



OpenShift Container Platform 4.8

CI/CD

OpenShift Container Platform의 빌드, 파이프라인, GitOps에 대한 정보 제공

OpenShift Container Platform 4.8 CI/CD

OpenShift Container Platform의 빌드, 파이프라인, GitOps에 대한 정보 제공

법적 공지

Copyright © 2023 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

초록

OpenShift Container Platform의 CI/CD

차례

1장. OPENSIFT CONTAINER PLATFORM CI/CD 개요	3
1.1. OPENSIFT BUILDS	3
1.2. OPENSIFT PIPELINES	3
1.3. OPENSIFT GITOPS	3
1.4. JENKINS	3
2장. 빌드	4
2.1. 이미지 빌드 이해	4
2.2. 빌드 구성 이해	5
2.3. 빌드 입력 생성	6
2.4. 빌드 출력 관리	32
2.5. 빌드 전략 사용	34
2.6. BUILDDAH를 사용한 사용자 정의 이미지 빌드	52
2.7. 기본 빌드 수행	55
2.8. 빌드 트리거 및 수정	60
2.9. 고급 빌드 수행	73
2.10. 빌드에서 RED HAT 서브스크립션 사용	79
2.11. 전략에 따른 빌드 보안	84
2.12. 빌드 구성 리소스	87
2.13. 빌드 문제 해결	91
2.14. 빌드에 대해 신뢰할 수 있는 추가 인증 기관 설정	92
3장. JENKINS에서 TEKTON으로 마이그레이션	94
3.1. JENKINS에서 TEKTON으로 마이그레이션	94
4장. PIPELINE	102
4.1. RED HAT OPENSIFT PIPELINES 릴리스 정보	102
4.2. OPENSIFT PIPELINES 이해	144
4.3. OPENSIFT PIPELINES 설치	166
4.4. OPENSIFT PIPELINES 설치 제거	171
4.5. OPENSIFT PIPELINES를 사용하여 애플리케이션용 CI/CD 솔루션 작성	172
4.6. 개발자 화면을 사용하여 RED HAT OPENSIFT PIPELINES 작업	193
4.7. 파이프라인의 리소스 사용량 감소	208
4.8. 권한 있는 보안 컨텍스트에서 POD 사용	211
4.9. 이벤트 리스너로 WEBHOOK 보안	217
4.10. GIT 보안을 사용하여 파이프라인 인증	220
4.11. OPENSIFT LOGGING OPERATOR를 사용하여 파이프라인 로그 보기	228
5장. GITOPS	234
5.1. RED HAT OPENSIFT GITOPS 릴리스 정보	234
5.2. OPENSIFT GITOPS 이해	267
5.3. OPENSIFT GITOPS 설치	269
5.4. OPENSIFT GITOPS 설치 제거	271
5.5. 클러스터 구성으로 애플리케이션을 배포하여 OPENSIFT 클러스터 구성	273
5.6. ARGO CD로 SPRING BOOT 애플리케이션 배포	283
5.7. DEX를 사용하여 ARGO CD에 대한 SSO 구성	288
5.8. KEYCLOAK을 사용하여 ARGO CD에 대한 SSO 구성	290
5.9. 인프라 노드에서 GITOPS 컨트롤 플레인 워크로드 실행	294
5.10. GITOPS OPERATOR의 크기 조정 요구 사항	297

1장. OPENSIFT CONTAINER PLATFORM CI/CD 개요

OpenShift Container Platform은 개발자를 위한 엔터프라이즈급 Kubernetes 플랫폼으로, CI(Continuous Integration) 및 CD(Continuous Delivery)와 같은 DevOps 사례를 통해 애플리케이션 제공 프로세스를 자동화할 수 있습니다. 조직의 요구 사항을 충족하기 위해 OpenShift Container Platform은 다음과 같은 CI/CD 솔루션을 제공합니다.

- OpenShift Builds
- OpenShift Pipelines
- OpenShift GitOps

1.1. OPENSIFT BUILDS

OpenShift 빌드를 사용하면 선언적 빌드 프로세스를 사용하여 클라우드 네이티브 앱을 생성할 수 있습니다. BuildConfig 오브젝트를 생성하는 데 사용하는 YAML 파일에서 빌드 프로세스를 정의할 수 있습니다. 이 정의에는 빌드 트리거, 입력 매개 변수, 소스 코드와 같은 속성이 포함됩니다. 배포된 경우 BuildConfig 오브젝트는 일반적으로 실행 가능한 이미지를 빌드하여 컨테이너 이미지 레지스트리로 푸시합니다.

OpenShift 빌드에서는 빌드 전략에 대해 다음과 같은 확장 가능한 지원을 제공합니다.

- Docker 빌드
- S2I(Source-to-Image) 빌드
- 사용자 정의 빌드

자세한 내용은 [이미지 빌드 이해](#)를 참조하십시오.

1.2. OPENSIFT PIPELINES

OpenShift Pipelines는 Kubernetes 네이티브 CI/CD 프레임워크를 제공하여 자체 컨테이너에서 CI/CD 파이프라인의 각 단계를 설계하고 실행합니다. 예측 가능한 결과를 통해 온디맨드 파이프라인을 충족하도록 독립적으로 확장할 수 있습니다.

자세한 내용은 [OpenShift Pipelines 이해](#)를 참조하십시오.

1.3. OPENSIFT GITOPS

OpenShift GitOps는 Argo CD를 선언적 GitOps 엔진으로 사용하는 Operator입니다. 다중 클러스터 OpenShift 및 Kubernetes 인프라에서 GitOps 워크플로를 활성화합니다. 관리자는 OpenShift GitOps를 사용하여 클러스터 및 개발 라이프사이클 전체에 Kubernetes 기반 인프라 및 애플리케이션을 일관되게 구성하고 배포할 수 있습니다.

자세한 내용은 [OpenShift GitOps 이해](#)를 참조하십시오.

1.4. JENKINS

Jenkins는 애플리케이션 및 프로젝트 빌드, 테스트 및 배포 프로세스를 자동화합니다. OpenShift Developer Tools는 OpenShift Container Platform과 직접 통합하는 Jenkins 이미지를 제공합니다. Samples Operator 템플릿 또는 인증된 Helm 차트를 사용하여 Jenkins를 OpenShift에 배포할 수 있습니다.

2장. 빌드

2.1. 이미지 빌드 이해

2.1.1. 빌드

빌드는 입력 매개변수를 결과 오브젝트로 변환하는 프로세스입니다. 대부분의 경우 프로세스는 입력 매개변수 또는 소스 코드를 실행 가능한 이미지로 변환하는 데 사용됩니다. **BuildConfig** 오브젝트는 전체 빌드 프로세스에 대한 정의입니다.

OpenShift Container Platform에서는 빌드 이미지에서 컨테이너를 생성하고 이를 컨테이너 이미지 레지스트리로 내보내는 방식으로 Kubernetes를 사용합니다.

빌드 오브젝트는 빌드에 대한 입력, 즉 빌드 프로세스를 완료하고 빌드 프로세스를 로깅한 후 성공한 빌드의 리소스를 게시하고 빌드의 최종 상태를 게시하는 데 필요한 요구 사항과 같은 공통 특징을 공유합니다. 빌드에서는 CPU 사용량, 메모리 사용량, 빌드 또는 Pod 실행 시간과 같은 리소스 제한 사항을 활용합니다.

OpenShift Container Platform 빌드 시스템은 빌드 API에서 지정한 선택 가능한 유형을 기반으로 하는 빌드 전략에 확장 가능한 지원을 제공합니다. 다음은 세 가지 주요 빌드 전략입니다.

- Docker 빌드
- S2I(Source-to-Image) 빌드
- 사용자 정의 빌드

기본적으로 Docker 빌드 및 S2I 빌드가 지원됩니다.

빌드의 결과 오브젝트는 빌드를 생성하는 데 사용된 빌더에 따라 다릅니다. Docker 및 S2I 빌드의 경우 결과 오브젝트는 실행 가능한 이미지입니다. 사용자 정의 빌드의 경우 결과 오브젝트는 빌더 이미지 작성자가 지정한 모든 항목입니다.

또한 파이프라인 빌드 전략을 사용하여 정교한 워크플로를 구현할 수 있습니다.

- 연속 통합
- 연속 배포

2.1.1.1. Docker 빌드

OpenShift Container Platform은 Buildah를 사용하여 Dockerfile에서 컨테이너 이미지를 빌드합니다. Dockerfile을 사용하여 컨테이너 이미지를 빌드하는 방법에 대한 자세한 내용은 [Dockerfile 참조 문서](#)를 참조하십시오.

작은 정보

buildArgs 배열을 사용하여 Docker 빌드 인수를 설정하는 경우 Dockerfile 참조 문서에서 [ARG 및 FROM 이 상호 작용하는 방법 이해](#)를 참조하십시오.

2.1.1.2. S2I(Source-to-Image) 빌드

S2I(Source-to-Image)는 재현 가능한 컨테이너 이미지를 빌드하는 툴입니다. 컨테이너 이미지에 애플리케이션 소스를 삽입하고 새 이미지를 어셈블하여 실행할 수 있는 이미지를 생성합니다. 새 이미지는 기본 이미지, 빌더, 빌드 소스를 통합하고 **buildah run** 명령과 함께 사용할 수 있습니다. S2I는 이전에 다운로드

한 종속 항목, 이전에 빌드한 아티팩트 등을 다시 사용하는 증분 빌드를 지원합니다.

2.1.1.3. 사용자 정의 빌드

사용자 정의 빌드 전략을 사용하면 개발자가 전체 빌드 프로세스를 담당하는 특정 빌더 이미지를 정의할 수 있습니다. 자체 빌더 이미지를 사용하면 빌드 프로세스를 사용자 정의할 수 있습니다.

사용자 정의 빌더 이미지는 빌드 프로세스 논리가 포함된 일반 컨테이너 이미지입니다(예: RPM 또는 기본 이미지 빌드).

사용자 정의 빌드는 높은 권한으로 실행되며 기본적으로 사용자에게 제공되지 않습니다. 클러스터 관리 권한이 있는 신뢰할 수 있는 사용자에게만 사용자 정의 빌드를 실행할 수 있는 권한을 부여해야 합니다.

2.1.1.4. 파이프라인 빌드



중요

파이프라인 빌드 전략은 OpenShift Container Platform 4에서 더 이상 사용되지 않습니다. 동등하고 향상된 기능은 Tekton 기반 OpenShift Container Platform Pipelines에 있습니다.

OpenShift Container Platform의 Jenkins 이미지는 완전히 지원되며 사용자는 Jenkins 사용 설명서를 따라 작업에 **jenkinsfile**을 정의하거나 소스 제어 관리 시스템에 저장해야 합니다.

파이프라인 빌드 전략을 사용하면 개발자가 Jenkins 파이프라인 플러그인에서 사용할 Jenkins 파이프라인을 정의할 수 있습니다. 다른 빌드 유형과 동일한 방식으로 OpenShift Container Platform에서 빌드를 시작, 모니터링, 관리할 수 있습니다.

파이프라인 워크플로는 빌드 구성에 직접 포함하거나 Git 리포지토리에 제공하는 방식으로 **jenkinsfile**에 정의하고 빌드 구성에서 참조합니다.

2.2. 빌드 구성 이해

다음 섹션에서는 빌드의 개념과 빌드 구성을 정의하고 사용 가능한 주요 빌드 전략을 간략히 설명합니다.

2.2.1. BuildConfigs

빌드 구성은 단일 빌드 정의와 새 빌드가 생성될 때의 트리거 세트를 나타냅니다. 빌드 구성은 **BuildConfig**에 의해 정의되는데 BuildConfig는 새 인스턴스를 생성하기 위해 API 서버에 대한 POST에 사용할 수 있는 REST 오브젝트입니다.

빌드 구성 또는 **BuildConfig**는 빌드 전략 및 하나 이상의 소스가 특징입니다. 전략에 따라 프로세스가 결정되고 소스에서는 입력을 제공합니다.

OpenShift Container Platform을 사용하여 애플리케이션을 생성하기 위해 선택하는 방법에 따라 **BuildConfig**는 일반적으로 웹 콘솔 또는 CLI를 사용하는 경우 자동으로 생성되며 언제든지 편집할 수 있습니다. **BuildConfig**를 구성하는 부분과 해당 부분의 사용 가능한 옵션을 이해하면 나중에 구성을 수동으로 변경할 때 도움이 될 수 있습니다.

다음 예제 **BuildConfig**에서는 컨테이너 이미지 태그 또는 소스 코드가 변경될 때마다 새 빌드를 생성합니다.

BuildConfig 오브젝트 정의

```

kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: "ruby-sample-build" ❶
spec:
  runPolicy: "Serial" ❷
  triggers: ❸
  -
    type: "GitHub"
    github:
      secret: "secret101"
  - type: "Generic"
    generic:
      secret: "secret101"
  -
    type: "ImageChange"
  source: ❹
  git:
    uri: "https://github.com/openshift/ruby-hello-world"
  strategy: ❺
  sourceStrategy:
    from:
      kind: "ImageStreamTag"
      name: "ruby-20-centos7:latest"
  output: ❻
  to:
    kind: "ImageStreamTag"
    name: "origin-ruby-sample:latest"
  postCommit: ❼
  script: "bundle exec rake test"

```

- ❶ 이 사양은 **ruby-sample-build**라는 새 **BuildConfig**를 생성합니다.
- ❷ **runPolicy** 필드는 이 빌드 구성에서 생성한 빌드를 동시에 실행할 수 있는지 여부를 제어합니다. 기본값은 **Serial**입니다. 즉 새 빌드가 동시에 실행되지 않고 순차적으로 실행됩니다.
- ❸ 트리거 목록을 지정할 수 있으며 이 경우 새 빌드가 생성될 수 있습니다.
- ❹ **source** 섹션은 빌드의 소스를 정의합니다. 소스 유형에 따라 기본 입력 소스가 결정되는데, 소스 유형은 코드 리포지토리 위치를 가리키는 **Git**, 인라인 Dockerfile에서 빌드하는 **Dockerfile** 또는 바이너리 페이로드를 허용하는 **Binary**일 수 있습니다. 한번에 여러 개의 소스를 사용할 수 있습니다. 각 소스 유형에 대한 자세한 내용은 "빌드 입력 생성"을 참조하십시오.
- ❺ **strategy** 섹션에서는 빌드를 실행하는 데 사용하는 빌드 전략에 대해 설명합니다. 여기에서 **Source**, **Docker** 또는 **Custom** 전략을 지정할 수 있습니다. 이 예에서는 S2I(Source-to-Image)에서 애플리케이션 빌드에 사용하는 **ruby-20-centos7** 컨테이너 이미지를 사용합니다.
- ❻ 컨테이너 이미지가 성공적으로 빌드되면 **output** 섹션에 설명된 리포지토리로 푸시됩니다.
- ❼ **postCommit** 섹션에서는 선택적 빌드 후크를 정의합니다.

2.3. 빌드 입력 생성

다음 섹션에서는 빌드 입력에 대한 개요, 입력을 사용하여 빌드가 작동하도록 소스 콘텐츠를 제공하는 방법, 빌드 환경을 사용하고 보안을 생성하는 방법에 대한 지침을 확인할 수 있습니다.

2.3.1. 빌드 입력

빌드 입력에서는 빌드가 작동하도록 소스 콘텐츠를 제공합니다. 다음 빌드 입력을 사용하여 OpenShift Container Platform에 소스를 제공할 수 있습니다(우선순위 순으로 표시됨).

- 인라인 Dockerfile 정의
- 기존 이미지에서 추출한 콘텐츠
- Git 리포지토리
- 바이너리(로컬) 입력
- 입력 보안
- 외부 아티팩트

단일 빌드에서 여러 입력을 결합할 수 있습니다. 그러나 인라인 Dockerfile이 우선하므로 다른 입력에서 제공하는 Dockerfile이라는 기타 파일을 덮어쓸 수 있습니다. 바이너리(local) 입력과 Git 리포지토리는 서로 함께 사용할 수 없는 입력입니다.

빌드 중 사용한 특정 리소스 또는 자격 증명을 빌드에서 생성한 최종 애플리케이션 이미지에 사용하지 않으려는 경우 또는 보안 리소스에 정의된 값을 사용하려는 경우에는 입력 보안을 사용하면 됩니다. 외부 아티팩트를 사용하면 기타 빌드 입력 유형 중 하나로 사용할 수 없는 추가 파일을 가져올 수 있습니다.

빌드를 실행하면 다음이 수행됩니다.

1. 작업 디렉터리가 구성되고 모든 입력 콘텐츠가 작업 디렉터리에 배치됩니다. 예를 들어 입력 Git 리포지토리가 작업 디렉터리에 복제되고 입력 이미지에서 지정된 파일이 타겟 경로를 사용하여 작업 디렉터리에 복사됩니다.
2. 빌드 프로세스에서는 **contextDir**이 정의된 경우 디렉터리를 해당 디렉터리로 변경합니다.
3. 인라인 Dockerfile(있는 경우)은 현재 디렉터리에 기록됩니다.
4. 현재 디렉터리의 콘텐츠는 Dockerfile, 사용자 지정 빌더 논리 또는 **assemble** 스크립트에서 참조할 수 있도록 빌드 프로세스에 제공됩니다. 즉 **contextDir** 외부에 있는 모든 입력 콘텐츠는 빌드에서 무시합니다.

다음 소스 정의 예제에는 다양한 입력 유형 및 이러한 유형이 결합되는 방법에 대한 설명이 포함되어 있습니다. 각 입력 유형이 정의되는 방법에 대한 자세한 내용은 각 입력 유형별 섹션을 참조하십시오.

```
source:
  git:
    uri: https://github.com/openshift/ruby-hello-world.git 1
    ref: "master"
  images:
  - from:
    kind: ImageStreamTag
    name: myinputimage:latest
    namespace: mynamespace
  paths:
  - destinationDir: app/dir/injected/dir 2
```

```
sourcePath: /usr/lib/somefile.jar
contextDir: "app/dir" ❸
dockerfile: "FROM centos:7\nRUN yum install -y httpd" ❹
```

- ❶ 빌드를 위한 작업 디렉터리에 복제할 리포지토리입니다.
- ❷ `myinputimage`의 `/usr/lib/somefile.jar`는 `<workingdir>/app/dir/injected/dir`에 저장됩니다.
- ❸ 빌드용 작업 디렉터리는 `<original_workingdir>/app/dir`입니다.
- ❹ 이 콘텐츠가 포함된 Dockerfile은 `<original_workingdir>/app/dir`에 생성되어 해당 이름의 기존 파일을 덮어씁니다.

2.3.2. Dockerfile 소스

`dockerfile` 값을 제공하면 이 필드의 콘텐츠가 `dockerfile`이라는 파일로 디스크에 작성됩니다. 이 작업은 다른 입력 소스를 처리한 후 수행되므로 입력 소스 리포지토리의 루트 디렉터리에 Dockerfile이 포함된 경우 해당 콘텐츠가 이를 덮어씁니다.

소스 정의는 `BuildConfig`에 있는 `spec` 섹션의 일부입니다.

```
source:
  dockerfile: "FROM centos:7\nRUN yum install -y httpd" ❶
```

- ❶ `dockerfile` 필드에는 빌드된 인라인 Dockerfile이 포함됩니다.

추가 리소스

- 이 필드는 일반적으로 Docker 전략 빌드에 Dockerfile을 제공하는 데 사용됩니다.

2.3.3. 이미지 소스

이미지가 포함된 빌드 프로세스에 파일을 추가할 수 있습니다. 입력 이미지는 `From` 및 `To` 이미지 타겟을 정의하는 방식과 동일한 방식으로 참조합니다. 즉 컨테이너 이미지와 이미지 스트림 태그를 모두 참조할 수 있습니다. 이미지와 함께 이미지를 복사할 파일 또는 디렉터리의 경로와 빌드 컨텍스트에서 해당 이미지를 배치할 대상을 나타내는 하나 이상의 경로 쌍을 제공해야 합니다.

소스 경로는 지정된 이미지 내의 모든 절대 경로일 수 있습니다. 대상은 상대 디렉터리 경로여야 합니다. 빌드 시 이미지가 로드되고 표시된 파일과 디렉터리가 빌드 프로세스의 컨텍스트 디렉터리로 복사됩니다. 이 디렉터리는 소스 리포지토리 콘텐츠가 복제되는 것과 동일한 디렉터리입니다. 소스 경로가 `/`로 종료되면 디렉터리의 콘텐츠가 복사되지만 디렉터리 자체는 대상에 생성되지 않습니다.

이미지 입력은 `BuildConfig`의 `source` 정의에 지정됩니다.

```
source:
  git:
    uri: https://github.com/openshift/ruby-hello-world.git
    ref: "master"
  images: ❶
  - from: ❷
    kind: ImageStreamTag
    name: myinputimage:latest
```

```

namespace: mynamespace
paths: ❸
- destinationDir: injected/dir ❹
  sourcePath: /usr/lib/somefile.jar ❺
- from:
  kind: ImageStreamTag
  name: myotherinputimage:latest
  namespace: myothernamespace
pullSecret: mysecret ❻
paths:
- destinationDir: injected/dir
  sourcePath: /usr/lib/somefile.jar

```

- ❶ 하나 이상의 입력 이미지 및 파일로 이루어진 배열입니다.
- ❷ 복사할 파일이 포함된 이미지에 대한 참조입니다.
- ❸ 소스/대상 경로로 이루어진 배열입니다.
- ❹ 빌드 프로세스에서 파일에 액세스할 수 있는, 빌드 루트의 상대 디렉터리입니다.
- ❺ 참조한 이미지에서 복사할 파일의 위치입니다.
- ❻ 입력 이미지에 액세스하는 데 자격 증명이 필요한 경우 제공되는 선택적 보안입니다.

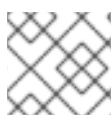


참고

클러스터에서 **ImageContentSourcePolicy** 오브젝트를 사용하여 저장소 미러링을 구성하는 경우 미러링된 레지스트리에 대한 글로벌 풀 시크릿만 사용할 수 있습니다. 프로젝트에 풀 시크릿을 추가할 수 없습니다.

입력 이미지에 가져오기 보안이 필요한 경우 선택적으로 빌드에서 사용하는 서비스 계정에 가져오기 보안을 연결할 수 있습니다. 기본적으로 빌드에서는 **builder** 서비스 계정을 사용합니다. 보안에 입력 이미지를 호스팅하는 리포지토리와 일치하는 자격 증명이 포함된 경우 가져오기 보안이 빌드에 자동으로 추가됩니다. 빌드에서 사용하는 서비스 계정에 가져오기 보안을 연결하려면 다음을 실행합니다.

```
$ oc secrets link builder dockerhub
```



참고

이 기능은 사용자 정의 전략을 사용하는 빌드에는 지원되지 않습니다.

2.3.4. Git 소스

지정하는 경우 입력한 위치에서 소스 코드를 가져옵니다.

인라인 Dockerfile을 제공하는 경우 Git 리포지토리의 **contextDir**에 Dockerfile을 덮어씁니다.

소스 정의는 **BuildConfig**에 있는 **spec** 섹션의 일부입니다.

```

source:
  git: ❶
  uri: "https://github.com/openshift/ruby-hello-world"

```

```
ref: "master"
contextDir: "app/dir" 2
dockerfile: "FROM openshift/ruby-22-centos7\nUSER example" 3
```

- 1 **git** 필드에는 소스 코드의 원격 Git 리포지토리에 대한 URI가 포함되어 있습니다. 필요한 경우 **ref** 필드를 지정하여 특정 Git 참조를 확인합니다. 유효한 **ref**는 SHA1 태그 또는 분기 이름이 될 수 있습니다.
- 2 **contextDir** 필드를 사용하면 빌드에서 애플리케이션 소스 코드를 찾는 소스 코드 리포지토리 내부의 기본 위치를 덮어쓸 수 있습니다. 애플리케이션이 하위 디렉터리에 있는 경우 이 필드를 사용하여 기본 위치(루트 폴더)를 덮어쓸 수 있습니다.
- 3 선택적 **dockerfile** 필드를 제공하는 경우 소스 리포지토리에 있을 수 있는 모든 Dockerfile을 덮어쓰는 Dockerfile이 문자열에 포함되어야 합니다.

ref 필드가 가져오기 요청을 나타내는 경우 시스템은 **git fetch** 작업을 사용한 후 **FETCH_HEAD**를 점검합니다.

ref 값을 제공하지 않으면 OpenShift Container Platform에서 부분 복제(**--depth=1**)를 수행합니다. 이 경우 기본 분기(일반적으로 **master**)의 최근 커밋과 관련된 파일만 다운로드됩니다. 그러면 리포지토리는 더 빠르게 다운로드되지만 전체 커밋 내역은 다운로드되지 않습니다. 지정된 리포지토리의 기본 분기에 대한 전체 **git clone**을 수행하려면 기본 분기(예: **master**)의 이름을 **ref**로 설정합니다.



주의

MITM(Man in the middle) TLS 하이재킹을 수행하거나 프록시 연결을 재암호화하고 있는 프록시를 통과하는 Git 복제 작업이 작동하지 않습니다.

2.3.4.1. 프록시 사용

프록시를 사용하는 경우에만 Git 리포지토리에 액세스할 수 있는 경우 빌드 구성의 **source** 섹션에 사용할 프록시를 정의할 수 있습니다. 사용할 HTTP 및 HTTPS 프록시를 둘 다 구성할 수 있습니다. 두 필드 모두 선택 사항입니다. 프록시를 사용할 수 없는 도메인도 **NoProxy** 필드에 지정할 수 있습니다.



참고

이를 위해서는 소스 URI에서 HTTP 또는 HTTPS 프로토콜을 사용해야 합니다.

```
source:
git:
  uri: "https://github.com/openshift/ruby-hello-world"
  ref: "master"
httpProxy: http://proxy.example.com
httpsProxy: https://proxy.example.com
noProxy: somedomain.com, otherdomain.com
```



참고

Pipeline 전략 빌드의 경우 Jenkins용 Git 플러그인에 대한 현재 제한 사항을 고려할 때 Git 플러그인을 통한 모든 Git 작업은 **BuildConfig**에 정의된 HTTP 또는 HTTPS 프록시를 사용하지 않습니다. Git 플러그인은 플러그인 관리자 패널에서 Jenkins UI에 구성된 프록시만 사용합니다. 그런 다음 이 프록시는 모든 작업에서 Jenkins 내의 모든 Git 상호 작용에 사용됩니다.

추가 리소스

- [JenkinsBehindProxy](#)에서 Jenkins UI를 통해 프록시를 구성하는 방법에 대한 지침을 확인할 수 있습니다.

2.3.4.2. 소스 복제 보안

빌더 Pod는 빌드의 소스로 정의된 모든 Git 리포지토리에 액세스해야 합니다. 소스 복제 보안은 자체 서명되거나 신뢰할 수 없는 SSL 인증서가 있는 프라이빗 리포지토리 또는 리포지토리나 같이 일반적으로 액세스할 수 없는 리포지토리에 대한 액세스 권한을 제공하는 데 사용됩니다.

다음과 같은 소스 복제 보안 구성이 지원됩니다.

- .gitconfig 파일
- 기본 인증
- SSH 키 인증
- 신뢰할 수 있는 인증 기관



참고

이러한 구성의 조합을 사용하여 특정 요구 사항을 충족할 수도 있습니다.

2.3.4.2.1. 빌드 구성에 소스 복제 보안 자동 추가

BuildConfig가 생성되면 OpenShift Container Platform에서 소스 복제 보안 참조를 자동으로 채울 수 있습니다. 이 동작을 사용하면 생성된 빌드에서 참조된 보안에 저장된 자격 증명을 자동으로 사용하여 추가 구성없이 원격 Git 리포지토리에 인증할 수 있습니다.

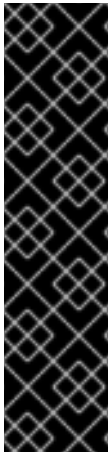
이 기능을 사용하려면 Git 리포지토리 자격 증명에 포함된 보안이 나중에 **BuildConfig**가 생성되는 네임스페이스에 있어야 합니다. 이 보안에는 **build.openshift.io/source-secret-match-uri-** 접두사가 있는 주석이 하나 이상 포함되어야 합니다. 이러한 주석의 각 값은 다음과 같이 정의되는 URI(Uniform Resource Identifier) 패턴입니다. 소스 복제 보안 참조 없이 **BuildConfig**를 생성하고 해당 Git 소스 URI가 보안 주석의 URI 패턴과 일치하는 경우 OpenShift Container Platform은 **BuildConfig**에 해당 보안에 대한 참조를 자동으로 삽입합니다.

사전 요구 사항

URI 패턴은 다음으로 구성되어야 합니다.

- 유효 스키마: ***://, git://, http://, https://** 또는 **ssh://**
- 호스트: ***** 또는 유효한 호스트 이름이나 필요한 경우 앞에 *****가 있는 IP 주소
- 경로: **/*** 또는 **/** 뒤에 ***** 문자를 선택적으로 포함하는 모든 문자

위의 모든 항목에서 * 문자는 와일드카드로 해석됩니다.



중요

URI 패턴은 [RFC3986](#)을 준수하는 Git 소스 URI와 일치해야 합니다. URI 패턴에 사용자 이름(또는 암호) 구성 요소를 포함하지 마십시오.

예를 들어 Git 리포지토리 URL에 **ssh://git@bitbucket.atlassian.com:7999/ATLASSIAN/jira.git**을 사용하는 경우 소스 보안을 **ssh://bitbucket.atlassian.com:7999/(ssh://git@bitbucket.atlassian.com:7999/* 아님)**로 지정해야 합니다.

```
$ oc annotate secret mysecret \
    'build.openshift.io/source-secret-match-uri-1=ssh://bitbucket.atlassian.com:7999/*'
```

프로세스

특정 **BuildConfig**의 Git URI와 일치하는 보안이 여러 개인 경우 OpenShift Container Platform은 가장 긴 일치 항목이 있는 보안을 선택합니다. 이렇게 하면 다음 예와 같이 기본 덮어쓰기가 가능합니다.

다음 조각은 두 개의 부분적인 소스 복제 보안을 보여줍니다. 첫 번째는 HTTPS를 통해 액세스하는 도메인 **mycorp.com**의 모든 서버와 일치하고, 두 번째는 서버 **mydev1.mycorp.com** 및 **mydev2.mycorp.com**에 대한 액세스 권한을 덮어씁니다.

```
kind: Secret
apiVersion: v1
metadata:
  name: matches-all-corporate-servers-https-only
  annotations:
    build.openshift.io/source-secret-match-uri-1: https://*.mycorp.com/*
data:
  ...
---
kind: Secret
apiVersion: v1
metadata:
  name: override-for-my-dev-servers-https-only
  annotations:
    build.openshift.io/source-secret-match-uri-1: https://mydev1.mycorp.com/*
    build.openshift.io/source-secret-match-uri-2: https://mydev2.mycorp.com/*
data:
  ...
```

- 다음을 사용하여 기존 보안에 **build.openshift.io/source-secret-match-uri-** 주석을 추가합니다.

```
$ oc annotate secret mysecret \
    'build.openshift.io/source-secret-match-uri-1=https://*.mycorp.com/*'
```

2.3.4.2.2. 수동으로 소스 복제 보안 추가

소스 복제 보안은 **BuildConfig** 내부의 **source** 필드에 **sourceSecret** 필드를 추가한 후 사용자가 생성한 보안의 이름으로 설정하는 방식으로 빌드 구성에 수동으로 추가할 수 있습니다. 다음 예에서는 **basicsecret**입니다.


```

apiVersion: "v1"
kind: "BuildConfig"
metadata:
  name: "sample-build"
spec:
  output:
    to:
      kind: "ImageStreamTag"
      name: "sample-image:latest"
  source:
    git:
      uri: "https://github.com/user/app.git"
    sourceSecret:
      name: "basicsecret"
  strategy:
    sourceStrategy:
      from:
        kind: "ImageStreamTag"
        name: "python-33-centos7:latest"

```

프로세스

oc set build-secret 명령을 사용하여 기존 빌드 구성에 소스 복제 보안을 설정할 수도 있습니다.

