Red Hat OpenShift Pipelines 1.15

코드로서의 파이프라인

Pipeline을 코드로 구성 및 사용

Last Updated: 2024-09-02

Red Hat OpenShift Pipelines 1.15 코드로서의 파이프라인

Pipeline을 코드로 구성 및 사용

법적 공지

초록

이 문서에서는 Git 소스 코드 리포지토리의 일부로 파이프라인 템플릿을 정의할 수 있는 OpenShift Pipelines 하위 시스템인 Pipeline 구성 및 사용에 대한 정보를 제공합니다.

차례

1장. PIPELINE AS CODE 정보

Pipeline을 Code로 사용하면 클러스터 관리자와 필요한 권한이 있는 사용자가 파이프라인 템플릿을 소스 코드 Git 리포지토리의 일부로 정의할 수 있습니다. 소스 코드 푸시 또는 구성된 Git 리포지토리에 대한 가져오기 요청에 의해 트리거되는 경우 Code로 Pipeline이 파이프라인을 실행하고 상태를 보고합니다.

1.1. 주요 기능

코드로서의 파이프라인은 다음 기능을 지원합니다.

- Git 리포지토리를 호스팅하는 플랫폼에서 요청 상태 및 제어 권한을 가져옵니다.
- GitHub Checks API를 사용하여 재확인을 포함하여 파이프라인 실행 상태를 설정합니다.
- GitHub 가져오기 요청 및 커밋 이벤트입니다.
- 주석에서 요청 작업을 가져옵니다(예: **/retest**).
- 각 이벤트에 대해 Git 이벤트 필터링 및 별도의 파이프라인입니다.
- 로컬 작업, Tekton Hub 및 원격 URL을 포함하여 OpenShift Pipelines의 자동 작업 확인
- GitHub Blob 및 오브젝트 API를 사용하여 구성 검색.
- GitHub 조직 또는 Prow 스타일 **OWNERS** 파일을 통해 ACL(액세스 제어 목록)을 사용합니다.
- 코드 리포지토리로 부트스트랩 및 Pipeline을 관리하기 위한 **tkn pac** CLI 플러그인입니다.
- GitHub 앱, GitHub Webhook, Bitbucket 서버 및 Bitbucket Cloud 지원

2장. PIPELINE을 코드로 설치 및 구성

Red Hat OpenShift Pipelines 설치의 일부로 Pipeline을 코드로 설치할 수 있습니다.

2.1. OPENSHIFT CONTAINER PLATFORM에서 PIPELINE을 코드로 설치

Red Hat OpenShift Pipelines Operator를 설치할 때 코드로서의 파이프라인은 **openshift-pipelines** 네임스페이스에 설치됩니다. 자세한 내용은 *추가 리소스* 섹션에서 *OpenShift Pipelines* 설치를 참조하십시오.

Operator를 사용하여 Pipeline의 기본 설치를 비활성화하려면 **TektonConfig** 사용자 정의 리소스에서 **enable** 매개변수 값을 **false** 로 설정합니다.

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  platforms:
    openshift:
      pipelinesAsCode:
        enable: false
        settings:
          application-name: Pipelines as Code CI
          auto-configure-new-github-repo: "false"
          bitbucket-cloud-check-source-ip: "true"
          hub-catalog-name: tekton
          hub-url: https://api.hub.tekton.dev/v1
          remote-tasks: "true"
          secret-auto-create: "true"
# ...
```

선택적으로 다음 명령을 실행할 수 있습니다.

```
$ oc patch tektonconfig config --type="merge" -p '{"spec": {"platforms": {"openshift":
{"pipelinesAsCode": {"enable": false}}}}}'
```

Red Hat OpenShift Pipelines Operator를 사용하여 Pipeline을 코드로 기본 설치를 활성화하려면 **TektonConfig** 사용자 정의 리소스에서 **enable** 매개변수 값을 **true** 로 설정합니다.

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  platforms:
    openshift:
      pipelinesAsCode:
        enable: true
        settings:
          application-name: Pipelines as Code CI
          auto-configure-new-github-repo: "false"
          bitbucket-cloud-check-source-ip: "true"
          hub-catalog-name: tekton
          hub-url: https://api.hub.tekton.dev/v1
```

```
        remote-tasks: "true"
        secret-auto-create: "true"
# ...
```

선택적으로 다음 명령을 실행할 수 있습니다.

```
$ oc patch tektonconfig config --type="merge" -p '{"spec": {"platforms": {"openshift":
{"pipelinesAsCode": {"enable": true}}}}}'
```

2.2. PIPELINE을 코드 CLI로 설치

클러스터 관리자는 로컬 머신 또는 테스트를 위한 컨테이너로 **tkn pac** 및 **opc** CLI 툴을 사용할 수 있습니다. Red Hat OpenShift Pipelines용 **tkn** CLI를 설치할 때 **tkn pac** 및 **opc** CLI 도구가 자동으로 설치됩니다.

지원되는 플랫폼에 대해 **tkn pac** 및 **opc** 버전 **1.15.0** 바이너리를 설치할 수 있습니다.

- Linux (x86_64, amd64)
- Linux on IBM zSystems and IBM® LinuxONE (s390x)
- Linux on IBM Power (ppc64le)
- Linux on ARM (aarch64, arm64)
- macOS
- Windows

2.3. PIPELINE을 코드 구성으로 사용자 정의

클러스터 관리자는 Code로 Pipeline을 사용자 지정하기 위해 **platforms.openshift.pipelinesAsCode.settings** 사양의 **TektonConfig** 사용자 지정 리소스에서 다음 매개변수를 구성할 수 있습니다.

표 2.1. Pipeline을 코드 구성으로 사용자 정의

매개변수	설명	기본
application-name	애플리케이션 이름입니다. 예를 들어 GitHub Checks 라벨에 표시되는 이름의 경우입니다.	**"pipelines as Code CI"**
secret-auto-create	GitHub 애플리케이션에서 생성된 토큰을 사용하여 시크릿을 자동으로 생성해야 하는지 여부를 나타냅니다. 그런 다음 이 시크릿을 프라이빗 리포지토리와 함께 사용할 수 있습니다.	**enabled**
remote-tasks	활성화하면 파이프라인 실행 주석에서 원격 작업을 허용합니다.	**enabled**

매개변수	설명	기본
hub-url	Tekton Hub API 의 기본 URL입니다.	**https://hub.tekton.dev/**
hub-catalog-name	Tekton Hub 카탈로그 이름입니다.	**Tekton**
tekton-dashboard-url	Tekton Hub 대시보드의 URL입니다. 코드로서의 파이프라인은 이 URL을 사용하여 Tekton Hub 대시보드에서 **PipelineRun** URL을 생성합니다.	해당 없음
bitbucket-cloud-check-source-ip	공용 Bitbucket의 IP 범위를 쿼리하여 서비스 요청을 보호할지 여부를 나타냅니다.Indicates whether to secure the service requests by querying IP ranges for a public Bitbucket. 매개변수의 기본값을 변경하면 보안 문제가 발생할 수 있습니다.	**enabled**
bitbucket-cloud-additional-source-ip	쉼표로 구분된 추가 IP 범위 또는 네트워크 세트를 제공할지 여부를 나타냅니다.	해당 없음
max-keep-run-upper-limit	파이프라인 실행의 **max-keep-run** 값에 대한 최대 제한입니다.	해당 없음
default-max-keep-runs	파이프라인 실행의 **max-keep-run** 값에 대한 기본 제한입니다. 정의된 경우 **max-keep-run** 주석이 없는 모든 파이프라인 실행에 값이 적용됩니다.	해당 없음
auto-configure-new-github-repo	새 GitHub 리포지토리를 자동으로 구성합니다. 코드로서의 파이프라인은 네임스페이스를 설정하고 리포지토리에 대한 사용자 지정 리소스를 생성합니다. 이 매개변수는 GitHub 애플리케이션에서만 지원됩니다.	**비활성화됨**
auto-configure-repo-namespace-template	**auto-configure-new-github-repo** 가 활성화된 경우 새 리포지토리에 대한 네임스페이스를 자동으로 생성하도록 템플릿을 구성합니다.	**{repo_name}-pipelines**

매개변수	설명	기본
error-log-snippet	파이프라인에 오류가 있고 실패한 작업의 로그 스니펫 보기를 활성화하거나 비활성화합니다. 파이프라인에서 데이터 누출의 경우 이 매개변수를 비활성화할 수 있습니다.	**true**
error-detection-from-container-logs	컨테이너 로그 검사를 활성화하거나 비활성화하여 오류 메시지를 감지하고 가져오기 요청에 주석으로 노출합니다. 이 설정은 GitHub 앱을 사용하는 경우에만 적용됩니다.	**true**
error-detection-max-number-of-lines	컨테이너 로그에서 검사한 최대 행 수로 오류 메시지를 검색합니다. 무제한의 행 수를 검사하려면 **-1** 로 설정합니다.	50
secret-github-app-token-scoped	**true** 로 설정하면 GitHub 앱을 사용하여 Pipeline이 생성하는 GitHub 액세스 토큰의 범위는 Code에서 Pipeline 정의를 가져오는 리포지토리에만 범위가 지정됩니다. **false** 로 설정하면 **TektonConfig** 사용자 정의 리소스와 **Repository** 사용자 정의 리소스를 모두 사용하여 토큰의 범위를 추가 리포지토리로 지정할 수 있습니다.	**true**
secret-github-app-scope-extra-repos	생성된 GitHub 액세스 토큰의 범위를 지정하는 추가 리포지토리입니다.	

2.4. 추가 GITHUB 앱을 지원하기 위해 추가 PIPELINE을 코드 컨트롤러로 구성

기본적으로 파이프라인을 코드로 구성하여 하나의 GitHub 애플리케이션과 상호 작용할 수 있습니다. 예를 들어 다른 GitHub 계정 또는 GitHub Enterprise 또는 GitHub SaaS와 같은 다른 GitHub 인스턴스를 사용해야 하는 경우와 같이 둘 이상의 GitHub 앱을 사용해야 하는 경우도 있습니다. 둘 이상의 GitHub 앱을 사용하려면 추가 Pipeline을 모든 추가 GitHub 앱에 대한 코드 컨트롤러로 구성해야 합니다.

프로세스

1. **TektonConfig** 사용자 지정 리소스에서 다음 예와 같이 **platform.openshift.pipelinesAsCode** 사양에 **additionalPACControllers** 섹션을 추가합니다.

   additionalPACControllers 섹션의 예

```
apiVersion: operator.tekton.dev/v1
kind: TektonConfig
metadata:
  name: config
spec:
  platforms:
    openshift:
      pipelinesAsCode:
        additionalPACControllers:
          pac_controller_2: (1)
            enable: true (2)
            secretName: pac_secret_2 (3)
            settings: # (4)
# ...
```

(1) 컨트롤러의 이름입니다. 이름은 고유해야 하며 길이 25자를 초과할 수 없습니다.

(2) 이 매개변수는 선택 사항입니다. 추가 컨트롤러를 활성화하려면 이 매개변수를 **true** 로 설정하여 추가 컨트롤러를 활성화하거나 **false** 로 설정합니다. 기본 vaule은 **true** 입니다.

(3) 이 매개변수를 GitHub 앱에 생성해야 하는 시크릿 이름으로 설정합니다.

(4) 이 섹션은 선택 사항입니다. 이 섹션에서는 설정이 코드 컨트롤러의 기본 Pipeline과 달라야 하는 경우 이 컨트롤러의 코드 설정으로 Pipeline을 설정할 수 있습니다.

2. 선택 사항: 두 개 이상의 GitHub 앱을 사용하려면 **pipelinesAsCode.additionalPACControllers** 사양에 추가 섹션을 생성하여 Pipeline을 모든 GitHub 인스턴스에 대한 코드 컨트롤러로 구성합니다. 모든 컨트롤러에 고유한 이름을 사용합니다.

추가 리소스

- Pipeline을 코드 구성으로 사용자 정의
- 수동으로 GitHub 앱 구성 및 Pipeline의 시크릿 생성

2.5. 추가 리소스

- OpenShift Pipelines 설치
- tkn 설치
- Red Hat OpenShift Pipelines 릴리스 정보

3장. GIT 리포지토리 호스팅 서비스 공급자와 함께 PIPELINE을 코드로 사용

Pipeline을 코드로 설치한 후 클러스터 관리자는 Git 리포지토리 호스팅 서비스 공급자를 구성할 수 있습니다. 현재 다음 서비스가 지원됩니다.

- GitHub App
- GitHub Webhook
- GitLab
- Bitbucket 서버
- Bitbucket 클라우드

참고

GitHub App은 Pipeline과 함께 Code를 사용하는 데 권장되는 서비스입니다.

3.1. GITHUB 앱에서 PIPELINE을 코드로 사용

GitHub 앱은 Red Hat OpenShift Pipelines와의 통합 지점 역할을 하며 OpenShift Pipelines에 Git 기반 워크플로를 활용할 수 있습니다. 클러스터 관리자는 모든 클러스터 사용자에 대해 단일 GitHub 앱을 구성할 수 있습니다. GitHub Apps가 코드로 Pipeline에서 작동하도록 하려면 GitHub 앱의 Webhook가 GitHub 이벤트를 수신하는 코드 컨트롤러 경로(또는 수신 끝점)로 Pipeline을 가리키는지 확인합니다.

Pipeline용 GitHub 앱을 코드로 설정하는 방법은 다음 세 가지가 있습니다.

- **tkn** 명령줄 유틸리티를 사용합니다.
- 웹 콘솔의 관리자 화면을 사용합니다.
- GitHub에서 수동으로 앱을 설정한 다음 코드로 Pipeline의 시크릿을 생성합니다.

기본적으로 Pipeline as Code는 하나의 GitHub 애플리케이션과 통신할 수 있습니다. 추가 Pipeline을 추가 GitHub 앱과 통신하도록 코드 컨트롤러로 구성한 경우 각 GitHub 앱을 별도로 구성합니다. 추가 컨트롤러의 GitHub 앱을 수동으로 설정해야 합니다.

3.1.1. 명령줄 인터페이스를 사용하여 GitHub 앱 구성

tkn 명령줄 유틸리티를 사용하여 GitHub 앱을 생성하고 Pipeline을 GitHub 앱의 코드 컨트롤러로 구성할 수 있습니다.

