하는 기능입니다. 예를 들어 활성화 또는 비활성화된 파이프라인을 검색할 수 있습니다. 이러한 검색 작업이 필요한 이유는 비활성화된 파이프라인은 실행할 수 없기 때문입니다.

## 검색할 수 있는 대상

검색할 수 있는 대상

- 프로젝트
- 끝점
- 파이프라인
- 실행
- 파이프라인 대시보드, 사용자 지정 대시보드
- Gerrit 트리거 및 서버

- Git Webhook
- Docker Webhook

열 기반 필터 검색을 수행할 수 있는 대상

- 사용자 작업
- 변수
- Gerrit, Git 및 Docker에 대한 트리거 작업

각 트리거의 **작업** 페이지에서 그리드 기반 필터 검색을 수행할 수 있습니다.

## 검색 작동 방식

검색 조건은 현재 위치한 페이지에 따라 다릅니다. 각 페이지마다 검색 조건이 다릅니다.

검색 위치	검색에 사용할 조건
파이프라인 대시보드	프로젝트, 이름, 설명, 태그, 링크
사용자 지정 대시보드	프로젝트, 이름, 설명, 링크(대시보드에 있는 항목의 UUID)
실행	이름, 주석, 이유, 태그, 인덱스, 상태, 프로젝트, 표시, 실행한 사람, 내가 실행한 항목, 링크(실행 UUID) 및 다음 형식을 사용한 입력 매개 변수, 출력 매개 변수 또는 상태 메시지: <key>:<value>
파이프라인	이름, 설명, 상태, 태그, 생성한 사용자, 내가 생성한 항목, 업데이트한 사용자, 내가 업데이트한 항목, 프로젝트
프로젝트	이름, 설명
끝점	이름, 설명, 유형, 업데이트한 사용자, 프로젝트
Gerrit 트리거	이름, 상태, 프로젝트
Gerrit 서버	이름, 서버 URL, 프로젝트
Git Webhook	이름, 서버 유형, 저장소, 분기, 프로젝트

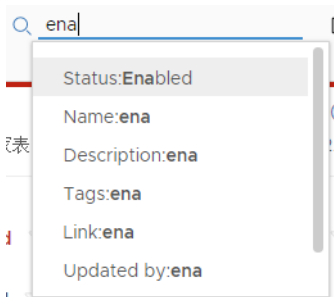
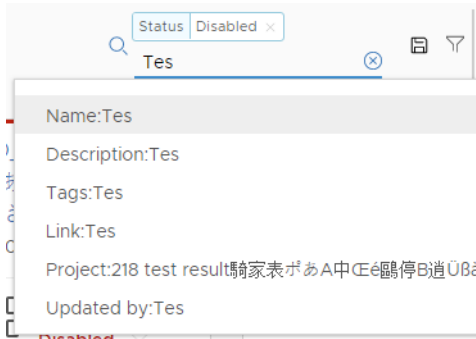
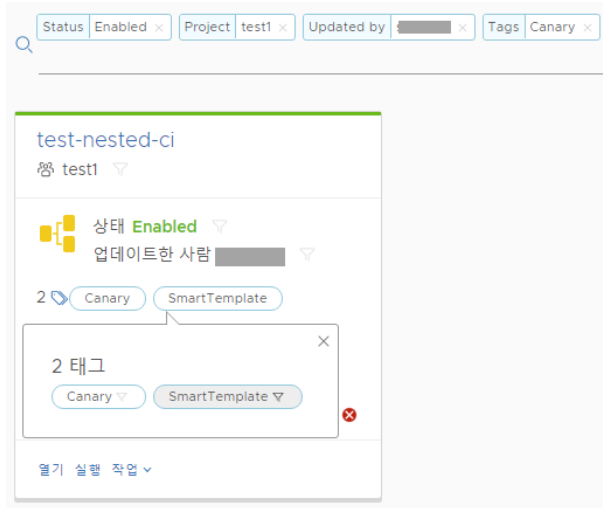
설명

- 링크는 대시보드의 위젯, 실행 또는 파이프라인의 UUID입니다.
- 입력 매개 변수, 출력 매개 변수 및 상태 메시지 표기법과 예는 다음과 같습니다.
  - 표기법: `input.<inputKey>:<inputValue>`  
예: `input.GERRIT_CHANGE_OWNER_EMAIL:joe_user`
  - 표기법: `output.<outputKey>:<outputValue>`  
예: `output.BuildNo:29`
  - 표기법: `statusMessage:<value>`  
예: `statusMessage:Execution failed`



- 상태는 검색 페이지에 따라 다릅니다.
  - 실행의 경우 가능한 값은 **completed**, **failed**, **rollback\_failed** 또는 **canceled**입니다.
  - 파이프라인의 경우 가능한 상태 값은 **사용**, **사용 안 함** 또는 **릴리스됨**입니다.
  - 트리거의 경우 가능한 상태 값은 **사용** 또는 **사용 안 함**입니다.
- 내가 실행한 항목, 내가 생성한 항목 또는 내가 업데이트한 항목은 현재 로그인한 사용자 본인의 항목을 나타냅니다.