- 기존 빌드 구성에 소스 복제 보안을 설정하려면 다음 명령을 입력합니다.

```
$ oc set build-secret --source bc/sample-build basicsecret
```

2.3.4.2.3. .gitconfig 파일에서 보안 생성

애플리케이션 복제에서 **.gitconfig** 파일을 사용하는 경우 이 파일을 포함하는 보안을 생성할 수 있습니다. 빌더 서비스 계정에 추가한 다음 **BuildConfig**에 추가합니다.

프로세스

- .gitconfig** 파일에서 보안을 생성하려면 다음을 수행합니다.

```
$ oc create secret generic <secret_name> --from-file=<path/to/.gitconfig>
```



참고

.gitconfig 파일의 **http** 섹션에 **sslVerify=false**가 설정되어 있는 경우 SSL 확인을 해제할 수 있습니다.

```
[http]
sslVerify=false
```

2.3.4.2.4. 보안 Git의 .gitconfig 파일에서 보안 생성

Git 서버가 양방향 SSL과 사용자 이름 및 암호로 보호되는 경우 소스 빌드에 인증서 파일을 추가하고 **.gitconfig** 파일의 인증서 파일에 참조를 추가해야 합니다.

사전 요구 사항

- Git 자격 증명이 있어야 합니다.

프로세스

소스 빌드에 인증서 파일을 추가하고 **.gitconfig** 파일의 인증서 파일에 대한 참조를 추가합니다.

1. 애플리케이션 소스 코드의 **/var/run/secrets/openshift.io/source/** 폴더에 **client.crt, cacert.crt, client.key** 파일을 추가합니다.
2. 서버의 **.gitconfig** 파일에서 다음 예에 표시된 **[http]** 섹션을 추가합니다.

```
# cat .gitconfig
```

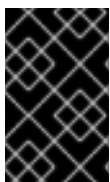
출력 예

```
[user]
  name = <name>
  email = <email>
[http]
  sslVerify = false
  sslCert = /var/run/secrets/openshift.io/source/client.crt
  sslKey = /var/run/secrets/openshift.io/source/client.key
  sslCaInfo = /var/run/secrets/openshift.io/source/cacert.crt
```

3. 보안을 생성합니다.

```
$ oc create secret generic <secret_name> \
--from-literal=username=<user_name> \ 1
--from-literal=password=<password> \ 2
--from-file=.gitconfig=.gitconfig \
--from-file=client.crt=/var/run/secrets/openshift.io/source/client.crt \
--from-file=cacert.crt=/var/run/secrets/openshift.io/source/cacert.crt \
--from-file=client.key=/var/run/secrets/openshift.io/source/client.key
```

- 1 사용자의 Git 사용자 이름입니다.
- 2 이 사용자의 암호입니다.



중요

암호를 다시 입력하지 않으려면 빌드에서 S2I(Source-to-Image) 이미지를 지정해야 합니다. 그러나 리포지토리를 복제할 수 없는 경우 빌드를 승격하려면 사용자 이름과 암호를 계속 지정해야 합니다.

추가 리소스

- 애플리케이션 소스 코드의 **/var/run/secrets/openshift.io/source/** 폴더입니다.

2.3.4.2.5. 소스 코드 기본 인증에서 보안 생성

기본 인증에는 **--username** 및 **--password**의 조합 또는 SCM(소프트웨어 구성 관리) 서버에 대해 인증하는 토큰이 필요합니다.

사전 요구 사항

- 개인 리포지토리에 액세스할 수 있는 사용자 이름 및 암호입니다.

프로세스

1. 먼저 보안을 생성한 후 **--username** 및 **--password**를 사용하여 개인 리포지토리에 액세스합니다.

```
$ oc create secret generic <secret_name> \
  --from-literal=username=<user_name> \
  --from-literal=password=<password> \
  --type=kubernetes.io/basic-auth
```

2. 토큰을 사용하여 기본 인증 보안을 생성합니다.

```
$ oc create secret generic <secret_name> \
  --from-literal=password=<token> \
  --type=kubernetes.io/basic-auth
```

2.3.4.2.6. 소스 코드 SSH 키 인증에서 보안 생성

SSH 키 기반 인증에는 개인 SSH 키가 필요합니다.

리포지토리 키는 일반적으로 **\$HOME/.ssh/** 디렉터리에 있으며 기본적으로 이름이 **id_dsa.pub**, **id_ecdsa.pub**, **id_ed25519.pub** 또는 **id_rsa.pub**입니다.

프로세스

1. SSH 키 자격 증명을 생성합니다.

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"
```



참고

SSH 키의 암호를 생성하면 OpenShift Container Platform이 빌드되지 않습니다. 암호를 입력하라는 메시지가 표시되면 비워 두십시오.

두 파일(공개 키 및 해당 개인 키(**id_dsa**, **id_ecdsa**, **id_ed25519** 또는 **id_rsa**))이 생성됩니다. 두 파일이 모두 있는 상태에서 공개 키를 업로드하는 방법에 대한 SCM(소스 제어 관리) 시스템의 설명서를 참조하십시오. 개인 키는 개인 리포지토리에 액세스하는 데 사용됩니다.

2. SSH 키를 사용하여 개인 리포지토리에 액세스하기 전에 보안을 생성합니다.

```
$ oc create secret generic <secret_name> \
  --from-file=ssh-privatekey=<path/to/ssh/private/key> \
  --from-file=<path/to/known_hosts> \ 1
  --type=kubernetes.io/ssh-auth
```

- 1** 선택 사항: 이 필드를 추가하면 엄격한 서버 호스트 키 점검이 가능합니다.



주의

시크릿을 생성하는 동안 **known_hosts** 파일을 건너뛰면 잠재적인 MIT(man-in-the-middle) 공격에 빌드가 취약해집니다.



참고

known_hosts 파일에 소스 코드 호스트의 항목이 포함되어 있는지 확인합니다.

2.3.4.2.7. 신뢰할 수 있는 소스 코드 인증 기관에서 보안 생성

Git 복제 작업 중 신뢰할 수 있는 일련의 TLS(Transport Layer Security) CA(인증 기관)가 OpenShift Container Platform 인프라 이미지로 빌드됩니다. Git 서버에서 자체 서명된 인증서 또는 이미지에서 신뢰할 수 없는 인증 기관에서 서명한 인증서를 사용하는 경우 인증서가 포함된 보안을 생성하거나 TLS 확인을 비활성화할 수 있습니다.

CA 인증서에 대한 보안을 생성하는 경우 OpenShift Container Platform에서는 Git 복제 작업 중 Git 서버에 액세스합니다. 이 방법을 사용하면 제공되는 모든 TLS 인증서를 허용하는 Git SSL 확인을 비활성화하는 것보다 더 안전합니다.

프로세스

CA 인증서 파일을 사용하여 보안을 생성합니다.

1. CA에서 중간 인증 기관을 사용하는 경우 **ca.crt** 파일의 모든 CA 인증서를 결합합니다. 다음 명령을 실행합니다.

```
$ cat intermediateCA.crt intermediateCA.crt rootCA.crt > ca.crt
```

- a. 보안을 생성합니다.

```
$ oc create secret generic mycert --from-file=ca.crt=</path/to/file> 1
```

- 1** 키 이름으로 **ca.crt**를 사용해야 합니다.

2.3.4.2.8. 소스 보안 조합

특정 요구 사항에 맞는 소스 복제 보안을 생성하기 위해 다양한 방법을 결합할 수 있습니다.

2.3.4.2.8.1. .gitconfig 파일을 사용하여 SSH 기반 인증 보안 생성

다양한 방법을 결합하여 특정 요구 사항에 맞는 소스 복제 보안을 생성할 수 있습니다(예: **.gitconfig** 파일을 사용하는 SSH 기반 인증 보안).

사전 요구 사항

- SSH 인증
- .gitconfig 파일

프로세스

- **.gitconfig** 파일을 사용하여 SSH 기반 인증 보안을 생성하려면 다음을 실행합니다.

```
$ oc create secret generic <secret_name> \
  --from-file=ssh-privatekey=<path/to/ssh/private/key> \
  --from-file=<path/to/.gitconfig> \
  --type=kubernetes.io/ssh-auth
```

2.3.4.2.8.2. .gitconfig 파일 및 CA 인증서를 결합하는 보안 생성

다양한 방법을 결합하여 특정 요구 사항에 맞는 소스 복제 보안을 생성할 수 있습니다(예: **.gitconfig** 파일 및 CA(인증 기관) 인증서를 결합하는 보안).

사전 요구 사항

- .gitconfig 파일
- CA 인증서

프로세스

- **.gitconfig** 파일 및 CA 인증서를 결합하는 보안을 생성하려면 다음을 실행합니다.

```
$ oc create secret generic <secret_name> \
  --from-file=ca.crt=<path/to/certificate> \
  --from-file=<path/to/.gitconfig>
```

2.3.4.2.8.3. CA 인증서를 사용하여 기본 인증 보안 생성

다양한 방법을 결합하여 특정 요구 사항에 맞는 소스 복제 보안을 생성할 수 있습니다(예: 기본 인증 및 CA(인증 기관) 인증서를 결합하는 보안).

사전 요구 사항

- 기본 인증 자격 증명
- CA 인증서

프로세스

- CA 인증서로 기본 인증 보안을 생성하려면 다음을 실행합니다.

```
$ oc create secret generic <secret_name> \
  --from-literal=username=<user_name> \
  --from-literal=password=<password> \
  --from-file=ca-cert=</path/to/file> \
  --type=kubernetes.io/basic-auth
```

2.3.4.2.8.4. .gitconfig 파일을 사용하여 기본 인증 보안 생성

다양한 방법을 결합하여 특정 요구 사항에 맞는 소스 복제 보안을 생성할 수 있습니다(예: 기본 인증 및 **.gitconfig** 파일을 결합하는 보안).

사전 요구 사항

- 기본 인증 자격 증명
- **.gitconfig** 파일

프로세스

- **.gitconfig** 파일을 사용하여 기본 인증 보안을 생성하려면 다음을 실행합니다.

```
$ oc create secret generic <secret_name> \
  --from-literal=username=<user_name> \
  --from-literal=password=<password> \
  --from-file=</path/to/.gitconfig> \
  --type=kubernetes.io/basic-auth
```

2.3.4.2.8.5. .gitconfig 파일 및 CA 인증서를 사용하여 기본 인증 보안 생성

다양한 방법을 결합하여 특정 요구 사항에 맞는 소스 복제 보안을 생성할 수 있습니다(예: 기본 인증, **.gitconfig** 파일, CA(인증 기관) 인증서를 결합하는 보안).

사전 요구 사항

- 기본 인증 자격 증명
- **.gitconfig** 파일
- CA 인증서

프로세스

- **.gitconfig** 파일 및 CA 인증서를 사용하여 기본 인증 보안을 생성하려면 다음을 실행합니다.

```
$ oc create secret generic <secret_name> \
  --from-literal=username=<user_name> \
  --from-literal=password=<password> \
  --from-file=</path/to/.gitconfig> \
  --from-file=ca-cert=</path/to/file> \
  --type=kubernetes.io/basic-auth
```

2.3.5. 바이너리(로컬) 소스

로컬 파일 시스템의 콘텐츠를 빌더로 스트리밍하는 것을 **Binary** 빌드라고 합니다. 이러한 빌드의 경우 **BuildConfig.spec.source.type**의 해당 값이 **Binary**입니다.

이 소스 유형은 **oc start-build**를 사용할 때만 활용하므로 고유합니다.



참고

바이너리 유형 빌드에서는 로컬 파일 시스템의 콘텐츠를 스트리밍해야 하므로 이미지 변경 트리거와 같이 바이너리 유형 빌드를 자동으로 트리거할 수 없습니다. 바이너리 파일을 제공할 수 없기 때문입니다. 마찬가지로 웹 콘솔에서 바이너리 유형 빌드를 시작할 수 없습니다.

바이너리 빌드를 사용하려면 다음 옵션 중 하나를 사용하여 **oc start-build**를 호출합니다.

- **--from-file**: 지정한 파일의 콘텐츠는 빌더에 바이너리 스트림으로 전송됩니다. 파일에 URL을 지정할 수도 있습니다. 그러면 빌더에서 빌드 컨텍스트 상단에 있는 것과 동일한 이름으로 파일에 데이터를 저장합니다.
- **--from-dir** 및 **--from-repo**: 콘텐츠는 보관되어 빌더에 바이너리 스트림으로 전송됩니다. 그러면 빌더가 빌드 컨텍스트 디렉터리 내에서 아카이브 콘텐츠를 추출합니다. **--from-dir**을 사용하면 추출된 아카이브에 URL을 지정할 수도 있습니다.
- **--from-archive**: 지정한 아카이브는 빌더로 전송되며 빌드 컨텍스트 디렉터리 내에서 추출됩니다. 이 옵션은 **--from-dir**과 동일하게 작동합니다. 이러한 옵션에 대한 인수가 디렉터리인 경우 먼저 호스트에서 아카이브가 생성됩니다.

위에 나열된 각 사례에서 다음을 수행합니다.

- **BuildConfig**에 이미 **Binary** 소스 유형이 정의되어 있는 경우 효과적으로 무시되고 클라이언트에서 전송하는 내용으로 교체됩니다.
- **BuildConfig**에 **Git** 소스 유형이 정의되어 있는 경우 **Binary** 및 **Git**을 함께 사용할 수 없으므로 해당 **BuildConfig**가 동적으로 비활성화되고 빌더에 제공하는 바이너리 스트림의 데이터에 우선순위가 지정됩니다.

HTTP 또는 HTTPS 스키마를 사용하여 파일 이름 대신 URL을 **--from-file** 및 **--from-archive**로 전달할 수 있습니다. URL과 함께 **--from-file**을 사용하는 경우 빌더 이미지의 파일 이름은 웹 서버에서 전송한 **Content-Disposition** 헤더 또는 헤더가 없는 경우 URL 경로의 마지막 구성 요소에 따라 결정됩니다. 지원되는 인증 형식이 없는 경우 사용자 정의 TLS 인증서를 사용하거나 인증서 검증 작업을 비활성화할 수 없습니다.

oc new-build --binary=true를 사용하면 명령에서 바이너리 빌드와 관련된 제한을 적용합니다. 생성된 **BuildConfig**의 소스 유형이 **Binary**이므로 이 **BuildConfig**로 빌드를 실행하는 유일한 방법은 **--from** 옵션 중 하나와 함께 **oc start-build**를 사용하여 필수 바이너리 데이터를 제공하는 것입니다.

Dockerfile 및 **contextDir** 소스 옵션에는 바이너리 빌드에서 특별한 의미가 있습니다.

Dockerfile은 바이너리 빌드 소스와 함께 사용할 수 있습니다. Dockerfile을 사용하고 바이너리 스트림이 아카이브인 경우 해당 콘텐츠는 아카이브의 모든 Dockerfile에 대한 대체 Dockerfile 역할을 합니다. Dockerfile을 **--from-file** 인수와 함께 사용하고 파일 인수의 이름이 Dockerfile인 경우 Dockerfile의 값이 바이너리 스트림의 값을 대체합니다.

추출된 아카이브 콘텐츠를 캡슐화하는 바이너리 스트림의 경우 **contextDir** 필드의 값이 아카이브 내 하위 디렉터리로 해석되고, 유효한 경우 빌드를 실행하기 전에 빌더가 해당 하위 디렉터리로 변경됩니다.

2.3.6. 입력 보안 및 구성 맵

일부 시나리오에서는 빌드 작업을 수행하려면 종속 리소스에 액세스하기 위해 자격 증명 또는 기타 구성 데이터가 필요합니다. 그러나 이러한 정보가 소스 제어에 배치되는 것은 바람직하지 않습니다. 이러한 용도를 위해 입력 보안 및 입력 구성 맵을 정의할 수 있습니다.

예를 들어 Maven으로 Java 애플리케이션을 빌드할 때 개인 키로 액세스할 수 있는 Maven Central 또는 JCenter의 개인 미러를 설정할 수 있습니다. 해당 개인 미러에서 라이브러리를 다운로드하려면 다음을 제공해야 합니다.

1. 미러의 URL 및 연결 설정으로 구성된 **settings.xml** 파일
2. 설정 파일에서 참조하는 개인 키(예: `~/.ssh/id_rsa`)

보안상의 이유로 애플리케이션 이미지에 자격 증명을 노출해서는 안 됩니다.

이 예제에서는 Java 애플리케이션을 설명하지만 **/etc/ssl/certs** 디렉터리, API 키 또는 토큰, 라이선스 파일 등에 SSL 인증서를 추가하는 데 동일한 접근 방식을 사용할 수 있습니다.

2.3.6.1. 비밀이란?

Secret 오브젝트 유형에서는 암호, OpenShift Container Platform 클라이언트 구성 파일, **dockercfg** 파일, 개인 소스 리포지토리 자격 증명 등과 같은 중요한 정보를 보유하는 메커니즘을 제공합니다. 보안은 Pod에서 중요한 콘텐츠를 분리합니다. 볼륨 플러그인을 사용하여 컨테이너에 보안을 마운트하거나 시스템에서 시크릿을 사용하여 Pod 대신 작업을 수행할 수 있습니다.

YAML 보안 오브젝트 정의

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
  namespace: my-namespace
type: Opaque ①
data: ②
  username: dmFsdWUtMQ0K ③
  password: dmFsdWUtMg0KDQo=
stringData: ④
  hostname: myapp.mydomain.com ⑤
```

- ① 보안의 키 이름과 값의 구조를 나타냅니다.
- ② **data** 필드에서 허용되는 키 형식은 Kubernetes 구분자 용어집의 **DNS_SUBDOMAIN** 값에 있는 지침을 충족해야 합니다.
- ③ **data** 맵의 키와 관련된 값은 base64로 인코딩되어야 합니다.
- ④ **stringData** 맵의 항목이 base64로 변환된 후 해당 항목이 자동으로 **data** 맵으로 이동합니다. 이 필드는 쓰기 전용입니다. 해당 값은 **data** 필드에서만 반환합니다.
- ⑤ **stringData** 맵의 키와 관련된 값은 일반 텍스트 문자열로 구성됩니다.

2.3.6.1.1. 보안 속성

주요 속성은 다음과 같습니다.

- 보안 데이터는 정의와는 별도로 참조할 수 있습니다.
- 보안 데이터 볼륨은 임시 파일 저장 기능(tmpfs)에 의해 지원되며 노드에 저장되지 않습니다.
- 보안 데이터는 네임스페이스 내에서 공유할 수 있습니다.

2.3.6.1.2. 보안 유형

type 필드의 값은 보안의 키 이름과 값의 구조를 나타냅니다. 유형을 사용하면 보안 오브젝트에 사용자 이름과 키를 적용할 수 있습니다. 검증을 수행하지 않으려면 기본값인 **opaque** 유형을 사용합니다.

보안 데이터에 특정 키 이름이 있는지 확인하기 위해 서버 측 최소 검증을 트리거하려면 다음 유형 중 하나를 지정합니다.

- **kubernetes.io/service-account-token**. 서비스 계정 토큰을 사용합니다.
- **kubernetes.io/dockercfg**. 필수 Docker 자격 증명으로 **.dockercfg** 파일을 사용합니다.
- **kubernetes.io/dockerconfigjson**. 필수 Docker 자격 증명으로 **.docker/config.json** 파일을 사용합니다.
- **kubernetes.io/basic-auth**. 기본 인증에 사용합니다.
- **kubernetes.io/ssh-auth**. SSH 키 인증에 사용합니다.
- **kubernetes.io/tls**. TLS 인증 기관에 사용합니다.

검증을 수행하지 않으려면 **type= Opaque**를 지정합니다. 즉 보안에서 키 이름 또는 값에 대한 규칙을 준수하도록 요청하지 않습니다. **opaque** 보안에는 임의의 값을 포함할 수 있는 비정형 **key:value** 쌍을 사용할 수 있습니다.



참고

example.com/my-secret-type과 같은 다른 임의의 유형을 지정할 수 있습니다. 이러한 유형은 서버 측에 적용되지 않지만 보안 생성자가 해당 유형의 키/값 요구 사항을 준수하도록 의도했음을 나타냅니다.

2.3.6.1.3. 보안 업데이트

보안 값을 수정해도 이미 실행 중인 Pod에서 사용하는 값은 동적으로 변경되지 않습니다. 보안을 변경하려면 동일한 **PodSpec**을 사용하는 일부 경우에서 원래 Pod를 삭제하고 새 Pod를 생성해야 합니다.

보안 업데이트 작업에서는 새 컨테이너 이미지를 배포하는 것과 동일한 워크플로를 따릅니다. **kubectl rolling-update** 명령을 사용할 수 있습니다.

보안의 **resourceVersion** 값을 참조 시 지정되지 않습니다. 따라서 Pod가 시작되는 동시에 보안이 업데이트되는 경우 Pod에 사용되는 보안의 버전이 정의되지 않습니다.



참고

현재는 Pod가 생성될 때 사용된 보안 오브젝트의 리소스 버전을 확인할 수 없습니다. 컨트롤러에서 이전 **resourceVersion**을 사용하여 재시작할 수 있도록 Pod에서 이 정보를 보고하도록 계획되어 있습니다. 그동안 기존 보안 데이터를 업데이트하지 말고 고유한 이름으로 새 보안을 생성하십시오.

2.3.6.2. 보안 생성

먼저 보안을 생성한 후 해당 보안을 사용하는 Pod를 생성해야 합니다.

보안 생성 시 다음을 수행합니다.

- 보안 데이터를 사용하여 보안 오브젝트를 생성합니다.
- Pod 서비스 계정을 업데이트하여 보안에 대한 참조를 허용합니다.
- 보안을 환경 변수로 사용하거나 **secret** 볼륨을 사용하여 파일로 사용하는 Pod를 생성합니다.


```
$ oc create -f <your_yaml_file>.yaml
```

- 로그를 가져옵니다.

```
$ oc logs secret-example-pod
```

- Pod를 삭제합니다.

```
$ oc delete pod secret-example-pod
```

추가 리소스

- 보안 데이터가 있는 YAML 파일의 예:

파일 4개를 생성할 YAML 보안

```
apiVersion: v1
kind: Secret
metadata:
  name: test-secret
data:
  username: dmFsdWUtMQ0K 1
  password: dmFsdWUtMQ0KDQo= 2
stringData:
  hostname: myapp.mydomain.com 3
secret.properties: |- 4
  property1=valueA
  property2=valueB
```

- 1** 파일에 디코딩된 값이 포함되어 있습니다.
- 2** 파일에 디코딩된 값이 포함되어 있습니다.
- 3** 파일에 제공된 문자열이 포함되어 있습니다.
- 4** 파일에 제공된 데이터가 포함되어 있습니다.

보안 데이터로 볼륨의 파일을 채우는 Pod의 YAML

```
apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  containers:
    - name: secret-test-container
      image: busybox
      command: [ "/bin/sh", "-c", "cat /etc/secret-volume/*" ]
      volumeMounts:
        # name must match the volume name below
        - name: secret-volume
          mountPath: /etc/secret-volume
```

```

    readOnly: true
  volumes:
  - name: secret-volume
    secret:
      secretName: test-secret
  restartPolicy: Never

```

보안 데이터로 환경 변수를 채우는 Pod의 YAML

```

apiVersion: v1
kind: Pod
metadata:
  name: secret-example-pod
spec:
  containers:
  - name: secret-test-container
    image: busybox
    command: [ "/bin/sh", "-c", "export" ]
    env:
    - name: TEST_SECRET_USERNAME_ENV_VAR
      valueFrom:
        secretKeyRef:
          name: test-secret
          key: username
  restartPolicy: Never

```

보안 데이터로 환경 변수를 채우는 빌드 구성의 YAML

```

apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: secret-example-bc
spec:
  strategy:
    sourceStrategy:
      env:
      - name: TEST_SECRET_USERNAME_ENV_VAR
        valueFrom:
          secretKeyRef:
            name: test-secret
            key: username

```

2.3.6.4. 입력 보안 및 구성 맵 추가

일부 시나리오에서는 빌드 작업을 수행하려면 종속 리소스에 액세스하기 위해 자격 증명 또는 기타 구성 데이터가 필요합니다. 그러나 이러한 정보가 소스 제어에 배치되는 것은 바람직하지 않습니다. 이러한 용도를 위해 입력 보안 및 입력 구성 맵을 정의할 수 있습니다.

프로세스

입력 보안이나 구성 맵 또는 둘 다를 기존 **BuildConfig** 오브젝트에 추가하려면 다음을 수행합니다.

1. **ConfigMap** 오브젝트가 없는 경우 해당 오브젝트를 생성합니다.

```
$ oc create configmap settings-mvn \
  --from-file=settings.xml=<path/to/settings.xml>
```

그러면 **settings-mvn**이라는 새 구성 맵이 생성됩니다. 이 맵에는 **settings.xml** 파일의 일반 텍스트 내용이 포함됩니다.

2. **Secret** 오브젝트가 없는 경우 해당 오브젝트를 생성합니다.

```
$ oc create secret generic secret-mvn \
  --from-file=id_rsa=<path/to/.ssh/id_rsa>
```

그러면 **secret-mvn**이라는 새 보안이 생성됩니다. 이 보안에는 **id_rsa** 개인 키의 base64 인코딩 콘텐츠가 포함됩니다.

3. 다음과 같이 기존 **BuildConfig** 오브젝트의 **source** 섹션에 구성 맵과 보안을 추가합니다.

```
source:
  git:
    uri: https://github.com/wildfly/quickstart.git
  contextDir: helloworld
  configMaps:
  - configMap:
    name: settings-mvn
  secrets:
  - secret:
    name: secret-mvn
```

새 **BuildConfig** 오브젝트에 구성 맵과 보안을 포함하려면 다음 명령을 실행합니다.

```
$ oc new-build \
  openshift/wildfly-101-centos7~https://github.com/wildfly/quickstart.git \
  --context-dir helloworld --build-secret "secret-mvn" \
  --build-config-map "settings-mvn"
```

빌드 중 **settings.xml** 및 **id_rsa** 파일이 소스 코드가 있는 디렉터리로 복사됩니다. OpenShift Container Platform S2I 빌더 이미지의 경우 이 디렉터리는 **Dockerfile**에 **WORKDIR** 명령을 사용하여 설정하는 이미지 작업 디렉터리입니다. 다른 디렉터를 지정하려면 정의에 **destinationDir**을 추가합니다.

```
source:
  git:
    uri: https://github.com/wildfly/quickstart.git
  contextDir: helloworld
  configMaps:
  - configMap:
    name: settings-mvn
    destinationDir: ".m2"
  secrets:
  - secret:
    name: secret-mvn
    destinationDir: ".ssh"
```

새 **BuildConfig** 오브젝트를 생성할 때 대상 디렉터를 지정할 수도 있습니다.

```
$ oc new-build \
```

```
openshift/wildfly-101-centos7~https://github.com/wildfly/quickstart.git \
--context-dir helloworld --build-secret "secret-mvn:.ssh" \
--build-config-map "settings-mvn:.m2"
```

두 경우 모두 **settings.xml** 파일은 빌드 환경의 **./m2** 디렉터리에 추가되고 **id_rsa** 키는 **./ssh** 디렉터리에 추가됩니다.

2.3.6.5. S2I(Source-to-Image) 전략

Source 전략을 사용하면 정의된 모든 입력 보안이 해당 **destinationDir**에 복사됩니다. **destinationDir**을 비워 두면 보안이 빌더 이미지의 작업 디렉터리에 배치됩니다.

destinationDir이 상대 경로인 경우 동일한 규칙이 사용됩니다. 보안은 이미지 작업 디렉터리의 상대 경로에 배치됩니다. **destinationDir** 경로의 최종 디렉터리가 빌더 이미지에 없는 경우 해당 디렉터리가 생성됩니다. **destinationDir**의 선행 디렉터리가 모두 존재해야 합니다. 없는 경우 오류가 발생합니다.



참고

입력 보안은 전역 쓰기 가능으로 추가되고 **0666** 권한이 있으며 **assemble** 스크립트 실행 후 크기가 0으로 잘립니다. 즉 보안 파일은 결과 이미지에 존재하지만 보안상의 이유로 비어 있습니다.

assemble 스크립트가 완료되면 입력 구성 맵이 잘리지 않습니다.

2.3.6.6. Docker 전략

docker 전략을 사용하는 경우 **Dockerfile**의 **ADD** 및 **COPY** 명령을 사용하여 정의된 모든 입력 보안을 컨테이너 이미지에 추가할 수 있습니다.

보안의 **destinationDir**을 지정하지 않으면 **Dockerfile**이 있는 동일한 디렉터리로 파일이 복사됩니다. 상대 경로를 **destinationDir**로 지정하면 보안이 **Dockerfile** 위치와 상대되는 해당 디렉터리에 복사됩니다. 그러면 **Docker** 빌드 작업에서 빌드 중 사용하는 컨텍스트 디렉터리의 일부로 보안 파일을 사용할 수 있습니다.

보안 및 구성 맵 데이터를 참조하는 **Dockerfile**의 예

```
FROM centos/ruby-22-centos7

USER root
COPY ./secret-dir /secrets
COPY ./config /

# Create a shell script that will output secrets and ConfigMaps when the image is run
RUN echo '#!/bin/sh' > /input_report.sh
RUN echo '(test -f /secrets/secret1 && echo -n "secret1=" && cat /secrets/secret1)' >>
/input_report.sh
RUN echo '(test -f /config && echo -n "relative-configMap=" && cat /config)' >> /input_report.sh
RUN chmod 755 /input_report.sh

CMD ["/bin/sh", "-c", "/input_report.sh"]
```



참고

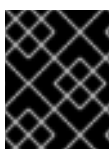
일반적으로 사용자는 해당 이미지에서 실행되는 컨테이너에 보안이 표시되지 않도록 최종 애플리케이션 이미지에서 입력 보안을 제거합니다. 그러나 보안은 추가된 계층에 있는 이미지 자체에 계속 있습니다. 이 제거는 Dockerfile 자체에 포함됩니다.

2.3.6.7. 사용자 정의 전략

사용자 정의 전략을 사용하는 경우 정의된 입력 보안 및 구성 맵을 `/var/run/secrets/openshift.io/build` 디렉터리의 빌더 컨테이너에서 모두 사용할 수 있습니다. 사용자 정의 빌드 이미지에서는 이러한 보안 및 구성 맵을 적절하게 사용해야 합니다. 사용자 정의 전략을 사용하면 사용자 정의 전략 옵션에 설명된 대로 보안을 정의할 수 있습니다.

기존 전략 보안과 입력 보안은 기술적으로 차이가 없습니다. 하지만 빌더 이미지는 빌드 사용 사례에 따라 해당 항목을 구분하고 다르게 사용할 수 있습니다.

입력 보안은 항상 `/var/run/secrets/openshift.io/build` 디렉터리에 마운트되거나 빌더에서 전체 빌드 오브젝트를 포함하는 `$BUILD` 환경 변수를 구문 분석할 수 있습니다.



중요

레지스트리에 대한 가져오기 보안이 네임스페이스와 노드 모두에 있는 경우 빌드는 기본적으로 네임스페이스의 가져오기 보안을 사용합니다.

2.3.7. 외부 아티팩트

바이너리 파일을 소스 리포지토리에 저장하지 않는 것이 좋습니다. 따라서 빌드 프로세스 중 Java `.jar` 종속 항목과 같은 추가 파일을 가져오는 빌드를 정의해야 합니다. 이 작업을 수행하는 방법은 사용 중인 빌드 전략에 따라 다릅니다.

소스 빌드 전략의 경우 `assemble` 스크립트에 적절한 셸 명령을 배치해야 합니다.

`.s2i/bin/assemble` 파일

```
#!/bin/sh
APP_VERSION=1.0
wget http://repository.example.com/app/app-$APP_VERSION.jar -O app.jar
```

`.s2i/bin/run` 파일

```
#!/bin/sh
exec java -jar app.jar
```

Docker 빌드 전략의 경우 Dockerfile을 수정하고 `RUN` 명령을 사용하여 셸 명령을 호출해야 합니다.

Dockerfile 발췌 내용

```
FROM jboss/base-jdk:8

ENV APP_VERSION 1.0
RUN wget http://repository.example.com/app/app-$APP_VERSION.jar -O app.jar

EXPOSE 8080
CMD [ "java", "-jar", "app.jar" ]
```

실제로 Dockerfile 또는 **assemble** 스크립트를 업데이트하는 대신 **BuildConfig**에 정의된 환경 변수를 사용하여 다운로드할 특정 파일을 사용자 정의할 수 있도록 파일 위치에 대한 환경 변수를 사용할 수 있습니다.