중요

추가 GitHub 앱을 지원하기 위해 추가 Pipeline을 코드 컨트롤러로 생성한 경우 기본 컨트롤러에만 이 절차를 사용할 수 있습니다. 추가 컨트롤러의 GitHub 앱을 생성하려면 수동 절차를 사용합니다.

사전 요구 사항

- 클러스터 관리자로 OpenShift Container Platform 클러스터에 로그인되어 있습니다.

- **tkn pac** 플러그인을 사용하여 tkn 명령줄 유틸리티를 설치했습니다.

프로세스

- 다음 명령을 실행합니다.

```
$ tkn pac bootstrap github-app
```

이 명령은 계정에서 표준 github.com API 끝점을 사용한다고 가정합니다. 예를 들어 GitHub Enterprise를 사용하는 경우 다른 GitHub API 끝점을 사용하는 경우 다음 예와 같이 **--github-api-url** 옵션을 사용하여 엔드포인트를 지정합니다.

명령 예

```
$ tkn pac bootstrap github-app --github-api-url https://github.com/enterprises/example-enterprise
```

3.1.2. 관리자 화면에서 GitHub 앱 생성

클러스터 관리자는 OpenShift Container Platform 클러스터로 GitHub 앱을 구성하여 Pipeline을 코드로 사용할 수 있습니다. 이 구성을 사용하면 빌드 배포에 필요한 작업 세트를 실행할 수 있습니다.

중요

추가 GitHub 앱을 지원하기 위해 추가 Pipeline을 코드 컨트롤러로 생성한 경우 기본 컨트롤러에만 이 절차를 사용할 수 있습니다. 추가 컨트롤러의 GitHub 앱을 생성하려면 수동 절차를 사용합니다.

사전 요구 사항

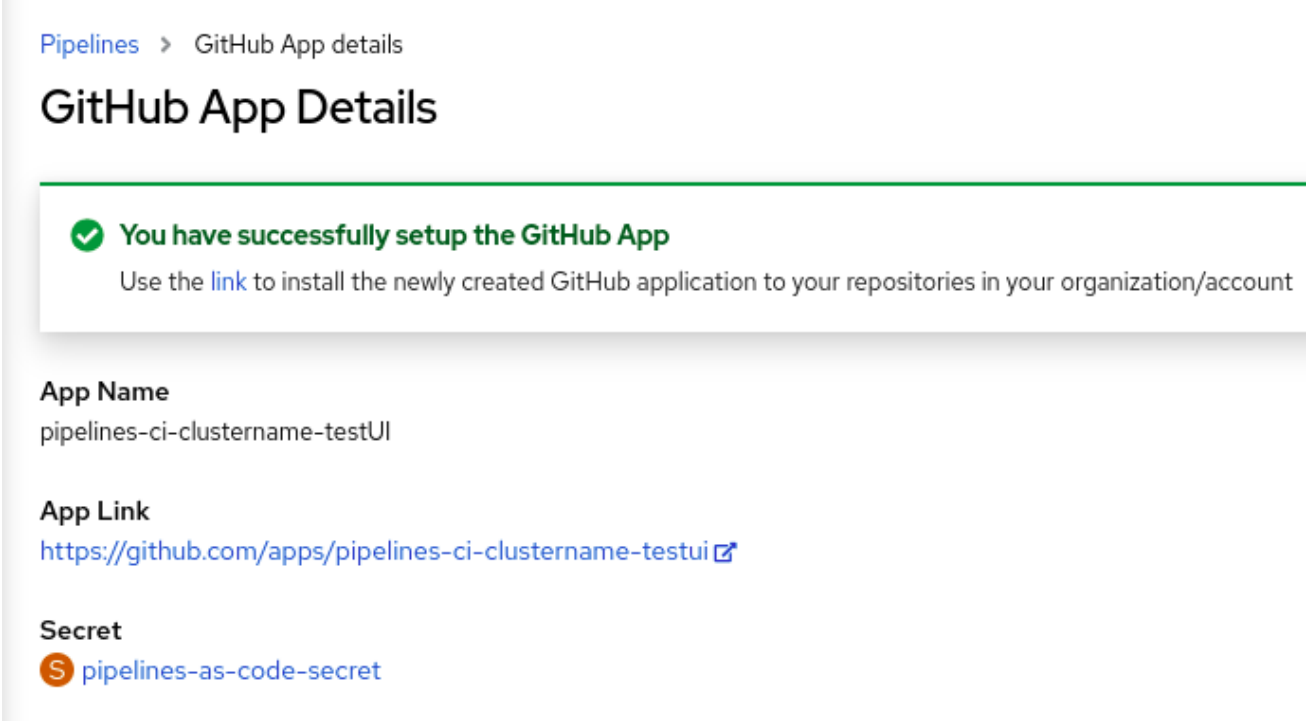
Operator Hub에서 Red Hat OpenShift Pipelines **pipelines-1.15** Operator를 설치했습니다.

프로세스

1. 관리자 화면에서 탐색 창을 **사용하여 파이프라인** 으로 이동합니다.
2. **파이프라인 페이지에서 GitHub 앱** 설정을 클릭합니다.
3. GitHub 앱 이름을 입력합니다. 예를 들면 **pipelines-ci-clustername-testui** 입니다.
4. 설정을 **클릭합니다**.
5. 브라우저에 메시지가 표시되면 Git 암호를 입력합니다.
6. **Create GitHub App for <username>**을 클릭합니다. 여기서 < **username** >은 GitHub 사용자 이름입니다.

검증

GitHub 앱을 성공적으로 생성하면 OpenShift Container Platform 웹 콘솔이 열리고 애플리케이션에 대한 세부 정보가 표시됩니다.



GitHub 앱의 세부 정보는 **openShift-pipelines** 네임스페이스에 시크릿으로 저장됩니다.

GitHub 애플리케이션과 연결된 이름, 링크 및 시크릿과 같은 세부 정보를 보려면 파이프라인으로 **이동하여 GitHub 앱 보기를** 클릭합니다.

3.1.3. 수동으로 GitHub 앱 구성 및 Pipeline의 시크릿 생성

GitHub 사용자 인터페이스를 사용하여 GitHub 앱을 만들 수 있습니다. 그러면 GitHub 앱에 연결하기 위해 Pipeline을 코드로 구성하는 시크릿을 생성해야 합니다.

추가 GitHub 앱을 지원하기 위해 추가 Pipeline을 코드 컨트롤러로 생성한 경우 추가 컨트롤러에 이 절차를 사용해야 합니다.

프로세스

1. GitHub 계정에 로그인합니다.
2. GitHub 메뉴에서 **설정 → 개발자 설정 → GitHub 앱을** 선택한 다음 **새 GitHub 앱** 을 클릭합니다.
3. GitHub 앱 양식에 다음 정보를 제공합니다.
   - **GitHub 애플리케이션 이름:OpenShift Pipelines**
   - **홈페이지 URL**: OpenShift 콘솔 URL
   - **Webhook URL**: 코드 경로 또는 인그레스 URL로 파이프라인입니다. 다음 명령을 실행하여 찾을 수 있습니다.

     ```
     $ echo https://$(oc get route -n openshift-pipelines pipelines-as-code-controller -o
     jsonpath='{.spec.host}')
     ```

     또는 추가 Pipeline에 대한 GitHub 앱을 코드 컨트롤러로 구성하려면 다음 예와 같이 **pipelines-as-code-controller** 를 구성한 컨트롤러 이름으로 교체합니다.

     명령 예

```
$ echo https://$(oc get route -n openshift-pipelines pac_controller_2 -o
jsonpath='{.spec.host}')
```

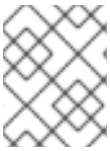
- **Webhook 보안**: 임의의 시크릿입니다. 다음 명령을 실행하여 보안을 생성할 수 있습니다.

```
$ openssl rand -hex 20
```

4. **리포지토리 권한** 섹션에서 다음 항목을 선택합니다.
   - **검사:읽기 및 쓰기**
   - **내용:읽기 & 쓰기**
   - **문제:읽기 & 쓰기**
   - **metadata:읽기 전용**
   - **pull request:Read & Write**
5. **조직 권한** 섹션에서 다음 항목을 선택합니다.
   - **멤버:읽기 전용**
   - **계획:읽기 전용**
6. 다음 이벤트를 구독합니다.
   - **실행 확인**
   - **도구 모음 확인**
   - **커밋 주석**
   - **문제 주석**
   - **가져오기 요청**
   - **push**
7. **GitHub 앱 생성을** 클릭합니다.
8. 새로 생성된 GitHub 앱의 **세부 정보** 페이지에서 맨 위에 표시된 **앱 ID** 를 확인합니다.
9. **개인 키** 섹션에서 **개인 키 생성** 을 클릭하여 GitHub 앱의 개인 키를 자동으로 생성하고 다운로드합니다. 나중에 참조 및 사용을 위해 개인 키를 안전하게 저장합니다.
10. Pipeline과 함께 사용할 리포지토리에 생성된 앱을 코드로 설치합니다.
11. 다음 명령을 입력하여 새로 생성된 GitHub 앱에 액세스하도록 Pipeline을 코드로 구성합니다.

```
$ oc -n openshift-pipelines create secret generic pipelines-as-code-secret \ 1
    --from-literal github-private-key="$(cat <PATH_PRIVATE_KEY>)" \ 2
    --from-literal github-application-id="<APP_ID>" \ 3
    --from-literal webhook.secret="<WEBHOOK_SECRET>" 4
```

1. 추가 GitHub 앱을 지원하기 위해 코드 컨트롤러로 추가 Pipeline을 생성하고 추가 컨트롤러의 앱을 구성하는 경우 **pipelines-as-code-secret** 을 컨트롤러의 **secretName** 매개변수에 구성한 이름으로 교체합니다.
2. GitHub 앱을 구성하는 동안 다운로드한 개인 키의 경로입니다.
3. GitHub **앱의 앱 ID** 입니다.
4. GitHub 앱을 생성할 때 제공되는 Webhook 시크릿입니다.



참고

코드로서의 파이프라인은 GitHub Enterprise에서 설정된 헤더를 감지하고 GitHub Enterprise API 권한 부여 URL에 사용하여 GitHub Enterprise에서 자동으로 작동합니다.

추가 리소스

- 추가 GitHub 앱을 지원하기 위해 추가 Pipeline을 코드 컨트롤러로 구성

3.1.4. GitHub 토큰을 추가 리포지토리로 범위 지정

코드로서의 파이프라인은 GitHub 앱을 사용하여 GitHub 액세스 토큰을 생성합니다. 코드로서의 파이프라인은 이 토큰을 사용하여 리포지토리에서 파이프라인 페이로드를 검색하고 CI/CD 프로세스가 GitHub 리포지토리와 상호 작용할 수 있도록 합니다.

기본적으로 액세스 토큰은 Pipeline이 파이프라인 정의를 검색하는 리포지토리에만 범위가 지정됩니다. 경우에 따라 토큰이 추가 리포지토리에 액세스할 수 있도록 할 수 있습니다. 예를 들어 **.tekton/pr.yaml** 파일 및 소스 페이로드가 있는 CI 리포지토리가 있을 수 있지만 **pr.yaml** 에 정의된 빌드 프로세스는 별도의 개인 CD 리포지토리에서 작업을 가져옵니다.

다음 두 가지 방법으로 GitHub 토큰의 범위를 확장할 수 있습니다.

- *글로벌 구성*: GitHub 토큰을 다른 네임스페이스의 리포지토리 목록으로 확장할 수 있습니다. 이 구성을 설정하려면 관리 권한이 있어야 합니다.
- *리포지토리 수준 구성*: GitHub 토큰을 원래 리포지토리와 동일한 네임스페이스에 있는 리포지토리 목록으로 확장할 수 있습니다. 이 구성을 설정하는 데 관리자 권한이 필요하지 않습니다.

프로세스

1. **TektonConfig** CR(사용자 정의 리소스)에서 **pipelinesAsCode.settings** 사양의 **secret-github-app-token-scoped** 매개변수를 **false** 로 설정합니다. 이 설정을 사용하면 GitHub 토큰의 범위를 글로벌 및 리포지토리 수준 구성에 나열된 프라이빗 및 공용 리포지토리로 지정할 수 있습니다.
2. GitHub 토큰 범위를 지정하는 글로벌 구성을 설정하려면 **pipelinesAsCode.settings** 사양의 **TektonConfig** CR에서 다음 예와 같이 **secret-github-app-scope-extra-repos** 매개변수에 추가 리포지토리를 지정합니다.

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  platforms:
    openshift:
```

```
    pipelinesAsCode:
      enable: true
      settings:
        secret-github-app-token-scoped: false
        secret-github-app-scope-extra-repos: "owner2/project2, owner3/project3"
```

3. GitHub 토큰 범위를 지정하는 리포지토리 수준 구성을 설정하려면 다음 예와 같이 **Repository** CR의 **github_app_token_scope_repos** 매개변수에 추가 리포지토리를 지정합니다.

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: test
  namespace: test-repo
spec:
  url: "https://github.com/linda/project"
  settings:
    github_app_token_scope_repos:
    - "owner/project"
    - "owner1/project1"
```

이 예제에서 **Repository** 사용자 지정 리소스는 **test-repo** 네임스페이스의 **linda/project** 리포지토리와 연결됩니다. 생성된 GitHub 토큰의 범위는 **owner/project** 및 **owner1/project1** 리포지토리와 **linda/project** 리포지토리로 확장됩니다. 이러한 리포지토리는 **test-repo** 네임스페이스에 있어야 합니다.

참고

추가 리포지토리는 공용 또는 개인 리포지토리일 수 있지만 리포지토리 리소스가 연결된 리포지토리와 동일한 네임스페이스에 있어야 합니다.

네임스페이스에 리포지토리가 없는 경우 GitHub 토큰의 범위가 오류 메시지와 함께 실패합니다.

```
failed to scope GitHub token as repo owner1/project1 does not exist in namespace test-repo
```

결과

생성된 GitHub 토큰을 사용하면 글로벌 및 리포지토리 수준 구성에 구성한 추가 리포지토리와 Pipeline as Code 페이로드 파일이 있는 원본 리포지토리에 액세스할 수 있습니다.

글로벌 구성 및 리포지토리 수준 구성을 모두 제공하는 경우 다음 예와 같이 두 구성의 모든 리포지토리에 범위가 지정됩니다.