검색은 유효한 모든 페이지의 오른쪽 상단에 나타납니다. 검색 빈 칸에 내용을 입력하기 시작하면 Code Stream은 페이지의 컨텍스트를 토대로 검색 옵션을 제안합니다.

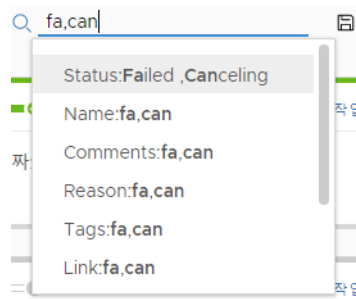
검색 시 사용할 수 있는 방법	입력 방법
<p>검색 매개 변수의 일부를 입력합니다.</p> <p>예를 들어, 사용하도록 설정된 모든 파이프라인을 나열하는 상태 필터를 추가하려면 <b>ena</b>를 입력합니다.</p>	
<p>발견된 항목 수를 줄이면 필터를 추가합니다.</p> <p>예를 들어 <b>Tes</b>를 입력하여 이름 필터를 추가합니다. 필터는 기존의 <b>상태:disabled</b> 필터에 대해 AND로 연결되어 비활성화되어 있고 이름에 <b>Tes</b>가 들어 있는 파이프라인만 표시합니다.</p> <p>다른 필터를 추가하면 나머지 옵션(<b>이름, 설명, 태그, 링크, 프로젝트</b> 및 <b>업데이트한 사람</b>)이 나타납니다.</p>	
<p>표시되는 항목 수를 줄이려면 파이프라인 또는 파이프라인 실행의 속성에서 필터 아이콘을 클릭합니다.</p> <ul style="list-style-type: none"> <li>■ 파이프라인의 경우 <b>상태</b>, <b>태그</b>, <b>프로젝트</b> 및 <b>업데이트한 사람</b>마다 필터 아이콘이 있습니다.</li> <li>■ 실행의 경우 <b>태그</b>, <b>실행한 사람</b> 및 <b>상태 메시지</b>마다 필터 아이콘이 있습니다.</li> </ul> <p>예를 들어 파이프라인 카드에서 아이콘을 클릭하여 <b>SmartTemplate</b> 태그에 대한 필터를 <b>Status:Enabled</b>, <b>Project:test</b>, <b>Updated by:user</b> 및 <b>Tags:Canary</b>의 기존 필터에 추가합니다.</p>	

## 검색 시 사용할 수 있는 방법

쉽표 구분 기호를 사용하여 두 가지 실행 상태의 모든 항목을 포함합니다.

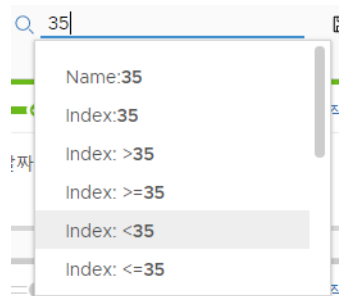
예를 들어 **fa,can**을 입력하여 실패하거나 취소된 실행 모두 나열하도록 OR로 연결된 상태 필터를 생성합니다.

## 입력 방법



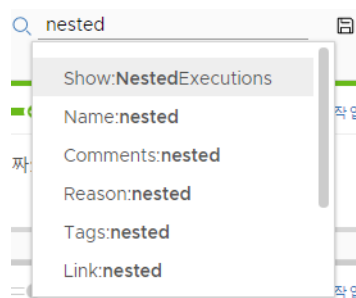
숫자를 입력하여 인덱스 범위 내에 있는 모든 항목을 포함합니다.

예를 들어 **35**를 입력한 후 **<**를 선택하여, 인덱스 번호가 35 미만인 모든 실행을 나열합니다.



작업으로 모델링된 파이프라인은 중첩된 실행이 되기 때문에 기본적으로 다른 모든 실행과 함께 나열되지 않습니다.

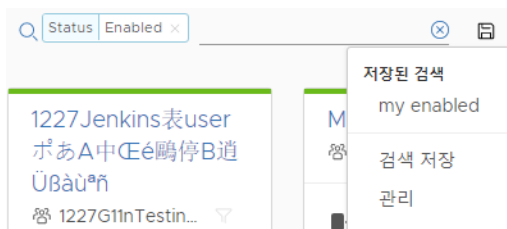
중첩된 실행을 표시하려면 **nested**를 입력한 후 **표시** 필터를 선택합니다.



## 즐거찾기 검색을 저장하는 방법

검색 영역 옆에 있는 디스크 아이콘을 클릭하면 각 페이지에서 사용할 즐거찾기 검색을 저장할 수 있습니다.