다음과 같이 환경 변수를 정의하는 다양한 방법 중에서 선택할 수 있습니다.

- **.s2i/environment** 파일 사용(소스 빌드 전략 전용)
- **BuildConfig**에 설정
- **oc start-build --env**를 사용하여 명시적으로 제공(수동으로 트리거하는 빌드 전용)

2.3.8. 개인 레지스트리에 Docker 자격 증명 사용

개인 컨테이너 레지스트리에 유효한 자격 증명이 있는 **docker/config.json** 파일을 사용하여 빌드를 제공할 수 있습니다. 이 경우 출력 이미지를 개인 컨테이너 이미지 레지스트리로 내보내거나 인증이 필요한 개인 컨테이너 이미지 레지스트리에서 빌더 이미지를 가져올 수 있습니다.



참고

OpenShift Container Platform 컨테이너 이미지 레지스트리의 경우 OpenShift Container Platform에서 보안이 자동으로 생성되므로 필요하지 않습니다.

.docker/config.json 파일은 기본적으로 홈 디렉터리에 있으며 다음과 같은 형식을 취합니다.

```
auths:
  https://index.docker.io/v1/: 1
    auth: "YWRFbGZhcGU6R2labnRib21ifTE=" 2
    email: "user@example.com" 3
```

- 1 레지스트리의 URL입니다.
- 2 암호화된 암호입니다.
- 3 로그인에 사용할 이메일 주소입니다.

이 파일에서 여러 컨테이너 이미지 레지스트리 항목을 정의할 수 있습니다. 또는 **docker login** 명령을 실행하여 이 파일에 인증 항목을 추가할 수도 있습니다. 파일이 없는 경우 생성됩니다.

Kubernetes는 구성 및 암호를 저장하는 데 사용할 수 있는 **Secret** 오브젝트를 제공합니다.

사전 요구 사항

- **.docker/config.json** 파일이 있어야 합니다.

프로세스

1. 로컬 **.docker/config.json** 파일에서 보안을 생성합니다.

```
$ oc create secret generic dockerhub \
  --from-file=.dockerconfigjson=<path/to/.docker/config.json> \
  --type=kubernetes.io/dockerconfigjson
```


이 명령은 **dockerhub**라는 보안의 JSON 사양을 생성한 후 오브젝트를 생성합니다.

2. **BuildConfig**의 **output** 섹션에 **pushSecret** 필드를 추가하고 생성한 **secret** 이름(위 예의 경우 **dockerhub**)으로 설정합니다.

```
spec:
  output:
    to:
      kind: "DockerImage"
      name: "private.registry.com/org/private-image:latest"
    pushSecret:
      name: "dockerhub"
```

oc set build-secret 명령을 사용하여 빌드 구성에 내보내기 보안을 설정할 수 있습니다.

```
$ oc set build-secret --push bc/sample-build dockerhub
```

pushSecret 필드를 지정하는 대신 빌드에서 사용하는 서비스 계정에 내보내기 보안을 연결할 수도 있습니다. 기본적으로 빌드에서는 **builder** 서비스 계정을 사용합니다. 보안에 빌드의 출력 이미지를 호스팅하는 리포지토리와 일치하는 자격 증명이 포함된 경우 내보내기 보안이 빌드에 자동으로 추가됩니다.

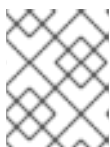
```
$ oc secrets link builder dockerhub
```

3. 빌드 전략 정의의 일부인 **pullSecret** 필드를 지정하여 개인 컨테이너 이미지 레지스트리에서 빌더 컨테이너 이미지를 가져옵니다.

```
strategy:
  sourceStrategy:
    from:
      kind: "DockerImage"
      name: "docker.io/user/private_repository"
    pullSecret:
      name: "dockerhub"
```

oc set build-secret 명령을 사용하여 빌드 구성에 가져오기 보안을 설정할 수 있습니다.

```
$ oc set build-secret --pull bc/sample-build dockerhub
```



참고

이 예제에서는 소스 빌드에 **pullSecret**을 사용하지만 Docker 및 Custom 빌드에도 적용할 수 있습니다.

pullSecret 필드를 지정하는 대신 빌드에서 사용하는 서비스 계정에 가져오기 보안을 연결할 수도 있습니다. 기본적으로 빌드에서는 **builder** 서비스 계정을 사용합니다. 보안에 빌드의 입력 이미지를 호스팅하는 리포지토리와 일치하는 자격 증명이 포함된 경우 가져오기 보안이 빌드에 자동으로 추가됩니다. **pullSecret** 필드를 지정하는 대신 빌드에서 사용하는 서비스 계정에 가져오기 보안을 연결하려면 다음을 실행합니다.

```
$ oc secrets link builder dockerhub
```



참고

이 기능을 사용하려면 **BuildConfig** 사양에 **from** 이미지를 지정해야 합니다. **oc new-build** 또는 **oc new-app**으로 생성한 Docker 전략 빌드는 특정 상황에서 이러한 작업을 수행하지 못할 수 있습니다.

2.3.9. 빌드 환경

Pod 환경 변수와 마찬가지로 빌드 환경 변수는 다른 리소스 또는 변수에 대한 참조 측면에서 Downward API를 사용하여 정의할 수 있습니다. 여기에는 잘 알려진 몇 가지 예외가 있습니다.

oc set env 명령을 사용하면 **BuildConfig**에 정의된 환경 변수도 관리할 수 있습니다.



참고

빌드 환경 변수에서 **valueFrom**을 사용하여 컨테이너 리소스를 참조하는 기능은 컨테이너를 생성하기 전에 참조를 확인하기 때문에 지원되지 않습니다.

2.3.9.1. 빌드 필드를 환경 변수로 사용

값을 가져올 필드의 **JsonPath**에 **fieldPath** 환경 변수 소스를 설정하면 빌드 오브젝트에 대한 정보를 삽입할 수 있습니다.



참고

Jenkins Pipeline 전략에서는 환경 변수에 **valueFrom** 구문을 지원하지 않습니다.

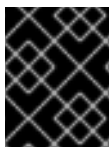
프로세스

- **fieldPath** 환경 변수 소스를 값을 가져올 필드의 **JsonPath**로 설정합니다.

```
env:
  - name: FIELDREF_ENV
    valueFrom:
      fieldRef:
        fieldPath: metadata.name
```

2.3.9.2. 보안을 환경 변수로 사용

valueFrom 구문을 사용하여 보안의 키 값을 환경 변수로 사용하도록 설정할 수 있습니다.



중요

이 메서드는 빌드 포드 콘솔의 출력에 보안을 일반 텍스트로 표시합니다. 이 문제를 방지하려면 입력 보안 및 구성 맵을 대신 사용합니다.

절차

- 보안을 환경 변수로 사용하려면 **valueFrom** 구문을 설정합니다.

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
```

```

name: secret-example-bc
spec:
  strategy:
    sourceStrategy:
      env:
        - name: MYVAL
          valueFrom:
            secretKeyRef:
              key: myval
              name: mysecret

```

추가 리소스

- [입력 보안 및 구성 맵](#)

2.3.10. 서비스 제공 인증서 보안

서비스 제공 인증서 보안은 즉시 사용 가능한 인증서가 필요한 복잡한 미들웨어 애플리케이션을 지원하기 위한 것입니다. 해당 설정은 관리자 툴에서 노드 및 마스터에 대해 생성하는 서버 인증서와 동일합니다.

프로세스

서비스와의 통신을 보호하려면 클러스터에서 서명된 제공 인증서/키 쌍을 네임스페이스의 보안에 생성하도록 합니다.

- 보안에 사용할 이름으로 설정된 값을 사용하여 서비스에 **service.beta.openshift.io/serving-cert-secret-name** 주석을 설정합니다. 그러면 **PodSpec**에서 해당 보안을 마운트할 수 있습니다. 사용 가능한 경우 Pod가 실행됩니다. 인증서는 내부 서비스 DNS 이름인 **<service.name>.<service.namespace>.svc**에 적합합니다.

인증서 및 키는 PEM 형식이며 각각 **tls.crt** 및 **tls.key**에 저장됩니다. 인증서/키 쌍은 만료 시기가 다가오면 자동으로 교체됩니다. 보안의 **service.beta.openshift.io/expiry** 주석에서 RFC3339 형식으로 된 만료 날짜를 확인합니다.



참고

대부분의 경우 서비스 DNS 이름 **<service.name>.<service.namespace>.svc**는 외부에서 라우팅할 수 없습니다. **<service.name>.<service.namespace>.svc**는 주로 클러스터 내 또는 서비스 내 통신과 경로 재암호화에 사용됩니다.

기타 Pod는 해당 Pod에 자동으로 마운트되는 **/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt** 파일의 CA(인증 기관) 번들을 사용하여 내부 DNS 이름에만 서명되는 클러스터 생성 인증서를 신뢰할 수 있습니다.

이 기능의 서명 알고리즘은 **x509.SHA256WithRSA**입니다. 직접 교대하려면 생성된 보안을 삭제합니다. 새 인증서가 생성됩니다.

2.3.11. 보안 제한 사항

보안을 사용하려면 Pod에서 보안을 참조해야 합니다. 보안은 다음 세 가지 방법으로 Pod에서 사용할 수 있습니다.

- 컨테이너에 환경 변수를 채우기 위해 사용.
- 하나 이상의 컨테이너에 마운트된 볼륨에서 파일로 사용.

- Pod에 대한 이미지를 가져올 때 kubelet으로 사용.

볼륨 유형 보안은 볼륨 메커니즘을 사용하여 데이터를 컨테이너에 파일로 작성합니다. **imagePullSecrets** 는 서비스 계정을 사용하여 네임스페이스의 모든 포트에 보안을 자동으로 삽입합니다.

템플릿에 보안 정의가 포함된 경우 템플릿에 제공된 보안을 사용할 수 있는 유일한 방법은 보안 볼륨 소스를 검증하고 지정된 오브젝트 참조가 유형이 **Secret**인 오브젝트를 실제로 가리키는 것입니다. 따라서 보안을 생성한 후 해당 보안을 사용하는 Pod를 생성해야 합니다. 가장 효과적인 방법은 서비스 계정을 사용하여 자동으로 삽입되도록 하는 것입니다.

Secret API 오브젝트는 네임스페이스에 있습니다. 동일한 네임스페이스에 있는 Pod만 참조할 수 있습니다.

개별 보안은 1MB로 제한됩니다. 이는 대규모 보안이 생성되어 apiserver 및 kubelet 메모리가 소진되는 것을 막기 위한 것입니다. 그러나 작은 보안을 많이 생성해도 메모리가 소진될 수 있습니다.

2.4. 빌드 출력 관리

빌드 출력 관리에 대한 개요 및 지침은 다음 섹션에서 확인하십시오.

2.4.1. 빌드 출력

Docker 또는 S2I(source-to-image) 전략을 사용하는 빌드에서는 새 컨테이너 이미지를 생성합니다. 그런 다음 이미지를 **Build** 사양의 **output** 섹션에 지정된 컨테이너 이미지 레지스트리로 푸시됩니다.

출력 종류가 **ImageStreamTag**인 경우 이미지를 통합된 OpenShift Container Platform 레지스트리로 푸시되고 지정된 이미지 스트림에 태그를 지정합니다. 출력 유형이 **DockerImage**인 경우에는 출력 참조 이름이 Docker 내보내기 사양으로 사용됩니다. 사양은 레지스트리를 포함할 수 있으며 레지스트리가 지정되지 않은 경우 기본적으로 DockerHub로 설정됩니다. 빌드 사양의 출력 섹션이 비어 있으면 빌드 종료 시 이미지를 푸시하지 않습니다.

ImageStreamTag로 출력

```
spec:
  output:
    to:
      kind: "ImageStreamTag"
      name: "sample-image:latest"
```

Docker 내보내기 사양으로 출력

```
spec:
  output:
    to:
      kind: "DockerImage"
      name: "my-registry.mycompany.com:5000/myimages/myimage:tag"
```

2.4.2. 이미지 환경 변수 출력

Docker 및 S2I(Source-to-Image) 전략 빌드에서는 출력 이미지에 다음 환경 변수를 설정합니다.

변수	설명
OPENSIFT_BUILD_NAME	빌드 이름
OPENSIFT_BUILD_NAMESPACE	빌드의 네임스페이스
OPENSIFT_BUILD_SOURCE	빌드의 소스 URL
OPENSIFT_BUILD_REFERENCE	빌드에 사용된 Git 참조
OPENSIFT_BUILD_COMMIT	빌드에 사용된 소스 커밋

또한 모든 사용자 정의 환경 변수(예: S2I 또는 Docker 전략 옵션으로 구성된 환경 변수)도 출력 이미지 환경 변수 목록의 일부입니다.

2.4.3. 출력 이미지 라벨

Docker 및 S2I(Source-to-Image)의 빌드에서는 출력 이미지에 다음 라벨을 설정합니다.

레이블	설명
io.openshift.build.commit.author	빌드에 사용된 소스 커밋 작성자
io.openshift.build.commit.date	빌드에 사용된 소스 커밋의 날짜
io.openshift.build.commit.id	빌드에 사용된 소스 커밋의 해시
io.openshift.build.commit.message	빌드에 사용된 소스 커밋의 메시지
io.openshift.build.commit.ref	소스에 지정된 분기 또는 참조
io.openshift.build.source-location	빌드의 소스 URL

BuildConfig.spec.output.imageLabels 필드를 사용하여 빌드 구성에서 빌드하는 각 이미지에 적용할 사용자 정의 라벨 목록을 지정할 수도 있습니다.

빌드한 이미지에 적용할 사용자 정의 라벨

```
spec:
  output:
    to:
      kind: "ImageStreamTag"
      name: "my-image:latest"
    imageLabels:
      - name: "vendor"
        value: "MyCompany"
      - name: "authoritative-source-url"
        value: "registry.mycompany.com"
```

2.5. 빌드 전략 사용

다음 섹션에서는 지원되는 주요 빌드 전략과 이러한 전략을 사용하는 방법을 정의합니다.

2.5.1. Docker 빌드

OpenShift Container Platform은 Buildah를 사용하여 Dockerfile에서 컨테이너 이미지를 빌드합니다. Dockerfile을 사용하여 컨테이너 이미지를 빌드하는 방법에 대한 자세한 내용은 [Dockerfile 참조 문서](#)를 참조하십시오.

작은 정보

buildArgs 배열을 사용하여 Docker 빌드 인수를 설정하는 경우 Dockerfile 참조 문서에서 [ARG 및 FROM 이 상호 작용하는 방법 이해](#)를 참조하십시오.

2.5.1.1. Dockerfile FROM 이미지 교체

Dockerfile의 **FROM** 명령을 **BuildConfig** 오브젝트의 **from**으로 교체할 수 있습니다. Dockerfile에서 다중 단계 빌드를 사용하는 경우 마지막 **FROM** 명령의 이미지가 교체됩니다.

프로세스

Dockerfile의 **FROM** 명령을 **BuildConfig** 오브젝트의 **from**으로 교체

```
strategy:
  dockerStrategy:
    from:
      kind: "ImageStreamTag"
      name: "debian:latest"
```

2.5.1.2. Dockerfile 경로 사용

기본적으로 Docker 빌드는 **BuildConfig.spec.source.contextDir** 필드에 지정된 컨텍스트의 루트에 있는 Dockerfile을 사용합니다.

dockerfilePath 필드를 사용하면 **BuildConfig.spec.source.contextDir** 필드와 상대되는 다른 경로를 사용하여 Dockerfile을 찾을 수 있습니다. 파일 이름은 기본 Dockerfile(예: **MyDockerfile**) 또는 하위 디렉터리의 Dockerfile 경로(예: **dockerfiles/app1/Dockerfile**)와 다를 수 있습니다.

프로세스

빌드에서 다른 경로를 사용하여 Dockerfile을 찾으려면 **dockerfilePath** 필드를 사용하려면 다음과 같이 설정합니다.

```
strategy:
  dockerStrategy:
    dockerfilePath: dockerfiles/app1/Dockerfile
```

2.5.1.3. Docker 환경 변수 사용

Docker 빌드 프로세스 및 생성된 이미지에 환경 변수를 사용할 수 있도록 빌드 구성의 **dockerStrategy** 정의에 환경 변수를 추가할 수 있습니다.

정의된 환경 변수는 **FROM** 명령 직후 단일 **ENV** Dockerfile 명령으로 삽입되어 나중에 Dockerfile 내에서 참조할 수 있습니다.

프로세스

변수는 빌드 중 정의되고 출력 이미지에 유지되므로 해당 이미지를 실행하는 모든 컨테이너에도 존재합니다.

예를 들어 다음은 빌드 및 런타임 중 사용할 사용자 정의 HTTP 프록시를 정의합니다.

```
dockerStrategy:
...
  env:
    - name: "HTTP_PROXY"
      value: "http://myproxy.net:5187/"
```

oc set env 명령을 사용하면 빌드 구성에 정의된 환경 변수도 관리할 수 있습니다.

2.5.1.4. Docker 빌드 인수 추가

buildArgs 배열을 사용하여 **Docker 빌드 인수**를 설정할 수 있습니다. 빌드 인수는 빌드가 시작될 때 Docker로 전달됩니다.

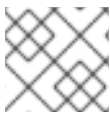
작은 정보

Dockerfile 참조 문서에서 **ARG** 및 **FROM**이 상호 작용하는 방법 이해 를 참조하십시오.

절차

Docker 빌드 인수를 설정하려면 **BuildConfig** 오브젝트의 **dockerStrategy** 정의에 있는 **buildArgs** 배열에 항목을 추가합니다. 예를 들면 다음과 같습니다.

```
dockerStrategy:
...
  buildArgs:
    - name: "foo"
      value: "bar"
```



참고

name 및 **value** 필드만 지원됩니다. **valueFrom** 필드의 설정은 모두 무시됩니다.

2.5.1.5. Docker 빌드가 포함된 스쿼시 계층

Docker 빌드는 일반적으로 Dockerfile의 각 명령을 나타내는 계층을 생성합니다.

imageOptimizationPolicy를 **SkipLayers**로 설정하면 모든 명령을 기본 이미지 상단의 단일 계층으로 병합합니다.

프로세스

- **imageOptimizationPolicy**를 **SkipLayers**로 설정합니다.

```
strategy:
  dockerStrategy:
    imageOptimizationPolicy: SkipLayers
```

2.5.2. S2I(Source-to-Image) 빌드

S2I(Source-to-Image)는 재현 가능한 컨테이너 이미지를 빌드하는 틀입니다. 컨테이너 이미지에 애플리케이션 소스를 삽입하고 새 이미지를 어셈블하여 실행할 수 있는 이미지를 생성합니다. 새 이미지는 기본 이미지, 빌더, 빌드 소스를 통합하고 **buildah run** 명령과 함께 사용할 수 있습니다. S2I는 이전에 다운로드한 종속 항목, 이전에 빌드한 아티팩트 등을 다시 사용하는 증분 빌드를 지원합니다.

2.5.2.1. S2I(Source-to-Image) 증분 빌드 수행

S2I(Source-to-Image)는 증분 빌드를 수행할 수 있으므로 이전에 빌드한 이미지의 아티팩트를 재사용할 수 있습니다.

프로세스

- 증분 빌드를 생성하려면 전략 정의를 다음과 같이 수정합니다.

```
strategy:
  sourceStrategy:
    from:
      kind: "ImageStreamTag"
      name: "incremental-image:latest" 1
    incremental: true 2
```

- 1 증분 빌드를 지원하는 이미지를 지정합니다. 빌더 이미지 설명서를 참조하여 이 동작을 지원하는지 확인합니다.
- 2 이 플래그는 증분 빌드 시도 여부를 제어합니다. 빌더 이미지에서 증분 빌드를 지원하지 않는 경우 빌드는 성공하지만 **save-artifacts** 스크립트 누락으로 인해 증분 빌드가 성공하지 못했다는 로그 메시지가 표시됩니다.

추가 리소스

- 증분 빌드를 지원하는 빌더 이미지를 생성하는 방법에 대한 내용은 S2I 요구 사항을 참조하십시오.

2.5.2.2. S2I(Source-to-Image) 빌더 이미지 스크립트 덮어쓰기

빌더 이미지에서 제공하는 **assemble**, **run**, **save-artifacts** S2I(Source-to-Image) 스크립트를 덮어쓸 수 있습니다.

프로세스

빌더 이미지에서 제공하는 **assemble**, **run**, **save-artifacts** S2I 스크립트를 덮어쓰려면 다음 중 하나를 수행합니다.

- 애플리케이션 소스 리포지토리의 **.s2i/bin** 디렉터리에 **assemble**, **run**, 또는 **save-artifacts** 스크립트를 제공합니다.

- 스크립트가 포함된 디렉터리의 URL을 전략 정의의 일부로 제공합니다. 예를 들면 다음과 같습니다.

```
strategy:
  sourceStrategy:
    from:
      kind: "ImageStreamTag"
      name: "builder-image:latest"
      scripts: "http://somehost.com/scripts_directory" 1
```

- 1 이 경로에는 **run, assemble, save-artifacts**가 추가됩니다. 일부 또는 모든 스크립트가 확인되면 해당 스크립트가 이미지에 제공된 동일한 이름의 스크립트 대신 사용됩니다.



참고

scripts URL에 있는 파일은 소스 리포지토리의 **.s2i/bin**에 있는 파일보다 우선합니다.

2.5.2.3. S2I(Source-to-Image) 환경 변수

소스 빌드 프로세스 및 결과 이미지에서 환경 변수를 사용할 수 있도록 하는 방법에는 환경 파일과 BuildConfig 환경 값을 사용하는 것입니다. 제공되는 변수는 빌드 프로세스 중 출력 이미지에 제공됩니다.

2.5.2.3.1. S2I(Source-to-Image) 환경 파일 사용

소스 빌드를 사용하면 소스 리포지토리의 **.s2i/environment** 파일에 지정하는 방식으로 애플리케이션 내에서 해당 하나씩 환경 값을 설정할 수 있습니다. 이 파일에 지정된 환경 변수는 빌드 프로세스 중 출력 이미지에 제공됩니다.

소스 리포지토리에 **.s2i/environment** 파일을 제공하는 경우 빌드 중 S2I(Source-to-Image)에서 이 파일을 읽습니다. 그러면 **assemble** 스크립트에서 이러한 변수를 사용할 수 있으므로 빌드 동작을 사용자 정의할 수 있습니다.

프로세스

예를 들어 빌드 중 Rails 애플리케이션의 자산 컴파일을 비활성화하려면 다음을 수행합니다.

- **.s2i/environment** 파일에 **DISABLE_ASSET_COMPILATION=true**를 추가합니다.

빌드 외에 지정된 환경 변수도 실행 중인 애플리케이션 자체에서 사용할 수 있습니다. 예를 들어 Rails 애플리케이션이 **production** 대신 **development** 모드에서 시작되도록 하려면 다음을 수행합니다.

- **RAILS_ENV=development**를 **.s2i/environment** 파일에 추가합니다.

지원되는 환경 변수의 전체 목록은 각 이미지의 이미지 사용 섹션에서 확인할 수 있습니다.

2.5.2.3.2. S2I(Source-to-Image) 빌드 구성 환경 사용

빌드 구성의 **sourceStrategy** 정의에 환경 변수를 추가할 수 있습니다. 여기에 정의된 환경 변수는 **assemble** 스크립트를 실행하는 동안 표시되고 출력 이미지에 정의되어 **run** 스크립트 및 애플리케이션 코드에서도 사용할 수 있습니다.

프로세스

- 예를 들어 Rails 애플리케이션의 자산 컴파일을 비활성화하려면 다음을 수행합니다.

```
sourceStrategy:
...
env:
  - name: "DISABLE_ASSET_COMPILATION"
    value: "true"
```

추가 리소스

- 빌드 환경 섹션에서는 고급 지침을 제공합니다.
- **oc set env** 명령을 사용하면 빌드 구성에 정의된 환경 변수도 관리할 수 있습니다.

2.5.2.4. S2I(Source-to-Image) 소스 파일 무시

S2I(Source-to-Image)는 무시해야 하는 파일 패턴 목록이 포함된 **.s2iignore** 파일을 지원합니다. **.s2iignore** 파일에 있는 패턴과 일치하고 다양한 입력 소스에서 제공하는 빌드 작업 디렉터리의 파일은 **assemble** 스크립트에서 사용할 수 없습니다.

2.5.2.5. S2I(Source-to-Image)를 사용하여 소스 코드에서 이미지 생성

S2I(Source-to-Image)는 애플리케이션 소스 코드를 입력으로 사용하고 어셈블된 애플리케이션을 실행하는 새 이미지를 출력으로 생성하는 이미지를 쉽게 작성할 수 있는 프레임워크입니다.

재현 가능한 컨테이너 이미지를 빌드하는 데 S2I를 사용하는 주요 장점은 개발자가 쉽게 사용할 수 있다는 점입니다. 빌더 이미지 작성자는 이미지에서 최상의 S2I 성능, 빌드 프로세스, S2I 스크립트를 제공하도록 두 가지 기본 개념을 이해해야 합니다.

2.5.2.5.1. S2I(Source-to-Image) 빌드 프로세스 이해

빌드 프로세스는 최종 컨테이너 이미지로 통합되는 다음 세 가지 기본 요소로 구성됩니다.

- 소스
- S2I(Source-to-Image) 스크립트
- 빌더 이미지

S2I는 첫 번째 **FROM** 명령으로 빌더 이미지가 포함된 Dockerfile을 생성합니다. 그런 다음 S2I에서 생성된 Dockerfile은 Buildah로 전달됩니다.

2.5.2.5.2. S2I(Source-to-Image) 스크립트를 작성하는 방법

스크립트를 빌더 이미지 내에서 실행할 수 있는 경우 모든 프로그래밍 언어로 S2I(Source-to-Image) 스크립트를 작성할 수 있습니다. S2I는 **assemble/run/save-artifacts** 스크립트를 제공하는 다양한 옵션을 지원합니다. 이러한 위치는 모두 다음 순서에 따라 각 빌드에서 확인합니다.

1. 빌드 구성에 지정된 스크립트입니다.
2. 애플리케이션 소스 **.s2i/bin** 디렉터리에 있는 스크립트입니다.
3. 라벨이 **io.openshift.s2i.scripts-url**인 기본 이미지 URL에 있는 스크립트입니다.

이미지에 지정된 **io.openshift.s2i.scripts-url** 라벨과 빌드 구성에 지정된 스크립트 모두 다음 양식 중 하나를 취할 수 있습니다.

- **image:///path_to_scripts_dir**: S2I 스크립트가 있는 디렉터리에 대한 이미지 내부의 절대 경로입니다.
- **file:///path_to_scripts_dir**: S2I 스크립트가 있는 호스트의 디렉터리에 대한 상대 또는 절대 경로입니다.
- **http(s)://path_to_scripts_dir**: S2I 스크립트가 있는 디렉터리의 URL입니다.

표 2.1. S2I 스크립트

스크립트	설명
assemble	<p>assemble 스크립트는 소스에서 애플리케이션 아티팩트를 빌드하여 이미지 내부의 적절한 디렉터리에 배치합니다. 이 스크립트는 필수입니다. 이 스크립트의 워크플로는 다음과 같습니다.</p> <ol style="list-style-type: none"> 1. 선택 사항: 빌드 아티팩트를 복구합니다. 증분 빌드를 지원하려면 save-artifacts도 정의해야 합니다. 2. 애플리케이션 소스를 원하는 위치에 배치합니다. 3. 애플리케이션 아티팩트를 빌드합니다. 4. 아티팩트를 실행할 수 있는 적절한 위치에 설치합니다.
run	<p>run 스크립트는 애플리케이션을 실행합니다. 이 스크립트는 필수입니다.</p>
save-artifacts	<p>save-artifacts 스크립트는 이어지는 빌드 프로세스의 속도를 높일 수 있는 모든 종속 항목을 수집합니다. 이 스크립트는 선택 사항입니다. 예를 들면 다음과 같습니다.</p> <ul style="list-style-type: none"> • Ruby의 경우 Bundler에서 설치한 gems입니다. • Java의 경우 .m2 콘텐츠입니다. <p>이러한 종속 항목은 tar 파일로 수집되어 표준 출력으로 스트리밍됩니다.</p>
usage	<p>usage 스크립트를 사용하면 사용자에게 이미지를 올바르게 사용하는 방법을 알릴 수 있습니다. 이 스크립트는 선택 사항입니다.</p>

스크립트	설명
<p>test/run</p>	<p>test/run 스크립트를 사용하면 이미지가 올바르게 작동하는지 확인하는 프로세스를 생성할 수 있습니다. 이 스크립트는 선택 사항입니다. 해당 프로세스의 제안된 흐름은 다음과 같습니다.</p> <ol style="list-style-type: none"> 1. 이미지를 빌드합니다. 2. 이미지를 실행하여 usage 스크립트를 확인합니다. 3. s2i build를 실행하여 assemble 스크립트를 확인합니다. 4. 선택 사항: s2i build 를 다시 실행하여 save-artifacts 및 assemble 스크립트에서 아티팩트를 저장 및 복원하는지 확인합니다. 5. 이미지를 실행하여 테스트 애플리케이션이 작동하는지 확인합니다. <div data-bbox="517 707 625 875" style="float: left; margin-right: 10px;"> </div> <p>참고</p> <p>test/run 스크립트로 빌드한 테스트 애플리케이션을 배치하도록 제안된 위치는 이미지 리포지토리의 test/test-app 디렉터리입니다.</p>

S2I 스크립트의 예

다음 예제 S2I 스크립트는 Bash로 작성됩니다. 각 예에서는 **tar** 콘텐츠가 **/tmp/s2i** 디렉터리에 압축 해제되어 있다고 가정합니다.

assemble 스크립트:

```
#!/bin/bash

# restore build artifacts
if [ "$(ls /tmp/s2i/artifacts/ 2>/dev/null)" ]; then
    mv /tmp/s2i/artifacts/* $HOME/.
fi

# move the application source
mv /tmp/s2i/src $HOME/src

# build application artifacts
pushd ${HOME}
make all

# install the artifacts
make install
popd
```

run 스크립트:

```
#!/bin/bash

# run the application
/opt/application/run.sh
```

save-artifacts 스크립트:

```
#!/bin/bash

pushd ${HOME}
if [ -d deps ]; then
    # all deps contents to tar stream
    tar cf - deps
fi
popd
```

usage 스크립트:

```
#!/bin/bash

# inform the user how to use the image
cat <<EOF
This is a S2I sample builder image, to use it, install
https://github.com/openshift/source-to-image
EOF
```

추가 리소스

- [S2I 이미지 생성 튜토리얼](#)

2.5.3. 사용자 정의 빌드

사용자 정의 빌드 전략을 사용하면 개발자가 전체 빌드 프로세스를 담당하는 특정 빌더 이미지를 정의할 수 있습니다. 자체 빌더 이미지를 사용하면 빌드 프로세스를 사용자 정의할 수 있습니다.

사용자 정의 빌더 이미지는 빌드 프로세스 논리가 포함된 일반 컨테이너 이미지입니다(예: RPM 또는 기본 이미지 빌드).

사용자 정의 빌드는 높은 권한으로 실행되며 기본적으로 사용자에게 제공되지 않습니다. 클러스터 관리 권한이 있는 신뢰할 수 있는 사용자에게만 사용자 정의 빌드를 실행할 수 있는 권한을 부여해야 합니다.

2.5.3.1. 사용자 정의 빌드에 FROM 이미지 사용

customStrategy.from 섹션을 사용하여 사용자 정의 빌드에 사용할 이미지를 표시할 수 있습니다.

프로세스

- **customStrategy.from** 섹션을 설정합니다.

```
strategy:
  customStrategy:
    from:
      kind: "DockerImage"
      name: "openshift/sti-image-builder"
```

2.5.3.2. 사용자 정의 빌드에서 보안 사용

사용자 정의 전략에서는 모든 빌드 유형에 추가할 수 있는 소스 및 이미지 보안 외에 임의의 보안 목록을 빌더 Pod에 추가할 수 있습니다.

프로세스

- 각 보안을 특정 위치에 마운트하려면 **strategy** YAML 파일의 **secretSource** 및 **mountPath** 필드를 편집합니다.