TektonConfig 사용자 정의 리소스

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  platforms:
    openshift:
      pipelinesAsCode:
        enable: true
        settings:
          secret-github-app-token-scoped: false
          secret-github-app-scope-extra-repos: "owner2/project2, owner3/project3"
```

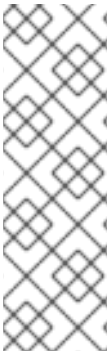
리포지토리 사용자 정의 리소스

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
 name: test
 namespace: test-repo
spec:
 url: "https://github.com/linda/project"
 settings:
   github_app_token_scope_repos:
   - "owner/project"
   - "owner1/project1"
```

GitHub 토큰의 범위는 owner/project,owner1/project1,owner2/project2,owner3/project3 및 linda/project respositories로 지정됩니다.

3.2. GITHUB WEBHOOK에서 PIPELINE을 코드로 사용

GitHub 앱을 생성할 수 없는 경우 리포지토리에서 GitHub Webhook와 함께 Pipeline을 코드로 사용합니다. 그러나 GitHub Webhook에서 Pipeline을 코드로 사용하면 GitHub Check Runs API에 액세스할 수 없습니다. 작업 상태는 가져오기 요청에 대한 주석으로 추가되며 Checks 탭에서는 사용할 수 없습니다.



참고

GitHub Webhook의 코드로서의 파이프라인은 /retest 및 /ok-to-test 와 같은 GitOps 주석을 지원하지 않습니다. 연속 통합(CI)을 다시 시작하려면 리포지토리에 대한 새 커밋을 생성합니다. 예를 들어 변경 없이 새 커밋을 생성하려면 다음 명령을 사용할 수 있습니다.

```
$ git --amend -a --no-edit && git push --force-with-lease <origin> <branchname>
```

사전 요구 사항

- Pipeline as Code가 클러스터에 설치되어 있는지 확인합니다.
- 권한 부여를 위해 GitHub에서 개인 액세스 토큰을 생성합니다.
  - 안전하고 세분화된 토큰을 생성하려면 해당 범위를 특정 리포지토리로 제한하고 다음 권한을 부여합니다.

표 3.1. 세분화된 토큰에 대한 권한

이름	액세스
관리	읽기 전용
메타데이터	읽기 전용
내용	읽기 전용
커밋 상태	읽기 및 쓰기
가져오기 요청	읽기 및 쓰기
Webhook	읽기 및 쓰기

  - 클래식 토큰을 사용하려면 범위를 공용 리포지토리 및 개인 리포지토리에 대해 public_repo 로 설정합니다. 또한 짧은 토큰 만료 기간을 제공하고 대체 위치에서 토큰을 기록해 둡니다.

참고

tkn pac CLI를 사용하여 **Webhook**를 구성하려면 **admin:repo_hook** 범위를 추가합니다.

프로세스

1. **Webhook**를 구성하고 **Repository CR**(사용자 정의 리소스)을 생성합니다.

- **tkn pac CLI** 툴을 사용하여 **Webhook**를 구성하고 리포지토리 **CR**을 *자동으로* 생성하려면 다음 명령을 사용합니다.

```
$ tkn pac create repo
```

대화형 출력 샘플

```
? Enter the Git repository url (default: https://github.com/owner/repo):
? Please enter the namespace where the pipeline should run (default: repo-
pipelines):
! Namespace repo-pipelines is not found
? Would you like me to create the namespace repo-pipelines? Yes
✓ Repository owner-repo has been created in repo-pipelines namespace
✓ Setting up GitHub Webhook for Repository https://github.com/owner/repo
  I have detected a controller url: https://pipelines-as-code-controller-openshift-
pipelines.apps.example.com
? Do you want me to use it? Yes
? Please enter the secret to configure the webhook for payload validation (default:
sJNwdmTifHTs):  sJNwdmTifHTs
i  You now need to create a GitHub personal access token, please checkout the
docs at https://docs.github.com/en/authentication/keeping-your-account-and-data-
secure/creating-a-personal-access-token for the required scopes
? Please enter the GitHub access token:  ****************************************
✓ Webhook has been created on repository owner/repo
  Webhook Secret owner-repo has been created in the repo-pipelines namespace.
  Repository CR owner-repo has been updated with webhook secret in the repo-
pipelines namespace
i Directory .tekton has been created.
✓ We have detected your repository using the programming language Go.
✓ A basic template has been created in
/home/Go/src/github.com/owner/repo/.tekton/pipelinerun.yaml, feel free to
customize it.
```

- Webhook를 구성하고 리포지토리 CR을 수동으로 생성하려면 다음 단계를 수행합니다.

i. OpenShift 클러스터에서 Pipeline의 공용 URL을 코드 컨트롤러로 추출합니다.

```
$ echo https://$(oc get route -n openshift-pipelines pipelines-as-code-
controller -o jsonpath='{.spec.host}')
```

ii. GitHub 리포지토리 또는 조직에서 다음 단계를 수행합니다.

A. 설정 -> Webhook 로 이동하여 Webhook 추가 를 클릭합니다.

B. Payload URL 을 코드 컨트롤러 공용 URL로 Pipeline으로 설정합니다.

C. 콘텐츠 유형을 application/json 으로 선택합니다.

D. 웹 후크 시크릿을 추가하고 대체 위치에서 기록해 둡니다. 로컬 시스템에 openssl 을 설치하면 임의의 보안을 생성합니다.

```
$ openssl rand -hex 20
```

E. 개별 이벤트 선택을 클릭하고 Commit comments,Issue comments,Pull request, Pushes 이벤트를 선택합니다.

F. Webhook 추가를 클릭합니다.

iii. OpenShift 클러스터에서 개인 액세스 토큰 및 Webhook 시크릿을 사용하여 Secret 오브젝트를 생성합니다.

```
$ oc -n target-namespace create secret generic github-webhook-config \
  --from-literal provider.token="<GITHUB_PERSONAL_ACCESS_TOKEN>" \
  --from-literal webhook.secret="<WEBHOOK_SECRET>"
```

iv. 리포지토리 CR을 생성합니다.

예: 리포지토리 CR

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
  namespace: target-namespace
spec:
  url: "https://github.com/owner/repo"
  git_provider:
    secret:
      name: "github-webhook-config"
      key: "provider.token" # Set this if you have a different key in your secret
    webhook_secret:
      name: "github-webhook-config"
      key: "webhook.secret" # Set this if you have a different key for your secret
```

참고

코드로서의 파이프라인은 **OpenShift Secret** 오브젝트 및 **Repository CR**이 동일한 네임스페이스에 있다고 가정합니다.

2. 선택 사항: 기존 리포지토리 **CR**의 경우 여러 **GitHub Webhook** 보안을 추가하거나 삭제된 보안을 대신 제공합니다.

   a. **tkn pac CLI** 툴을 사용하여 **Webhook**를 추가합니다.

예: tkn pac CLI를 사용한 추가 **Webhook**

```
$ tkn pac webhook add -n repo-pipelines
```

대화형 출력 샘플

-

```
✓ Setting up GitHub Webhook for Repository https://github.com/owner/repo
  I have detected a controller url: https://pipelines-as-code-controller-openshift-pipelines.apps.example.com
? Do you want me to use it? Yes
? Please enter the secret to configure the webhook for payload validation (default: AeHdHTJVfAeH):  AeHdHTJVfAeH
✓ Webhook has been created on repository owner/repo
  Secret owner-repo has been updated with webhook secert in the repo-pipelines namespace.
```

b. 기존 **OpenShift Secret** 오브젝트에서 **webhook.secret** 키를 업데이트합니다.

3. **선택 사항: 기존 리포지토리 CR**의 경우 개인 액세스 토큰을 업데이트합니다.

- **tkn pac CLI** 툴을 사용하여 개인 액세스 토큰을 업데이트합니다.

예: tkn pac CLI를 사용하여 개인 액세스 토큰 업데이트

```
$ tkn pac webhook update-token -n repo-pipelines
```

대화형 출력 샘플

```
? Please enter your personal access token:  ****************************************
  Secret owner-repo has been updated with new personal access token in the repo-pipelines namespace.
```

- 또는 **Repository CR**을 수정하여 개인 액세스 토큰을 업데이트합니다.

i. Repository CR에서 시크릿 이름을 찾습니다.

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
  namespace: target-namespace
spec:
# ...
  git_provider:
    secret:
      name: "github-webhook-config"
# ...
```

ii. oc patch 명령을 사용하여 $target_namespace 네임스페이스에서 $NEW_TOKEN 값을 업데이트합니다.

```
$ oc -n $target_namespace patch secret github-webhook-config -p "{\"data\":
{\"provider.token\": \"$(echo -n $NEW_TOKEN|base64 -w0)\"}}"
```

추가 리소스

- GitHub의 GitHub Webhook 문서
- GitHub Check Runs documentation on GitHub
- GitHub에서 개인 액세스 토큰 생성
- 사전 입력된 권한이 있는 클래식 토큰

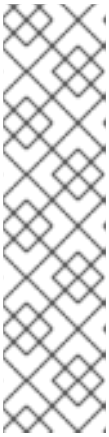
3.3. GITLAB에서 PIPELINE을 코드로 사용

조직 또는 프로젝트에서 GitLab을 기본 플랫폼으로 사용하는 경우 GitLab에서 Webhook가 있는 리포지토리의 코드로 Pipeline을 사용할 수 있습니다.

사전 요구 사항

- Pipeline as Code가 클러스터에 설치되어 있는지 확인합니다.

- 승인을 위해 **GitLab**에서 프로젝트 또는 조직 관리자로 개인 액세스 토큰을 생성합니다.



참고

- **tkn pac CLI**를 사용하여 **Webhook**를 구성하려면 **admin:repo_hook** 범위를 토큰에 추가합니다.
- 특정 프로젝트에 토큰 범위를 사용하면 분기된 리포지토리에서 전송된 병합 요청**(MR)**에 대한 **API** 액세스를 제공할 수 없습니다. 이러한 경우 **Code**로 **Pipeline**은 파이프라인의 결과를 **MR**에 대한 주석으로 표시합니다.

프로세스

1. **Webhook**를 구성하고 **Repository CR**(사용자 정의 리소스)을 생성합니다.

- **tkn pac CLI** 툴을 사용하여 **Webhook**를 구성하고 리포지토리 **CR**을 *자동으로* 생성하려면 다음 명령을 사용합니다.

```
$ tkn pac create repo
```

대화형 출력 샘플

```
? Enter the Git repository url (default: https://gitlab.com/owner/repo):
? Please enter the namespace where the pipeline should run (default: repo-
pipelines):
! Namespace repo-pipelines is not found
? Would you like me to create the namespace repo-pipelines? Yes
✓ Repository repositories-project has been created in repo-pipelines namespace
✓ Setting up GitLab Webhook for Repository https://gitlab.com/owner/repo
? Please enter the project ID for the repository you want to be configured,
  project ID refers to an unique ID (e.g. 34405323) shown at the top of your GitLab
project : 17103
  I have detected a controller url: https://pipelines-as-code-controller-openshift-
pipelines.apps.example.com
? Do you want me to use it? Yes
? Please enter the secret to configure the webhook for payload validation (default:
lFjHlEcaGFlF):  lFjHlEcaGFlF
i  You now need to create a GitLab personal access token with `api` scope
i  Go to this URL to generate one https://gitlab.com/-
/profile/personal_access_tokens, see https://is.gd/rOEo9B for documentation
? Please enter the GitLab access token:  ****************************
? Please enter your GitLab API URL::  https://gitlab.com
```

```
✓ Webhook has been created on your repository
  Webhook Secret repositories-project has been created in the repo-pipelines
namespace.
  Repository CR repositories-project has been updated with webhook secret in the
repo-pipelines namespace
i Directory .tekton has been created.
✓ A basic template has been created in
/home/Go/src/gitlab.com/repositories/project/.tekton/pipelinerun.yaml, feel free to
customize it.
```

- Webhook를 구성하고 리포지토리 CR을 수동으로 생성하려면 다음 단계를 수행합니다.

  i. OpenShift 클러스터에서 Pipeline의 공용 URL을 코드 컨트롤러로 추출합니다.

  ```
  $ echo https://$(oc get route -n openshift-pipelines pipelines-as-code-
  controller -o jsonpath='{.spec.host}')
  ```

  ii. GitLab 프로젝트에서 다음 단계를 수행합니다.

  A. 왼쪽 사이드바를 사용하여 설정 -> Webhook 로 이동합니다.

  B. URL 을 코드 컨트롤러 공용 URL로 Pipeline으로 설정합니다.

  C. 웹 후크 시크릿을 추가하고 대체 위치에서 기록해 둡니다. 로컬 시스템에 openssl 을 설치하면 임의의 보안을 생성합니다.

  ```
  $ openssl rand -hex 20
  ```

  D. 개별 이벤트 선택을 클릭하고 Commit comments,Issue comments,Pull request, Pushes 이벤트를 선택합니다.

  E. 변경 사항 저장을 클릭합니다.

  iii. OpenShift 클러스터에서 개인 액세스 토큰 및 Webhook 시크릿을 사용하여 Secret 오브젝트를 생성합니다.

```
$ oc -n target-namespace create secret generic gitlab-webhook-config \
  --from-literal provider.token="<GITLAB_PERSONAL_ACCESS_TOKEN>" \
  --from-literal webhook.secret="<WEBHOOK_SECRET>"
```

iv. 리포지토리 CR을 생성합니다.