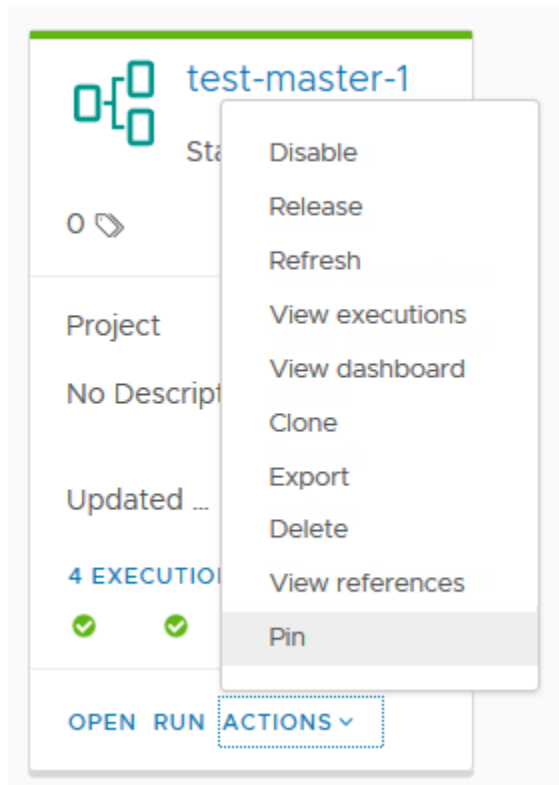
- 검색할 매개 변수를 입력하고 아이콘을 클릭하여 검색 이름(예: **my enabled**)을 지정하는 방법으로 검색을 저장합니다.
- 검색을 저장한 후에는 아이콘을 클릭하여 해당 검색을 액세스할 수 있습니다. **관리**를 선택하면 저장된 검색 목록에서 검색을 삭제하거나, 이동하거나, 이름을 바꿀 수도 있습니다.



검색은 사용자 이름에 연결되며 해당 검색이 적용되는 페이지에만 나타납니다. 예를 들어 파이프라인 페이지에서 **상태:enabled** 검색을 **my enabled**라는 이름으로 저장한 경우, Gerrit 트리거 페이지에서도 **상태:enabled**가 유효한 트리거 검색이지만 이 페이지에서는 **my enabled** 검색을 사용할 수 없습니다.

## 즐거찾기 파이프라인을 저장할 수 있는지 여부

즐거찾기 파이프라인 또는 대시보드가 있는 경우에는 이를 고정하여 파이프라인 또는 대시보드 페이지 상단에 항상 표시되도록 할 수 있습니다. 파이프라인 카드에서 **작업 > 고정**을 클릭합니다.



## Code Stream 관리자 및 개발자를 위한 추가 리소스

Code Stream 관리자 또는 개발자는 Code Stream에 대해 자세히 알아볼 수 있습니다.

표 9-1. 관리자를 위한 추가 리소스

주제...	참조할 리소스...
<p>관리자가 Code Stream을 사용할 수 있는 다른 방식:</p> <ul style="list-style-type: none"> <li>클라우드 네이티브 애플리케이션의 테스트 및 릴리스를 자동화하기 위한 파이프라인을 구성합니다.</li> <li>테스트에서 운영에 이르기까지 개발자 소스 코드를 자동화하고 테스트합니다.</li> <li>개발자가 변경 사항을 기본 분기에 커밋하기 전에 변경 사항을 테스트하기 위한 파이프라인을 구성합니다.</li> <li>주요 파이프라인 메트릭을 추적합니다.</li> </ul>	<p>Code Stream</p> <ul style="list-style-type: none"> <li><a href="#">vRealize Automation 설명서</a></li> <li><a href="#">vRealize Automation 제품 웹 사이트</a></li> </ul> <p>VMware Hands On</p> <ul style="list-style-type: none"> <li><a href="#">vRealize Automation 커뮤니티 활용</a></li> <li><a href="#">VMware Learning Zone 활용</a></li> <li><a href="#">VMware 블로그 검색</a></li> <li><a href="#">VMware Hands On Labs 사용</a></li> </ul>

표 9-2. 개발자를 위한 추가 리소스

주제...	참조할 리소스...
<p>개발자가 Code Stream를 사용할 수 있는 다른 방식:</p> <ul style="list-style-type: none"> <li>■ 공용 및 전용 레지스트리 이미지를 사용하여 새 애플리케이션 또는 서비스에 대한 환경을 구축합니다.</li> <li>■ 최신 안정적 빌드에서 분기를 생성할 수 있도록 개발 환경을 설정합니다.</li> <li>■ 최신 코드 변경 내용 및 아티팩트로 개발 환경을 업데이트합니다.</li> <li>■ 다른 종속된 서비스의 최신 안정적 빌드를 기준으로 커밋되지 않은 코드 변경을 테스트합니다.</li> <li>■ 기본 CICD 파이프라인에 커밋된 변경 사항으로 인해 다른 서비스가 중단될 때 알림을 받습니다.</li> </ul>	<p>Code Stream</p> <ul style="list-style-type: none"> <li>■ <a href="#">vRealize Automation 설명서</a></li> <li>■ <a href="#">vRealize Automation 제품 웹 사이트</a></li> </ul> <p>VMware Hands On</p> <ul style="list-style-type: none"> <li>■ <a href="#">vRealize Automation 커뮤니티 활용</a></li> <li>■ <a href="#">VMware Learning Zone 활용</a></li> <li>■ <a href="#">VMware 블로그 검색</a></li> <li>■ <a href="#">VMware Hands On Labs 사용</a></li> </ul>