```
strategy:
  customStrategy:
    secrets:
      - secretSource: ❶
        name: "secret1"
        mountPath: "/tmp/secret1" ❷
      - secretSource:
        name: "secret2"
        mountPath: "/tmp/secret2"
```

❶ **secretSource**는 빌드와 동일한 네임스페이스에 있는 보안에 대한 참조입니다.

❷ **mountPath**는 보안을 마운트해야 하는 사용자 정의 빌더 내부의 경로입니다.

2.5.3.3. 사용자 정의 빌드에 환경 변수 사용

사용자 정의 빌드 프로세스에서 환경 변수를 사용할 수 있도록 하려면 빌드 구성의 **customStrategy** 정의에 환경 변수를 추가하면 됩니다.

여기에서 정의한 환경 변수는 사용자 정의 빌드를 실행하는 Pod로 전달됩니다.

프로세스

1. 빌드 중 사용할 사용자 정의 HTTP 프록시를 정의합니다.

```
customStrategy:
  ...
  env:
    - name: "HTTP_PROXY"
      value: "http://myproxy.net:5187/"
```

2. 빌드 구성에 정의된 환경 변수를 관리하려면 다음 명령을 입력합니다.

```
$ oc set env <enter_variables>
```

2.5.3.4. 사용자 정의 빌더 이미지 사용

OpenShift Container Platform의 사용자 정의 빌드 전략을 사용하면 전체 빌드 프로세스를 담당하는 특정 빌더 이미지를 정의할 수 있습니다. 패키지, JAR, WAR, 설치 가능한 ZIP 또는 기본 이미지와 같은 개별 아티팩트를 생성하는 빌드가 필요한 경우 사용자 정의 빌드 전략을 사용하는 사용자 정의 빌더 이미지를 사용합니다.

사용자 정의 빌더 이미지는 RPM 또는 기본 컨테이너 이미지와 같은 아티팩트를 구축하는 데 사용되는 빌드 프로세스 논리에 내장된 일반 컨테이너 이미지입니다.

또한 사용자 정의 빌더를 사용하면 단위 테스트 또는 통합 테스트를 실행하는 CI/CD 흐름과 같이 확장된 빌드 프로세스를 구현할 수 있습니다.

2.5.3.4.1. 사용자 정의 빌더 이미지

사용자 정의 빌더 이미지는 호출 시 빌드를 진행하는 데 필요한 정보와 함께 다음 환경 변수를 수신합니다.

표 2.2. 사용자 정의 빌더 환경 변수

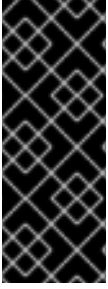
변수 이름	설명
BUILD	Build 오브젝트 정의의 전체 직렬화 JSON입니다. 직렬화에 특정 API 버전을 사용해야 하는 경우 빌드 구성의 사용자 정의 전략 사양에 buildAPIVersion 매개변수를 설정하면 됩니다.
SOURCE_REPOSITORY	빌드할 소스가 있는 Git 리포지토리의 URL입니다.
SOURCE_URI	SOURCE_REPOSITORY 와 동일한 값을 사용합니다. 둘 중 하나를 사용할 수 있습니다.
SOURCE_CONTEXT_DIR	빌드할 때 사용할 Git 리포지토리의 하위 디렉터리를 지정합니다. 정의한 경우에만 존재합니다.
SOURCE_REF	빌드할 Git 참조입니다.
ORIGIN_VERSION	이 빌드 오브젝트를 생성한 OpenShift Container Platform 마스터의 버전입니다.
OUTPUT_REGISTRY	이미지를 내보낼 컨테이너 이미지 레지스트리입니다.
OUTPUT_IMAGE	빌드 중인 이미지의 컨테이너 이미지 태그 이름입니다.
PUSH_DOCKERCFG_PATH	podman push 작업을 실행하는 데 필요한 컨테이너 레지스트리 자격 증명의 경로입니다.

2.5.3.4.2. 사용자 정의 빌더 워크플로

사용자 정의 빌더 이미지 작성자는 빌드 프로세스를 정의하는 데 유연성이 있지만 빌더 이미지는 OpenShift Container Platform 내에서 빌드를 실행하는 데 필요한 다음 단계를 준수해야 합니다.

1. **Build** 오브젝트 정의에는 빌드의 입력 매개변수에 대한 모든 필수 정보가 포함되어 있습니다.
2. 빌드 프로세스를 실행합니다.
3. 빌드에서 이미지를 생성하면 빌드의 출력 위치가 정의된 경우 해당 위치로 내보냅니다. 다른 출력 위치는 환경 변수를 통해 전달할 수 있습니다.

2.5.4. 파이프라인 빌드



중요

파이프라인 빌드 전략은 OpenShift Container Platform 4에서 더 이상 사용되지 않습니다. 동등하고 향상된 기능은 Tekton 기반 OpenShift Container Platform Pipelines에 있습니다.

OpenShift Container Platform의 Jenkins 이미지는 완전히 지원되며 사용자는 Jenkins 사용 설명서를 따라 작업에 **jenkinsfile**을 정의하거나 소스 제어 관리 시스템에 저장해야 합니다.

파이프라인 빌드 전략을 사용하면 개발자가 Jenkins 파이프라인 플러그인에서 사용할 Jenkins 파이프라인을 정의할 수 있습니다. 다른 빌드 유형과 동일한 방식으로 OpenShift Container Platform에서 빌드를 시작, 모니터링, 관리할 수 있습니다.

파이프라인 워크플로는 빌드 구성에 직접 포함하거나 Git 리포지토리에 제공하는 방식으로 **jenkinsfile**에 정의하고 빌드 구성에서 참조합니다.

2.5.4.1. OpenShift Container Platform 파이프라인 이해



중요

파이프라인 빌드 전략은 OpenShift Container Platform 4에서 더 이상 사용되지 않습니다. 동등하고 향상된 기능은 Tekton 기반 OpenShift Container Platform Pipelines에 있습니다.

OpenShift Container Platform의 Jenkins 이미지는 완전히 지원되며 사용자는 Jenkins 사용 설명서를 따라 작업에 **jenkinsfile**을 정의하거나 소스 제어 관리 시스템에 저장해야 합니다.

파이프라인을 사용하면 OpenShift Container Platform에서 애플리케이션의 빌드, 배포, 승격을 제어할 수 있습니다. Jenkins Pipeline 빌드 전략, **jenkinsfiles** 및 OpenShift Container Platform Domain Specific Language (DSL)의 조합을 사용하여 Jenkins 클라이언트 플러그인에서 제공하는 DSL(OpenShift Container Platform Domain Specific Language)을 사용하면 모든 시나리오에 대해 고급 빌드, 테스트, 배포 및 승격을 수행할 수 있습니다.

OpenShift Container Platform Jenkins 동기화 플러그인

OpenShift Container Platform Jenkins 동기화 플러그인은 Jenkins 작업 및 빌드와 동기화된 빌드 구성 및 빌드 오브젝트를 유지하고 다음 기능을 제공합니다.

- Jenkins에서 동적 작업 및 실행 생성
- 이미지 스트림, 이미지 스트림 태그 또는 구성 맵에서 에이전트 Pod 템플릿 동적 생성
- 환경 변수 할당
- OpenShift Container Platform 웹 콘솔의 파이프라인 시각화
- Jenkins Git 플러그인과의 통합으로 OpenShift Container Platform 빌드의 커밋 정보를 Jenkins Git 플러그인에 전달합니다.
- Jenkins 자격 증명 항목에 시크릿 동기화.

OpenShift Container Platform Jenkins 클라이언트 플러그인

OpenShift Container Platform Jenkins 클라이언트 플러그인은 Jenkins 플러그인 중 하나로, OpenShift Container Platform API 서버와의 다양한 상호 작용을 위해 읽기 쉽고 간결하고 포괄적이며 유창한

Jenkins Pipeline 구문을 제공하기 위한 것입니다. 플러그인은 OpenShift Container Platform 명령줄 툴인 **oc**를 사용하는데 스크립트를 실행하는 노드에서 이 툴을 사용할 수 있어야 합니다.

Jenkins 클라이언트 플러그인은 애플리케이션의 **jenkinsfile** 내에서 OpenShift Container Platform DSL을 사용할 수 있도록 Jenkins 마스터에 설치해야 합니다. 이 플러그인은 OpenShift Container Platform Jenkins 이미지를 사용할 때 기본적으로 설치 및 활성화됩니다.

프로젝트 내 OpenShift Container Platform Pipeline의 경우 Jenkins Pipeline 빌드 전략을 사용해야 합니다. 이 전략에서는 기본적으로 소스 리포지토리의 루트에서 **jenkinsfile**을 사용하지만 다음과 같은 구성 옵션을 제공합니다.

- 빌드 구성 내의 인라인 **jenkinsfile** 필드
- 소스 **contextDir**에 상대적으로 사용할 **jenkinsfile**의 위치를 참조하는, 빌드 구성 내의 **jenkinsfilePath** 필드



참고

선택적 **jenkinsfilePath** 필드는 소스 **contextDir**에 상대적으로 사용할 파일의 이름을 지정합니다. **contextDir**이 생략된 경우 기본값은 리포지토리의 루트입니다. **jenkinsfilePath**가 생략된 경우 기본값은 **jenkinsfile**입니다.

2.5.4.2. 파이프라인 빌드를 위한 Jenkins 파일 제공



중요

파이프라인 빌드 전략은 OpenShift Container Platform 4에서 더 이상 사용되지 않습니다. 동등하고 향상된 기능은 Tekton 기반 OpenShift Container Platform Pipelines에 있습니다.

OpenShift Container Platform의 Jenkins 이미지는 완전히 지원되며 사용자는 Jenkins 사용 설명서를 따라 작업에 **jenkinsfile**을 정의하거나 소스 제어 관리 시스템에 저장해야 합니다.

jenkinsfile에서는 표준 Groovy 언어 구문을 사용하여 애플리케이션의 구성, 빌드, 배포를 세부적으로 제어할 수 있습니다.

다음 방법 중 하나로 **jenkinsfile**을 제공할 수 있습니다.

- 소스 코드 리포지토리에 있는 파일
- **jenkinsfile** 필드를 사용하여 빌드 구성의 일부로 포함

첫 번째 옵션을 사용하는 경우 다음 위치 중 하나의 애플리케이션 소스 코드 리포지토리에 **jenkinsfile**을 포함해야 합니다.

- 리포지토리 루트에 있는 **jenkinsfile**이라는 파일
- 리포지토리의 소스 **contextDir** 루트에 있는 **jenkinsfile**이라는 파일
- BuildConfig의 **JenkinsPipelineStrategy** 섹션에 있는 **jenkinsfilePath** 필드를 통해 지정되는 파일 이름(제공되는 경우 소스 **contextDir**에 상대적이고 제공되지 않는 경우 기본값은 리포지토리의 루트임)

jenkinsfile은 Jenkins 에이전트 Pod에서 실행됩니다. OpenShift Container Platform DSL을 사용하려면 해당 Pod에 사용 가능한 OpenShift Container Platform 클라이언트 바이너리가 있어야 합니다.

프로세스

다음 중 하나를 수행하여 Jenkins 파일을 제공할 수 있습니다.

- 빌드 구성에 Jenkins 파일 포함
- 빌드 구성에 Jenkins 파일이 포함된 Git 리포지토리에 대한 참조 포함

포함된 정의

```
kind: "BuildConfig"
apiVersion: "v1"
metadata:
  name: "sample-pipeline"
spec:
  strategy:
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        node('agent') {
          stage 'build'
          openshiftBuild(buildConfig: 'ruby-sample-build', showBuildLogs: 'true')
          stage 'deploy'
          openshiftDeploy(deploymentConfig: 'frontend')
        }
```

Git 리포지토리에 대한 참조

```
kind: "BuildConfig"
apiVersion: "v1"
metadata:
  name: "sample-pipeline"
spec:
  source:
    git:
      uri: "https://github.com/openshift/ruby-hello-world"
  strategy:
    jenkinsPipelineStrategy:
      jenkinsfilePath: some/repo/dir/filename ❶
```

❶ 선택적 **jenkinsfilePath** 필드는 소스 **contextDir**에 상대적으로 사용할 파일의 이름을 지정합니다. **contextDir**이 생략된 경우 기본값은 리포지토리의 루트입니다. **jenkinsfilePath**가 생략된 경우 기본값은 **jenkinsfile**입니다.

2.5.4.3. 파이프라인 빌드에 환경 변수 사용



중요

파이프라인 빌드 전략은 OpenShift Container Platform 4에서 더 이상 사용되지 않습니다. 동등하고 향상된 기능은 Tekton 기반 OpenShift Container Platform Pipelines에 있습니다.

OpenShift Container Platform의 Jenkins 이미지는 완전히 지원되며 사용자는 Jenkins 사용 설명서를 따라 작업에 **jenkinsfile**을 정의하거나 소스 제어 관리 시스템에 저장해야 합니다.

파이프라인 빌드 프로세스에서 환경 변수를 사용할 수 있도록 하려면 빌드 구성의 `jenkinsPipelineStrategy` 정의에 환경 변수를 추가하면 됩니다.

정의된 환경 변수는 빌드 구성과 관련된 모든 Jenkins 작업의 매개변수로 설정됩니다.

프로세스

- 빌드 중 사용할 환경 변수를 정의하려면 YAML 파일을 다음과 같이 편집합니다.

```
jenkinsPipelineStrategy:
...
  env:
    - name: "FOO"
      value: "BAR"
```

`oc set env` 명령을 사용하면 빌드 구성에 정의된 환경 변수도 관리할 수 있습니다.

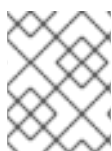
2.5.4.3.1. BuildConfig 환경 변수 및 Jenkins 작업 매개변수 간 매핑

파이프라인 전략 빌드 구성의 변경 사항에 따라 Jenkins 작업이 생성되거나 업데이트되면 빌드 구성의 모든 환경 변수가 Jenkins 작업 매개 변수 정의에 매핑됩니다. 여기서 Jenkins 작업 매개변수 정의의 기본값은 연결된 환경 변수의 현재 값입니다.

Jenkins 작업의 초기 생성 후에도 Jenkins 콘솔에서 작업에 매개변수를 추가할 수 있습니다. 매개변수 이름은 빌드 구성의 환경 변수 이름과 다릅니다. 매개변수는 해당 Jenkins 작업에 대한 빌드가 시작될 때 적용됩니다.

Jenkins 작업의 빌드를 시작하는 방법에 따라 매개변수 설정 방법이 결정됩니다.

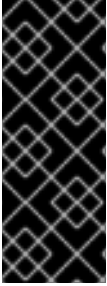
- `oc start-build`로 시작하는 경우 빌드 구성의 환경 변수 값은 해당 작업 인스턴스에 설정된 매개변수입니다. Jenkins 콘솔에서 매개변수 기본값을 변경하면 해당 변경 사항이 무시됩니다. 빌드 구성 값이 우선합니다.
- `oc start-build -e`로 시작하는 경우 `-e` 옵션에 지정된 환경 변수의 값이 우선합니다.
 - 빌드 구성에 나열되지 않은 환경 변수를 지정하면 Jenkins 작업 매개변수 정의로 추가됩니다.
 - Jenkins 콘솔에서 환경 변수에 해당하는 매개변수를 변경하면 해당 변경 사항이 무시됩니다. 빌드 구성 및 `oc start-build -e`로 지정하는 항목이 우선합니다.
- Jenkins 콘솔에서 Jenkins 작업을 시작하면 작업의 빌드를 시작하는 과정의 일부로 Jenkins 콘솔을 사용하여 매개변수 설정을 제어할 수 있습니다.



참고

빌드 구성에서 작업 매개변수와 연결할 수 있는 모든 환경 변수를 지정하는 것이 좋습니다. 이렇게 하면 디스크 I/O가 줄어들고 Jenkins 처리 중 성능이 향상됩니다.

2.5.4.4. 파이프라인 빌드 튜토리얼



중요

파이프라인 빌드 전략은 OpenShift Container Platform 4에서 더 이상 사용되지 않습니다. 동등하고 향상된 기능은 Tekton 기반 OpenShift Container Platform Pipelines에 있습니다.

OpenShift Container Platform의 Jenkins 이미지는 완전히 지원되며 사용자는 Jenkins 사용 설명서를 따라 작업에 **jenkinsfile**을 정의하거나 소스 제어 관리 시스템에 저장해야 합니다.

이 예제에서는 **nodejs-mongodb.json** 템플릿을 사용하여 **Node.js/MongoDB** 애플리케이션을 빌드, 배포, 확인할 OpenShift Container Platform Pipeline을 생성하는 방법을 보여줍니다.

프로세스

- 1. Jenkins 마스터를 생성합니다.

```
$ oc project <project_name>
```

사용할 프로젝트를 선택하거나 **oc new-project <project_name>**을 사용하여 새 프로젝트를 생성합니다.

```
$ oc new-app jenkins-ephemeral 1
```

영구 스토리지를 사용하려면 대신 **jenkins-persistent**를 사용합니다.

- 2. 다음 콘텐츠를 사용하여 **nodejs-sample-pipeline.yaml**이라는 파일을 생성합니다.



참고

이 과정에서 Jenkins Pipeline 전략을 사용하여 **Node.js/MongoDB** 예제 애플리케이션을 빌드, 배포, 스케일링하는 **BuildConfig** 오브젝트가 생성됩니다.

```
kind: "BuildConfig"
apiVersion: "v1"
metadata:
  name: "nodejs-sample-pipeline"
spec:
  strategy:
    jenkinsPipelineStrategy:
      jenkinsfile: <pipeline content from below>
    type: JenkinsPipeline
```

- 3. **jenkinsPipelineStrategy**를 사용하여 **BuildConfig** 오브젝트를 생성한 후에는 인라인 **jenkinsfile**을 사용하여 파이프라인에 수행할 작업을 지시합니다.



참고

이 예에서는 애플리케이션에 대한 Git 리포지토리를 설정하지 않습니다.

다음 **jenkinsfile** 콘텐츠는 OpenShift Container Platform DSL을 사용하여 Groovy로 작성됩니다. 소스 리포지토리에 **jenkinsfile**을 포함하는 것이 기본 방법이지만 이 예제에서는 YAML 리터럴 스타일을 사용하여 **BuildConfig** 오브젝트에 인라인 콘텐츠를 포함합니다.

```

def templatePath = 'https://raw.githubusercontent.com/openshift/nodejs-
ex/master/openshift/templates/nodejs-mongodb.json' ❶
def templateName = 'nodejs-mongodb-example' ❷
pipeline {
  agent {
    node {
      label 'nodejs' ❸
    }
  }
  options {
    timeout(time: 20, unit: 'MINUTES') ❹
  }
  stages {
    stage('preamble') {
      steps {
        script {
          openshift.withCluster() {
            openshift.withProject() {
              echo "Using project: ${openshift.project()}"
            }
          }
        }
      }
    }
    stage('cleanup') {
      steps {
        script {
          openshift.withCluster() {
            openshift.withProject() {
              openshift.selector("all", [ template : templateName ]).delete() ❺
              if (openshift.selector("secrets", templateName).exists()) { ❻
                openshift.selector("secrets", templateName).delete()
              }
            }
          }
        }
      }
    }
    stage('create') {
      steps {
        script {
          openshift.withCluster() {
            openshift.withProject() {
              openshift.newApp(templatePath) ❼
            }
          }
        }
      }
    }
    stage('build') {
      steps {
        script {
          openshift.withCluster() {
            openshift.withProject() {
              def builds = openshift.selector("bc", templateName).related('builds')
            }
          }
        }
      }
    }
  }
}

```

```

        timeout(5) { 8
            builds.untilEach(1) {
                return (it.object().status.phase == "Complete")
            }
        }
    }
}
}
}
}
}
}
}
}
stage('deploy') {
    steps {
        script {
            openshift.withCluster() {
                openshift.withProject() {
                    def rm = openshift.selector("dc", templateName).rollout()
                    timeout(5) { 9
                        openshift.selector("dc", templateName).related('pods').untilEach(1) {
                            return (it.object().status.phase == "Running")
                        }
                    }
                }
            }
        }
    }
}
stage("tag") {
    steps {
        script {
            openshift.withCluster() {
                openshift.withProject() {
                    openshift.tag("${templateName}:latest", "${templateName}-staging:latest") 10
                }
            }
        }
    }
}
}
}
}
}
}
}
}

```

- 1 사용할 템플릿의 경로입니다.
- 1 2 생성할 템플릿의 이름입니다.
- 3 이 빌드를 실행할 **node.js** 에이전트 Pod를 구동합니다.
- 4 이 파이프라인에 타임아웃을 20분으로 설정합니다.
- 5 이 템플릿 라벨이 있는 모든 항목을 삭제합니다.
- 6 이 템플릿 라벨을 사용하여 모든 보안을 삭제합니다.
- 7 **templatePath**에서 새 애플리케이션을 생성합니다.
- 8 빌드가 완료될 때까지 최대 5분 동안 기다립니다.

- 9 배포가 완료될 때까지 최대 5분 정도 기다립니다.
- 10 다른 모든 과정이 성공하면 **\$ {templateName}:latest** 이미지에 **\$ {templateName}-staging:latest** 태그를 지정합니다. 스테이징 환경에 대한 파이프라인 빌드 구성에서는 **\$ {templateName}-staging:latest** 이미지가 변경될 때까지 기다린 다음 해당 이미지를 스테이징 환경에 배포할 수 있습니다.



참고

위 예제는 선언적 파이프라인 스타일로 작성되었지만 기존에 스크립팅된 파이프라인 스타일도 지원됩니다.

- 4. OpenShift Container Platform 클러스터에서 파이프라인 **BuildConfig**를 생성합니다.

```
$ oc create -f nodejs-sample-pipeline.yaml
```

- a. 자체 파일을 생성하지 않으려면 다음을 실행하여 원래 리포지토리의 샘플을 사용하면 됩니다.

```
$ oc create -f
https://raw.githubusercontent.com/openshift/origin/master/examples/jenkins/pipeline/nodejs-sample-pipeline.yaml
```

- 5. 파이프라인을 시작합니다.

```
$ oc start-build nodejs-sample-pipeline
```



참고

또는 빌드 → 파이프라인 섹션으로 이동하여 **파이프라인 시작**을 클릭하거나 Jenkins 콘솔을 방문하여 생성한 파이프라인으로 이동한 다음 **지금 빌드**를 클릭하여 OpenShift Container Platform 웹 콘솔에서 파이프라인을 시작할 수 있습니다.

파이프라인이 시작되면 프로젝트 내에서 수행되는 다음 작업이 표시되어야 합니다.

- Jenkins 서버에서 작업 인스턴스가 생성됩니다.
- 파이프라인에 필요한 경우 에이전트 Pod가 시작됩니다.
- 파이프라인은 에이전트 Pod에서 실행되지만 에이전트가 필요하지 않은 경우에는 마스터에서 실행됩니다.
 - **template=nodejs-mongodb-example** 라벨을 사용하여 이전에 생성한 리소스는 삭제됩니다.
 - 새 애플리케이션 및 이 애플리케이션의 모든 관련 리소스는 **nodejs-mongodb-example** 템플릿에서 생성됩니다.
 - **nodejs-mongodb-example BuildConfig**를 사용하여 빌드가 시작됩니다.
 - 파이프라인은 빌드가 완료될 때까지 기다린 후 다음 단계를 트리거합니다.
 - 배포는 **nodejs-mongodb-example** 배포 구성을 사용하여 시작됩니다.

- 파이프라인은 배포가 완료될 때까지 기다린 후 다음 단계를 트리거합니다.
- 빌드 및 배포가 성공하면 **nodejs-mongodb-example:latest** 이미지에 **nodejs-mongodb-example:stage** 태그가 지정됩니다.
- 파이프라인에 필요한 경우 에이전트 pod가 삭제됩니다.



참고

파이프라인 실행을 시각화하는 가장 좋은 방법은 OpenShift Container Platform 웹 콘솔에서 확인하는 것입니다. 웹 콘솔에 로그인하고 빌드 → 파이프라인으로 이동하여 파이프라인을 확인할 수 있습니다.

2.5.5. 웹 콘솔을 사용하여 보안 추가

개인 리포지토리에 액세스할 수 있도록 빌드 구성에 보안을 추가할 수 있습니다.

프로세스

OpenShift Container Platform 웹 콘솔에서 개인 리포지토리에 액세스할 수 있도록 빌드 구성에 보안을 추가하려면 다음을 수행합니다.

1. 새 OpenShift Container Platform 프로젝트를 생성합니다.
2. 개인 소스 코드 리포지토리에 액세스하는 데 필요한 자격 증명이 포함된 보안을 생성합니다.
3. 빌드 구성을 생성합니다.
4. 빌드 구성 편집기 페이지 또는 웹 콘솔의 **빌더 이미지에서 앱 생성** 페이지에서 **소스 보안**을 설정합니다.
5. **저장**을 클릭합니다.

2.5.6. 가져오기 및 내보내기 활성화

빌드 구성에 가져오기 보안을 설정하여 프라이빗 레지스트리로 가져오고 내보내기 보안을 설정하여 내보낼 수 있습니다.

프로세스

프라이빗 레지스트리로 가져오기를 활성화하려면 다음을 수행합니다.

- 빌드 구성에 가져오기 보안을 설정합니다.

내보내기를 활성화하려면 다음을 수행합니다.

- 빌드 구성에 내보내기 보안을 설정합니다.

2.6. BUILDHAH를 사용한 사용자 정의 이미지 빌드

OpenShift Container Platform 4.8에서는 호스트 노드에 Docker 소켓이 존재하지 않습니다. 즉 사용자 정의 빌드의 *Docker* 소켓 *마운트* 옵션에서 사용자 정의 빌드 이미지 내에서 사용하도록 액세스 가능한 Docker 소켓을 제공하지 않을 수 있습니다.

이미지를 빌드하고 내보내기 위해 이 기능이 필요한 경우 사용자 정의 빌드 이미지에 Buildah 툴을 추가한 후 이 툴을 사용하여 사용자 정의 빌드 논리 내에서 이미지를 빌드하고 내보내십시오. 다음은 Buildah를 사용하여 사용자 정의 빌드를 실행하는 방법의 예입니다.



참고

사용자 정의 빌드 전략을 사용하려면 사용자가 클러스터에서 실행되는 권한 있는 컨테이너 내에서 임의의 코드를 실행할 수 있으므로 기본적으로 일반 사용자에게는 없는 권한이 필요합니다. 이 수준의 액세스 권한은 사용 시 클러스터를 손상시킬 수 있으므로 클러스터에 대한 관리 권한이 있는 신뢰할 수 있는 사용자에게만 부여해야 합니다.

2.6.1. 사전 요구 사항

- **사용자 정의 빌드 권한을 부여** 하는 방법을 검토합니다.

2.6.2. 사용자 정의 빌드 아티팩트 생성

사용자 정의 빌드 이미지로 사용할 이미지를 생성해야 합니다.

프로세스

1. 빈 디렉터리에서부터 다음 콘텐츠를 사용하여 **Dockerfile**이라는 파일을 생성합니다.

```
FROM registry.redhat.io/rhel8/buildah
# In this example, `tmp/build` contains the inputs that build when this
# custom builder image is run. Normally the custom builder image fetches
# this content from some location at build time, by using git clone as an example.
ADD dockerfile.sample /tmp/input/Dockerfile
ADD build.sh /usr/bin
RUN chmod a+x /usr/bin/build.sh
# /usr/bin/build.sh contains the actual custom build logic that will be run when
# this custom builder image is run.
ENTRYPOINT ["/usr/bin/build.sh"]
```

2. 동일한 디렉터리에서 **dockerfile.sample**이라는 파일을 생성합니다. 이 파일은 사용자 정의 빌드 이미지에 포함되어 있으며 사용자 정의 빌드에서 생성하는 이미지를 정의합니다.

```
FROM registry.access.redhat.com/ubi8/ubi
RUN touch /tmp/build
```

3. 동일한 디렉터리에 **build.sh**라는 파일을 생성합니다. 이 파일에는 사용자 정의 빌드가 실행될 때 실행되는 논리가 포함되어 있습니다.

```
#!/bin/sh
# Note that in this case the build inputs are part of the custom builder image, but normally this
# is retrieved from an external source.
cd /tmp/input
# OUTPUT_REGISTRY and OUTPUT_IMAGE are env variables provided by the custom
# build framework
TAG="${OUTPUT_REGISTRY}/${OUTPUT_IMAGE}"

# performs the build of the new image defined by dockerfile.sample
buildah --storage-driver vfs bud --isolation chroot -t ${TAG} .
```

```
# buildah requires a slight modification to the push secret provided by the service
# account to use it for pushing the image
cp /var/run/secrets/openshift.io/push/.dockercfg /tmp
(echo "{\"auths\": \"\" ; cat /var/run/secrets/openshift.io/push/.dockercfg ; echo \"}") >
/tmp/.dockercfg

# push the new image to the target for the build
buildah --storage-driver vfs push --tls-verify=false --authfile /tmp/.dockercfg ${TAG}
```

2.6.3. 사용자 정의 빌더 이미지 빌드

OpenShift Container Platform을 사용하여 사용자 정의 전략에서 사용할 사용자 정의 빌더 이미지를 빌드하고 배포할 수 있습니다.

사전 요구 사항

- 새 사용자 정의 빌더 이미지를 생성하는 데 사용할 모든 입력을 정의합니다.

프로세스

1. 사용자 정의 빌더 이미지를 빌드할 **BuildConfig** 오브젝트를 정의합니다.

```
$ oc new-build --binary --strategy=docker --name custom-builder-image
```

2. 사용자 정의 빌드 이미지를 생성한 디렉터리에서 빌드를 실행합니다.

```
$ oc start-build custom-builder-image --from-dir . -F
```

빌드가 완료되면 **custom-builder-image:latest**라는 이미지 스트림 태그의 프로젝트에서 새 사용자 정의 빌더 이미지를 사용할 수 있습니다.

2.6.4. 사용자 정의 빌더 이미지 사용

사용자 정의 빌더 이미지와 함께 사용자 정의 전략을 사용하여 사용자 정의 빌드 논리를 실행하는 **BuildConfig** 오브젝트를 정의할 수 있습니다.

사전 요구 사항

- 새 사용자 정의 빌더 이미지에 필요한 모든 입력을 정의합니다.
- 사용자 정의 빌더 이미지를 빌드합니다.

프로세스

1. **buildconfig.yaml**이라는 파일을 생성합니다. 이 파일은 프로젝트에서 생성되어 실행되는 **BuildConfig** 오브젝트를 정의합니다.

```
kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
```

```

name: sample-custom-build
labels:
  name: sample-custom-build
annotations:
  template.alpha.openshift.io/wait-for-ready: 'true'
spec:
  strategy:
    type: Custom
    customStrategy:
      forcePull: true
    from:
      kind: ImageStreamTag
      name: custom-builder-image:latest
      namespace: <yourproject> ❶
  output:
    to:
      kind: ImageStreamTag
      name: sample-custom:latest

```

❶ 프로젝트 이름을 지정합니다.

2. **BuildConfig**를 생성합니다.

```
$ oc create -f buildconfig.yaml
```

3. **imagestream.yaml**이라는 파일을 생성합니다. 이 파일은 빌드에서 이미지를 내보낼 이미지 스트림을 정의합니다.

```

kind: ImageStream
apiVersion: image.openshift.io/v1
metadata:
  name: sample-custom
spec: {}

```

4. 이미지 스트림을 생성합니다.

```
$ oc create -f imagestream.yaml
```

5. 사용자 정의 빌드를 실행합니다.

```
$ oc start-build sample-custom-build -F
```

빌드를 실행하면 빌드에서 이전에 빌드한 사용자 정의 빌더 이미지를 실행하는 Pod를 시작합니다. Pod는 사용자 정의 빌더 이미지의 진입점으로 정의된 **build.sh** 논리를 실행합니다. **build.sh** 논리는 Buildah를 호출하여 사용자 정의 빌더 이미지에 포함된 **dockerfile.sample**을 빌드한 다음 Buildah를 사용하여 새 이미지를 **sample-custom image stream**으로 내보냅니다.

2.7. 기본 빌드 수행

다음 섹션에서는 빌드 시작 및 취소, BuildConfig 삭제, 빌드 세부 정보 보기, 빌드 로그 액세스 등 기본 빌드 작업에 대한 지침을 제공합니다.

2.7.1. 빌드 시작

현재 프로젝트의 기존 빌드 구성에서 새 빌드를 수동으로 시작할 수 있습니다.

프로세스

빌드를 수동으로 시작하려면 다음 명령을 입력합니다.