예: 리포지토리 CR

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
  namespace: target-namespace
spec:
  url: "https://gitlab.com/owner/repo" # The repository URL
  git_provider:
    #url: "https://gitlab.example.com/" 1
    secret:
      name: "gitlab-webhook-config"
      key: "provider.token" # Set this if you have a different key in your secret
    webhook_secret:
      name: "gitlab-webhook-config"
      key: "webhook.secret" # Set this if you have a different key for your secret
```

1 GitLab.com이 아닌 GitLab의 개인 인스턴스를 사용하는 경우 이 필드의 주석을 제거하고 GitLab API의 URL로 설정합니다. GitLab API는 리포지토리와 동일한 호스트입니다. 예를 들어 리포지터리가 https://gitlab.example.com/owner/repo 이면 API URL은 https://gitlab.example.com/ 입니다.

참고

- 코드로서의 파이프라인은 OpenShift Secret 오브젝트 및 Repository CR이 동일한 네임스페이스에 있다고 가정합니다.

2. 선택 사항: 기존 리포지토리 CR의 경우 GitLab Webhook 보안을 여러 개 추가하거나 삭제된 보안을 대신 제공합니다.

a. **tkn pac CLI** 툴을 사용하여 **Webhook**를 추가합니다.

예: tkn pac CLI를 사용하여 추가 **Webhook** 추가

```
$ tkn pac webhook add -n repo-pipelines
```

대화형 출력 샘플

```
✓ Setting up GitLab Webhook for Repository https://gitlab.com/owner/repo
  I have detected a controller url: https://pipelines-as-code-controller-openshift-pipelines.apps.example.com
? Do you want me to use it? Yes
? Please enter the secret to configure the webhook for payload validation (default: AeHdHTJVfAeH):  AeHdHTJVfAeH
✓ Webhook has been created on repository owner/repo
  Secret owner-repo has been updated with webhook secert in the repo-pipelines namespace.
```

b. 기존 **OpenShift Secret** 오브젝트에서 **webhook.secret** 키를 업데이트합니다.

3. **선택 사항: 기존 리포지토리 CR**의 경우 개인 액세스 토큰을 업데이트합니다.

- **tkn pac CLI** 툴을 사용하여 개인 액세스 토큰을 업데이트합니다.

예: tkn pac CLI를 사용하여 개인 액세스 토큰 업데이트

```
$ tkn pac webhook update-token -n repo-pipelines
```

대화형 출력 샘플

```
? Please enter your personal access token: ****************************************
  Secret owner-repo has been updated with new personal access token in the
repo-pipelines namespace.
```

- 또는 Repository CR을 수정하여 개인 액세스 토큰을 업데이트합니다.

  i. Repository CR에서 시크릿 이름을 찾습니다.

```
...
spec:
  git_provider:
    secret:
      name: "gitlab-webhook-config"
...
```

  ii. oc patch 명령을 사용하여 $target_namespace 네임스페이스에서 $NEW_TOKEN 값을 업데이트합니다.

```
$ oc -n $target_namespace patch secret gitlab-webhook-config -p "{\"data\":
{\"provider.token\": \"$(echo -n $NEW_TOKEN|base64 -w0)\"}}"
```

추가 리소스

- GitLab의 GitLab Webhook 문서

3.4. BITBUCKET CLOUD에서 PIPELINE을 코드로 사용

조직 또는 프로젝트에서 Bitbucket Cloud를 기본 플랫폼으로 사용하는 경우 Bitbucket Cloud에서 Webhook가 있는 리포지토리의 코드로 Pipeline을 사용할 수 있습니다.

사전 요구 사항

- **Pipeline as Code**가 클러스터에 설치되어 있는지 확인합니다.

- **Bitbucket Cloud**에서 앱 암호를 생성합니다.

  - 다음 확인란을 선택하여 토큰에 적절한 권한을 추가합니다.

    - 계정: 이메일,읽기
    - 작업 공간 멤버십: 읽기,쓰기
    - 프로젝트: 읽기,쓰기
    - 문제: 읽기,쓰기
    - 가져오기 요청: 읽기,쓰기

참고

- **tkn pac CLI**를 사용하여 **Webhook**를 구성하려면 **Webhook:**읽기 및 쓰기 권한을 토큰에 추가합니다.
- 생성된 암호 또는 토큰 사본을 대체 위치에 저장합니다.

프로세스

1. **Webhook**를 구성하고 리포지토리 **CR**을 생성합니다.

   - **tkn pac CLI** 툴을 사용하여 **Webhook**를 구성하고 리포지토리 **CR**을 *자동으로* 생성하려면 다음 명령을 사용합니다.

```
$ tkn pac create repo
```

대화형 출력 샘플

```
? Enter the Git repository url (default: https://bitbucket.org/workspace/repo):
? Please enter the namespace where the pipeline should run (default: repo-pipelines):
! Namespace repo-pipelines is not found
? Would you like me to create the namespace repo-pipelines? Yes
✓ Repository workspace-repo has been created in repo-pipelines namespace
✓ Setting up Bitbucket Webhook for Repository https://bitbucket.org/workspace/repo
? Please enter your bitbucket cloud username:  <username>
i  You now need to create a Bitbucket Cloud app password, please checkout the docs at https://is.gd/fqMHiJ for the required permissions
? Please enter the Bitbucket Cloud app password:  ************************************
  I have detected a controller url: https://pipelines-as-code-controller-openshift-pipelines.apps.example.com
? Do you want me to use it? Yes
✓ Webhook has been created on repository workspace/repo
  Webhook Secret workspace-repo has been created in the repo-pipelines namespace.
  Repository CR workspace-repo has been updated with webhook secret in the repo-pipelines namespace
i Directory .tekton has been created.
✓ A basic template has been created in /home/Go/src/bitbucket/repo/.tekton/pipelinerun.yaml, feel free to customize it.
```

- Webhook를 구성하고 리포지토리 CR을 수동으로 생성하려면 다음 단계를 수행합니다.

  i. OpenShift 클러스터에서 Pipeline의 공용 URL을 코드 컨트롤러로 추출합니다.

  ```
  $ echo https://$(oc get route -n openshift-pipelines pipelines-as-code-controller -o jsonpath='{.spec.host}')
  ```

  ii. Bitbucket Cloud에서 다음 단계를 수행합니다.

  A. Bitbucket Cloud 리포지토리의 왼쪽 탐색 창을 사용하여 리포지토리 설정 -> Webhook로 이동하고 Webhook 추가 를 클릭합니다.

  B. 제목 을 설정합니다. 예를 들어 "Pipelines as Code"가 있습니다.

C.

URL 을 코드 컨트롤러 공용 URL로 Pipeline으로 설정합니다.

D.

다음 이벤트를 선택합니다: Repository: Push,Pull Request: Created,Pull Request: Updated, Pull Request: Comment created.

E.

저장을 클릭합니다.

iii.

OpenShift 클러스터에서 대상 네임스페이스에 app 암호를 사용하여 Secret 오브젝트를 생성합니다.

```
$ oc -n target-namespace create secret generic bitbucket-cloud-token \
  --from-literal provider.token="<BITBUCKET_APP_PASSWORD>"
```

iv.

리포지토리 CR을 생성합니다.

예: 리포지토리 CR

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
  namespace: target-namespace
spec:
  url: "https://bitbucket.com/workspace/repo"
  branch: "main"
  git_provider:
    user: "<BITBUCKET_USERNAME>" (1)
    secret:
      name: "bitbucket-cloud-token" (2)
      key: "provider.token" # Set this if you have a different key in your secret
```

(1)

소유자 파일의 ACCOUNT_ID 만 사용자를 참조할 수 있습니다.

(2)

참고

- Bitbucket Cloud에서는 **tkn pac create** 및 **tkn pac bootstrap** 명령이 지원되지 않습니다.

- Bitbucket Cloud는 Webhook 보안을 지원하지 않습니다. 페이로드를 보호하고 CI의 하이재킹을 방지하기 위해 코드로서의 파이프라인은 Bitbucket Cloud IP 주소 목록을 가져와서 Webhook 수신이 해당 IP 주소에서만 제공되도록 합니다.

  - 기본 동작을 비활성화하려면 **pipelinesAsCode.settings** 사양에서 **TektonConfig** 사용자 정의 리소스에서 **bitbucket-cloud-check-source-ip** 매개변수를 **false** 로 설정합니다.

  - 추가 안전 IP 주소 또는 네트워크를 허용하려면 **pipelinesAsCode.settings** 사양에서 **TektonConfig** 사용자 정의 리소스의 **bitbucket-cloud-additional-source-ip** 매개변수에 쉼표로 구분된 값으로 추가합니다.

2. 선택 사항: 기존 리포지토리 CR의 경우 여러 Bitbucket Cloud Webhook 시크릿을 추가하거나 삭제된 시크릿을 대체할 수 있습니다.

   a. tkn pac CLI 툴을 사용하여 Webhook를 추가합니다.

   예: tkn pac CLI를 사용하여 추가 Webhook 추가

```
$ tkn pac webhook add -n repo-pipelines
```

   대화형 출력 샘플

```
✓ Setting up Bitbucket Webhook for Repository
```

```
https://bitbucket.org/workspace/repo
? Please enter your bitbucket cloud username:  <username>
  I have detected a controller url: https://pipelines-as-code-controller-openshift-pipelines.apps.example.com
? Do you want me to use it? Yes
✓ Webhook has been created on repository workspace/repo
  Secret workspace-repo has been updated with webhook secret in the repo-pipelines namespace.
```

참고

tkn pac webhook add 명령과 함께 [-n <namespace>] 옵션을 기본 네임스페이스 이외의 네임스페이스에 리포지토리 CR이 있는 경우에만 사용합니다.

b. 기존 OpenShift Secret 오브젝트에서 webhook.secret 키를 업데이트합니다.

3. 선택 사항: 기존 리포지토리 CR의 경우 개인 액세스 토큰을 업데이트합니다.

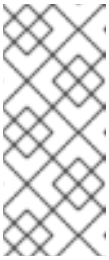
- tkn pac CLI 툴을 사용하여 개인 액세스 토큰을 업데이트합니다.

예: tkn pac CLI를 사용하여 개인 액세스 토큰 업데이트

```
$ tkn pac webhook update-token -n repo-pipelines
```

대화형 출력 샘플

```
? Please enter your personal access token: ****************************************
  Secret owner-repo has been updated with new personal access token in the repo-pipelines namespace.
```



참고

기본 네임스페이스 이외의 네임스페이스에 리포지토리 CR이 있는 경우에만 tkn pac webhook update-token 명령에 [-n <namespace>] 옵션을 사용합니다.

- 또는 Repository CR을 수정하여 개인 액세스 토큰을 업데이트합니다.

  i. Repository CR에서 시크릿 이름을 찾습니다.

```
...
spec:
  git_provider:
    user: "<BITBUCKET_USERNAME>"
    secret:
      name: "bitbucket-cloud-token"
      key: "provider.token"
...
```

  ii. oc patch 명령을 사용하여 $target_namespace 네임스페이스에서 $password 값을 업데이트합니다.

```
$ oc -n $target_namespace patch secret bitbucket-cloud-token -p "{\"data\":
{\"provider.token\": \"$(echo -n $NEW_TOKEN|base64 -w0)\"}}"
```

추가 리소스

- Bitbucket Cloud에서 앱 암호 생성
- Altassian 계정 ID 및 닉네임 소개

3.5. BITBUCKET SERVER에서 PIPELINE을 코드로 사용

조직 또는 프로젝트에서 Bitbucket 서버를 기본 플랫폼으로 사용하는 경우 Bitbucket Server에서 Webhook가 있는 리포지토리의 코드로 Pipeline을 사용할 수 있습니다.

사전 요구 사항

- Pipeline as Code가 클러스터에 설치되어 있는지 확인합니다.
- Bitbucket 서버에서 프로젝트의 관리자로 개인 액세스 토큰을 생성하고 다른 위치에 저장합니다.

참고

- 토큰에는 PROJECT_ADMIN 및 REPOSITORY_ADMIN 권한이 있어야 합니다.
- 토큰에는 가져오기 요청에서 분기된 리포지토리에 액세스할 수 있어야 합니다.

프로세스

1. OpenShift 클러스터에서 Pipeline의 공용 URL을 코드 컨트롤러로 추출합니다.

```
$ echo https://$(oc get route -n openshift-pipelines pipelines-as-code-controller -o
jsonpath='{.spec.host}')
```

2. Bitbucket 서버에서 다음 단계를 수행합니다.

a. Bitbucket Data Center 리포지토리의 왼쪽 탐색 창을 사용하여 리포지토리 설정 -> Webhook로 이동하고 Webhook 추가 를 클릭합니다.

b. 제목 을 설정합니다. 예를 들어 "Pipelines as Code"가 있습니다.

c. URL 을 코드 컨트롤러 공용 URL로 Pipeline으로 설정합니다.

d. 웹 후크 시크릿을 추가하고 복사본을 대체 위치에 저장합니다. 로컬 머신에 openssl 을 설치한 경우 다음 명령을 사용하여 임의의 시크릿을 생성합니다.

```
$ openssl rand -hex 20
```

e. 다음 이벤트를 선택합니다.

- 리포지토리: 푸시
- 리포지토리: Cryostat
- 가져오기 요청: 개설됨
- 가져오기 요청: 소스 분기 업데이트
- 가져오기 요청: 주석 추가

f. 저장을 클릭합니다.

3. OpenShift 클러스터에서 대상 네임스페이스에 app 암호를 사용하여 Secret 오브젝트를 생성합니다.

```
$ oc -n target-namespace create secret generic bitbucket-server-webhook-config \
  --from-literal provider.token="<PERSONAL_TOKEN>" \
  --from-literal webhook.secret="<WEBHOOK_SECRET>"
```

4. 리포지토리 CR을 생성합니다.

예: 리포지토리 CR

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
  namespace: target-namespace
spec:
  url: "https://bitbucket.com/workspace/repo"
  git_provider:
    url: "https://bitbucket.server.api.url/rest" (1)
    user: "<BITBUCKET_USERNAME>" (2)
```

```
  secret: 3
    name: "bitbucket-server-webhook-config"
    key: "provider.token" # Set this if you have a different key in your secret
  webhook_secret:
    name: "bitbucket-server-webhook-config"
    key: "webhook.secret" # Set this if you have a different key for your secret
```

1

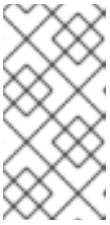
/api/v1.0 접미사가 없는 올바른 Bitbucket 서버 API URL이 있는지 확인합니다. 일반적으로 기본 설치에는 /rest 접미사가 있습니다.

2

소유자 파일의 ACCOUNT_ID 만 사용자를 참조할 수 있습니다.

3

Code와 같은 파이프라인에서는 git_provider.secret 사양에서 참조하는 시크릿과 Repository CR이 동일한 네임스페이스에 있다고 가정합니다.



참고

Bitbucket 서버에서 tkn pac create 및 tkn pac bootstrap 명령은 지원되지 않습니다.

추가 리소스

- Bitbucket 서버에서 개인 토큰 생성
- Bitbucket 서버에서 Webhook 생성

3.6. 사용자 정의 인증서와 함께 PIPELINE을 코드로 연결

개인 서명 또는 사용자 정의 인증서로 액세스할 수 있는 Git 리포지토리를 사용하여 Pipeline을 코드로 구성하려면 인증서를 코드로 노출할 수 있습니다.

프로세스

- Red Hat OpenShift Pipelines Operator를 사용하여 Pipeline을 코드로 설치한 경우 Proxy 오브젝트를 사용하여 사용자 정의 인증서를 클러스터에 추가할 수 있습니다. Operator는 Pipeline을 코드로 포함한 모든 Red Hat OpenShift Pipelines 구성 요소 및 워크로드에 인증서를 노출합니다.

추가 리소스

- 클러스터 전체 프록시 사용

3.7. PIPELINE이 포함된 개인 리포지토리를 코드로 사용

코드로서의 파이프라인은 대상 네임스페이스에서 사용자 토큰을 사용하여 시크릿을 생성하거나 업데이트하여 개인 리포지토리를 지원합니다. Tekton Hub의 git-clone 작업에서는 사용자 토큰을 사용하여 개인 리포지토리를 복제합니다.

코드가 대상 네임스페이스에서 새 파이프라인을 실행할 때마다 pac-gitauth-<REPOSITORY_OWNER>-<REPOSITORY_NAME>-<RANDOM_STRING > 형식으로 보안을 생성하거나 업데이트합니다.

파이프라인 실행 및 파이프라인 정의에서 basic-auth 작업 영역을 사용하여 시크릿을 참조해야 합니다. 그런 다음 git-clone 작업으로 전달됩니다.

```
...
  workspace:
  - name: basic-auth
    secret:
      secretName: "{{ git_auth_secret }}"
...
```

파이프라인에서 git-clone 작업을 재사용할 수 있는 basic-auth 작업 영역을 참조할 수 있습니다.

```
...
workspaces:
  - name basic-auth
params:
    - name: repo_url
    - name: revision
...
tasks:
  workspaces:
    - name: basic-auth
      workspace: basic-auth
  ...
```

```
  tasks:
  - name: git-clone-from-catalog
      taskRef:
        name: git-clone (1)
      params:
        - name: url
          value: $(params.repo_url)
        - name: revision
          value: $(params.revision)
...
```

(1)

git-clone 작업에서는 **basic-auth** 작업 영역을 선택하고 이를 사용하여 개인 리포지토리를 복제합니다.

pipelinesAsCode.settings 사양의 **TektonConfig** 사용자 지정 리소스에서 필요에 따라 **secret-auto-create** 매개변수를 **false** 또는 **true** 값으로 설정하여 이 구성을 수정할 수 있습니다.

4장. REPOSITORY 사용자 정의 리소스 사용

Repository CR(사용자 정의 리소스)에는 다음과 같은 기본 기능이 있습니다.

- 파이프라인에 URL에서 이벤트를 처리하는 방법에 대한 코드로 알립니다.
- 파이프라인을 파이프라인 실행의 네임스페이스에 대한 코드로 알립니다.
- Webhook 방법을 사용할 때 Git 공급자 플랫폼에 필요한 API 시크릿, 사용자 이름 또는 API URL을 참조합니다.
- 리포지토리의 마지막 파이프라인 실행 상태를 제공합니다.

4.1. 리포지토리 사용자 정의 리소스 생성

tkn pac CLI 또는 기타 대체 방법을 사용하여 대상 네임스페이스에 리포지토리 CR (사용자 정의 리소스)을 생성할 수 있습니다. 예를 들면 다음과 같습니다.

```
cat <<EOF|kubectl create -n my-pipeline-ci -f- 1

apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: project-repository
spec:
  url: "https://github.com/<repository>/<project>"
EOF
```

1 my-pipeline-ci 는 대상 네임스페이스입니다.

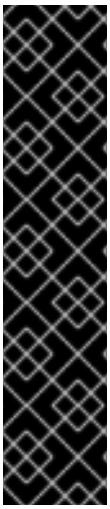
https://github.com/<repository>/<project >와 같은 URL에서 이벤트가 발생할 때마다 Code와 일치하는 Pipeline이 일치하고 .tekton / 디렉터리의 콘텐츠와 일치하도록 파이프라인 실행을 위한 <repository>/<project > 리포지토리의 콘텐츠를 확인하기 시작합니다.

참고

- 소스 코드 리포지토리와 연결된 파이프라인이 실행될 동일한 네임스페이스에 **Repository CR**을 생성해야 합니다. 다른 네임스페이스를 대상으로 지정할 수 없습니다.
- 여러 **Repository CR**이 동일한 이벤트와 일치하는 경우 **Code**와 **Pipelines**는 가장 오래된 이벤트만 처리합니다. 특정 네임스페이스와 일치해야 하는 경우 **pipelinesascode.tekton.dev/target-namespace: "<mynamespace>"** 주석을 추가합니다. 이러한 명시적 대상에서는 악의적인 행위자가 액세스 권한이 없는 네임스페이스에서 파이프라인 실행을 실행하지 못하도록 합니다.

4.2. 글로벌 리포지토리 사용자 정의 리소스 생성

선택적으로 **OpenShift Pipelines**가 설치된 네임스페이스에 글로벌 리포지토리 **CR**(사용자 정의 리소스)을 생성할 수 있습니다(일반적으로 **openshift-pipelines**). 이 **CR**을 생성하는 경우 해당 **CR**에 지정하는 설정이 기본적으로 사용자가 생성한 모든 **Repository CR**에 적용됩니다.



중요

글로벌 리포지토리 **CR**은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(**SLA**)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

사전 요구 사항

- **openshift-pipelines** 네임스페이스에 대한 관리자 액세스 권한이 있습니다.
- **oc** 명령줄 유틸리티를 사용하여 **OpenShift** 클러스터에 로그온했습니다.

프로세스

- **openshift-pipelines** 네임스페이스에 **pipeline-as-code** 라는 리포지토리 **CR**을 생성합니다. 이 **CR**에서 필요한 모든 기본 설정을 지정합니다.

CR을 생성하는 명령 예

```
$ cat <<EOF|oc create -n openshift-pipelines -f -

apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: pipelines-as-code
spec:
  git_provider:
    secret:
      name: "gitlab-webhook-config"
      key: "provider.token"
    webhook_secret:
      name: "gitlab-webhook-config"
      key: "webhook.secret"
EOF
```

이 예에서 생성하는 모든 Repository CR에는 GitLab 리포지토리에 액세스하기 위한 공통 시크릿이 포함됩니다. CR에서 다른 리포지토리 URL 및 기타 설정을 설정할 수 있습니다.

4.3. 동시성 제한 설정

Repository CRD(사용자 정의 리소스 정의)에서 concurrency_limit 사양을 사용하여 리포지토리에 대해 동시에 실행되는 최대 파이프라인 실행 수를 정의할 수 있습니다.

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
  namespace: target-namespace
spec:
# ...
  concurrency_limit: <number>
# ...
```

이벤트와 일치하는 파이프라인이 여러 개 있는 경우 파이프라인은 이벤트 시작과 알파벳순으로 실행됩니다.

예를 들어 .tekton 디렉터리에 3개의 파이프라인 실행이 있고 리포지토리 구성에서 concurrency_limit를 사용하여 가져오기 요청을 생성하는 경우 모든 파이프라인 실행이 알파벳순으로 실행됩니다. 언제든

지 하나의 파이프라인 실행만 실행 중 상태인 동안 나머지 실행은 대기열에 있습니다.

4.4. 파이프라인 정의의 소스 분기 변경

기본적으로 푸시 이벤트 또는 가져오기 요청 이벤트를 처리할 때 **Pipeline**은 이벤트를 트리거한 분기에서 파이프라인 정의를 가져옵니다. **Repository CRD**(사용자 정의 리소스 정의)에서 **pipelinerun_provenance** 설정을 사용하여 기본 **,master** 또는 **trunk** 와 같은 **Git** 리포지토리 공급자에 구성된 기본 분기에서 정의를 가져올 수 있습니다.

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: my-repo
  namespace: target-namespace
spec:
# ...
  settings:
    pipelinerun_provenance: "default_branch"
# ...
```

참고

이 설정을 보안 예방 조치로 사용할 수 있습니다. 기본 동작으로 코드로서의 파이프라인은 제출된 가져오기 요청에서 파이프라인 정의를 사용합니다. **default-branch** 설정을 사용하면 파이프라인 정의를 실행하기 전에 기본 분기에 병합해야 합니다. 이 요구 사항은 병합 검토 중에 변경 사항을 가능한 최대로 확인합니다.

4.5. 사용자 정의 매개변수 확장

Pipeline을 코드로 사용하여 **params** 필드를 사용하여 **PipelineRun** 리소스 내에서 사용자 정의 매개변수를 확장할 수 있습니다. **Repository CR**(사용자 정의 리소스) 템플릿 내에 사용자 정의 매개변수 값을 지정할 수 있습니다. 지정된 값은 파이프라인 실행의 사용자 정의 매개변수를 대체합니다.

다음 시나리오에서는 사용자 지정 매개변수를 사용할 수 있습니다.

- 푸시 또는 가져오기 요청에 따라 다른 레지스트리 **URL**과 같은 **URL** 매개변수를 정의하려면 다음을 수행합니다.

- 관리자가 **Git** 리포지토리에서 **PipelineRun** 실행을 변경하지 않고도 관리할 수 있는 계정 **UUID**와 같은 매개변수를 정의하려면 다음을 수행합니다.

참고

Tekton 매개변수가 Pipeline 리소스에 정의되어 Git 리포지토리 내에서 함께 사용자 정의되므로 Tekton PipelineRun 매개변수를 사용할 수 없는 경우에만 사용자 정의 매개변수 확장 기능을 사용합니다. 그러나 사용자 지정 매개변수는 Repository CR이 있는 경우 정의 및 사용자 지정됩니다. 따라서 CI/CD 파이프라인을 단일 지점에서 관리할 수 없습니다.

다음 예제에서는 Repository CR에서 company 라는 사용자 정의 매개변수를 보여줍니다.

```
...
spec:
  params:
    - name: company
      value: "ABC Company"
...
```

value Cryo stat Company 는 파이프라인 실행 및 원격으로 가져온 작업에서 매개 변수 이름 회사로 대체됩니다.

다음 예와 같이 Kubernetes 시크릿에서 사용자 정의 매개변수 값을 검색할 수도 있습니다.

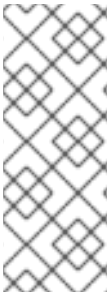
```
...
spec:
  params:
    - name: company
      secretRef:
        name: my-secret
        key: companyname
...
```

코드화로 파이프라인은 다음과 같은 방식으로 사용자 지정 매개변수를 구문 분석하고 사용합니다.

- 값 과 secretRef 가 정의된 경우 코드로 Pipeline은 값을 사용합니다.
- params 섹션에 이름이 없는 경우 Code로 Pipeline이 매개변수를 구문 분석하지 않습니다.
- 이름이 붙은 매개변수가 여러 개 있는 경우 Code와 Pipelines는 마지막 매개변수를 사용합니다.

사용자 지정 매개변수를 정의하고 지정된 조건이 **CEL** 필터에 일치하는 경우에만 확장을 사용할 수 있습니다. 다음 예제에서는 **pull request** 이벤트가 트리거될 때 **company** 라는 사용자 지정 매개변수에 적용 가능한 **CEL** 필터를 보여줍니다.

```
...
spec:
  params:
    - name: company
      value: "ABC Company"
      filter:
        - name: event
          value: |
      pac.event_type == "pull_request"
...
```



참고

이름과 다른 필터가 여러 개 있는 경우 **Code**와 **Pipeline**은 필터와 일치하는 첫 번째 매개변수를 사용합니다. 따라서 **Pipeline as Code**를 사용하면 다양한 이벤트 유형에 따라 매개변수를 확장할 수 있습니다. 예를 들어 **push** 및 **pull request** 이벤트를 결합할 수 있습니다.

5장. PIPELINE을 코드 확인기로 사용

Pipeline as Code resolver를 사용하면 실행 중인 파이프라인 실행이 다른 파이프라인 실행과 충돌하지 않습니다.

5.1. PIPELINE AS CODE RESOLVER 정보

파이프라인 및 파이프라인 실행을 분할하려면 파일을 .tekton/ 디렉터리 또는 해당 하위 디렉터리에 저장합니다.

코드상 Pipeline이 .tekton/ 디렉터리에 있는 모든 YAML 파일의 작업 또는 파이프라인에 대한 참조로 파이프라인 실행을 관찰하는 경우, 코드로서 Pipeline은 참조된 작업을 자동으로 해결하여 PipelineRun 오브젝트에 포함된 사양으로 단일 파이프라인 실행을 제공합니다.

Code로 Pipeline을 통해 Pipeline 또는 PipelineSpec 정의에서 참조된 작업을 확인할 수 없는 경우 클러스터에 변경 사항을 적용하기 전에 실행이 실패합니다. Git 공급자 플랫폼과 Repository CR이 있는 대상 네임스페이스의 이벤트 내부에서 문제를 확인할 수 있습니다.

해결자는 다음 유형의 작업을 관찰하는 경우 확인을 건너뜁니다.

- 클러스터 작업에 대한 참조입니다.
- 작업 또는 파이프라인 번들입니다.
- tekton.dev/ 접두사가 없는 API 버전이 있는 사용자 정의 작업입니다.

해결자는 변환 없이 이러한 작업을 문자 그대로 사용합니다.

가져오기 요청으로 보내기 전에 파이프라인 실행을 로컬에서 테스트하려면 tkn pac resolve 명령을 사용합니다.

원격 파이프라인 및 작업을 참조할 수도 있습니다.

5.2. PIPELINE의 원격 작업 주석 사용

코드로서의 파이프라인은 파이프라인 실행에서 주석을 사용하여 원격 작업 또는 파이프라인 가져오기를 지원합니다. 파이프라인 실행에서 원격 작업 또는 **PipelineRun** 또는 **PipelineSpec** 오브젝트의 파이프라인을 참조하는 경우 **Code resolver**로서의 **Pipeline**이 자동으로 포함됩니다. 원격 작업을 가져오거나 구문 분석하는 동안 오류가 있는 경우 코드로 파이프라인은 작업 처리를 중지합니다.

원격 작업을 포함하려면 다음 주석 예제를 참조하십시오.

Tekton Hub에서 원격 작업 참조

- **Tekton Hub**에서 단일 원격 작업을 참조합니다.

```
...
 pipelinesascode.tekton.dev/task: "git-clone" (1)
...
```

(1) **Code**로서의 파이프라인에는 **Tekton Hub**의 최신 작업 버전이 포함됩니다.

- **Tekton Hub**에서 여러 원격 작업 참조

```
...
 pipelinesascode.tekton.dev/task: "[git-clone, golang-test, tkn]"
...
```

- **-<NUMBER >** 접미사를 사용하여 **Tekton Hub**에서 여러 원격 작업을 참조합니다.