```
$ oc start-build <buildconfig_name>
```

2.7.1.1. 빌드 재실행

--from-build 플래그를 사용하여 빌드를 수동으로 다시 실행할 수 있습니다.

프로세스

- 빌드를 수동으로 다시 실행하려면 다음 명령을 입력합니다.

```
$ oc start-build --from-build=<build_name>
```

2.7.1.2. 빌드 로그 스트리밍

--follow 플래그를 지정하여 빌드의 로그를 **stdout**에서 스트리밍할 수 있습니다.

프로세스

- **stdout**에서 빌드 로그를 수동으로 스트리밍하려면 다음 명령을 입력합니다.

```
$ oc start-build <buildconfig_name> --follow
```

2.7.1.3. 빌드 시작 시 환경 변수 설정

--env 플래그를 지정하여 빌드에 원하는 환경 변수를 설정할 수 있습니다.

프로세스

- 원하는 환경 변수를 지정하려면 다음 명령을 입력합니다.

```
$ oc start-build <buildconfig_name> --env=<key>=<value>
```

2.7.1.4. 소스를 사용하여 빌드 시작

빌드에 Git 소스 가져오기 또는 Dockerfile을 사용하는 대신 소스를 직접 내보내 빌드를 시작할 수 있습니다. 소스는 Git 또는 SVN 작업 디렉터리, 배포하려는 사전 빌드된 바이너리 아티팩트 세트 또는 단일 파일의 콘텐츠일 수 있습니다. 이 작업은 **start-build** 명령에 다음 옵션 중 하나를 지정하여 수행할 수 있습니다.

옵션	설명
--from-dir=<directory>	보관하여 빌드에 바이너리 입력으로 사용할 디렉터리를 지정합니다.

옵션	설명
--from-file=<file>	빌드 소스에서 유일한 파일이 될 단일 파일을 지정합니다. 이 파일은 제공된 원래 파일과 파일 이름이 동일한 빈 디렉터리의 루트에 배치됩니다.
--from-repo=<local_source_repo>	빌드에 바이너리 입력으로 사용할 로컬 리포지토리의 경로를 지정합니다. 빌드에 사용되는 분기, 태그 또는 커밋을 제어하는 --commit 옵션을 추가합니다.

이러한 옵션을 빌드에 직접 전달하면 해당 콘텐츠가 빌드로 스트리밍되어 현재 빌드 소스 설정을 덮어씁니다.



참고

바이너리 입력에서 트리거된 빌드는 서버의 소스를 유지하지 않으므로 기본 이미지 변경에 의해 트리거된 리빌드는 빌드 구성에 지정된 소스를 사용합니다.

프로세스

- 다음 명령을 통해 소스에서 빌드를 시작하여 태그 **v2**의 아카이브로 로컬 Git 리포지토리의 콘텐츠를 보냅니다.

```
$ oc start-build hello-world --from-repo=./hello-world --commit=v2
```

2.7.2. 빌드 취소

웹 콘솔을 사용하거나 다음 CLI 명령을 사용하여 빌드를 취소할 수 있습니다.

프로세스

- 빌드를 수동으로 취소하려면 다음 명령을 입력합니다.

```
$ oc cancel-build <build_name>
```

2.7.2.1. 여러 빌드 취소

다음 CLI 명령으로 여러 빌드를 취소할 수 있습니다.

프로세스

- 여러 빌드를 수동으로 취소하려면 다음 명령을 입력합니다.

```
$ oc cancel-build <build1_name> <build2_name> <build3_name>
```

2.7.2.2. 모든 빌드 취소

다음 CLI 명령을 사용하여 빌드 구성에서 모든 빌드를 취소할 수 있습니다.

프로세스

- 모든 빌드를 취소하려면 다음 명령을 입력합니다.

```
$ oc cancel-build bc/<buildconfig_name>
```

2.7.2.3. 지정된 상태의 모든 빌드 취소

지정된 상태(**new** 또는 **pending** 등)의 빌드는 모두 취소하고 다른 상태의 빌드는 무시할 수 있습니다.

프로세스

- 지정된 상태의 빌드를 모두 취소하려면 다음 명령을 입력합니다.

```
$ oc cancel-build bc/<buildconfig_name>
```

2.7.3. BuildConfig 삭제

다음 명령을 사용하여 **BuildConfig**를 삭제할 수 있습니다.

프로세스

- **BuildConfig**를 삭제하려면 다음 명령을 입력합니다.

```
$ oc delete bc <BuildConfigName>
```

이 명령은 이 **BuildConfig**에서 인스턴스화한 빌드도 모두 삭제합니다.

- **BuildConfig**를 삭제하고 **BuildConfig**에서 인스턴스화한 빌드는 유지하려면 다음 명령을 입력할 때 **--cascade=false** 플래그를 지정하십시오.

```
$ oc delete --cascade=false bc <BuildConfigName>
```

2.7.4. 빌드 세부 정보 보기

웹 콘솔을 사용하거나 **oc describe** CLI 명령을 사용하여 빌드 세부 정보를 볼 수 있습니다.

다음은 포함된 정보가 표시됩니다.

- 빌드 소스입니다.
- 빌드 전략입니다.
- 출력 대상입니다.
- 대상 레지스트리의 이미지 다이제스트입니다.
- 빌드를 생성한 방법입니다.

빌드에서 **Docker** 또는 **Source** 전략을 사용하는 경우 **oc describe** 출력에 커밋 ID, 작성자, 커밋, 메시지 등 해당 빌드에 사용된 소스 리버전 정보도 포함됩니다.

프로세스

- 빌드 세부 정보를 보려면 다음 명령을 입력합니다.

```
$ oc describe build <build_name>
```

2.7.5. 빌드 로그에 액세스

웹 콘솔 또는 CLI를 사용하여 빌드 로그에 액세스할 수 있습니다.

프로세스

- 빌드를 직접 사용하여 로그를 스트리밍하려면 다음 명령을 입력합니다.

```
$ oc describe build <build_name>
```

2.7.5.1. BuildConfig 로그에 액세스

웹 콘솔 또는 CLI를 사용하여 **BuildConfig** 로그에 액세스할 수 있습니다.

프로세스

- **BuildConfig**의 최신 빌드 로그를 스트리밍하려면 다음 명령을 입력합니다.

```
$ oc logs -f bc/<buildconfig_name>
```

2.7.5.2. 특정 버전 빌드의 BuildConfig 로그에 액세스

웹 콘솔 또는 CLI를 사용하여 특정 버전 빌드의 **BuildConfig** 로그에 액세스할 수 있습니다.

프로세스

- 특정 버전 빌드의 **BuildConfig** 로그를 스트리밍하려면 다음 명령을 입력합니다.

```
$ oc logs --version=<number> bc/<buildconfig_name>
```

2.7.5.3. 로그 세부 정보 표시 활성화

BuildConfig에서 **sourceStrategy** 또는 **dockerStrategy**의 일부로 **BUILD_LOGLEVEL** 환경 변수를 전달하여 더 세부적인 출력을 제공할 수 있습니다.



참고

관리자는 **env/BUILD_LOGLEVEL**을 구성하여 전체 OpenShift Container Platform 인스턴스에 대한 기본 빌드 세부 정보 표시 수준을 설정할 수 있습니다. 이 기본값은 지정된 **BuildConfig**에 **BUILD_LOGLEVEL**을 지정하여 덮어쓸 수 있습니다. 명령줄에서 **--build-loglevel**을 **oc start-build**로 전달하여 바이너리가 아닌 빌드에 더 높은 우선순위를 지정할 수 있습니다.

소스 빌드에 사용 가능한 로그 수준은 다음과 같습니다.

수준 0	assemble 스크립트를 실행하는 컨테이너의 출력과 발생한 모든 오류를 생성합니다. 이는 기본값입니다.
------	---

수준 1	실행된 프로세스에 대한 기본 정보를 생성합니다.
수준 2	실행된 프로세스에 대한 매우 자세한 정보를 생성합니다.
수준 3	실행된 프로세스에 대한 자세한 정보와 아카이브 콘텐츠 목록을 생성합니다.
수준 4	현재는 수준 3과 동일한 정보를 제공합니다.
수준 5	위 수준에서 언급한 모든 정보를 생성하고 Docker 내보내기 메시지도 제공합니다.

프로세스

- 더 세부적인 출력을 제공하기 위해 **BuildConfig**에서 **sourceStrategy** 또는 **dockerStrategy**의 일부로 **BUILD_LOGLEVEL** 환경 변수를 전달합니다.

```
sourceStrategy:
...
env:
  - name: "BUILD_LOGLEVEL"
    value: "2" 1
```

1 이 값을 원하는 로그 수준으로 조정합니다.

2.8. 빌드 트리거 및 수정

다음 섹션에서는 빌드 후크를 사용하여 빌드를 트리거하고 수정하는 방법을 간략하게 설명합니다.

2.8.1. 빌드 트리거

BuildConfig를 정의할 때 **BuildConfig**를 실행해야 하는 상황을 제어하기 위해 트리거를 정의할 수 있습니다. 다음과 같은 빌드 트리거를 사용할 수 있습니다.

- Webhook
- 이미지 변경
- 구성 변경

2.8.1.1. Webhook 트리거

Webhook 트리거를 사용하면 OpenShift Container Platform API 끝점에 요청을 전송하여 새 빌드를 트리거할 수 있습니다. 이러한 트리거는 GitHub, GitLab, Bitbucket 또는 Generic Webhook를 사용하여 정의할 수 있습니다.

현재는 OpenShift Container Platform Webhook에서 Git 기반 SCM(소스 코드 관리) 시스템 각각에 대해 유사한 버전의 내보내기 이벤트만 지원합니다. 기타 모든 이벤트 유형은 무시됩니다.

내보내기 이벤트가 처리되면 OpenShift Container Platform 컨트롤 플레인 호스트 (마스터 호스트라고도 함)에서 이벤트 내부의 분기 참조가 해당 **BuildConfig**의 분기 참조와 일치하는지 확인합니다. 일치하는 경우 OpenShift Container Platform 빌드의 Webhook 이벤트에 언급된 커밋 참조가 정확한지 확인합니다. 일치하지 않는 경우에는 빌드가 트리거되지 않습니다.



참고

oc new-app 및 **oc new-build**는 GitHub 및 Generic Webhook 트리거를 자동으로 생성하지만 필요한 다른 Webhook 트리거는 수동으로 추가해야 합니다. 트리거 설정을 통해 트리거를 수동으로 추가할 수 있습니다.

모든 Webhook에 대해 **WebHookSecretKey**라는 키와 Webhook를 호출할 때 제공할 값이 되는 값을 사용하여 보안을 정의해야 합니다. 그런 다음 Webhook 정의에서 보안을 참조해야 합니다. 보안을 사용하면 URL의 고유성이 보장되어 다른 사용자가 빌드를 트리거할 수 없습니다. 키 값은 Webhook 호출 중 제공되는 보안과 비교됩니다.

예를 들면 아래 예제에는 **mysecret**이라는 보안에 대한 참조가 포함된 GitHub Webhook가 있습니다.

```
type: "GitHub"
github:
  secretReference:
    name: "mysecret"
```

그런 다음 보안이 다음과 같이 정의됩니다. 보안 값은 **Secret** 오브젝트의 **data** 필드에 필요하므로 base64로 인코딩됩니다.

```
- kind: Secret
  apiVersion: v1
  metadata:
    name: mysecret
    creationTimestamp:
  data:
    WebHookSecretKey: c2VjcmV0dmFsdWUx
```

2.8.1.1.1. GitHub Webhook 사용

GitHub Webhook는 리포지토리를 업데이트할 때 GitHub에서 생성하는 호출을 처리합니다. 트리거를 정의할 때는 보안을 지정해야 하는데, 보안은 Webhook를 구성할 때 GitHub에 제공하는 URL의 일부입니다.

GitHub Webhook 정의의 예:

```
type: "GitHub"
github:
  secretReference:
    name: "mysecret"
```



참고

Webhook 트리거 구성에 사용되는 보안은 GitHub UI에서 Webhook를 구성할 때 표시되는 **secret** 필드와 동일하지 않습니다. 전자는 Webhook URL을 고유하고 예측하기 어렵게 만들고, 후자는 **X-Hub-Signature** 헤더로 전송되는 본문의 HMAC 16진수 다이제스트를 생성하는 데 사용되는 선택적 문자열 필드입니다.

페이로드 URL은 **oc describe** 명령에 의해 GitHub Webhook URL로 반환되고(Webhook URL 표시 참조) 다음과 같이 구성됩니다.

출력 예

```
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/github
```

사전 요구 사항

- GitHub 리포지토리에서 **BuildConfig**를 생성합니다.

프로세스

1. GitHub Webhook를 구성하려면 다음을 수행합니다.

- a. GitHub 리포지토리에서 **BuildConfig**를 생성한 후 다음을 실행합니다.

```
$ oc describe bc/<name-of-your-BuildConfig>
```

그러면 다음과 같은 Webhook GitHub URL이 생성됩니다.

출력 예

```
<https://api.starter-us-east-1.openshift.com:443/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/github
```

- b. GitHub 웹 콘솔에서 이 URL을 잘라내어 GitHub에 붙여넣습니다.
- c. GitHub 리포지토리의 **설정** → **Webhook**에서 **Webhook** 추가를 선택합니다.
- d. URL 출력을 **페이로드 URL** 필드에 붙여넣습니다.
- e. GitHub의 기본 **application/x-www-form-urlencoded**에서 **콘텐츠 유형**을 **application/json**으로 변경합니다.
- f. **Webhook** 추가를 클릭합니다.
GitHub에서 Webhook가 성공적으로 구성되었음을 알리는 메시지가 표시됩니다.

이제 GitHub 리포지토리에 변경 사항을 내보내면 새 빌드가 자동으로 시작되고 빌드가 성공하면 새 배포가 시작됩니다.



참고

Gogs는 GitHub와 동일한 Webhook 페이로드 형식을 지원합니다. 따라서 Gogs 서버를 사용하는 경우 **BuildConfig**에 GitHub Webhook 트리거를 정의하고 Gogs 서버에서도 트리거할 수 있습니다.

2. **payload.json**과 같이 유효한 JSON 페이로드가 포함된 파일의 경우 **curl**을 사용하여 Webhook를 수동으로 트리거할 수 있습니다.

```
$ curl -H "X-GitHub-Event: push" -H "Content-Type: application/json" -k -X POST --data-binary @payload.json https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/github
```

-k 인수는 API 서버에 올바르게 서명된 인증서가 없는 경우에만 필요합니다.

추가 리소스

- [Gogs](#)

2.8.1.1.2. GitLab Webhook 사용

GitLab Webhook는 리포지토리를 업데이트할 때 GitLab에서 생성하는 호출을 처리합니다. GitHub 트리거와 마찬가지로 보안을 지정해야 합니다. 다음 예제는 **BuildConfig** 내의 트리거 정의 YAML입니다.

```
type: "GitLab"
gitlab:
  secretReference:
    name: "mysecret"
```

페이로드 URL은 **oc describe** 명령을 통해 GitLab Webhook URL로 반환되고 다음과 같이 구조화됩니다.

출력 예

```
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/gitlab
```

프로세스

1. GitLab Webhook를 구성하려면 다음을 수행합니다.
 - a. Webhook URL을 가져오도록 **BuildConfig**를 지정합니다.


```
$ oc describe bc <name>
```
 - b. Webhook URL을 복사하여 **<secret>**을 보안 값으로 교체합니다.
 - c. [GitLab 설정 지침](#)에 따라 Webhook URL을 GitLab 리포지토리 설정에 붙여넣습니다.
2. **payload.json**과 같이 유효한 JSON 페이로드가 포함된 파일의 경우 **curl**을 사용하여 Webhook를 수동으로 트리거할 수 있습니다.

```
$ curl -H "X-GitLab-Event: Push Hook" -H "Content-Type: application/json" -k -X POST --
data-binary @payload.json
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildcon
gs/<name>/webhooks/<secret>/gitlab
```

-k 인수는 API 서버에 올바르게 서명된 인증서가 없는 경우에만 필요합니다.

2.8.1.1.3. Bitbucket Webhook 사용

Bitbucket Webhook는 리포지토리가 업데이트될 때 Bitbucket에서 만든 호출을 처리합니다. 이전 트리거와 유사하게 보안을 지정해야 합니다. 다음 예제는 **BuildConfig** 내의 트리거 정의 YAML입니다.

```
type: "Bitbucket"
bitbucket:
  secretReference:
    name: "mysecret"
```

페이로드 URL은 oc describe 명령을 통해 BuildConfig 리소스의 URL로 반환되고 다음과 같이 구조화됩니다.

페이로드 URL은 **oc describe** 명령을 통해 Bitbucket Webhook URL로 반환되고 다음과 같이 구조화됩니다.

출력 예

```
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/bitbucket
```

프로세스

1. Bitbucket Webhook를 구성하려면 다음을 수행합니다.
 - a. Webhook URL을 가져오도록 'BuildConfig'를 지정합니다.


```
$ oc describe bc <name>
```
 - b. Webhook URL을 복사하여 **<secret>**을 보안 값으로 교체합니다.
 - c. [Bitbucket 설정 지침](#)에 따라 Webhook URL을 Bitbucket 리포지토리 설정에 붙여넣습니다.
2. **payload.json**과 같이 유효한 JSON 페이로드가 포함된 파일의 경우 **curl**을 사용하여 Webhook를 수동으로 트리거할 수 있습니다.

```
$ curl -H "X-Event-Key: repo:push" -H "Content-Type: application/json" -k -X POST --data-binary @payload.json https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/bitbucket
```

-k 인수는 API 서버에 올바르게 서명된 인증서가 없는 경우에만 필요합니다.

2.8.1.1.4. 일반 Webhook 사용

일반 Webhook는 웹 요청을 수행할 수 있는 모든 시스템에서 호출합니다. 다른 Webhook와 마찬가지로 보안을 지정해야 하는데 보안을 호출자가 빌드를 트리거하는 데 사용해야 하는 URL의 일부입니다. 보안을 사용하면 URL의 고유성이 보장되어 다른 사용자가 빌드를 트리거할 수 없습니다. 다음은 **BuildConfig** 내 트리거 정의 YAML의 예입니다.

```
type: "Generic"
generic:
  secretReference:
    name: "mysecret"
  allowEnv: true 1
```

- 1** 일반 Webhook에서 환경 변수를 전달하도록 허용하려면 **true**로 설정합니다.

프로세스

1. 호출자를 설정하기 위해 빌드에 대한 일반 Webhook 끝점의 URL을 호출 시스템에 제공합니다.

출력 예

```
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildconfigs/<name>/webhooks/<secret>/generic
```

호출자는 Webhook를 **POST** 작업으로 호출해야 합니다.

2. Webhook를 수동으로 호출하려면 **curl**을 사용하면 됩니다.

```
$ curl -X POST -k
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildcon
gs/<name>/webhooks/<secret>/generic
```

HTTP 동사를 **POST**로 설정해야 합니다. 비보안 **-k** 플래그는 인증서 검증을 무시하도록 지정됩니다. 클러스터에 올바르게 서명된 인증서가 있는 경우 이 두 번째 플래그는 필요하지 않습니다.

끝점은 다음 형식을 사용하여 선택적 페이로드를 허용할 수 있습니다.

```
git:
  uri: "<url to git repository>"
  ref: "<optional git reference>"
  commit: "<commit hash identifying a specific git commit>"
  author:
    name: "<author name>"
    email: "<author e-mail>"
  committer:
    name: "<committer name>"
    email: "<committer e-mail>"
  message: "<commit message>"
env: ①
  - name: "<variable name>"
    value: "<variable value>"
```

- ① **BuildConfig** 환경 변수와 유사하게 여기에 정의된 환경 변수도 빌드에 사용할 수 있습니다. 이러한 변수가 **BuildConfig** 환경 변수와 충돌하는 경우 해당 변수가 우선합니다. 기본적으로 Webhook에서 전달하는 환경 변수는 무시됩니다. 이 동작을 활성화하려면 Webhook 정의에서 **allowEnv** 필드를 **true**로 설정합니다.

3. **curl**을 사용하여 이 페이로드를 전달하려면 **payload_file.yaml**이라는 파일에 페이로드를 정의하고 다음을 실행합니다.

```
$ curl -H "Content-Type: application/yaml" --data-binary @payload_file.yaml -X POST -k
https://<openshift_api_host:port>/apis/build.openshift.io/v1/namespaces/<namespace>/buildcon
gs/<name>/webhooks/<secret>/generic
```

인수는 위 예제와 동일하고 헤더와 페이로드가 추가되었습니다. **-H** 인수는 페이로드 형식에 따라 **Content-Type** 헤더를 **application/yaml** 또는 **application/json**으로 설정합니다. **--data-binary** 인수는 **POST** 요청을 사용하여 온전한 새 줄로 바이너리 페이로드를 보내는 데 사용됩니다.



참고

OpenShift Container Platform에서는 유효하지 않은 요청 페이로드(예: 유효하지 않은 콘텐츠 유형, 구문 분석할 수 없거나 유효하지 않은 콘텐츠 등)가 제공되는 경우에도 일반 Webhook에서 빌드를 트리거할 수 있습니다. 이 동작은 이전 버전과의 호환성을 위해 유지됩니다. 유효하지 않은 요청 페이로드가 제공되면 OpenShift Container Platform에서 **HTTP 200 OK** 응답의 일부로 JSON 형식의 알림을 반환합니다.

2.8.1.1.5. Webhook URL 표시

다음 명령을 사용하여 빌드 구성과 관련된 Webhook URL을 표시할 수 있습니다. 이 명령에서 Webhook URL을 표시하지 않으면 해당 빌드 구성에 Webhook 트리거가 정의되지 않습니다.

프로세스

- **BuildConfig**와 관련된 Webhook URL을 표시하려면 다음을 실행합니다.

```
$ oc describe bc <name>
```

2.8.1.2. 이미지 변경 트리거 사용

개발자는 기본 이미지가 변경될 때마다 자동으로 실행되도록 빌드를 구성할 수 있습니다.

이미지 변경 트리거를 사용하여 업스트림 이미지의 새 버전을 사용할 수 있을 때 빌드를 자동으로 호출할 수 있습니다. 예를 들어 빌드가 RHEL 이미지를 기반으로 하는 경우 해당 빌드를 트리거하여 RHEL 이미지를 변경할 때마다 실행할 수 있습니다. 결과적으로 애플리케이션 이미지는 항상 최신 RHEL 기본 이미지에서 실행됩니다.



참고

[v1 컨테이너 레지스트리](#)의 컨테이너 이미지를 가리키는 이미지 스트림은 이미지 스트림 태그를 사용할 수 있을 때 빌드를 한 번만 트리거하고 후속 이미지 업데이트에서는 빌드를 트리거하지 않습니다. 이는 v1 컨테이너 레지스트리에서 고유하게 확인할 수 있는 이미지가 없기 때문입니다.

절차

1. 트리거로 사용하려는 업스트림 이미지를 가리키는 **ImageStream**을 정의합니다.

```
kind: "ImageStream"
apiVersion: "v1"
metadata:
  name: "ruby-20-centos7"
```

이는 **<system-registry>/<namespace>/ruby-20-centos7**에 있는 컨테이너 이미지 리포지토리에 연결된 이미지 스트림을 정의합니다. **<system-registry>**는 OpenShift Container Platform에서 실행 중인 **docker-registry**라는 이름을 사용하여 서비스로 정의됩니다.

2. 이미지 스트림이 빌드의 기본 이미지인 경우 빌드 전략의 **from** 필드를 **ImageStream**을 가리키도록 설정합니다.

```
strategy:
  sourceStrategy:
    from:
      kind: "ImageStreamTag"
      name: "ruby-20-centos7:latest"
```

이 경우 **sourceStrategy** 정의에서는 이 네임스페이스 내에 있는 **ruby-20-centos7**이라는 이미지 스트림의 **latest** 태그를 사용합니다.

3. **ImageStreams**를 가리키는 하나 이상의 트리거를 사용하여 빌드를 정의합니다.

```
type: "ImageChange" 1
imageChange: {}
```

```

type: "ImageChange" 2
imageChange:
  from:
    kind: "ImageStreamTag"
    name: "custom-image:latest"

```

- 1 빌드 전략의 **from** 필드에 정의된 **ImageStream** 및 **Tag**를 모니터링하는 이미지 변경 트리거입니다. 여기에서 **imageChange** 오브젝트는 비어 있어야 합니다.
- 2 임의의 이미지 스트림을 모니터링하는 이미지 변경 트리거입니다. 이 경우 **imageChange** 부분에 모니터링할 **ImageStreamTag**를 참조하는 **from** 필드를 포함해야 합니다.

전략 이미지 스트림에 이미지 변경 트리거를 사용하는 경우 생성된 빌드에 해당 태그와 일치하는 최신 이미지를 가리키는 변경 불가능한 Docker 태그가 제공됩니다. 이 새 이미지 참조는 빌드에 대해 실행할 때 전략에서 사용합니다.

전략 이미지를 참조하지 않는 다른 이미지 변경 트리거의 경우 새 빌드가 시작되지만 빌드 전략은 고유 이미지 참조로 업데이트되지 않습니다.

이 예제에는 전략에 대한 이미지 변경 트리거가 있으므로 결과 빌드는 다음과 같습니다.

```

strategy:
  sourceStrategy:
    from:
      kind: "DockerImage"
      name: "172.30.17.3:5001/mynamespace/ruby-20-centos7:<immutableid>"

```

그 결과 트리거된 빌드는 리포지토리로 방금 푸시된 새 이미지를 사용하고 동일한 입력을 사용하여 빌드를 다시 실행할 수 있습니다.

이미지 변경 트리거를 일시 정지하여 빌드를 시작하기 전에 참조 이미지 스트림에 대한 다양한 변경 사항을 허용할 수 있습니다. 처음에 **ImageChangeTrigger**를 **BuildConfig**에 추가할 때 빌드가 즉시 트리거되지 않도록 **paused** 특성을 **true**로 설정할 수도 있습니다.

```

type: "ImageChange"
imageChange:
  from:
    kind: "ImageStreamTag"
    name: "custom-image:latest"
  paused: true

```

모든 **Strategy** 유형의 이미지 필드를 설정하는 것 외에 사용자 지정 빌드의 경우 **OPENSIFT_CUSTOM_BUILD_BASE_IMAGE** 환경 변수도 확인합니다. 존재하지 않는 경우 변경 불가능한 이미지 참조를 사용하여 생성됩니다. 존재하는 경우 변경 불가능한 이미지 참조를 사용하여 업데이트됩니다.

Webhook 트리거 또는 수동 요청으로 인해 빌드가 트리거되는 경우 생성된 빌드는 **Strategy**에서 참조한 **ImageStream**의 확인된 **<immutableid>**를 사용합니다. 그러면 쉽게 재현할 수 있도록 일관된 이미지 태그를 사용하여 빌드를 수행할 수 있습니다.

추가 리소스

- [v1 컨테이너 레지스트리](#)

2.8.1.3. 빌드의 이미지 변경 트리거 식별

개발자는 이미지 변경 트리거가 있는 경우 마지막 빌드를 시작한 이미지 변경 사항을 확인할 수 있습니다. 이는 빌드를 디버깅하거나 문제 해결하는 데 유용할 수 있습니다.

BuildConfig 예

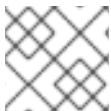
```

apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: bc-ict-example
  namespace: bc-ict-example-namespace
spec:
  # ...

  triggers:
  - imageChange:
    from:
      kind: ImageStreamTag
      name: input:latest
      namespace: bc-ict-example-namespace
  - imageChange:
    from:
      kind: ImageStreamTag
      name: input2:latest
      namespace: bc-ict-example-namespace
    type: ImageChange
status:
  imageChangeTriggers:
  - from:
    name: input:latest
    namespace: bc-ict-example-namespace
    lastTriggerTime: "2021-06-30T13:47:53Z"
    lastTriggeredImageID: image-registry.openshift-image-registry.svc:5000/bc-ict-example-namespace/input@sha256:0f88ffbeb9d25525720bfa3524cb1bf0908b7f791057cf1acfae917b11266a69
  - from:
    name: input2:latest
    namespace: bc-ict-example-namespace
    lastTriggeredImageID: image-registry.openshift-image-registry.svc:5000/bc-ict-example-namespace/input2@sha256:0f88ffbeb9d25525720bfa3524cb2ce0908b7f791057cf1acfae917b11266a69

  lastVersion: 1

```



참고

이 예제에서는 이미지 변경 트리거와 관련이 없는 요소를 생략합니다.

사전 요구 사항

- 여러 이미지 변경 트리거를 구성했습니다. 이러한 트리거는 하나 이상의 빌드를 트리거했습니다.

절차

1. `buildConfig.status.imageChangeTriggers`에서 최신 타임 스탬프가 있는 `lastTriggerTime`을 식별합니다.

ImageChangeTriggerStatus

Then you use the ``name`` and ``namespace`` from that build to find the corresponding image change trigger in ``buildConfig.spec.triggers``.

2. `imageChangeTriggers`에서 타임스탬프를 비교하여 최신 정보를 식별합니다.

이미지 변경 트리거

빌드 구성에서 `buildConfig.spec.triggers`는 빌드 트리거 정책인 `BuildTriggerPolicy`의 배열입니다.

각 `BuildTriggerPolicy`에는 `type` 필드와 포인터 필드 세트가 있습니다. 각 포인터 필드는 `type` 필드에 허용된 값 중 하나에 해당합니다. 따라서 `BuildTriggerPolicy`를 하나의 포인터 필드로만 설정할 수 있습니다.

이미지 변경 트리거의 경우 `type` 값은 `ImageChange`입니다. 그런 다음 `imageChange` 필드는 다음 필드가 있는 `ImageChangeTrigger` 오브젝트의 포인터입니다.

- **lastTriggeredImageID**: 예제에 표시되지 않는 이 필드는 OpenShift Container Platform 4.8에서 더 이상 사용되지 않으며 향후 릴리스에서는 무시됩니다. 이 `BuildConfig`에서 마지막 빌드가 트리거된 경우 `ImageStreamTag`에 대한 확인된 이미지 참조가 포함됩니다.
- **paused**: 예제에 표시되지 않는 이 필드를 사용하여 이 특정 이미지 변경 트리거를 일시적으로 비활성화할 수 있습니다.
- **from**: 이 필드를 사용하여 이 이미지 변경 트리거를 구동하는 `ImageStreamTag`를 참조합니다. 유형은 핵심 Kubernetes 유형인 `OwnerReference`입니다.

`from` 필드에는 다음 필드가 있습니다. 종류: 이미지 변경 트리거의 경우 지원되는 유일한 값은 `ImageStreamTag`입니다. 네임 스페이스: 이 필드를 사용하여 `ImageStreamTag`의 네임스페이스를 지정합니다. ** 이름: 이 필드를 사용하여 `ImageStreamTag`를 지정합니다.

이미지 변경 트리거 상태

빌드 구성에서 `buildConfig.status.imageChangeTriggers`는 `ImageChangeTriggerStatus` 요소의 배열입니다. 각 `ImageChangeTriggerStatus` 요소에는 앞의 예에서 표시된 `from`, `lastTriggeredImageID` 및 `lastTriggerTime` 요소가 포함됩니다.

가장 최근의 `lastTriggerTime`이 있는 `ImageChangeTriggerStatus`가 가장 최근 빌드를 트리거했습니다. 해당 `name` 및 `namespace`를 사용하여 빌드를 트리거한 `buildConfig.spec.triggers`에서 이미지 변경 트리거를 식별합니다.

가장 최근 타임스탬프를 사용하는 `lastTriggerTime`은 마지막 빌드의 `ImageChangeTriggerStatus`를 나타냅니다. 이 `ImageChangeTriggerStatus`에는 빌드를 트리거한 `buildConfig.spec.triggers`의 이미지 변경 트리거와 동일한 `name`과 `namespace`가 있습니다.

추가 리소스

- [v1 컨테이너 레지스트리](#)

2.8.1.4. 구성 변경 트리거

구성 변경 트리거를 사용하면 새 **BuildConfig**가 생성되는 즉시 빌드가 자동으로 호출됩니다.

다음은 **BuildConfig** 내 트리거 정의 **YAML**의 예입니다.