```
...
 pipelinesascode.tekton.dev/task: "git-clone"
 pipelinesascode.tekton.dev/task-1: "golang-test"
 pipelinesascode.tekton.dev/task-2: "tkn" (1)
...
```

(1) 기본적으로 **Code**로 **Pipeline**은 문자열을 **Tekton Hub**에서 가져올 최신 작업으로 해석합니다.

- Tekton Hub에서 특정 버전의 원격 작업을 참조합니다.

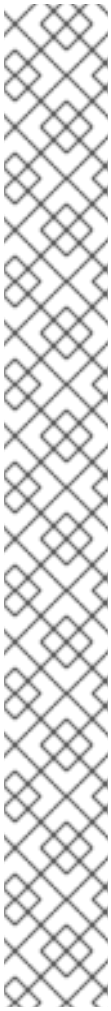
```
...
  pipelinesascode.tekton.dev/task: "[git-clone:0.1]" (1)
...
```

(1) Tekton Hub에서 git-clone 원격 작업의 0.1 버전을 나타냅니다.

URL을 사용하는 원격 작업

```
...
  pipelinesascode.tekton.dev/task: "<https://remote.url/task.yaml>" (1)
...
```

(1) 원격 작업의 공용 URL입니다.



참고

- GitHub를 사용하고 원격 작업 URL이 Repository CRD(사용자 정의 리소스 정의)와 동일한 호스트를 사용하는 경우 Code와 Pipeline은 GitHub 토큰을 사용하고 GitHub API를 사용하여 URL을 가져옵니다.

  예를 들어 https://github.com/<organization>/<repository>와 유사한 리포지토리 URL이 있고 원격 HTTP URL이 https://github.com/<organization>/<repository>/blob/<mainbranch>/<path>/<file>과 유사한 GitHub Blob 을 참조하는 경우 코드로서의 Pipeline은 GitHub 앱 토큰을 사용하여 해당 프라이빗 리포지토리에서 해당 개인 리포지터리에서 작업 정의 파일을 가져옵니다.

  공용 GitHub 리포지토리에서 작업하는 경우 Code로서의 Pipeline은 https://raw.githubusercontent.com/<organization>/<repository>/<main branch>/<path>/<file >과 같은 GitHub 원시 URL에서 유사하게 작동합니다.

- GitHub 앱 토큰의 범위는 리포지토리가 있는 소유자 또는 조직에 따라 지정됩니다. GitHub Webhook 방법을 사용하는 경우 개인 토큰이 허용되는 모든 조직에서 개인 또는 공용 리포지토리를 가져올 수 있습니다.

리포지토리 내부의 YAML 파일에서 작업 참조

```
...
pipelinesascode.tekton.dev/task: "<share/tasks/git-clone.yaml>" (1)
...
```

(1) 작업 정의가 포함된 로컬 파일의 상대 경로입니다.

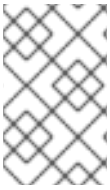
5.3. PIPELINE의 원격 파이프라인 주석 사용

원격 파이프라인 주석을 사용하여 여러 리포지토리에서 파이프라인 정의를 공유할 수 있습니다.

```
...
  pipelinesascode.tekton.dev/pipeline: "<https://git.provider/raw/pipeline.yaml>" 1
...
```

1

원격 파이프라인 정의에 대한 URL입니다. 동일한 리포지토리 내에 파일의 위치를 제공할 수도 있습니다.



참고

주석을 사용하여 하나의 파이프라인 정의만 참조할 수 있습니다.

5.3.1. 원격 파이프라인의 작업 덮어쓰기

기본적으로 파이프라인 실행에서 원격 파이프라인 주석을 사용하면 코드로서 Pipeline은 원격 파이프라인의 일부인 모든 작업을 사용합니다.

파이프라인 실행에 작업 주석을 추가하여 원격 파이프라인의 작업을 덮어쓸 수 있습니다. 추가된 작업에는 원격 파이프라인의 작업과 동일한 이름이 있어야 합니다.

예를 들어 다음 파이프라인 실행 정의를 사용할 수 있습니다.

원격 파이프라인을 참조하는 파이프라인 실행 정의의 예 및 작업 덮어쓰기

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  annotations:
    pipelinesascode.tekton.dev/pipeline: "https://git.provider/raw/pipeline.yaml"
    pipelinesascode.tekton.dev/task: "./my-git-clone-task.yaml"
```

이 예제에서는 https://git.provider/raw/pipeline.yaml 에 있는 원격 작업에 git-clone이라는 작업과 my- git-clone -task.yaml 파일의 이름이 git-clone 라고도 하는 작업이 포함되어 있다고 가정합니다.

이 경우 파이프라인 실행이 원격 파이프라인을 실행하지만 파이프라인의 **git-clone** 작업을 정의한 작업으로 교체합니다.

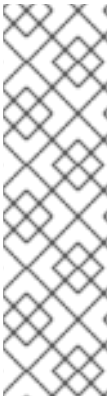
6장. 파이프라인 실행 관리

Pipeline을 코드로 사용하여 코드 리포지토리에서 파이프라인을 생성하고 이러한 파이프라인을 실행할 수 있습니다.

6.1. PIPELINE을 코드로 사용하여 파이프라인 실행 생성

코드로 파이프라인을 사용하여 파이프라인을 실행하려면 리포지토리의 .tekton/ 디렉터리에 있는 파이프라인 실행 정의 또는 템플릿을 YAML 파일로 생성할 수 있습니다. 원격 URL을 사용하여 다른 리포지토리의 YAML 파일을 참조할 수 있지만 파이프라인 실행은 .tekton/ 디렉터리가 포함된 리포지토리의 이벤트에서만 트리거됩니다.

코드 확인자인 파이프라인은 외부 종속 항목 없이 단일 파이프라인 실행으로 모든 작업을 통해 파이프라인을 번들로 실행합니다.



참고

- 파이프라인의 경우 spec 또는 분리된 Pipeline 오브젝트와 함께 하나 이상의 파이프라인 실행을 사용합니다.
- 작업의 경우 파이프라인 내부에 작업 사양을 포함하거나 Task 오브젝트로 별도로 정의합니다.

커밋 및 URL 매개 변수화

{{<var>}} 형식으로 동적 확장 가능한 변수를 사용하여 커밋 및 URL의 매개변수를 지정할 수 있습니다. 현재 다음 변수를 사용할 수 있습니다.

- {{repo_owner}}: 리포지토리 소유자.
- {{REPO_NAME}}: 리포지토리 이름입니다.
- {{repo_url}}: 리포지토리 전체 URL입니다.
- {{revision}}: 커밋의 전체 SHA 버전.

- {{sender}}: 커밋 보낸 사람의 사용자 이름 또는 계정 ID입니다.
- {{source_branch}}: 이벤트가 시작된 분기 이름입니다.
- {{target_branch}}: 이벤트가 대상으로 하는 분기 이름입니다. 푸시 이벤트의 경우 source_branch 와 동일합니다.
- {{pull_request_number}}: pull_request 이벤트 유형에 대해서만 정의된 가져오기 또는 병합 요청 번호입니다.
- {{git_auth_secret}}: 개인 리포지터리를 확인하기 위해 Git 공급자의 토큰을 사용하여 자동으로 생성되는 시크릿 이름입니다.

파이프라인 실행에 이벤트 일치

파이프라인 실행에 특수 주석을 사용하여 각 파이프라인 실행과 다른 Git 공급자 이벤트를 일치시킬 수 있습니다. 이벤트와 일치하는 파이프라인이 여러 개 있는 경우 코드가 병렬로 실행되고 파이프라인 실행이 완료되면 결과를 Git 공급자에 게시합니다.

파이프라인 실행에 가져오기 이벤트 일치

다음 예제를 사용하여 기본 분기를 대상으로 하는 pull_request 이벤트와 함께 pipeline-pr- main 파이프라인 실행을 일치시킬 수 있습니다.

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: pipeline-pr-main
annotations:
  pipelinesascode.tekton.dev/on-target-branch: "[main]" 1
  pipelinesascode.tekton.dev/on-event: "[pull_request]"
# ...
```

1

쉼표로 구분된 항목을 추가하여 여러 분기를 지정할 수 있습니다. 예를 들면 "[main, release-nightly]" 입니다. 또한 다음을 지정할 수 있습니다.

- "refs/heads/main"과 같은 분기에 대한 전체 참조

- "refs/heads/*"와 같은 패턴 일치가 있는 글러입니다.
- "refs/tags/1.*"와 같은 태그

파이프라인 실행에 푸시 이벤트 일치

다음 예제를 사용하여 **refs/heads/main** 분기를 대상으로 하는 푸시 이벤트와 함께 **pipeline- push -on-main** 파이프라인 실행과 일치시킬 수 있습니다.

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: pipeline-push-on-main
annotations:
  pipelinesascode.tekton.dev/on-target-branch: "[refs/heads/main]" 1
  pipelinesascode.tekton.dev/on-event: "[push]"
# ...
```

1

쉼표로 구분된 항목을 추가하여 여러 분기를 지정할 수 있습니다. 예를 들면 **"[main, release-nightly]"** 입니다. 또한 다음을 지정할 수 있습니다.

- "refs/heads/main"과 같은 분기에 대한 전체 참조
- "refs/heads/*"와 같은 패턴 일치가 있는 글러입니다.
- "refs/tags/1.*"와 같은 태그

주석 이벤트와 파이프라인 실행 일치

다음 예제를 사용하여 주석의 텍스트가 **^/merge-pr** 정규식과 일치하는 경우 가져오기 요청에 대한 주석과 함께 **pipeline-comment** 파이프라인 실행을 일치시킬 수 있습니다.

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: pipeline-comment
```

```
annotations:
  pipelinesascode.tekton.dev/on-comment: "^/merge-pr"
# ...
```

파이프라인 실행이 주석 작성자가 다음 요구 사항 중 하나를 충족하는 경우에만 시작됩니다.

- 작성자는 리포지토리의 소유자입니다.
- 작성자는 리포지토리의 공동 작업자입니다.
- 작성자는 리포지토리 조직에 있는 공용 멤버입니다.
- 주석 작성자는 Kubernetes 문서에 정의된 대로 리포지토리 루트에 있는 **OWNERS** 파일의 승인자 또는 검토자 섹션에 나열됩니다. 코드로서의 파이프라인은 **OWNERS** 및 **OWNERS_ALIASES** 파일에 대한 사양을 지원합니다. **OWNERS** 파일에 필터 섹션이 포함된 경우 코드로서의 파이프라인은 승인자 및 검토자가 .* 필터에 대해서만 일치합니다.

중요

주석 이벤트를 파이프라인 실행에 일치시키는 것은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

고급 이벤트 일치

코드로서의 파이프라인은 고급 이벤트 일치에 대한 **CEL(Common Expression Language)** 기반 필터링 사용을 지원합니다. 파이프라인 실행에 **pipelinesascode.tekton.dev/on-cel-expression** 주석이 있는 경우 **Pipeline**은 **CEL** 표현식을 사용하고 **on-target-branch** 주석을 건너뜁니다. **CEL** 표현식은 간단한 **on-target-branch** 주석 일치와 비교하여 복잡한 필터링 및 부정을 허용합니다.

Pipeline과 함께 **CEL** 기반 필터링을 코드로 사용하려면 다음 주석 예제를 고려하십시오.

-

기본 분기를 대상으로 하는 **pull_request** 이벤트를 일치시키고 **wip** 분기에서 가져오는 경우 다음을 수행합니다.

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: pipeline-advanced-pr
annotations:
  pipelinesascode.tekton.dev/on-cel-expression: |
    event == "pull_request" && target_branch == "main" && source_branch == "wip"
...
```

- 경로가 변경된 경우에만 파이프라인을 실행하려면 **.pathChanged** 접미사 함수를 **glob** 패턴과 함께 사용하면 됩니다.

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: pipeline-advanced-pr-pathchanged
annotations:
  pipelinesascode.tekton.dev/on-cel-expression: |
    event == "pull_request" && "docs/\*.md".pathChanged() (1)
# ...
```

(1) **docs** 디렉터리의 모든 마크다운 파일과 일치합니다.

- 제목 **[DOWNSTREAM]** 으로 시작하는 모든 가져오기 요청을 일치시킵니다.

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: pipeline-advanced-pr-downstream
annotations:
  pipelinesascode.tekton.dev/on-cel-expression: |
    event == "pull_request && event_title.startsWith("[DOWNSTREAM]")
# ...
```

- **pull_request** 이벤트에서 파이프라인을 실행하려면 실험적 분기를 건너뜁니다.

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  name: pipeline-advanced-pr-not-experimental
annotations:
```

```
    pipelinesascode.tekton.dev/on-cel-expression: |
      event == "pull_request" && target_branch != experimental"
  # ...
```

Pipeline을 코드로 사용하는 동안 고급 CEL 기반 필터링의 경우 다음 필드 및 접미사 함수를 사용할 수 있습니다.

- 이벤트: push 또는 pull_request 이벤트입니다.
- target_branch: 대상 분기입니다.
- source_branch: pull_request 이벤트의 출처 분기입니다. 푸시 이벤트의 경우 target_branch 와 동일합니다.
- event_title: push 이벤트의 커밋 제목과 pull_request 이벤트에 대한 가져오기 또는 병합 요청과 같은 이벤트 제목과 일치합니다. 현재 GitHub, Gitlab 및 Bitbucket Cloud만 지원되는 공급자입니다.
- .pathChanged: 문자열에 대한 접미사 함수입니다. 문자열은 경로가 변경되었는지 확인하는 경로의 glob일 수 있습니다. 현재 GitHub 및 Gitlab만 공급자로 지원됩니다.

또한 Git 리포지토리 공급자가 전달하는 대로 전체 페이로드에 액세스할 수 있습니다. headers 필드를 사용하여 페이로드의 헤더(예: headers['x-github-event'])에 액세스합니다. body 필드를 사용하여 페이로드 본문(예: body.pull_request.state)에 액세스합니다.

중요

코드로 CEL 기반 필터링에 대한 페이로드의 헤더와 본문을 코드로 사용하는 것은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 Red Hat 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

다음 예제에서는 다음 조건이 모두 **true**인 경우에만 파이프라인 실행이 시작됩니다.

- 가져오기 요청은 주요 분기를 대상으로 합니다.
- 가져오기 요청 작성자는 슈퍼유저 입니다.
- 작업은 동기화됩니다. 이 작업은 가져오기 요청에서 업데이트가 수행될 때 트리거됩니다.

```
apiVersion: tekton.dev/v1
kind: PipelineRun
metadata:
  annotations:
    pipelinesascode.tekton.dev/on-cel-expression: |
      body.pull_request.base.ref == "main" &&
        body.pull_request.user.login == "superuser" &&
        body.action == "synchronize"
# ...
```

참고

이벤트 일치에 **header** 또는 **body** 필드를 사용하는 경우 **retest** 와 같은 **Git** 명령을 사용하여 파이프라인 실행을 트리거하지 못할 수 있습니다. **Git** 명령을 사용하는 경우 페이로드 본문은 원래 페이로드가 아닌 이 명령을 포함하는 주석입니다.

이벤트 일치에 **body** 필드를 사용할 때 파이프라인 실행을 다시 트리거하려면 가져오기 요청 또는 병합 요청을 닫고 다시 열거나 또는 다음 명령을 사용하여 새 **SHA** 커밋을 추가할 수 있습니다.

```
git commit --amend --no-edit && git push --force-with-lease
```

Github API 작업에 임시 **GitHub** 앱 토큰 사용

Pipelines에서 **GitHub App**에서 코드로 생성한 임시 설치 토큰을 사용하여 **GitHub API**에 액세스할 수 있습니다. 토큰 값은 **git-provider-token** 키의 개인 리포지토리에 대해 생성된 임시 **{{git_auth_secret}}** 동적 변수에 저장됩니다.

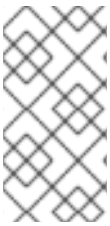
예를 들어 가져오기 요청에 주석을 추가하려면 **Pipelines**를 **Code** 주석으로 사용하여 **Tekton Hub**의 **github-add-comment** 작업을 사용할 수 있습니다.

```
...
  pipelinesascode.tekton.dev/task: "github-add-comment"
...
```

그런 다음 파이프라인 실행 정의의 **tasks** 섹션 또는 **finally** 작업에 작업을 추가할 수 있습니다.

```
[...]
tasks:
  - name:
      taskRef:
        name: github-add-comment
      params:
        - name: REQUEST_URL
          value: "{{ repo_url }}/pull/{{ pull_request_number }}" (1)
        - name: COMMENT_OR_FILE
          value: "Pipelines as Code IS GREAT!"
        - name: GITHUB_TOKEN_SECRET_NAME
          value: "{{ git_auth_secret }}"
        - name: GITHUB_TOKEN_SECRET_KEY
          value: "git-provider-token"
...
```

(1) 동적 변수를 사용하면 모든 리포지터리의 가져오기 요청에 이 스니펫 템플릿을 재사용할 수 있습니다.



참고

GitHub 앱에서는 생성된 설치 토큰을 8시간 동안 사용할 수 있으며 클러스터에서 다르게 구성하지 않는 한 이벤트가 시작된 저장소로 범위가 지정됩니다.

추가 리소스

- CEL 언어 사양

6.2. PIPELINE을 코드로 사용하여 파이프라인 실행

기본 구성을 사용하면 코드로서의 파이프라인은 리포지토리에서 가져오기 요청 또는 푸시와 같은 지정된 이벤트가 발생하면 리포지토리의 기본 분기의 **.tekton/** 디렉터리에서 모든 파이프라인을 실행합니다. 예를 들어 기본 분기에서 파이프라인 실행에 **pipelinesascode.tekton.dev/on-event: "[pull_request]"** 주석이 있는 경우 가져오기 요청 이벤트가 발생할 때마다 실행됩니다.

가져오기 요청 또는 병합 요청이 있는 경우 **Code**의 **Pipeline**은 가져오기 요청 작성자가 다음 조건을 충족하는 경우 기본 분기 이외의 분기에서 파이프라인을 실행합니다.

- 작성자는 리포지토리의 소유자입니다.
- 작성자는 리포지토리의 공동 작업자입니다.
- 작성자는 리포지토리 조직에 있는 공용 멤버입니다.
- 가져오기 요청 작성자는 Kubernetes 문서에 정의된 대로 리포지토리 루트에 있는 **OWNERS** 파일의 승인자 또는 검토자 섹션에 나열됩니다. 코드로서의 파이프라인은 **OWNERS** 및 **OWNERS_ALIASES** 파일에 대한 사양을 지원합니다. **OWNERS** 파일에 필터 섹션이 포함된 경우 코드로서의 파이프라인은 승인자 및 검토자가 **.*** 필터에 대해서만 일치합니다.

가져오기 요청 작성자가 요구 사항을 충족하지 않으면 요구 사항을 충족하는 다른 사용자가 가져오기 요청에 대해 **/ok-to-test** 를 처리하고 파이프라인 실행을 시작할 수 있습니다.

파이프라인 실행 실행

파이프라인 실행은 이벤트를 생성한 리포지토리와 연결된 **Repository CRD**(사용자 정의 리소스 정의)의 네임스페이스에서 항상 실행됩니다.

tkn pac CLI 툴을 사용하여 파이프라인 실행 실행을 확인할 수 있습니다.

- 마지막 파이프라인 실행의 실행을 수행하려면 다음 예제를 사용합니다.

```
$ tkn pac logs -n <my-pipeline-ci> -L 1
```

1 **my-pipeline-ci** 는 **Repository CRD**의 네임스페이스입니다.

- 대화형으로 파이프라인 실행을 수행하려면 다음 예제를 사용합니다.

```
$ tkn pac logs -n <my-pipeline-ci> 1
```

1

my-pipeline-ci 는 Repository CRD의 네임스페이스입니다. 마지막 파이프라인 이외의 파이프라인 실행을 확인해야 하는 경우 tkn pac logs 명령을 사용하여 리포지토리에 연결된 PipelineRun 을 선택할 수 있습니다.

GitHub 앱을 사용하여 Pipeline을 코드로 구성한 경우 코드로 Pipeline은 GitHub 앱의 Checks 탭에 URL을 게시합니다. URL을 클릭하고 파이프라인 실행을 추적할 수 있습니다.

6.3. PIPELINE을 코드로 사용하여 파이프라인 실행 다시 시작 또는 취소

새 커밋을 분기로 전송하거나 가져오기 요청 발생과 같은 이벤트 없이 파이프라인 실행을 재시작하거나 취소할 수 있습니다. 모든 파이프라인 실행을 다시 시작하려면 GitHub 앱의 모든 점검 기능을 사용합니다.

모두 또는 특정 파이프라인 실행을 다시 시작하려면 다음 주석을 사용합니다.

- /test 및 /retest 주석은 모든 파이프라인 실행을 다시 시작합니다.
- /test <pipeline_run_name > 및 /retest <pipeline_run_name > 주석이 시작되거나 특정 파이프라인 실행을 다시 시작합니다. 이 명령을 사용하여 이 파이프라인 실행에 대한 이벤트에서 트리거되었는지 여부에 관계없이 리포지토리에서 코드 파이프라인 실행으로 Pipeline을 시작할 수 있습니다.

모든 또는 특정 파이프라인 실행을 취소하려면 다음 주석을 사용합니다.

- /cancel 주석은 모든 파이프라인 실행을 취소합니다.
- /cancel <pipeline_run_name> 주석은 특정 파이프라인 실행을 취소합니다.

주석의 결과는 GitHub 앱의 확인 탭에 표시됩니다.

주석 작성자가 다음 요구 사항 중 하나를 충족하는 경우에만 파이프라인 실행이 시작, 재시작 또는 취소됩니다.

- 작성자는 리포지토리의 소유자입니다.
- 작성자는 리포지토리의 공동 작업자입니다.
- 작성자는 리포지토리 조직에 있는 공용 멤버입니다.
- 주석 작성자는 Kubernetes 문서에 정의된 대로 리포지토리 루트에 있는 **OWNERS** 파일의 승인자 또는 검토자 섹션에 나열됩니다. 코드로서의 파이프라인은 **OWNERS** 및 **OWNERS_ALIASES** 파일에 대한 사양을 지원합니다. **OWNERS** 파일에 필터 섹션이 포함된 경우 코드로서의 파이프라인은 승인자 및 검토자가 .* 필터에 대해서만 일치합니다.

중요

주석을 사용하여 이벤트와 일치하지 않는 파이프라인 실행을 시작하는 것은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

프로세스

- 가져오기 요청을 대상으로 하고 **GitHub** 앱을 사용하는 경우 확인 탭으로 이동하여 모든 검사를 다시 실행합니다.
- 가져오기 또는 병합 요청을 대상으로 하는 경우 가져오기 요청 내에서 주석을 사용합니다.

모든 파이프라인 실행을 취소하는 주석 예

```
This is a comment inside a pull request.
/cancel
```

- 내보내기 요청을 대상으로 하는 경우 커밋 메시지에 주석을 포함합니다.

참고

이 기능은 **GitHub** 공급자에서만 지원됩니다.

a. **GitHub** 리포지토리로 이동합니다.

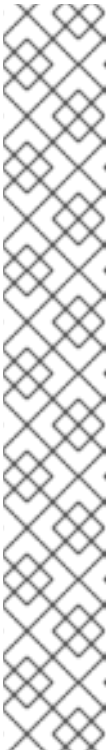
b. 커밋 섹션을 클릭합니다.

c. 파이프라인 실행을 다시 시작할 커밋을 클릭합니다.

d. 코멘트를 추가할 줄 번호를 클릭합니다.

특정 파이프라인 실행을 시작하거나 다시 시작하는 주석의 예

```
This is a comment inside a commit.
/retest example_pipeline_run
```



참고

내보내기 요청 내에서 여러 분기에 존재하는 커밋에 대해 명령을 실행하면 최신 커밋이 있는 분기가 사용됩니다.

이로 인해 두 가지 상황이 발생합니다.

- **/test** 와 같은 인수 없이 커밋에서 명령을 실행하면 기본 분기에서 테스트가 자동으로 수행됩니다.
- **/test branch:user-branch** 와 같은 분기 사양을 포함하는 경우, 해당 주석이 **user-branch** 분기의 컨텍스트와 함께 있는 커밋에 대해 테스트가 수행됩니다.

6.4. PIPELINE을 코드로 사용하여 파이프라인 실행 상태 모니터링

컨텍스트 및 지원되는 툴에 따라 파이프라인 실행 상태를 다른 방식으로 모니터링할 수 있습니다.

GitHub 앱의 상태

파이프라인 실행이 완료되면 파이프라인의 각 작업 및 **tkn pipelinerun describe** 명령의 출력에 대한 제한된 정보가 있는 **Check** 탭에 상태가 추가됩니다.

로그 오류 스니펫

Pipeline as Code가 파이프라인 작업 중 하나에서 오류를 감지하면 첫 번째 실패한 작업의 작업 분석에서 마지막 **3**행으로 구성된 작은 조각이 표시됩니다.

참고

코드로서 파이프라인을 사용하면 파이프라인 실행을 보고 시크릿 값을 숨겨진 문자로 교체하여 보안 누출을 방지할 수 있습니다. 그러나 코드로서의 파이프라인은 작업 공간 및 **envFrom** 소스에서 제공하는 보안을 숨길 수 없습니다.

로그 오류 스니펫에 대한 주석

TektonConfig 사용자 지정 리소스의 **pipelinesAsCode.settings** 사양에서 **error-detection-from-container-logs** 매개변수를 **true** 로 설정할 수 있습니다. 이 경우 코드로서의 파이프라인은 컨테이너 로

그의 오류를 감지하고 오류가 발생한 가져오기 요청에 주석을 추가합니다.

중요

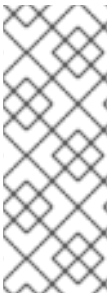
로그 오류 스니펫에 대한 주석을 추가하는 것은 기술 프리뷰 기능 전용입니다. 기술 프리뷰 기능은 **Red Hat** 프로덕션 서비스 수준 계약(SLA)에서 지원되지 않으며 기능적으로 완전하지 않을 수 있습니다. 따라서 프로덕션 환경에서 사용하는 것은 권장하지 않습니다. 이러한 기능을 사용하면 향후 제품 기능을 조기에 이용할 수 있어 개발 과정에서 고객이 기능을 테스트하고 피드백을 제공할 수 있습니다.

Red Hat 기술 프리뷰 기능의 지원 범위에 대한 자세한 내용은 기술 프리뷰 기능 지원 범위를 참조하십시오.

현재 **Pipeline**은 다음과 같은 형식의 **makefile** 또는 **grep** 출력처럼 보이는 간단한 사례만 지원합니다.

```
<filename>:<line>:<column>: <error message>
```

error-detection-simple-regexp 매개변수를 사용하여 오류를 감지하는 데 사용되는 정규식을 사용자 지정할 수 있습니다. 정규식은 이름이 지정된 그룹을 사용하여 일치 항목을 지정하는 방법에 대한 유연성을 제공합니다. 일치에 필요한 그룹은 파일 이름,행, 오류입니다. **Pipeline**을 기본 정규식의 코드 구성 맵으로 볼 수 있습니다.



참고

기본적으로 코드로서의 파이프라인은 컨테이너 로그의 마지막 **50**행만 검사합니다. **error-detection-max-number-of-lines** 필드에서 이 값을 늘리거나 무제한 줄 수에 대해 **-1**을 설정할 수 있습니다. 그러나 이러한 구성은 감시자의 메모리 사용량을 늘릴 수 있습니다.

Webhook 상태

Webhook의 경우 이벤트가 가져오기 요청이면 가져오기 또는 병합 요청에 대한 주석으로 상태가 추가됩니다.

실패

네임스페이스가 **Repository CRD**(사용자 정의 리소스 정의)와 일치하는 경우 코드로서의 **Pipeline**은 네임스페이스 내부의 **Kubernetes** 이벤트에서 실패 로그 메시지를 내보냅니다.

Repository CRD와 관련된 상태

파이프라인 실행에 대한 마지막 5개의 상태 메시지는 **Repository** 사용자 정의 리소스 내에 저장됩니다.