```
type: "ConfigChange"
```



참고

구성 변경 트리거는 현재 새 **BuildConfig**를 생성할 때만 작동합니다. 향후 릴리스에서는 **BuildConfig**가 업데이트될 때마다 구성 변경 트리거도 빌드를 시작할 수 있습니다.

2.8.1.4.1. 트리거 수동 설정

oc set triggers를 사용하여 빌드 구성에서 트리거를 추가 및 제거할 수 있습니다.

프로세스

- 빌드 구성에 **GitHub Webhook** 트리거를 설정하려면 다음을 사용합니다.

```
$ oc set triggers bc <name> --from-github
```

- 이미지 변경 트리거를 설정하려면 다음을 사용합니다.

```
$ oc set triggers bc <name> --from-image='<image>'
```

- 트리거를 제거하려면 **--remove**를 추가합니다.

```
$ oc set triggers bc <name> --from-bitbucket --remove
```



참고

Webhook 트리거가 이미 있는 경우 해당 트리거를 다시 추가하면 **Webhook** 보안이 다시 생성됩니다.

자세한 내용은 다음을 실행하여 도움말 문서를 참조하십시오.

```
$ oc set triggers --help
```

2.8.2. 빌드 후크

빌드 후크를 사용하면 빌드 프로세스에 동작을 삽입할 수 있습니다.

BuildConfig 오브젝트의 **postCommit** 필드는 빌드 출력 이미지를 실행하는 임시 컨테이너 내에서 명령을 실행합니다. 후크는 이미지의 마지막 계층을 커밋한 직후 그리고 이미지를 레지스트리로 푸시되기 전에 실행됩니다.

현재 작업 디렉터리는 이미지의 **WORKDIR**로 설정되어 있으며 이는 컨테이너 이미지의 기본 작업 디렉터리입니다. 대부분의 이미지에서 이 디렉터리는 소스 코드가 있는 위치입니다.

스크립트 또는 명령에서 **0**이 아닌 종료 코드를 반환하거나 임시 컨테이너를 시작하지 못하는 경우 후크가 실패합니다. 후크가 실패하면 빌드가 실패로 표시되고 이미지를 레지스트리로 푸시하지 않습니다. 실패 이유는 빌드 로그를 확인하여 검사할 수 있습니다.

빌드 후크를 사용하면 빌드를 완료로 표시하고 레지스트리에 이미지를 제공하기 전에 단위 테스트를 실행하여 이미지를 확인할 수 있습니다. 모든 테스트를 통과하고 테스트 실행기에서 종료 코드 **0**을 반환하면 빌드가 성공으로 표시됩니다. 실패한 테스트가 있는 경우 빌드가 실패로 표시됩니다. 어떠한 경우든 빌드 로그에는 테스트 실행기의 출력이 포함되므로 실패한 테스트를 확인할 수 있습니다.

postCommit 후크는 테스트 실행뿐만 아니라 다른 명령에도 사용할 수 있습니다. 이 후크는 임시 컨테이너에서 실행되기 때문에 후크에 의한 변경 사항은 지속되지 않습니다. 즉 후크를 실행해도 최종 이미지에는 영향을 미치지 않습니다. 이러한 동작으로 인해 특히 자동으로 삭제되어 최종 이미지에 존재하지 않는 테스트 종속 항목을 설치하고 사용할 수 있습니다.

2.8.2.1. post-commit 빌드 후크 구성

빌드 후 후크를 구성하는 방법은 다양합니다. 다음 예제에서 모든 양식은 동일하고 **bundle exec rake test --verbose**를 실행합니다.

프로세스

-

셸 스크립트:

```
postCommit:
  script: "bundle exec rake test --verbose"
```

script 값은 `/bin/sh -ic`를 사용하여 실행할 셸 스크립트입니다. 셸 스크립트가 빌드 후크를 실행하는 데 적합한 경우 이 값을 사용합니다. 예를 들면 위와 같이 단위 테스트를 실행하는 경우입니다. 이미지 항목 지점을 제어하려는 경우 또는 이미지에 `/bin/sh`가 없는 경우 **command** 및/또는 **args**를 사용합니다.



참고

추가 **-i** 플래그는 **CentOS** 및 **RHEL** 이미지 작업 환경을 개선하기 위해 도입되었으며 향후 릴리스에서 제거될 수 있습니다.

-

이미지 진입점으로서의 명령:

```
postCommit:
  command: ["/bin/bash", "-c", "bundle exec rake test --verbose"]
```

이 양식에서 **command**는 실행할 명령에 해당하며 [Dockerfile 참조](#)에 설명된 **exec** 형식의 이미지 진입점을 덮어씁니다. 이 명령은 이미지에 `/bin/sh`가 없거나 셸을 사용하지 않는 경우 필요합니다. 다른 모든 경우에는 **script**를 사용하는 것이 더 편리할 수 있습니다.

-

인수가 있는 명령:

```
postCommit:
  command: ["bundle", "exec", "rake", "test"]
  args: ["--verbose"]
```

이 형식은 **command**에 인수를 추가하는 것과 동일합니다.



참고

`script`와 `command`를 동시에 제공하면 유효하지 않은 빌드 후크가 생성됩니다.

2.8.2.2. CLI를 사용하여 `post-commit` 빌드 후크 설정

`oc set build-hook` 명령은 빌드 설정에 빌드 후크를 설정하는 데 사용할 수 있습니다.

프로세스

1.

명령을 `post-commit` 빌드 후크로 설정하려면 다음을 실행합니다.

```
$ oc set build-hook bc/mybc \
  --post-commit \
  --command \
  -- bundle exec rake test --verbose
```

2.

스크립트를 `post-commit` 빌드 후크로 설정하려면 다음을 실행합니다.

```
$ oc set build-hook bc/mybc --post-commit --script="bundle exec rake test --verbose"
```

2.9. 고급 빌드 수행

다음 섹션에서는 빌드 리소스 및 최대 기간 설정, 노드에 빌드 할당, 빌드 연결, 빌드 정리, 빌드 실행 정책 등 고급 빌드 작업에 대한 지침을 제공합니다.

2.9.1. 빌드 리소스 설정

기본적으로 빌드는 `Pod`에서 메모리 및 `CPU`와 같이 바인딩되지 않은 리소스를 사용하여 완료합니다. 이러한 리소스는 제한될 수 있습니다.

프로세스

다음 두 가지 방법으로 리소스 사용을 제한할 수 있습니다.

- 프로젝트의 기본 컨테이너 제한에 리소스 제한을 지정하여 리소스 사용 제한을 제한합니다.

리소스 제한을 빌드 구성의 일부로 지정하여 리소스 사용을 제한합니다. **다음 예제에서는 각 **resources, cpu, memory** 매개변수가 선택 사항입니다.

```

apiVersion: "v1"
kind: "BuildConfig"
metadata:
  name: "sample-build"
spec:
  resources:
    limits:
      cpu: "100m" ①
      memory: "256Mi" ②

```

①

CPU 는 CPU 단위입니다. 100m은 0.1 CPU 단위(100 * 1e-3)를 나타냅니다.

②

메모리 는 바이트 단위입니다. 256Mi 는 268435456 바이트(256 * 2 ^ 20)를 나타냅니다.

그러나 프로젝트에 할당량을 정의한 경우 다음 두 항목 중 하나가 필요합니다.

○

requests가 명시적으로 설정된 **resources** 섹션:

```

resources:
  requests: ①
    cpu: "100m"
    memory: "256Mi"

```

①

requests 오브젝트에 할당량의 리소스 목록에 해당하는 리소스 목록이 포함되어 있습니다.

○

프로젝트에 정의된 제한 범위: **LimitRange** 오브젝트의 기본값은 빌드 프로세스 중 생성된 **Pod**에 적용됩니다.

그러지 않으면 할당량을 충족하지 못하여 빌드 **Pod** 생성이 실패합니다.

2.9.2. 최대 기간 설정

BuildConfig 오브젝트를 정의할 때는 **completionDeadlineSeconds** 필드를 설정하여 최대 기간을 정의할 수 있습니다. 이는 초 단위로 지정되며 기본적으로 설정되어 있지 않습니다. 설정하지 않으면 최대 기간이 적용되지 않습니다.

최대 기간은 시스템에서 빌드 **Pod**를 예약하는 시점부터 계산되며 빌더 이미지를 가져오는 데 필요한 시간을 포함하여 활성화할 수 있는 기간을 정의합니다. 지정된 타임아웃에 도달하면 **OpenShift Container Platform**에서 빌드를 종료합니다.

프로세스

- 최대 기간을 설정하려면 **BuildConfig**에서 **completionDeadlineSeconds**를 지정합니다. 다음 예제에서는 **completionDeadlineSeconds** 필드를 30분으로 지정하는 **BuildConfig**의 일부를 보여줍니다.

```
spec:
  completionDeadlineSeconds: 1800
```



참고

파이프라인 전략 옵션에서는 이 설정이 지원되지 않습니다.

2.9.3. 특정 노드에 빌드 할당

빌드 구성의 **nodeSelector** 필드에 라벨을 지정하면 빌드가 특정 노드에서 실행되도록 타겟을 지정할 수 있습니다. **nodeSelector** 값은 빌드 **Pod**를 예약할 때 **Node** 라벨과 일치하는 일련의 키-값 쌍입니다.

nodeSelector 값은 클러스터 전체 기본값 및 덮어쓰기 값으로도 제어할 수 있습니다. 기본값은 빌드 구성에서 **nodeSelector**에 키-값 쌍을 정의하지 않고 명시적으로 비어 있는 맵 값(**nodeSelector: {}**)도 정의하지 않는 경우에만 적용됩니다. 덮어쓰기 값은 키에 따라 빌드 구성의 값을 대체합니다.



참고

지정된 **NodeSelector**가 해당 라벨이 있는 노드와 일치하지 않는 경우 빌드는 계속 **Pending** 상태로 무기한 유지됩니다.

프로세스

- **BuildConfig**의 **nodeSelector** 필드에 라벨을 할당하여 특정 노드에서 실행할 빌드를 할당합니다. 예를 들면 다음과 같습니다.

```

apiVersion: "v1"
kind: "BuildConfig"
metadata:
  name: "sample-build"
spec:
  nodeSelector: 1
    key1: value1
    key2: value2

```

1

이 빌드 구성과 관련된 빌드는 **key1=value1** 및 **key2=value2** 라벨이 있는 노드에서만 실행됩니다.

2.9.4. 연결된 빌드

Go, C, C++, Java와 같이 컴파일된 언어의 경우 애플리케이션 이미지에서 컴파일하는 데 필요한 종속 항목을 포함하면 이미지 크기가 늘어나거나 악용될 수 있는 취약점이 발생할 수 있습니다.

이러한 문제를 방지하기 위해 두 개의 빌드를 함께 연결할 수 있습니다. 하나는 컴파일된 아티팩트를 생성하는 빌드이고 다른 하나는 해당 아티팩트를 실행하는 별도의 이미지에 아티팩트를 배치하는 빌드입니다.

다음 예제에서는 **S2I(source-to-image)** 빌드가 **Docker** 빌드와 결합되어 아티팩트를 컴파일하고 해당 아티팩트는 별도의 런타임 이미지에 배치됩니다.



참고

이 예제에서는 **S2I** 빌드와 **Docker** 빌드를 연결하지만 첫 번째 빌드에서는 원하는 아티팩트를 포함하는 이미지를 생성하는 모든 전략을 사용할 수 있고, 두 번째 빌드에서는 이미지의 입력 콘텐츠를 소비하는 모든 전략을 사용할 수 있습니다.

첫 번째 빌드에서는 애플리케이션 소스를 가져와서 **WAR** 파일이 포함된 이미지를 생성합니다. 이미지는 **artifact-image** 이미지 스트림으로 푸쉬합니다. 출력 아티팩트의 경로는 사용된 **S2I** 빌더의 **assemble** 스크립트에 따라 달라집니다. 다음 예제의 경우 **/wildfly/standalone/deployments/ROOT.war**로 출력됩니다.

```

apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:

```



```

name: artifact-build
spec:
  output:
    to:
      kind: ImageStreamTag
      name: artifact-image:latest
  source:
    git:
      uri: https://github.com/openshift/openshift-jee-sample.git
      ref: "master"
  strategy:
    sourceStrategy:
      from:
        kind: ImageStreamTag
        name: wildfly:10.1
        namespace: openshift

```

두 번째 빌드에서는 이미지 소스와 첫 번째 빌드의 출력 이미지 내부에 있는 **WAR** 파일에 대한 경로를 사용합니다. 인라인 **dockerfile**은 해당 **WAR** 파일을 런타임 이미지에 복사합니다.

```

apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: image-build
spec:
  output:
    to:
      kind: ImageStreamTag
      name: image-build:latest
  source:
    dockerfile: |-
      FROM jee-runtime:latest
      COPY ROOT.war /deployments/ROOT.war
  images:
    - from: ①
      kind: ImageStreamTag
      name: artifact-image:latest
      paths: ②
    - sourcePath: /wildfly/standalone/deployments/ROOT.war
      destinationDir: "."
  strategy:
    dockerStrategy:
      from: ③
      kind: ImageStreamTag
      name: jee-runtime:latest
  triggers:
    - imageChange: {}
      type: ImageChange

```

①

from은 **Docker** 빌드에 이전 빌드의 타겟이었던 **artifact-image** 이미지 스트림의 이미지 출력이 포함되어야 함을 나타냅니다.

2

`paths`는 현재 **Docker** 빌드에 포함할 대상 이미지의 경로를 지정합니다.

3

런타임 이미지는 **Docker** 빌드의 소스 이미지로 사용됩니다.

이 설정으로 인해 두 번째 빌드의 출력 이미지에 **WAR** 파일을 생성하는 데 필요한 빌드 툴을 포함하지 않아도 됩니다. 또한 두 번째 빌드에는 이미지 변경 트리거가 포함되어 있기 때문에 첫 번째 빌드가 실행되어 바이너리 아티팩트가 포함된 새 이미지를 생성할 때마다 두 번째 빌드가 자동으로 트리거되어 해당 아티팩트가 포함된 런타임 이미지를 생성합니다. 따라서 두 빌드 모두 두 단계가 있는 단일 빌드로 작동합니다.

2.9.5. 빌드 정리

기본적으로 라이프사이클이 완료된 빌드는 무기한 유지됩니다. 유지되는 이전 빌드의 수를 제한할 수 있습니다.

프로세스

1.

BuildConfig의 `successfulBuildsHistoryLimit` 또는 `failedBuildsHistoryLimit`에 양의 정수 값을 제공하여 유지되는 이전 빌드의 수를 제한합니다. 예를 들면 다음과 같습니다.

```
apiVersion: "v1"
kind: "BuildConfig"
metadata:
  name: "sample-build"
spec:
  successfulBuildsHistoryLimit: 2 ①
  failedBuildsHistoryLimit: 2 ②
```

①

`successfulBuildsHistoryLimit`은 `completed` 상태의 빌드를 두 개까지 유지합니다.

②

`failedBuildsHistoryLimit`은 `failed`, `canceled` 또는 `error` 상태의 빌드를 두 개까지 유지합니다.

2.

다음 작업 중 하나로 빌드 정리를 트리거합니다.

- 빌드 구성 업데이트
- 빌드가 라이프사이클을 완료할 때까지 대기

빌드는 생성 타임스탬프에 따라 정렬되고 가장 오래된 빌드가 가장 먼저 정리됩니다.



참고

관리자는 'oc adm' 오브젝트 정리 명령을 사용하여 수동으로 빌드를 정리할 수 있습니다.

2.9.6. 빌드 정책 실행

빌드 실행 정책은 빌드 구성에서 생성한 빌드를 실행할 순서를 지정합니다. 이 작업은 **Build** 사양의 **spec** 섹션에서 **runPolicy** 필드 값을 변경하여 수행할 수 있습니다.

다음과 같이 기존 빌드 구성의 **runPolicy** 값을 변경할 수도 있습니다.

- **Parallel**을 **Serial** 또는 **SerialLatestOnly**로 변경하고 이 구성에서 새 빌드를 트리거하면 직렬 빌드는 단독으로만 실행할 수 있으므로 모든 병렬 빌드가 완료될 때까지 새 빌드가 대기합니다.
- **Serial**을 **SerialLatestOnly**로 변경하고 새 빌드를 트리거하면 현재 실행 중인 빌드와 최근 생성된 빌드를 제외하고 대기열에 있는 기존 빌드가 모두 취소됩니다. 최신 빌드는 다음에 실행됩니다.

2.10. 빌드에서 RED HAT 서브스크립션 사용

OpenShift Container Platform에서 권한이 있는 빌드를 실행하려면 다음 섹션을 사용합니다.

2.10.1. Red Hat Universal Base Image에 대한 이미지 스트림 태그 생성

빌드 내에서 **Red Hat** 서브스크립션을 사용하려면 **UBI(Universal Base Image)**를 참조하는 이미지 스트림 태그를 생성합니다.

클러스터의 모든 프로젝트에서 **UBI**를 사용할 수 있도록 하려면 **openshift** 네임스페이스에 이미지 스트림 태그를 추가합니다. 또는 **UBI**를 특정 프로젝트에서 사용할 수 있도록 하려면 해당 프로젝트에 이미지 스트림 태그를 추가합니다.

이미지 스트림 태그를 이러한 방식으로 사용할 때의 이점은 다른 사용자에게 가져오기 보안을 노출하지 않고 설치의 **registry.redhat.io** 자격 증명에 따라 **UBI**에 대한 액세스 권한을 부여할 수 있다는 점입니다. 이 방식은 각 개발자에게 각 프로젝트에서 **registry.redhat.io** 자격 증명을 사용하여 가져오기 보안을 설치하도록 요구하는 방식보다 편리합니다.

프로세스

- **openshift** 네임스페이스에 **ImageStreamTag**를 생성하려면 모든 프로젝트의 개발자가 다음을 입력하면 됩니다.

```
$ oc tag --source=docker registry.redhat.io/ubi7/ubi:latest ubi:latest -n openshift
```

- 단일 프로젝트에서 **ImageStreamTag**를 생성하려면 다음을 입력합니다.

```
$ oc tag --source=docker registry.redhat.io/ubi7/ubi:latest ubi:latest
```

2.10.2. 서브스크립션 자격을 빌드 보안으로 추가

Red Hat 서브스크립션을 사용하여 콘텐츠를 설치하는 빌드에는 자격 키가 빌드 보안으로 포함되어야 합니다.

사전 요구 사항

서브스크립션을 통해 **Red Hat** 인타이틀먼트에 액세스할 수 있어야 합니다. 인타이틀먼트 보안은 **Insights Operator**에 의해 자동으로 생성됩니다.

작은 정보

RHEL(Red Hat Enterprise Linux) 7을 사용하여 인타이틀먼트 빌드를 수행할 때 **yum** 명령을 실행하기 전에 **Dockerfile**에 다음 지침이 있어야 합니다.

```
RUN rm /etc/rhsm-host
```

프로세스

1. **etc-pki-entitlement** 보안을 빌드 구성에 빌드 입력으로 추가합니다.

```
source:
secrets:
- secret:
  name: etc-pki-entitlement
  destinationDir: etc-pki-entitlement
```

2.10.3. 서브스크립션 관리자를 사용한 빌드 실행

2.10.3.1. 서브스크립션 관리자를 사용하는 Docker 빌드

Docker 전략 빌드에서는 **Subscription Manager**를 사용하여 서브스크립션 콘텐츠를 설치할 수 있습니다.

사전 요구 사항

자격 키, 서브스크립션 관리자 구성, 서브스크립션 관리자 인증 기관을 빌드 입력으로 추가해야 합니다.

프로세스

다음은 예제 **Dockerfile**로 사용하여 서브스크립션 관리자를 통해 콘텐츠를 설치합니다.

```
FROM registry.redhat.io/rhel7:latest
USER root
# Copy entitlements
COPY ./etc-pki-entitlement /etc/pki/entitlement
# Copy subscription manager configurations
COPY ./rhsm-conf /etc/rhsm
COPY ./rhsm-ca /etc/rhsm/ca
# Delete /etc/rhsm-host to use entitlements from the build container
RUN rm /etc/rhsm-host && \
  # Initialize /etc/yum.repos.d/redhat.repo
  # See https://access.redhat.com/solutions/1443553
  yum repolist --disablerepo=* && \
  subscription-manager repos --enable <enabled-repo> && \
  yum -y update && \
  yum -y install <rpms> && \
  # Remove entitlements and Subscription Manager configs
  rm -rf /etc/pki/entitlement && \
  rm -rf /etc/rhsm
# OpenShift requires images to run as non-root by default
USER 1001
ENTRYPOINT ["/bin/bash"]
```

2.10.4. Red Hat Satellite 서브스크립션을 사용하여 빌드 실행

2.10.4.1. 빌드에 Red Hat Satellite 구성 추가

Red Hat Satellite를 사용하여 콘텐츠를 설치하는 빌드에서는 **Satellite** 리포지토리에서 콘텐츠를 가져오기 위해 적절한 구성을 제공해야 합니다.

사전 요구 사항

- **Satellite** 인스턴스에서 콘텐츠를 다운로드하는 **yum** 호환 리포지토리 구성 파일을 제공하거나 생성해야 합니다.

리포지토리 구성 샘플

```
[test-<name>]
name=test-<number>
baseurl = https://satellite.../content/dist/rhel/server/7/7Server/x86_64/os
enabled=1
gpgcheck=0
sslverify=0
sslclientkey = /etc/pki/entitlement/...-key.pem
sslclientcert = /etc/pki/entitlement/....pem
```

절차

1. **Satellite** 리포지토리 구성 파일이 포함된 **ConfigMap**을 생성합니다.

```
$ oc create configmap yum-repos-d --from-file /path/to/satellite.repo
```

2. **BuildConfig**에 **Satellite** 리포지토리 구성을 추가합니다.

```
source:
  configMaps:
  - configMap:
    name: yum-repos-d
    destinationDir: yum.repos.d
```

2.10.4.2. Red Hat Satellite 서브스크립션을 사용하는 Docker 빌드

Docker 전략 빌드에서는 **Red Hat Satellite** 리포지토리를 사용하여 서브스크립션 콘텐츠를 설치할 수 있습니다.

사전 요구 사항

- 자격 키 및 **Satellite** 리포지토리 구성을 빌드 입력으로 추가해야 합니다.

프로세스

다음은 예제 **Dockerfile**로 사용하여 **Satellite**를 통해 콘텐츠를 설치합니다.

```
FROM registry.redhat.io/rhel7:latest
USER root
# Copy entitlements
COPY ./etc-pki-entitlement /etc/pki/entitlement
# Copy repository configuration
COPY ./yum.repos.d /etc/yum.repos.d
# Delete /etc/rhsm-host to use entitlements from the build container
RUN sed -i".org" -e "s#^enabled=1#enabled=0#g" /etc/yum/pluginconf.d/subscription-
manager.conf 1
#RUN cat /etc/yum/pluginconf.d/subscription-manager.conf
RUN yum clean all
#RUN yum-config-manager
RUN rm /etc/rhsm-host && \
    # yum repository info provided by Satellite
    yum -y update && \
    yum -y install <rpms> && \
    # Remove entitlements
    rm -rf /etc/pki/entitlement
# OpenShift requires images to run as non-root by default
USER 1001
ENTRYPOINT ["/bin/bash"]
```

1

enabled=1을 사용하여 빌드에 **Satellite** 구성을 추가하는 데 실패하면 **Dockerfile**에 **RUN sed -i".org" -e "s#^enabled=1#enabled=0#g" /etc/yum/pluginconf.d/subscription-manager.conf**를 추가합니다.

2.10.5. 추가 리소스

- [이미지 스트림 관리](#)

● 빌드 전략

2.11. 전략에 따른 빌드 보안

OpenShift Container Platform의 빌드는 권한이 있는 컨테이너에서 실행됩니다. 권한이 있는 경우 사용한 빌드 전략에 따라 빌드를 실행하여 클러스터 및 호스트 노드에 대한 권한을 에스컬레이션할 수 있습니다. 그리고 일종의 보안 조치로 빌드 및 해당 빌드에 사용하는 전략을 실행할 수 있는 사람을 제한합니다. 사용자 정의 빌드는 권한 있는 컨테이너 내의 모든 코드를 실행할 수 있기 때문에 본질적으로 소스 빌드보다 안전하지 않으며, 기본적으로 비활성화되어 있습니다. Dockerfile 처리 논리의 취약성으로 인해 호스트 노드에 권한이 부여될 수 있으므로 Docker 빌드 권한을 주의하여 부여하십시오.

기본적으로 빌드를 생성할 수 있는 모든 사용자에게 Docker 및 S2I(Source-to-image) 빌드 전략을 사용할 수 있는 권한이 부여됩니다. 클러스터 관리자 권한이 있는 사용자는 '전역적으로 빌드 전략을 사용자로 제한' 섹션에 언급된 대로 사용자 정의 빌드 전략을 활성화할 수 있습니다.

권한 부여 정책을 사용하여 빌드할 수 있는 사용자와 이들이 사용할 수 있는 빌드 전략을 제어할 수 있습니다. 각 빌드 전략에는 해당 빌드의 하위 소스가 있습니다. 사용자는 빌드를 생성할 수 있는 권한과 해당 전략을 사용하여 빌드를 생성하기 위해 빌드 전략 하위 리소스에서 생성할 수 있는 권한이 있어야 합니다. 빌드 전략 하위 리소스에 생성 권한을 부여하는 기본 역할이 제공됩니다.

표 2.3. 빌드 전략 하위 리소스 및 역할

전략	하위 리소스	역할
Docker	빌드/Docker	system:build-strategy-docker
S2I(Source-to-Image)	빌드/소스	system:build-strategy-source
사용자 정의	빌드/사용자 정의	system:build-strategy-custom
JenkinsPipeline	builds/jenkinspipeline	system:build-strategy-jenkinspipeline

2.11.1. 전역적으로 빌드 전략에 대한 액세스 비활성화

특정 빌드 전략에 대한 액세스를 전역적으로 방지하려면 클러스터 관리자 권한이 있는 사용자로 로그인하여 `system:authenticated` 그룹에서 해당 역할을 제거한 후 주석 `rbac.authorization.kubernetes.io/autoupdate: "false"`를 적용하여 API를 재시작할 때마다 변경되지 않도록 보호하십시오. 다음 예제에서는 Docker 빌드 전략을 비활성화하는 방법을 보여줍니다.

프로세스

1.

`rbac.authorization.kubernetes.io/autoupdate` 주석을 적용합니다.

```
$ oc edit clusterrolebinding system:build-strategy-docker-binding
```

출력 예

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "false" ❶
  creationTimestamp: 2018-08-10T01:24:14Z
  name: system:build-strategy-docker-binding
  resourceVersion: "225"
  selfLink: /apis/rbac.authorization.k8s.io/v1/clusterrolebindings/system%3Abuild-
strategy-docker-binding
  uid: 17b1f3d4-9c3c-11e8-be62-0800277d20bf
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: system:build-strategy-docker
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:authenticated
```

❶

`rbac.authorization.kubernetes.io/autoupdate` 주석 값을 "false"로 변경합니다.

2.

역할을 제거합니다.

```
$ oc adm policy remove-cluster-role-from-group system:build-strategy-docker
system:authenticated
```

3.

빌드 전략 하위 소스도 이러한 역할에서 제거되었는지 확인합니다.

```
$ oc edit clusterrole admin
```

```
$ oc edit clusterrole edit
```

- 4. 각 역할에 대해 비활성화할 전략 리소스에 해당하는 하위 리소스를 지정합니다.
 - a. **admin**에 대한 **Docker** 빌드 전략을 비활성화합니다.

```

kind: ClusterRole
metadata:
  name: admin
...
- apiGroups:
  - ""
  - build.openshift.io
resources:
  - buildconfigs
  - buildconfigs/webhooks
  - builds/custom 1
  - builds/source
verbs:
  - create
  - delete
  - deletecollection
  - get
  - list
  - patch
  - update
  - watch
...

```

1

빌드/사용자 지정 및 빌드/소스 를 추가하여 **admin** 역할이 있는 사용자에게 대해 **Docker** 빌드를 전역적으로 비활성화합니다.

2.11.2. 전역적으로 빌드 전략을 사용자로 제한

특정 사용자 집합이 특정 전략을 사용하여 빌드를 생성하도록 허용할 수 있습니다.

사전 요구 사항

- 빌드 전략에 대한 글로벌 액세스 권한을 비활성화합니다.

프로세스

- 빌드 전략에 해당하는 역할을 특정 사용자에게 할당합니다. 예를 들어 **system:build-strategy-docker** 클러스터 역할을 사용자 **devuser**에 추가하려면 다음을 수행합니다.

```
$ oc adm policy add-cluster-role-to-user system:build-strategy-docker devuser
```



주의

클러스터 수준의 사용자 액세스 권한을 **builds/docker** 하위 리소스에 부여하면 사용자가 빌드를 생성할 수 있는 모든 프로젝트에서 **Docker** 전략을 사용하여 빌드를 생성할 수 있습니다.

2.11.3. 프로젝트 내 사용자로 빌드 전략 제한

전역적으로 사용자에게 빌드 전략 역할을 부여하는 것과 유사하게 프로젝트 내의 특정 사용자 집합이 특정 전략을 사용하여 빌드를 생성하도록 허용할 수 있습니다.

사전 요구 사항

- 빌드 전략에 대한 글로벌 액세스 권한을 비활성화합니다.

프로세스

- 빌드 전략에 해당하는 역할을 특정 프로젝트 내 사용자에게 할당합니다. 예를 들어 프로젝트 **devproject** 내에서 **system:build-strategy-docker** 역할을 사용자 **devuser**에 추가하려면 다음을 실행합니다.

```
$ oc adm policy add-role-to-user system:build-strategy-docker devuser -n devproject
```

2.12. 빌드 구성 리소스

빌드 설정을 구성하려면 다음 절차를 사용합니다.

2.12.1. 빌드 컨트롤러 구성 매개변수

build.config.openshift.io/cluster 리소스에서는 다음과 같은 구성 매개변수를 제공합니다.

매개변수	설명
Build	<p>빌드 처리 방법에 대한 클러스터 전체 정보가 들어 있습니다. 유일하게 유효한 정식 이름은 cluster입니다.</p> <p>spec: 빌드 컨트롤러 구성에 대한 사용자 설정 가능한 값을 포함합니다.</p>
buildDefaults	<p>빌드의 기본 정보를 제어합니다.</p> <p>defaultProxy: 이미지 가져오기 또는 내보내기 및 소스 다운로드를 포함하여 모든 빌드 작업에 대한 기본 프록시 설정이 포함되어 있습니다.</p> <p>BuildConfig 전략에 HTTP_PROXY, HTTPS_PROXY, NO_PROXY 환경 변수를 설정하여 값을 덮어쓸 수 있습니다.</p> <p>gitProxy: Git 작업에 대한 프록시 설정만 포함합니다. 설정하는 경우 git clone과 같은 모든 Git 명령의 모든 프록시 설정을 덮어씁니다.</p> <p>여기에 설정되지 않은 값은 DefaultProxy에서 상속됩니다.</p> <p>env: 지정된 변수가 빌드에 없는 경우 빌드에 적용되는 기본 환경 변수 집합입니다.</p> <p>imageLabels: 결과 이미지에 적용되는 레이블 목록입니다. BuildConfig에 동일한 이름의 라벨을 제공하여 기본 라벨을 덮어쓸 수 있습니다.</p> <p>resources: 빌드를 실행할 리소스 요구 사항을 정의합니다.</p>
ImageLabel	<p>name: 레이블 이름을 정의합니다. 길이가 0이 아니어야 합니다.</p>
buildOverrides	<p>빌드에 대한 덮어쓰기 설정을 제어합니다.</p> <p>imageLabels: 결과 이미지에 적용되는 레이블 목록입니다. 이 표에 있는 것과 같은 이름이 있는 BuildConfig에 라벨을 제공한 경우 해당 라벨을 덮어씁니다.</p> <p>nodeSelector: 빌드 Pod가 노드에 맞으려면 선택기가 true여야 합니다.</p> <p>허용 오차: 빌드 Pod에 설정된 기존 허용 오차를 덮어쓰는 허용 오차 목록입니다.</p>
BuildList	<p>items: 표준 개체의 메타데이터.</p>

2.12.2. 빌드 설정 구성

build.config.openshift.io/cluster 리소스를 편집하여 빌드 설정을 구성할 수 있습니다.