```
$ oc get repo -n <pipelines-as-code-ci>
```

```
NAME                 URL                                                NAMESPACE          SUCCEEDED
REASON      STARTTIME   COMPLETIONTIME
pipelines-as-code-ci   https://github.com/openshift-pipelines/pipelines-as-code   pipelines-as-
code-ci   True        Succeeded   59m         56m
```

tkn pac describe 명령을 사용하면 리포지토리 및 해당 메타데이터와 연결된 실행 상태를 추출할 수 있습니다.

알림

코드로서의 파이프라인은 알림을 관리하지 않습니다. 알림이 필요한 경우 파이프라인의 **finally** 기능을 사용합니다.

추가 리소스

- 성공 또는 실패 시 **Slack** 메시지를 보내는 작업의 예
- 내보내기 이벤트에서 트리거된 **finally** 작업이 포함된 파이프라인 실행 예

추가 리소스

- 프라이빗 리포지토리 복제에 사용되는 **git-clone** 작업의 예

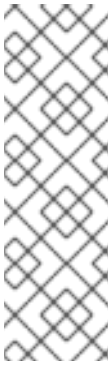
6.5. PIPELINE을 코드로 사용하여 파이프라인 실행 정리

사용자 네임스페이스에 많은 파이프라인 실행이 있을 수 있습니다. **max-keep-runs** 주석을 설정하면 이벤트와 일치하는 파이프라인 실행 수가 제한된 수를 유지하도록 **Pipeline**을 **Code**로 구성할 수 있습니다. 예를 들면 다음과 같습니다.

```
...
  pipelinesascode.tekton.dev/max-keep-runs: "<max_number>" 1
...
```

1

코드가 성공적으로 실행된 후 바로 정리가 시작되어 주석을 사용하여 구성된 최대 파이프라인 실행 수만 유지됩니다.



참고

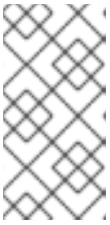
- **Code**는 실행 중인 파이프라인 정리를 건너뛰지만 파이프라인 실행을 알 수 없는 상태로 정리합니다.
- 코드가 실패한 가져오기 요청 정리를 건너뛰는 파이프라인입니다.

6.6. PIPELINE에서 코드로 들어오는 WEBHOOK 사용

들어오는 **Webhook URL**과 공유 보안을 사용하여 리포지토리에서 파이프라인 실행을 시작할 수 있습니다.

들어오는 **Webhook**를 사용하려면 **Repository CRD**(사용자 정의 리소스 정의)의 **spec** 섹션에서 다음을 지정합니다.

- **Pipeline**이 코드로 일치하는 들어오는 **Webhook URL**입니다.
- **Git** 공급자 및 사용자 토큰입니다. 현재 코드로서의 파이프라인은 **github,gitlab** 및 **bitbucket-cloud** 를 지원합니다.



참고

GitHub 앱 컨텍스트에서 들어오는 **Webhook URL**을 사용하는 경우 토큰을 지정해야 합니다.

- 대상 분기와 들어오는 **Webhook URL**의 시크릿입니다.

예: 들어오는 **Webhook**가 있는 리포지토리 **CRD**

```
apiVersion: "pipelinesascode.tekton.dev/v1alpha1"
kind: Repository
metadata:
  name: repo
  namespace: ns
spec:
  url: "https://github.com/owner/repo"
  git_provider:
    type: github
    secret:
      name: "owner-token"
  incoming:
    - targets:
      - main
      secret:
        name: repo-incoming-secret
      type: webhook-url
```

예: 들어오는 Webhook의 repo-incoming-secret 시크릿

```
apiVersion: v1
kind: Secret
metadata:
  name: repo-incoming-secret
  namespace: ns
type: Opaque
stringData:
  secret: <very-secure-shared-secret>
```

Git 리포지토리의 .tekton 디렉터리에 있는 파이프라인 실행을 트리거하려면 다음 명령을 사용합니다.

```
$ curl -X POST 'https://control.pac.url/incoming?secret=very-secure-shared-
secret&repository=repo&branch=main&pipelinerun=target_pipelinerun'
```

코드로서의 파이프라인은 들어오는 URL과 일치하고 푸시 이벤트로 처리합니다. 그러나 Code로서의 Pipeline은 이 명령으로 트리거된 파이프라인 실행의 상태를 보고하지 않습니다.

보고서 또는 알림을 가져오려면 finally 작업에서 파이프라인에 직접 추가합니다. 또는 tkn pac CLI 툴을 사용하여 리포지토리 CRD를 검사할 수 있습니다.

6.7. 추가 리소스

- 코드 리포지토리로 **Pipeline**의 **.tekton/** 디렉터리의 예
- 개발자 화면을 사용하여 애플리케이션 생성

7장. 코드 명령 참조로 파이프라인

tkn pac CLI 툴을 사용하여 Pipeline을 코드로 제어할 수 있습니다. TektonConfig 사용자 정의 리소스를 사용하여 Pipeline을 Code 로깅으로 구성하고 oc 명령을 사용하여 Pipeline을 Code 로그로 볼 수도 있습니다.

7.1. 코드 명령 참조로 파이프라인

tkn pac CLI 툴에서는 다음과 같은 기능을 제공합니다.

- 코드 설치 및 구성으로 Pipeline을 부트스트랩합니다.
- 코드 리포지토리로 새 Pipeline을 생성합니다.
- 모든 Pipeline을 코드 리포지토리로 나열합니다.
- Pipeline을 코드 리포지토리 및 관련 실행으로 설명합니다.
- 시작하려면 간단한 파이프라인 실행을 생성합니다.
- 파이프라인에서 코드로 실행된 것처럼 파이프라인 실행을 해결합니다.

작은 정보

애플리케이션 소스 코드가 포함된 Git 리포지토리를 변경할 필요가 없도록 테스트 및 실험 기능에 해당하는 명령을 사용할 수 있습니다.

7.1.1. 기본 구문

```
$ tkn pac [command or options] [arguments]
```

7.1.2. 글로벌 옵션

```
$ tkn pac --help
```

7.1.3. 유틸리티 명령

7.1.3.1. 부트스트랩

표 7.1. 코드 설치 및 구성으로 Pipeline 부트스트랩

명령	설명
tkn pac bootstrap	GitHub 및 GitHub Enterprise와 같은 Git 리포지토리 호스팅 서비스 공급자의 코드로 Pipeline을 설치하고 구성합니다.
tkn pac bootstrap --nightly	Nightly build of Pipeline을 코드로 설치합니다.
tkn pac bootstrap --route-url <public_url_to_ingress_spec>	OpenShift 경로 URL을 덮어씁니다. 기본적으로 **tkn pac bootstrap** 은 파이프라인과 코드 컨트롤러 서비스로 자동으로 연결된 OpenShift 경로를 감지합니다. OpenShift Container Platform 클러스터가 없는 경우 Ingress 끝점을 가리키는 공용 URL을 요청합니다.
tkn pac bootstrap github-app	**openshift-pipelines** 네임스페이스에 GitHub 애플리케이션 및 시크릿을 생성합니다.

7.1.3.2. 리포지토리

표 7.2. Pipeline을 코드 리포지토리로 관리

명령	설명
tkn pac create repository	파이프라인 실행 템플릿을 기반으로 새 파이프라인 리포지토리 및 네임스페이스를 생성합니다.
tkn pac list	모든 Pipeline을 Code 리포지토리로 나열하고 관련 실행의 마지막 상태를 표시합니다.
tkn pac repo describe	Pipeline을 코드 리포지토리 및 관련 실행으로 설명합니다.

7.1.3.3. generate

표 7.3. Pipeline을 코드로 사용하여 파이프라인 실행 생성

명령	설명

명령	설명
tkn pac generate	간단한 파이프라인 실행을 생성합니다. 소스 코드가 포함된 디렉터리에서 실행하면 현재 Git 정보를 자동으로 감지합니다. 또한 기본 언어 탐지 기능을 사용하고 언어에 따라 추가 작업을 추가합니다. 예를 들어 리포지토리 루트에서 **setup.py** 파일을 감지하면 pylint 작업이 생성된 파이프라인 실행에 자동으로 추가됩니다.

7.1.3.4. resolve

표 7.4. Pipeline을 코드로 사용하여 파이프라인 실행 해결 및 실행

명령	설명
tkn pac resolve	파이프라인이 서비스의 코드로 파이프라인을 소유한 것처럼 실행됩니다.
tkn pac resolve -f .tekton/pull-request.yaml | oc apply -f -	**.tekton/pull-request.yaml** 에서 템플릿을 사용하는 실시간 파이프라인 실행 상태를 표시합니다. 로컬 시스템에서 실행되는 Kubernetes 설치와 결합하여 새 커밋을 생성하지 않고 파이프라인 실행을 확인할 수 있습니다. 소스 코드 리포지토리에서 명령을 실행하는 경우 현재 Git 정보를 감지하고 현재 버전 또는 분기와 같은 매개변수를 자동으로 해결합니다.
tkn pac resolve -f .tekton/pr.yaml -p revision=main -p repo_name= <repository_name>	Git 리포지토리에서 파생된 기본 매개 변수 값을 재정의하여 파이프라인 실행을 실행합니다. **-f** 옵션은 디렉터리 경로를 수락하고 해당 디렉터리의 모든 **.yaml** 또는 **.yml** 파일에 **tkn pac resolve** 명령을 적용할 수도 있습니다. 동일한 명령에서 **-f** 플래그를 여러 번 사용할 수도 있습니다. **-p** 옵션을 사용하여 매개변수 값을 지정하여 Git 리포지토리에서 수집된 기본 정보를 덮어쓸 수 있습니다. 예를 들어 Git 분기를 버전으로 사용하고 다른 리포지토리 이름을 사용할 수 있습니다.

7.2. PIPELINE을 코드 로깅으로 구성

TektonConfig CR(사용자 정의 리소스)에서 pac-config-logging 구성 맵을 편집하여 Pipeline을 코드 로깅으로 구성할 수 있습니다.

사전 요구 사항

- 클러스터에 코드로 Pipeline이 설치되어 있어야 합니다.

프로세스

1. 웹 콘솔의 관리자 화면에서 Administration → CustomResourceDefinitions 로 이동합니다.
2. 이름으로 검색 필드를 사용하여 tektonconfigs.operator.tekton.dev CRD(사용자 정의 리소스 정의)를 검색하고 TektonConfig 를 클릭하여 CRD 세부 정보 페이지를 확인합니다.
3. Instances 탭을 클릭합니다.
4. config 인스턴스를 클릭하여 TektonConfig CR 세부 정보를 확인합니다.
5. YAML 탭을 클릭합니다.
6. 요구 사항에 따라 .options.configMaps.pac-config-logging.data 매개변수 아래의 loglevel. 필드를 편집합니다.

Pipelines가 Code 로그 수준 필드가 warn로 설정된 TektonConfig CR의 예

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  platforms:
    openshift:
      pipelinesAsCode:
        options:
          configMaps:
            pac-config-logging:
              data:
                loglevel.pac-watcher: warn (1)
```

```
    loglevel.pipelines-as-code-webhook: warn 2
    loglevel.pipelinesascode: warn 3
    zap-logger-config: |
     {
      "level": "info",
      "development": false,
      "sampling": {
       "initial": 100,
       "thereafter": 100
      },
      "outputPaths": ["stdout"],
      "errorOutputPaths": ["stderr"],
      "encoding": "json",
      "encoderConfig": {
       "timeKey": "ts",
       "levelKey": "level",
       "nameKey": "logger",
       "callerKey": "caller",
       "messageKey": "msg",
       "stacktraceKey": "stacktrace",
       "lineEnding": "",
       "levelEncoder": "",
       "timeEncoder": "iso8601",
       "durationEncoder": "",
       "callerEncoder": ""
      }
     }
```

1 **pipelines-as-code-watcher** 구성 요소의 로그 수준입니다.

2 **pipelines-as-code-webhook** 구성 요소의 로그 수준입니다.

3 **pipelines-as-code-controller** 구성 요소의 로그 수준입니다.

7. 선택 사항: **.options.deployments** 필드에서 각 구성 요소에 대해 **.env.value** 를 변경하여 **Pipeline**의 사용자 정의 로깅 구성 맵을 코드 구성 요소로 생성합니다. 아래 예제에서는 **custom-pac-config-logging** 이라는 사용자 정의 구성 맵이 있는 구성을 보여줍니다.

Pipeline을 **Code** 사용자 정의 로깅 구성 맵으로 사용하는 **TektonConfig CR**의 예

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  platforms:
    openshift:
      pipelinesAsCode:
        enable: true
        options:
          configMaps:
            custom-pac-config-logging:
              data:
                loglevel.pac-watcher: warn
                loglevel.pipelines-as-code-webhook: warn
                loglevel.pipelinesascode: warn
                zap-logger-config: |
                  {
                    "level": "info",
                    "development": false,
                    "sampling": {
                      "initial": 100,
                      "thereafter": 100
                    },
                    "outputPaths": ["stdout"],
                    "errorOutputPaths": ["stderr"],
                    "encoding": "json",
                    "encoderConfig": {
                      "timeKey": "ts",
                      "levelKey": "level",
                      "nameKey": "logger",
                      "callerKey": "caller",
                      "messageKey": "msg",
                      "stacktraceKey": "stacktrace",
                      "lineEnding": "",
                      "levelEncoder": "",
                      "timeEncoder": "iso8601",
                      "durationEncoder": "",
                      "callerEncoder": ""
                    }
                  }
          deployments:
            pipelines-as-code-controller:
              spec:
                template:
                  spec:
                    containers:
                    - name: pac-controller
                      env:
                      - name: CONFIG_LOGGING_NAME
                        value: custom-pac-config-logging
            pipelines-as-code-watcher:
              spec:
```

```
    template:
      spec:
        containers:
        - name: pac-watcher
          env:
          - name: CONFIG_LOGGING_NAME
            value: custom-pac-config-logging
pipelines-as-code-webhook:
  spec:
    template:
      spec:
        containers:
        - name: pac-webhook
          env:
          - name: CONFIG_LOGGING_NAME
            value: custom-pac-config-logging
```

7.3. 네임스페이스별 PIPELINE을 코드 로그로 분할

코드 로그로서의 파이프라인에는 로그를 필터링하거나 특정 네임스페이스로 로그를 분할할 수 있는 네임스페이스 정보가 포함되어 있습니다. 예를 들어, **Pipeline**을 **mynamespace** 네임스페이스와 관련된 코드 로그로 보려면 다음 명령을 입력합니다.

```
$ oc logs pipelines-as-code-controller-<unique-id> -n openshift-pipelines | grep mynamespace
```
(1)

(1) **pipelines-as-code-controller-<unique-id >**를 코드 컨트롤러 이름으로 **Pipeline**으로 바꿉니다.

7.4. 추가 리소스

- OpenShift Pipelines 설치
- tkn 설치