프로세스

- **build.config.openshift.io/cluster** 리소스를 편집합니다.

```
$ oc edit build.config.openshift.io/cluster
```

다음은 `build.config.openshift.io/cluster` 리소스의 예입니다.

```

apiVersion: config.openshift.io/v1
kind: Build 1
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-05-17T13:44:26Z"
  generation: 2
  name: cluster
  resourceVersion: "107233"
  selfLink: /apis/config.openshift.io/v1/builds/cluster
  uid: e2e9cc14-78a9-11e9-b92b-06d6c7da38dc
spec:
  buildDefaults: 2
  defaultProxy: 3
    httpProxy: http://proxy.com
    httpsProxy: https://proxy.com
    noProxy: internal.com
  env: 4
    - name: envkey
      value: envvalue
  gitProxy: 5
    httpProxy: http://gitproxy.com
    httpsProxy: https://gitproxy.com
    noProxy: internalgit.com
  imageLabels: 6
    - name: labelkey
      value: labelvalue
  resources: 7
    limits:
      cpu: 100m
      memory: 50Mi
    requests:
      cpu: 10m
      memory: 10Mi
  buildOverrides: 8
  imageLabels: 9
    - name: labelkey
      value: labelvalue
  nodeSelector: 10
    selectorkey: selectorvalue
  tolerations: 11
    - effect: NoSchedule
      key: node-role.kubernetes.io/builds
operator: Exists

```

1

빌드: 빌드 처리 방법에 대한 클러스터 전체 정보가 들어 있습니다. 유일하게 유효한 정식 이름은 `cluster`입니다.

2

buildDefaults: 빌드의 기본 정보를 제어합니다.

3

defaultProxy: 이미지 가져오기 또는 내보내기 및 소스 다운로드를 포함하여 모든 빌드 작업에 대한 기본 프록시 설정이 포함되어 있습니다.

4

env: 지정된 변수가 빌드에 없는 경우 빌드에 적용되는 기본 환경 변수 집합입니다.

5

gitProxy: Git 작업에 대한 프록시 설정만 포함합니다. 설정하는 경우 **git clone**과 같은 모든 **Git** 명령의 모든 프록시 설정을 덮어씁니다.

6

imageLabels: 결과 이미지에 적용되는 레이블 목록입니다. **BuildConfig**에 동일한 이름의 라벨을 제공하여 기본 라벨을 덮어쓸 수 있습니다.

7

resources: 빌드를 실행할 리소스 요구 사항을 정의합니다.

8

buildOverrides: 빌드에 대한 덮어쓰기 설정을 제어합니다.

9

imageLabels: 결과 이미지에 적용되는 레이블 목록입니다. 이 표에 있는 것과 같은 이름이 있는 **BuildConfig**에 라벨을 제공한 경우 해당 라벨을 덮어씁니다.

10

nodeSelector: 빌드 **Pod**가 노드에 맞으려면 선택기가 **true**여야 합니다.

11

허용 오차: 빌드 **Pod**에 설정된 기존 허용 오차를 덮어쓰는 허용 오차 목록입니다.

2.13. 빌드 문제 해결

다음을 사용하여 빌드 문제를 해결합니다.

2.13.1. 리소스에 대한 액세스 거부 문제 해결

리소스에 대한 액세스 요청이 거부되는 경우:

문제

다음과 같은 메시지가 표시되고 빌드가 실패합니다.

```
requested access to the resource is denied
```

해결

프로젝트에 설정된 이미지 할당량 중 하나를 초과했습니다. 현재 할당량을 확인하고 적용되는 제한과 사용 중인 스토리지를 확인합니다.

```
$ oc describe quota
```

2.13.2. 서비스 인증서 생성 실패

리소스에 대한 액세스 요청이 거부되는 경우:

문제

와 함께 서비스 인증서 생성이 실패하는 경우 (서비스의 `service.beta.openshift.io/serving-cert-generation-error` 주석에는 다음이 포함됩니다).

출력 예

```
secret/ssl-key references serviceUID 62ad25ca-d703-11e6-9d6f-0e9c0057b608, which does not match 77b6dd80-d716-11e6-9d6f-0e9c0057b60
```

해결

인증서를 생성한 서비스가 더 이상 존재하지 않거나 **serviceUID**가 다릅니다. 이전 보안을 제거하고 서비스에서 **service.beta.openshift.io/serving-cert-generation-error** 및 **service.beta.openshift.io/serving-cert-generation-error -num** 주석을 지워 인증서를 강제로 다시 생성해야 합니다.

```
$ oc delete secret <secret_name>
```

```
$ oc annotate service <service_name> service.beta.openshift.io/serving-cert-generation-error-
```

```
$ oc annotate service <service_name> service.beta.openshift.io/serving-cert-generation-error-  
num-
```



참고

주석을 제거하는 명령에는 제거할 주석 이름 뒤에 -가 있습니다.

2.14. 빌드에 대해 신뢰할 수 있는 추가 인증 기관 설정

이미지 레지스트리에서 이미지를 가져올 때 빌드에서 신뢰할 추가 **CA**(인증 기관)를 설정하려면 다음 섹션을 사용합니다.

이 절차를 수행하려면 클러스터 관리자가 **ConfigMap**을 생성하고 **ConfigMap**에 추가 **CA**를 키로 추가해야 합니다.

- **ConfigMap**은 **openshift-config** 네임스페이스에 생성해야 합니다.
- **domain**은 **ConfigMap**의 키이고 **value**는 **PEM** 형식으로 인코딩한 인증서입니다.
 - 각 **CA**는 도메인과 연결되어 있어야 합니다. 도메인 형식은 **hostname[..port]**입니다.
- **ConfigMap** 이름은 **image.config.openshift.io/cluster** 클러스터 범위 구성 리소스의 **spec.additionalTrustedCA** 필드에 설정해야 합니다.

2.14.1. 클러스터에 인증 기관 추가

다음 절차에 따라 이미지를 내보내고 가져올 때 사용할 클러스터에 인증서 **CA**(인증 기관)를 추가할 수

있습니다.

사전 요구 사항

- 클러스터 관리자 권한이 있어야 합니다.
- 레지스트리의 공용 인증서(일반적으로 `/etc/docker/certs.d/` 디렉터리에 있는 `hostname/ca.crt` 파일)에 액세스할 수 있어야 합니다.

절차

1. 자체 서명 인증서를 사용하는 레지스트리의 경우 신뢰할 수 있는 인증서가 있는 `openshift-config` 네임스페이스에 **ConfigMap**을 생성합니다. 각 **CA** 파일에 대해 **ConfigMap**의 키가 `hostname[..port]` 형식의 레지스트리 호스트 이름인지 확인하십시오.

```
$ oc create configmap registry-cas -n openshift-config \
--from-file=myregistry.corp.com..5000=/etc/docker/certs.d/myregistry.corp.com:5000/ca.crt \
--from-file=otherregistry.com=/etc/docker/certs.d/otherregistry.com/ca.crt
```

2. 클러스터 이미지 구성을 업데이트합니다.

```
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA": {"name":"registry-cas"}}}' --type=merge
```

2.14.2. 추가 리소스

- [ConfigMap 생성](#)
- [보안 및 ConfigMaps](#)
- [사용자 정의 PKI 구성](#)

3장. JENKINS에서 TEKTON으로 마이그레이션

3.1. JENKINS에서 TEKTON으로 마이그레이션

Jenkins 및 **Tekton**은 애플리케이션 및 프로젝트 구축, 테스트 및 배포 프로세스를 자동화하는 데 광범위하게 사용됩니다. 그러나 **Tekton**은 **Kubernetes** 및 **OpenShift Container Platform**과 원활하게 작동하는 클라우드 네이티브 **CI/CD** 솔루션입니다. 이 문서는 **Jenkins CI/CD** 워크플로를 **Tekton**으로 마이그레이션하는 데 도움이 됩니다.

3.1.1. Jenkins 및 Tekton 개념 비교

이 섹션에는 **Jenkins** 및 **Tekton**에 사용되는 기본 용어가 요약되어 있으며 동등한 용어를 비교합니다.

3.1.1.1. Jenkins 용어

Jenkins는 공유 라이브러리 및 플러그인을 사용하여 확장할 수 있는 선언적 및 스크립팅된 파이프라인을 제공합니다. **Jenkins**의 몇 가지 기본 용어는 다음과 같습니다.

- 파이프 라인: **Groovy** 구문을 사용하여 애플리케이션을 구축, 테스트 및 배포하는 전체 프로세스를 자동화합니다.
- 노드: 스크립팅된 파이프라인을 오케스트레이션하거나 실행할 수 있는 머신입니다.
- 단계: 파이프라인에서 수행되는 작업의 개념적으로 고유한 하위 집합입니다. 플러그인 또는 사용자 인터페이스에서는 이 블록을 사용하여 작업의 상태 또는 진행 상황을 표시하는 경우가 많습니다.
- 단계: 명령 또는 스크립트를 사용하여 수행할 정확한 작업을 지정하는 단일 작업입니다.

3.1.1.2. Tekton 용어

Tekton은 선언적 파이프라인에 **YAML** 구문을 사용하고 작업으로 구성됩니다. **Tekton**의 몇 가지 기본 용어는 다음과 같습니다.

- 파이프 라인: 시리즈, 병렬 또는 둘 다에 있는 작업 집합.

- 작업: 일련의 단계를 명령, 바이너리 또는 스크립트로 설정합니다.
- **PipelineRun**: 하나 이상의 작업을 사용하여 파이프라인 실행.
- **TaskRun**: 하나 이상의 단계를 사용하여 작업 실행.



참고

매개변수 및 작업 영역과 같은 입력 세트로 **PipelineRun** 또는 **TaskRun**을 시작하고 실행 결과 출력 및 아티팩트 세트가 생성됩니다.

- **Workspace**: Tekton에서 작업 공간은 다음과 같은 목적을 수행하는 개념적 블록입니다.
 - 입력, 출력 및 빌드 아티팩트의 저장.
 - 작업 간에 데이터를 공유하는 공용 공간.
 - 시크릿에 보유된 인증 정보, 구성 맵에 저장된 구성 및 조직에서 공유하는 공통 툴의 마운트 지점.



참고

Jenkins에는 **Tekton** 작업 공간에 직접적으로 해당하는 작업 공간이 없습니다. 복제된 코드 리포지토리를 저장하고 기록 및 아티팩트를 빌드하므로 컨트롤 노드를 작업 영역으로 간주할 수 있습니다. 작업이 다른 노드에 할당되는 경우 복제된 코드와 생성된 아티팩트는 해당 노드에 저장되지만 빌드 기록은 컨트롤 노드에 의해 유지 관리됩니다.

3.1.1.3. 개념 매핑

Jenkins 및 **Tekton**의 구성 요소는 동일하지 않으며 비교 결과에서는 기술적으로 정확한 매핑이 제공되지 않습니다. **Jenkins** 및 **Tekton**의 다음 용어 및 개념은 일반적으로 상관 관계가 있습니다.

표 3.1. Jenkins 및 Tekton - 기본 비교

Jenkins	Tekton
Pipeline	Pipeline 및 PipelineRun
Stage	Task
Step	작업의 단계

3.1.2. Jenkins에서 Tekton으로 샘플 파이프라인 마이그레이션

이 섹션에서는 **Jenkins** 및 **Tekton**에서 파이프라인과 동일한 예제를 제공하며 **Jenkins**에서 **Tekton**으로 파이프라인을 마이그레이션, 테스트 및 배포하는 데 유용한 정보를 제공합니다.

3.1.2.1. Jenkins 파이프라인

빌드, 테스트 및 배포를 위해 **Groovy**로 작성된 **Jenkins** 파이프라인을 고려하십시오.

```

pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        sh 'make'
      }
    }
    stage('Test'){
      steps {
        sh 'make check'
        junit 'reports/**/*.xml'
      }
    }
    stage('Deploy') {
      steps {
        sh 'make publish'
      }
    }
  }
}

```

3.1.2.2. Tekton 파이프라인

Tekton에서 **Jenkins** 파이프라인의 동등한 예는 세 가지 작업으로 구성되며 각각 **YAML** 구문을 사용하여 선언적으로 작성할 수 있습니다.

build 작업 예

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: myproject-build
spec:
  workspaces:
    - name: source
  steps:
    - image: my-ci-image
      command: ["make"]
      workingDir: $(workspaces.source.path)
```

test 작업 예:

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: myproject-test
spec:
  workspaces:
    - name: source
  steps:
    - image: my-ci-image
      command: ["make check"]
      workingDir: $(workspaces.source.path)
    - image: junit-report-image
      script: |
        #!/usr/bin/env bash
        junit-report reports/**/*.*.xml
      workingDir: $(workspaces.source.path)
```

deploy 작업 예:

```
apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: myprojectd-deploy
spec:
  workspaces:
```

```

- name: source
steps:
- image: my-deploy-image
  command: ["make deploy"]
  workingDir: $(workspaces.source.path)

```

세 가지 작업을 순차적으로 결합하여 **Tekton** 파이프라인을 구성할 수 있습니다.

예제: 구축, 테스트 및 배포를 위한 **Tekton** 파이프라인

```

apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: myproject-pipeline
spec:
  workspaces:
  - name: shared-dir
  tasks:
  - name: build
    taskRef:
      name: myproject-build
    workspaces:
    - name: source
      workspace: shared-dir
  - name: test
    taskRef:
      name: myproject-test
    workspaces:
    - name: source
      workspace: shared-dir
  - name: deploy
    taskRef:
      name: myproject-deploy
    workspaces:
    - name: source
      workspace: shared-dir

```

3.1.3. Jenkins 플러그인에서 Tekton Hub 작업으로 마이그레이션

[플러그인](#)을 사용하여 **Jenkins**의 기능을 확장할 수 있습니다. **Tekton**에서 유사한 확장성을 얻으려면 [Tekton Hub](#)에서 사용 가능한 작업을 사용합니다.

예를 들어 Jenkins의 [git plugin](#)에 해당하는 Tekton Hub에서 사용할 수 있는 [git-clone](#) 작업을 고려하십시오.

예: Tekton Hub에서 [git-clone](#) 작업

```

apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: demo-pipeline
spec:
  params:
    - name: repo_url
    - name: revision
  workspaces:
    - name: source
  tasks:
    - name: fetch-from-git
      taskRef:
        name: git-clone
      params:
        - name: url
          value: ${params.repo_url}
        - name: revision
          value: ${params.revision}
      workspaces:
        - name: output
          workspace: source

```

3.1.4. 사용자 정의 작업 및 스크립트를 사용하여 Tekton 기능 확장

Tekton의 Tekton Hub에서 올바른 작업을 찾지 못하거나 작업을 더 잘 제어해야 하는 경우 사용자 지정 작업 및 스크립트를 생성하여 Tekton의 기능을 확장할 수 있습니다.

예제: `maven test` 명령을 실행하는 사용자 지정 작업

```

apiVersion: tekton.dev/v1beta1
kind: Task
metadata:
  name: maven-test
spec:
  workspaces:
    - name: source

```

```

steps:
- image: my-maven-image
  command: ["mvn test"]
  workingDir: $(workspaces.source.path)

```

예제: 경로를 제공하여 사용자 정의 셸 스크립트 실행

```

...
steps:
  image: ubuntu
  script: |
    #!/usr/bin/env bash
    /workspace/my-script.sh
...

```

예제: 사용자 정의 Python 스크립트를 YAML 파일에 작성하여 실행합니다.

```

...
steps:
  image: python
  script: |
    #!/usr/bin/env python3
    print("hello from python!")
...

```

3.1.5. Jenkins 및 Tekton 실행 모델 비교

Jenkins 및 Tekton은 유사한 기능을 제공하지만 아키텍처 및 실행이 다릅니다. 이 섹션에서는 두 가지 실행 모델을 간단히 비교합니다.

표 3.2. Jenkins 및 Tekton의 실행 모델 비교

Jenkins	Tekton
<p>Jenkins에는 컨트롤 노드가 있습니다. Jenkins는 파이프라인을 실행하고 중앙 집중적으로 단계를 실행하거나 다른 노드에서 실행되는 작업을 오케스트레이션합니다.</p>	<p>Tekton은 서버리스 및 분산되어 있으며 실행을 위한 중앙 종속성이 없습니다.</p>
<p>컨테이너는 파이프라인을 통해 컨트롤 노드에서 시작됩니다.</p>	<p>Tekton은 모든 단계가 Pod에서 실행되는 컨테이너 (Jenkins의 노드와 동등)로 실행되는 '컨테이너 우선' 접근 방식을 채택합니다.</p>
<p>확장성은 플러그인을 사용하여 달성합니다.</p>	<p>확장성은 Tekton Hub의 작업을 사용하거나 사용자 지정 작업 및 스크립트를 만들어 얻을 수 있습니다.</p>

4장. PIPELINE

4.1. RED HAT OPENSIFT PIPELINES 릴리스 정보

Red Hat OpenShift Pipelines는 다음을 제공하는 Tekton 프로젝트를 기반으로 하는 클라우드 네이티브 CI/CD 환경입니다.

- 표준 CRD(Kubernetes 네이티브 Pipeline 정의)
- CI 서버 관리 오버헤드가 없는 서버리스 Pipeline
- S2I, Buildah, JIB 및 Kaniko와 같은 Kubernetes 도구를 사용하여 이미지를 빌드할 수 있는 확장성
- 모든 Kubernetes 배포판에서 이식성
- Pipeline과 상호 작용하기 위한 강력한 CLI
- OpenShift Container Platform 웹 콘솔의 개발자 화면과 통합된 사용자 경험

Red Hat OpenShift Pipelines 개요는 [OpenShift Pipelines 이해](#)를 참조하십시오.

4.1.1. 보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 [CTO Chris Wright의 메시지](#)를 참조하십시오.

4.1.2. Red Hat OpenShift Pipelines General Availability 1.5 릴리스 정보

Red Hat OpenShift Pipelines General Availability (GA) 1.5는 이제 OpenShift Container Platform 4.8에서 사용할 수 있습니다.

4.1.2.1. 호환성 및 지원 매트릭스

이 릴리스의 일부 기능은 현재 기술 프리뷰 단계에 있습니다. 이러한 실험적 기능은 프로덕션용이 아닙니다.

기술 프리뷰 기능 지원 범위

아래 표에서 기능은 다음과 같은 상태로 표시되어 있습니다.

- **TP: 기술 프리뷰**
- **GA: 정식 출시일 (GA)**

해당 기능은 **Red Hat Customer Portal**의 지원 범위를 참조하십시오.

표 4.1. 호환성 및 지원 매트릭스

기능	버전	지원 상태
Pipeline	0.24	GA
CLI	0.19	GA
카탈로그	0.24	GA
Trigger	0.14	TP
파이프 라인 리소스	-	TP

질문이나 의견이 있으시면 제품팀에 이메일(pipelines-interest@redhat.com)로 보내주시기 바랍니다.

4.1.2.2. 새로운 기능

다음 섹션에서는 수정 및 안정성 개선 사항 외에 **Red Hat OpenShift Pipelines 1.5**의 새로운 기능도 소개합니다.

- 대상 네임스페이스의 **cron** 작업으로 파이프라인 실행 및 작업 실행이 자동으로 제거됩니다. **cron** 작업에서는 **IMAGE_JOB_PRUNER_TKN** 환경 변수를 사용하여 **tkn image** 값을 가져옵니다. 이번 개선된 기능을 통해 **TektonConfig** 사용자 정의 리소스에 다음 필드가 도입되었습니다.

```

...
pruner:
  resources:
    - pipelinerun
    - taskrun
  schedule: "*/5 * * * * # cron schedule
  keep: 2 # delete all keeping n
...
    
```

- OpenShift Container Platform에서는 **Tekton Config** 사용자 정의 리소스의 새 매개 변수 **clusterTasks** 및 **pipelinesTemplates** 값을 수정하여 **Tekton Add-ons** 구성 요소 설치를 사용자 지정할 수 있습니다.

```

apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  profile: all
  targetNamespace: openshift-pipelines
  addon:
    params:
      - name: clusterTasks
        value: "true"
      - name: pipelineTemplates
        value: "true"
...
    
```

사용자 지정은 **TektonConfig**를 사용하여 애드온을 만들거나 **Tekton** 애드온을 사용하여 직접 만드는 경우에 허용됩니다. 그러나 매개 변수가 전달되지 않으면 컨트롤러는 기본값을 사용하여 매개 변수를 추가합니다.



참고

- TektonConfig** 사용자 지정 리소스를 사용하여 추가 기능이 생성되고 **Addon** 사용자 정의 리소스의 뒷부분에서 매개 변수 값을 변경하면 **TektonConfig** 사용자 지정 리소스의 값이 변경 사항을 덮어씁니다.
- clusterTasks** 매개 변수 값이 **true**인 경우에만 **pipelinesTemplates** 매개 변수의 값을 **true**로 설정할 수 있습니다.

- **enableMetrics** 매개변수는 **TektonConfig** 사용자 지정 리소스에 추가됩니다. 이를 사용하여 **OpenShift Container Platform**용 **Tekton Pipelines**의 일부인 서비스 모니터를 비활성화할 수 있습니다.

```

apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  profile: all
  targetNamespace: openshift-pipelines
  pipeline:
    params:
      - name: enableMetrics
        value: "true"
  ...

```

- 프로세스 수준에서 메트릭을 캡처하는 **EventListener OpenCensus** 메트릭이 추가되었습니다.
- 트리거에 레이블 선택기가 있습니다. 레이블을 사용하여 이벤트 리스너에 대한 트리거를 구성할 수 있습니다.
- 인터셉터 등록에 대한 **ClusterInterceptor** 사용자 정의 리소스 정의가 추가되어 연결할 수 있는 새로운 **Interceptor** 유형을 등록할 수 있습니다. 또한 다음과 같은 관련 변경 사항이 적용됩니다.
 - 트리거 사양에서는 클러스터 인터셉터를 참조하기 위해 **ref** 필드를 포함하는 새 **API**를 사용하여 인터셉터를 구성할 수 있습니다. 또한 **params** 필드를 사용하여 처리를 위해 인터셉터에 전달되는 매개변수를 추가할 수 있습니다.
 - 번들 인터셉터 **CEL**, **GitHub**, **GitLab**, **BitBucket**이 마이그레이션되었습니다. 새 **ClusterInterceptor** 사용자 정의 리소스 정의를 사용하여 구현됩니다.
 - 코어 인터셉터는 새 형식으로 마이그레이션되고 이전 구문을 사용하여 생성된 새 트리거가 새 **ref** 또는 **params** 기반 구문으로 자동 전환됩니다.
- 로그를 표시하는 동안 작업 이름 또는 단계의 접두사를 비활성화하려면 **log** 명령에 **--prefix** 옵션을 사용합니다.

- 특정 구성 요소의 버전을 표시하려면 `tkn version` 명령에서 새 `--component` 플래그를 사용합니다.
- `tkn hub check-upgrade` 명령이 추가되고 파이프라인 버전을 기반으로 다른 명령이 수정되었습니다. 또한 `search` 명령 출력에 카탈로그 이름이 표시됩니다.
- 선택적 작업 공간에 대한 지원이 `start` 명령에 추가됩니다.
- `plugins` 디렉토리에 플러그인이 없으면 현재 경로에서 검색됩니다.
- `tkn start [task | clustertask | pipeline]` 명령은 대화식으로 시작하고 기본 매개변수를 지정하는 경우에도 `params` 값을 요청합니다. 대화식 프롬프트를 중지하려면 명령을 호출할 때 `--use-param-defaults` 플래그를 전달합니다. 예를 들면 다음과 같습니다.


```
$ tkn pipeline start build-and-deploy \
  -w name=shared-workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.5/01_pipeline/03_persistent_volume_claim.yaml \
  -p deployment-name=pipelines-vote-api \
  -p git-url=https://github.com/openshift/pipelines-vote-api.git \
  -p IMAGE=image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/pipelines-vote-api \
  --use-param-defaults
```
- `version` 필드는 `tkn task describe` 명령에 추가됩니다.
- `TriggerTemplate` 또는 `TriggerBinding` 또는 `ClusterTriggerBinding` 또는 `EventListener`와 같은 리소스를 자동으로 선택하는 옵션은 하나만 있는 경우 `describe` 명령에 추가됩니다.
- `tkn pr describe` 명령에 건너뛰는 작업의 섹션이 추가됩니다.
- `tkn clustertask logs`에 대한 지원이 추가되었습니다.
- `config.yaml`에서 `YAML` 병합 및 변수가 제거됩니다. 또한 `kustomize` 및 `ytt`와 같은 툴에서 `release.yaml` 파일을 더 쉽게 사용할 수 있습니다.

- 리소스 이름에 점 문자(".")가 포함되도록 지원이 추가되었습니다.
- **PodTemplate** 사양의 **hostAliases** 어레이는 호스트 이름 확인의 **Pod** 수준 제정의에 추가됩니다. 이를 위해 **/etc/hosts** 파일을 수정합니다.
- **\$(tasks.status)** 변수가 도입되어 작업의 집계 실행 상태에 액세스합니다.
- **Windows**용 진입점 바이너리 빌드가 추가되었습니다.

4.1.2.3. 더 이상 사용되지 않는 기능

- **when** 표현식에서 작성된 필드에 대한 지원에서는 **PascalCase**가 제거되었습니다. **when** 표현식은 소문자로 작성된 필드만 지원합니다.



참고

Tekton Pipelines v0.16(Operator v 1.2.x)에서 **when** 표현식과 함께 파이프라인을 적용한 경우 다시 적용해야 합니다.

- **Red Hat OpenShift Pipelines Operator**를 **v1.5**로 업그레이드하면 **openshift-client** 및 **openshift-client-v-1-5-0** 클러스터 작업에 **SCRIPT** 매개변수가 있습니다. 그러나 **ARGS** 매개변수와 **git** 리소스는 **openshift-client** 클러스터 작업의 사양에서 제거됩니다. 이는 중단된 변경 사항이며 **ClusterTask** 리소스의 **name** 필드에 특정 버전이 없는 클러스터 작업만 원활하게 업그레이드됩니다.

파이프라인 실행이 중단되지 않도록 하려면 업그레이드 후 **ARGS** 매개변수에 지정된 값을 클러스터 작업의 **SCRIPT** 매개변수로 이동하기 때문에 **SCRIPT** 매개변수를 사용합니다. 예를 들면 다음과 같습니다.

```
...
- name: deploy
  params:
    - name: SCRIPT
      value: oc rollout status <deployment-name>
  runAfter:
    - build
  taskRef:
    kind: ClusterTask
    name: openshift-client
...
```

- Red Hat OpenShift Pipelines Operator v1.4에서 v1.5로 업그레이드하면 TektonConfig 사용자 정의 리소스가 설치된 프로필 이름이 변경됩니다.**

표 4.2. TektonConfig 사용자 정의 리소스의 프로필

Pipeline 1.5의 프로필	Pipelines 1.4의 해당 프로필	설치된 Tekton 구성 요소
모두(기본 프로필)	모두(기본 프로필)	파이프라인, 트리거, 애드온
Basic	Default	파이프라인, 트리거
Lite	Basic	파이프라인



참고

TektonConfig 사용자 정의 리소스의 **config** 인스턴스에서 **profile: all**을 사용한 경우 리소스 사양을 변경할 필요가 없습니다.

그러나 설치된 **Operator**가 업그레이드 전에 **Default** 또는 **Basic** 프로필에 있는 경우 업그레이드 후 **TektonConfig** 사용자 정의 리소스의 **config** 인스턴스를 편집해야 합니다. 예를 들어 구성이 업그레이드 전에 **profile: basic**인 경우 **Pipelines 1.5**로 업그레이드한 후 **profile: lite**인지 확인합니다.

- disable-home-env-overwrite** 및 **disable-working-dir-overwrite** 필드는 더 이상 사용되지 않으며 향후 릴리스에서 제거됩니다. 이번 릴리스에서는 이전 버전과의 호환성을 위해 이러한 플래그의 기본값이 **true**로 설정됩니다.



참고

다음 릴리스에서는 **HOME** 환경 변수가 자동으로 **/tekton/home**으로 설정되지 않으며 작업 실행을 위해 기본 작업 디렉터리가 **/workspace**로 설정되지 않습니다. 이러한 기본값은 단계의 이미지 **Dockerfile**로 설정된 값과 충돌합니다.

- ServiceType** 및 **podTemplate** 필드는 **EventListener** 사양에서 제거됩니다.

- 컨트롤러 서비스 계정은 더 이상 네임스페이스를 나열하고 감시하기 위한 클러스터 전체 권한을 요청하지 않습니다.

- **EventListener** 리소스의 상태에는 **Ready**라는 새로운 조건이 있습니다.



참고

나중에 **EventListener** 리소스의 기타 상태 조건이 더 이상 사용되지 않고 **Ready** 상태 조건이 사용됩니다.

- **EventListener** 응답의 **eventListener** 및 **namespace** 필드는 더 이상 사용되지 않습니다. 대신 **eventListenerUID** 필드를 사용합니다.

- **replicas** 필드는 **EventListener** 사양에서 더 이상 사용되지 않습니다. 대신 **spec.replicas** 필드가 **KubernetesResource** 사양의 **spec.resources.kubernetesResource.replicas**로 이동됩니다.



참고

replicas 필드는 향후 릴리스에서 제거됩니다.

- 코어 인터셉터를 구성하는 이전 방법은 더 이상 사용되지 않습니다. 그러나 향후 릴리스에서 제거될 때까지 계속 작동합니다. 대신 **Trigger** 리소스의 인터셉터가 새로운 **ref** 및 **params** 기반 구문을 사용하여 구성됩니다. 결과 기본 웹훅은 새 트리거에 대해 이전 구문의 사용법을 새 구문으로 자동 전환합니다.

- **ClusterRoleBinding** 리소스에 대해 더 이상 사용되지 않는 **rbac.authorization.k8s.io/v1beta1** 대신 **rbac.authorization.k8s.io/v1**을 사용합니다.

- 클러스터 역할에서는 **serviceaccounts**, **secrets**, **configmaps**, **limitranges**와 같은 리소스에 대한 클러스터 전체 쓰기 권한이 제거됩니다. 또한 **deployments**, **statefulsets**, **deployment/finalizers**와 같은 리소스에 대한 클러스터 전체 액세스도 제거됩니다.

- **caching.internal.knative.dev** 그룹의 **image** 사용자 지정 리소스 정의는 **Tekton**에서 더 이상 사용하지 않으며 이 릴리스에서 제외되었습니다.

4.1.2.4. 확인된 문제

- **git-cli** 클러스터 작업은 **/root**가 사용자의 홈 디렉터리로 예상되는 **alpine/git** 기본 이미지에

서 빌드됩니다. 그러나 **git-cli** 클러스터 작업에는 명시적으로 설정되어 있지 않습니다.

Tekton에서 달리 지정하지 않는 한, **Tekton**에서 기본 홈 디렉토리는 작업의 모든 단계에 대해 **/tekton/home**으로 덮어씁니다. 기본 이미지의 **\$HOME** 환경 변수를 덮어쓰면 **git-cli** 클러스터 작업이 실패합니다.

이 문제는 다음 릴리스에서 수정될 예정입니다. **Red Hat OpenShift Pipelines 1.5** 및 이전 버전의 경우 다음 해결 방법 중 하나를 사용하여 **git-cli** 클러스터 작업이 실패하지 않도록 할 수 있습니다.


- 단계에서 **\$HOME** 환경 변수를 설정하여 덮어쓰지 않도록 합니다.

1. [선택 사항] **Operator**를 사용하여 **Red Hat OpenShift Pipelines**를 설치한 경우 **git-cli** 클러스터 작업을 별도의 작업에 복제합니다. 이 방법을 사용하면 **Operator**에서 클러스터 작업의 변경 사항을 덮어쓰지 않습니다.
2. **oc edit clustertasks git-cli** 명령을 실행합니다.
3. 예상되는 **HOME** 환경 변수를 단계 **YAML**에 추가합니다.

```

...
steps:
- name: git
  env:
  - name: HOME
    value: /root
  image: $(params.BASE_IMAGE)
  workingDir: $(workspaces.source.path)
...


```



주의

Operator가 설치한 **Red Hat OpenShift Pipelines**의 경우 **HOME** 환경 변수를 변경하기 전에 **git-cli** 클러스터 작업을 별도의 작업으로 복제하지 않으면 **Operator** 조정 중에 변경 사항을 덮어씁니다.

- **feature-flags** 구성 맵에서 **HOME** 환경 변수 덮어쓰기를 비활성화합니다.
 1. **oc edit -n openshift-pipelines configmap feature-flags** 명령을 실행합니다.
 2. **disable-home-env-overwrite** 플래그 값을 **true**로 설정합니다.



주의

- **Operator**를 사용하여 **Red Hat OpenShift Pipelines**를 설치한 경우 **Operator** 조정 중에 변경 사항을 덮어씁니다.
- **disable-home-env-overwrite** 플래그의 기본값을 수정하면 모든 작업의 기본 동작을 변경하므로 다른 작업과 클러스터 작업이 중단될 수 있습니다.

- 파이프라인의 기본 서비스 계정이 사용될 때 **HOME** 환경 변수의 덮어쓰기가 수행되므로 **git-cli** 클러스터 작업에 다른 서비스 계정을 사용합니다.
 1. 새로운 서비스 계정을 생성합니다.
 2. 생성한 서비스 계정에 **Git** 시크릿을 연결합니다.
 3. 작업 또는 파이프라인을 실행하는 동안 서비스 계정을 사용합니다.
- **IBM Power Systems, IBM Z 및 LinuxONE**에서 **s2i-dotnet** 클러스터 작업 및 **tkn hub** 명령은 지원되지 않습니다.
- **IBM Power Systems(ppc64le), IBM Z 및 LinuxONE(s390x)** 클러스터에서 **Maven** 및 **Jib-Maven** 클러스터 작업을 실행하는 경우 **MAVEN_IMAGE** 매개변수 값을 **maven:3.6.3-adoptopenjdk-11**으로 설정합니다.

4.1.2.5. 해결된 문제

- dag** 작업의 **when** 표현식은 다른 작업의 실행상태 ($\$(tasks.<pipelineTask>.status)$)에 액세스하는 컨텍스트 변수를 지정할 수 없습니다.
- PipelineRun** 리소스가 신속하게 삭제되고 다시 생성되는 경우 **volumeClaimTemplate PVC**를 삭제하여 생성된 경쟁 조건을 방지할 수 있으므로 소유자 이름 대신 소유자 **UID**를 사용합니다.
- 루트가 아닌 사용자가 트리거한 **build-base** 이미지의 **pullrequest-init**에 대한 새 **Dockerfile**이 추가됩니다.
- 파이프라인 또는 작업을 **-f** 옵션으로 실행하고 정의의 **param**에 **type**이 정의되지 않은 경우 파이프라인 또는 작업 실행이 자동으로 실패하는 대신 유효성 검사 오류가 생성됩니다.
- tkn start [task | pipeline | clustertask]** 명령의 경우 **--workspace** 플래그에 대한 설명이 일관되게 표시됩니다.
- 매개 변수를 구문 분석하는 동안 빈 배열이 발생하는 경우 해당 대화형 도움말이 빈 문자열로 표시됩니다.

4.1.3. Red Hat OpenShift Pipelines General Availability 1.4 릴리스 정보

Red Hat OpenShift Pipelines General Availability (GA)는 이제 OpenShift Container Platform 4.7에서 사용할 수 있습니다.



참고

안정적인 프리뷰 Operator 채널 외에도 Red Hat OpenShift Pipelines Operator 1.4.0에는 **ocp-4.6**, **ocp-4.5** 및 **ocp-4.4** 사용 중단 채널이 함께 제공됩니다. 이러한 더 이상 사용되지 않는 채널과 이에 대한 지원은 다음 Red Hat OpenShift Pipelines 릴리스에서 제거됩니다.

4.1.3.1. 호환성 및 지원 매트릭스

이 릴리스의 일부 기능은 현재 기술 프리뷰 단계에 있습니다. 이러한 실험적 기능은 프로덕션용이 아닙니다.

기술 프리뷰 기능 지원 범위

아래 표에서 기능은 다음 상태로 표시됩니다.

- **TP: 기술 프리뷰**
- **GA: 정식 출시일 (GA)**

해당 기능은 **Red Hat Customer Portal**의 지원 범위를 참조하십시오.

표 4.3. 호환성 및 지원 매트릭스

기능	버전	지원 상태
파이프라인	0.22	GA
CLI	0.17	GA
카탈로그	0.22	GA
Trigger	0.12	TP
파이프 라인 리소스	-	TP

질문이나 의견이 있으시면 제품팀에 이메일(pipelines-interest@redhat.com)로 보내주시기 바랍니다.

4.1.3.2. 새로운 기능

다음 섹션에서는 수정 및 안정성 개선 사항 외에 **Red Hat OpenShift Pipelines 1.4**의 새로운 기능도 소개합니다.

- 사용자 지정 작업에는 다음과 같은 향상된 기능이 있습니다.
 - 파이프라인 결과는 사용자 지정 작업에서 생성된 결과를 참조할 수 있습니다.

- 사용자 지정 작업에서는 작업 영역, 서비스 계정 및 **Pod** 템플릿을 사용하여 더 복잡한 사용자 지정 작업을 빌드할 수 있습니다.
- **finally** 작업에는 다음과 같은 향상된 기능이 있습니다.
 - **finally** 작업에서 **when** 표현식이 지원되므로 효율적으로 보호되는 실행 및 작업 재사용성을 개선할 수 있습니다.
 - **finally** 작업에서는 동일한 파이프라인 내의 모든 작업 결과를 사용하도록 구성할 수 있습니다.



참고

OpenShift Container Platform 4.7 웹 콘솔에서 **when** 표현식 및 **finally** 작업을 지원하지 않습니다.

- **dockercfg** 또는 **dockerconfigjson** 유형의 여러 보안에 대한 지원이 런타임 시 인증에 추가되었습니다.
- **git-clone** 작업을 사용하여 스파스-체크아웃을 지원하는 기능이 추가되었습니다. 이를 통해 리포지토리의 하위 집합만 로컬 복사본으로 복제할 수 있으며 복제된 리포지토리의 크기를 제한할 수 있습니다.
- 실제로 시작하지 않고 보류 중인 상태에서 파이프라인 실행을 생성할 수 있습니다. 로드가 많은 클러스터에서 **Operator**는 파이프라인 실행 시작 시간을 제어할 수 있습니다.
- 컨트롤러에 대해 **SYSTEM_NAMESPACE** 환경 변수를 수동으로 설정했는지 확인합니다. 이전에는 기본적으로 설정되었습니다.
- 이제 루트가 아닌 사용자가 파이프라인의 빌드-기반 이미지에 추가되어 **git-init**가 루트가 아닌 사용자로 리포지토리를 복제할 수 있습니다.
- 파이프라인 실행이 시작되기 전에 해결된 리소스 간 종속성을 확인하는 지원이 추가되어 있습니다. 파이프라인의 모든 결과 변수가 유효해야 하며 파이프라인의 선택적 작업 공간을 파이프

라인 실행을 시작하기 위해 필요한 작업으로만 전달할 수 있습니다.

- 컨트롤러 및 **Webhook**는 루트가 아닌 그룹으로 실행되며, 보안을 강화하기 위해 해당 기능이 제거되었습니다.
- **tkn pr logs** 명령을 사용하여 재시도된 작업 실행에 대한 로그 스트림을 확인할 수 있습니다.
- **tkn tr delete** 명령에서 **--clustertask** 옵션을 사용하여 특정 클러스터 작업과 연관된 모든 작업을 삭제할 수 있습니다.
- 새 **customResource** 필드를 도입하여 **EventListener** 리소스와 함께 **Knative** 서비스 사용에 대한 지원이 추가되었습니다.
- 이벤트 페이로드에서 **JSON** 형식을 사용하지 않으면 오류 메시지가 표시됩니다.
- **GitLab**, **BitBucket** 및 **GitHub**와 같은 소스 제어 인터셉터는 이제 새로운 **InterceptorRequest** 또는 **InterceptorResponse** 유형 인터페이스를 사용합니다.
- **JSON** 개체 또는 배열을 문자열에 인코딩할 수 있도록 새로운 **CEL** 함수 **marshalJSON**이 구현됩니다.
- **CEL** 및 소스 제어 코어 인터셉터를 제공하는 **HTTP** 처리기가 추가되었습니다. **tekton-pipelines** 네임스페이스에 배포된 단일 **HTTP** 서버에 4개의 코어 인터셉터를 패키징합니다. **EventListener** 오브젝트는 **HTTP** 서버를 통해 이벤트를 인터셉터로 전달합니다. 각 인터셉터는 다른 경로에서 사용할 수 있습니다. 예를 들어 **/cel** 경로에서 **CEL** 인터셉터를 사용할 수 있습니다.
- **pipelines-scc SCC(Security Context Constraint)**는 파이프라인의 기본 **pipeline** 서비스 계정과 함께 사용됩니다. 이 새 서비스 계정은 **anyuid**와 유사하지만 **OpenShift Container Platform 4.7**의 **SCC**에 대해 **YAML**에 정의된 것과 약간의 차이점이 있습니다.

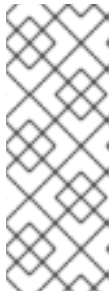
```
fsGroup:
  type: MustRunAs
```

4.1.3.3. 사용되지 않는 기능

-

파이프라인 리소스 스토리지의 **build-gcs** 하위 유형과 **gcs-fetcher** 이미지는 지원되지 않습니다.

- 클러스터 작업의 **taskRun** 필드에서 **tekton.dev/task** 레이블이 제거됩니다.
- Webhook의 경우 **admissionReviewVersions** 필드에 해당하는 **v1beta1** 값이 제거됩니다.
- 빌드 및 배포를 위한 **creds-init** 도우미 이미지가 제거됩니다.
- 트리거 사양 및 바인딩에서는 **template.ref**를 사용하도록 더 이상 사용되지 않는 필드 **template.name**이 제거됩니다. **ref** 필드를 사용하려면 모든 **eventListener** 정의를 업데이트해야 합니다.



참고

Pipelines 1.3.x 및 이전 버전에서 **Pipelines 1.4.0**으로 업그레이드하면 **template.name** 필드의 사용할 수 없으므로 이벤트 리스너가 중단됩니다. 이러한 경우 **Pipelines 1.4.1**을 사용하여 복원된 **template.name** 필드를 사용할 수 있습니다.

- **EventListener** 사용자 정의 리소스/개체의 경우 **Resource**가 사용되며 **PodTemplate** 및 **ServiceType** 필드는 더 이상 사용되지 않습니다.
- 더 이상 사용되지 않는 사양 스타일 내장 바인딩이 제거됩니다.
- **spec** 필드는 **triggerSpecBinding**에서 제거됩니다.
- 이벤트 ID 표현은 5자의 임의의 문자열에서 **UUID**로 변경됩니다.

4.1.3.4. 확인된 문제

- 개발자 화면에서 파이프라인 지표 및 트리거 기능은 **OpenShift Container Platform 4.7.6** 이상 버전에서만 사용할 수 있습니다.

- IBM Power Systems, IBM Z 및 LinuxONE에서는 `tkn hub` 명령이 지원되지 않습니다.
- IBM Power Systems(ppc64le), IBM Z 및 LinuxONE(s390x) 클러스터에서 Maven 및 Jib Maven 클러스터 작업을 실행하는 경우 `MAVEN_IMAGE` 매개변수 값을 `maven:3.6.3-adoptopenjdk-11`으로 설정합니다.
- 트리거 바인딩에 다음 구성이 있는 경우 **JSON** 형식이 잘못 처리되어 발생하는 오류를 트리거합니다.

```
params:
- name: github_json
  value: $(body)
```

문제를 해결하려면 다음을 수행합니다.

- 트리거 `v0.11.0` 이상을 사용하는 경우 `marshalJSON CEL` 함수를 사용하여 **JSON** 개체 또는 배열을 가져와 해당 오브젝트 또는 배열의 **JSON** 인코딩을 문자열로 반환합니다.
- 이전 트리거 버전을 사용하는 경우 트리거 템플릿에 다음 주석을 추가합니다.

```
annotations:
  triggers.tekton.dev/old-escape-quotes: "true"
```

- Pipelines 1.3.x에서 1.4.x로 업그레이드하는 경우 경로를 다시 생성해야 합니다.

4.1.3.5. 해결된 문제

- 이전에는 클러스터 작업의 작업 실행에서 `tekton.dev/task` 레이블이 제거되었으며 `tekton.dev/clusterTask` 레이블이 도입되었습니다. 해당 변경으로 인한 문제는 `clustertask describe` 및 `delete` 명령을 수정하여 해결됩니다. 또한 작업의 `lastrun` 기능이 수정되어 이전 버전의 파이프라인의 작업 실행에 적용되는 `tekton.dev/task` 레이블의 문제를 해결합니다.
- 대화형 `tkn pipeline start pipelinename`을 수행할 때 `PipelineResource`가 대화형으로 생성됩니다. 리소스 상태가 `nil`이 아닌 경우 `tkn p start` 명령은 리소스 상태를 출력합니다.
- 이전에는 `tekton.dev/task=name` 레이블이 클러스터 작업에서 생성된 작업 실행에서 제거되었습니다. 이번 수정에서는 `tkn clustertask start` 명령을 `--last` 플래그로 수정하여 생성된 작업

실행에서 `tekton.dev/task=name` 라벨을 확인합니다.

- 작업에서 인라인 작업 사양을 사용하는 경우 이제 `tkn pipeline describe` 명령을 실행할 때 해당 작업 실행이 파이프라인에 포함되고, 작업 이름이 포함된 것으로 반환됩니다.
- `tkn version` 명령은 구성된 `kubeConfiguration namespace` 또는 클러스터에 대한 액세스 없이 설치된 `Tekton CLI` 툴 버전을 표시하도록 수정되었습니다.
- 인수가 예기치 않은 것이거나 두 개 이상의 인수가 사용되는 경우 `tkn completion` 명령에서 오류가 발생합니다.
- 이전에는 파이프라인 사양에 중첩된 `finally` 작업으로 인해 `v1alpha1` 버전으로 변환되고 `v1beta1` 버전으로 복원된 `finally` 작업이 손실되었습니다. 변환 중에 발생하는 이 오류는 잠재적인 데이터 손실을 방지하기 위해 수정되었습니다. 파이프라인은 이제 파이프라인 사양에 중첩된 `finally` 작업에서 실행되며 알파 버전에 저장되지만 나중에 역직렬화됩니다.
- 이전에는 서비스 계정에 `{}`로 `secret` 필드가 있는 경우 `Pod` 생성에 오류가 발생했습니다. 빈 시크릿 이름을 가진 `GET` 요청이 오리소스 이름을 비워 둘 수 없다는 오류를 반환했기 때문에 이 작업은 `CouldntGetTask`로 실패합니다. 이 문제는 `kubeclient GET` 요청에서 시크릿 이름이 비어 있지 않도록 방지하여 해결되었습니다.
- 이제 `v1beta1 API` 버전이 있는 파이프라인은 `finally` 작업을 손실하지 않고 `v1alpha1` 버전과 함께 요청할 수 있습니다. 반환된 `v1alpha1` 버전을 적용하면 리소스가 `v1beta1`로 저장되고 `finally` 섹션은 원래 상태로 복원됩니다.
- 이전에는 컨트롤러의 설정되지 않은 `selfLink` 필드로 인해 `Kubernetes v1.20` 클러스터에서 오류가 발생했습니다. 임시 수정으로 `CloudEvent` 소스 필드는 자동으로 채워진 `selfLink` 필드의 값이 없이 현재 소스 `URI`와 일치하는 값으로 설정됩니다.
- 이전에는 `gcr.io`와 같은 점이 있는 시크릿 이름으로 인해 작업 실행 생성에 실패했습니다. 이는 내부적으로 볼륨 마운트 이름의 일부로 사용되는 시크릿 이름 때문에 발생했습니다. 볼륨 마운트 이름은 `RFC1123 DNS` 레이블을 준수하고, 이름의 일부로 점을 허용하지 않습니다. 이 문제는 점을 대시로 대체하여 읽을 수 있는 이름을 만들어 해결되었습니다.
- 이제 `finally` 작업에서 컨텍스트 변수의 유효성을 검사합니다.
- 이전 버전에서는 작업 실행 조정기에서 생성된 `Pod` 이름을 포함하는 이전 상태 업데이트가

없는 작업 실행을 통과하면 작업 실행 조정기는 작업 실행과 연결된 **Pod**를 나열했습니다. 작업 실행 조정기에서는 **Pod**에 전파된 작업 실행 레이블을 사용하여 **Pod**를 찾았습니다. 작업 실행 중에 이러한 레이블을 변경하면 코드에서 기존 **pod**를 찾지 못했습니다. 결과적으로 중복된 **pod**가 생성되었습니다. 이 문제는 **Pod**를 찾을 때 작업 실행 조정기를 **tekton.dev/taskRun Tekton-controlled** 라벨만 사용하도록 변경하여 해결되었습니다.

- - 이전 버전에서는 파이프라인에서 선택적 작업 영역을 수락하여 파이프라인 작업으로 전달하면 누락된 작업 공간 바인딩이 선택적 작업 공간에 유효한 상태인 경우에도 작업 영역을 제공하지 않은 경우 실행 조정기가 중지되어 오류가 발생했습니다. 이 문제는 선택적 작업 영역을 제공하지 않아도 파이프라인 실행 조정기에서 작업 실행을 생성하지 못하도록 하여 해결되었습니다.
- - 단계 상태의 정렬 순서는 단계 컨테이너의 순서와 일치합니다.
- - 이전에는 **Pod**에 **CreateContainerConfigError**가 발생할 때 작업 실행 상태가 **unknown**으로 설정되어 이로 인해 **Pod**가 시간 초과될 때까지 작업 및 파이프라인이 실행되었습니다. 이 문제는 **Pod**에 **CreateContainerConfigError** 이유가 발생할 때 작업이 실패로 설정되도록 작업 실행 상태를 **false**로 설정하여 해결되었습니다.
- - 이전에는 파이프라인 실행이 완료된 후 첫 번째 조정 시 파이프라인 결과가 해결되었습니다. 이로 인해 해결에 실패하여 파이프라인 실행의 **Succeeded** 조건을 덮어쓸 수 있습니다. 결과적으로 최종 상태 정보가 손실되어 파이프라인의 작동 조건을 모니터링하는 모든 서비스에 혼란을 줄 수 있습니다. 이 문제는 파이프라인 실행이 **Succeeded** 또는 **True** 조건이 될 때 파이프라인 결과의 해결을 조정의 끝으로 이동하여 해결됩니다.
- - 이제 실행 상태 변수가 확인됩니다. 이렇게 하면 실행 상태에 액세스하기 위해 컨텍스트 변수를 검증하는 동안 작업 결과를 확인하는 것을 방지할 수 있습니다.
- - 이전 버전에서는 잘못된 변수가 포함된 파이프라인 결과가 변수의 리터럴 표현식을 그대로 사용하여 파이프라인 실행에 추가되었습니다. 따라서 결과가 올바르게 설정되어 있는지를 평가하는 것은 쉽지 않았습니다. 이 문제는 실패한 작업 실행을 참조하는 파이프라인 실행 결과 필터링하여 해결되었습니다. 이제 잘못된 변수가 포함된 파이프라인 결과는 파이프라인 실행에서 전혀 배출되지 않습니다.
- - tkn eventlistener describe** 명령이 템플릿 없이 충돌하지 않도록 수정되었습니다. 트리거 참조에 대한 세부 정보도 표시합니다.
- - Pipelines 1.3.x** 및 이전 버전에서 **Pipelines 1.4.0**으로 업그레이드하면 **template.name**이 사용할 수 없으므로 이벤트 리스너가 중단됩니다. **Pipelines 1.4.1**에서는 트리거에서 이벤트 리스너 중단을 방지하기 위해 **template.name**이 복원되었습니다.

- **Pipelines 1.4.1**에서는 **OpenShift Container Platform 4.7** 기능 및 동작에 맞게 **ConsoleQuickStart** 사용자 정의 리소스가 업데이트되었습니다.

4.1.4. Red Hat OpenShift Pipelines Technology Preview 1.3 릴리스 정보

4.1.4.1. 새로운 기능

이제 **OpenShift Container Platform 4.7**에서 **Red Hat OpenShift Pipelines TP(Technology Preview) 1.3**을 사용할 수 있습니다. 다음을 지원하도록 **Red Hat OpenShift Pipelines TP 1.3**이 업데이트되었습니다.

- **Tekton Pipelines 0.19.0**
- **Tekton tkn CLI 0.15.0**
- **Tekton Triggers 0.10.2**
- **Tekton Catalog 0.19.0** 기반 클러스터 작업
- **OpenShift Container Platform 4.7**의 **IBM Power Systems**
- **OpenShift Container Platform 4.7**의 **IBM Z 및 LinuxONE**

다음 섹션에서는 수정 및 안정성 개선 사항 외에 **Red Hat OpenShift Pipelines 1.3**의 새로운 기능도 소개합니다.

4.1.4.1.1. 파이프라인

- **S2I** 및 **Buildah** 작업과 같은 이미지를 빌드하는 작업에서 이제 이미지 **SHA**를 포함하는 빌드된 이미지의 **URL**을 내보냅니다.
- **Conditions CRD**(사용자 정의 리소스 정의)가 더 이상 사용되지 않기 때문에 사용자 정의 작업을 참조하는 파이프라인 작업에서 조건이 비활성화되었습니다.

- **spec.steps[].imagePullPolicy** 및 **spec.sidecar[].imagePullPolicy**의 Task CRD에 변수 확장이 추가되었습니다.
- **disable-creds-init** 기능 플래그를 **true**로 설정하여 Tekton의 기본 제공 자격 증명 메커니즘을 비활성화할 수 있습니다.
- 해결된 **When** 표현식이 **PipelineRun** 구성의 **Status** 필드에 있는 **Skipped Tasks** 및 **Task Runs** 섹션에 나열됩니다.
- **git init** 명령으로 재귀 하위 모듈을 복제할 수 있습니다.
- **Task CR** 작성자가 **Task** 사양에 있는 단계의 타임아웃을 지정할 수 있습니다.
- 진입점 이미지의 기반을 **distroless/static:nonroot** 이미지로 하여 기본 이미지에 있는 **cp** 명령에 의존하지 않고도 대상에 자체 복사 모드를 제공할 수 있습니다.
- 구성 플래그 **require-git-ssh-secret-known-hosts**를 사용하여 **Git SSH** 보안에서 알려진 호스트를 생략하지 않도록 할 수 있습니다. 플래그 값이 **true**로 설정된 경우 **Git SSH** 보안에 **known_host** 필드를 포함해야 합니다. 플래그 기본값은 **false** 입니다.
- 선택적 작업 공간의 개념이 도입되었습니다. 작업 또는 파이프라인에서 작업 공간을 선택적으로 선언하고 작업 공간 존재 여부에 따라 조건부로 동작을 변경할 수 있습니다. 작업 실행 또는 파이프라인 실행에서도 해당 작업 공간을 생략하여 작업 또는 파이프라인 동작을 수정할 수 있습니다. 기본 작업 실행 작업 공간은 생략된 선택적 작업 공간 대신 추가되지 않습니다.
- Tekton의 자격 증명 초기화 과정에서 **SSH**가 아닌 **URL**과 함께 사용되는 **SSH** 자격 증명을 감지하고 **Git** 파이프라인 리소스에서 그 반대의 경우도 마찬가지이며, 단계 컨테이너에 경고를 기록합니다.
- **Pod** 템플릿에서 지정한 유사성을 유사성 도우미에서 덮어쓰는 경우 작업 실행 컨트롤러에서 경고 이벤트를 내보냅니다.
- 작업 실행이 완료되면 작업 실행 조정기에서 내보낸 클라우드 이벤트에 대한 지표를 기록합니다. 여기에는 재시도 횟수가 포함됩니다.

4.1.4.1.2. Pipeline CLI

- **tkn condition list, tkn triggerbinding list, tkn eventlistener list, tkn clustertask list, tkn clustertriggerbinding list** 명령에 **--no-headers flag**에 대한 지원이 추가되었습니다.
- **--last** 또는 **--use** 옵션은 함께 사용 시 **--prefix-name** 및 **--timeout** 옵션을 덮어씁니다.
- **EventListener** 로그를 볼 수 있도록 **tkn eventlistener logs** 명령이 추가되었습니다.
- **tekton hub** 명령이 **tkn CLI**에 통합되었습니다.
- **--nocolour** 옵션이 **--no-color**로 변경되었습니다.
- **tkn triggertemplate list, tkn condition list, tkn triggerbinding list, tkn eventlistener list** 명령에 **--all-namespaces** 플래그가 추가되었습니다.

4.1.4.1.3. Trigger

- **EventListener** 템플릿에 리소스 정보를 지정할 수 있습니다.
- **EventListener** 서비스 계정에 모든 트리거 리소스에 대한 **get** 동사 외에 **list** 및 **watch** 동사도 있어야 합니다. 따라서 **Listers**를 사용하여 **EventListener, Trigger, TriggerBinding, TriggerTemplate, ClusterTriggerBinding** 리소스에서 데이터를 가져올 수 있습니다. 이 기능을 사용하여 여러 정보원을 지정하지 않고 **Sink** 오브젝트를 생성하고 **API** 서버에 직접 호출할 수 있습니다.
- 변경 불가능한 입력 이벤트 본문을 지원하기 위해 새로운 **Interceptor** 인터페이스가 추가되었습니다. 인터셉터에서 새 **extensions** 필드에 데이터 또는 필드를 추가할 수는 있지만 입력 본문을 수정하여 변경 불가능으로 설정할 수는 없습니다. **CEL** 인터셉터는 이 새로운 **Interceptor** 인터페이스를 사용합니다.
- **namespaceSelector** 필드가 **EventListener** 리소스에 추가되었습니다. 이 필드는 **EventListener** 리소스에서 이벤트 처리를 위해 **Trigger** 오브젝트를 가져올 수 있는 네임스페이스를 지정하는 데 사용됩니다. **namespaceSelector** 필드를 사용하려면 **EventListener** 서비스 계정에 클러스터 역할이 있어야 합니다.

트리거 **EventListener** 리소스에서 **eventlistener Pod**에 대한 종단 간 보안 연결을 지원합니다.

- "를 \"로 교체하여 **TriggerTemplates** 리소스의 이스케이프 매개변수 동작이 제거되었습니다.
- **Kubernetes** 리소스를 지원하는 새로운 **resources** 필드가 **EventListener** 사양의 일부로 도입되었습니다.
- **ASCII** 문자열의 대문자 및 소문자를 지원하는 **CEL** 인터셉터의 새 기능이 추가되었습니다.
- 트리거의 **name** 및 **value** 필드를 사용하거나 이벤트 리스너를 사용하여 **TriggerBinding** 리소스를 포함할 수 있습니다.
- **PodSecurityPolicy** 구성이 제한된 환경에서 실행되도록 업데이트되었습니다. 따라서 컨테이너를 루트로 실행해서는 안 됩니다. 또한 **Pod** 보안 정책 사용을 위한 역할 기반 액세스 제어가 클러스터 범위에서 네임스페이스 범위로 이동했습니다. 그 결과 트리거에서 네임스페이스와 관련이 없는 다른 **Pod** 보안 정책을 사용할 수 없습니다.
- 포함된 트리거 템플릿에 대한 지원이 추가되었습니다. **name** 필드를 사용하여 포함된 템플릿을 참조하거나 템플릿을 **spec** 필드 내에 포함할 수 있습니다.

4.1.4.2. 사용되지 않는 기능

- **PipelineResources CRD**를 사용하는 파이프라인 템플릿이 더 이상 사용되지 않으며 향후 릴리스에서 제거됩니다.
- **template.name** 필드가 더 이상 **template.ref** 필드 대신 사용되지 않으며 향후 릴리스에서 제거됩니다.
- **--check** 명령에 대한 **-c** 약어가 제거되었습니다. 또한 전역 **tkn** 플래그가 **version** 명령에 추가되었습니다.

4.1.4.3. 확인된 문제

- **CEL** 오버레이는 들어오는 이벤트 본문을 수정하는 대신 새로운 최상위 **extensions** 함수에

필드를 추가합니다. **TriggerBinding** 리소스는 `$(extensions.<key>)` 구문을 사용하여 이 새로운 **extensions** 함수 내의 값에 액세스할 수 있습니다. `$(body.<overlay-key>)` 구문 대신 `$(extensions.<key>)` 구문을 사용하도록 바인딩을 업데이트합니다.

- "를 \"로 교체하여 이스케이프 매개변수 동작이 제거되었습니다. 이전 이스케이프 매개변수 동작을 유지해야 하는 경우 **TriggerTemplate** 사양에 `tekton.dev/old-escape-quotes: true`" 주석을 추가합니다.
- 트리거 내부의 **name** 및 **value** 필드를 사용하거나 이벤트 리스너를 사용하여 **TriggerBinding** 리소스를 포함할 수 있습니다. 그러나 단일 바인딩에 **name** 및 **ref** 필드를 둘 다 지정할 수는 없습니다. **ref** 필드를 사용하여 포함된 바인딩의 **TriggerBinding** 리소스 및 **name** 필드를 참조합니다.
- 인터셉터는 **EventListener** 리소스의 네임스페이스 외부에 있는 **secret**을 참조할 수 없습니다. 'EventListener' 리소스의 네임스페이스에 보안을 포함해야 합니다.
- **Triggers 0.9.0** 이상에서는 본문 또는 헤더 기반 **TriggerBinding** 매개변수가 이벤트 페이로드에서 누락되거나 잘못된 형식으로 되어 있는 경우 오류를 표시하는 대신 기본값을 사용합니다.
- **Tekton Pipelines 0.16.x**를 사용하여 **WhenExpressions** 개체로 생성된 작업 및 파이프라인을 다시 적용하여 **JSON** 주석을 수정해야 합니다.
- 파이프라인에서 선택적 작업 영역을 수락하고 이를 작업에 제공할 때 작업 영역을 제공하지 않으면 파이프라인 실행이 중단됩니다.
- 연결이 끊긴 환경에서 **Buildah** 클러스터 작업을 사용하려면 **Dockerfile**에서 내부 이미지 스트림을 기본 이미지로 사용하는지 확인한 다음 모든 **S2I** 클러스터 작업과 동일한 방식으로 사용합니다.

4.1.4.4. 해결된 문제

- 이벤트 본문에 **Extensions** 필드를 추가하면 **CEL** 인터셉터에서 추가한 확장이 **Webhook** 인터셉터에 전달됩니다.
- **LogOptions** 필드를 사용하여 로그 리더에 대한 활동 타임아웃을 구성할 수 있습니다. 그러나 10초 내의 기본 타임아웃 동작은 유지됩니다.

- **log** 명령은 작업 실행 또는 파이프라인 실행이 완료되면 **--follow** 플래그를 무시하고 라이브 로그 대신 사용 가능한 로그를 읽습니다.
- 다음 Tekton 리소스에 대한 참조: **EventListener, TriggerBinding, ClusterTriggerBinding, Condition, TriggerTemplate** 이 표준화되어 **tkn** 명령의 모든 사용자 대응 메시지에서 일관성을 유지합니다.
- 이전에는 **--use-taskrun <anceled-task-run-name>**, **--use-pipelinerun <anceled-pipeline-run-name>** 또는 **--last** 플래그를 사용하여 취소된 작업 실행 또는 파이프라인 실행을 시작하면 새 실행이 취소되었습니다. 이 버그가 해결되었습니다.
- 파이프라인이 조건에 따라 실행되는 경우 실패하지 않도록 **tkn pr desc** 명령이 향상되었습니다.
- **tkn tr delete** 명령을 **--task** 옵션과 함께 사용하여 작업을 삭제하고 이름이 동일한 클러스터 작업이 있는 경우 클러스터 작업의 작업 실행도 삭제됩니다. 이 문제를 해결하려면 **TaskRefKind** 필드를 사용하여 작업을 필터링합니다.
- **tkn triggertemplate describe** 명령을 실행하면 출력에 **apiVersion** 값의 일부만 표시됩니다. 예를 들어 **triggers.tekton.dev/v1alpha1** 대신 **triggers.tekton.dev**만 표시되었습니다. 이 버그가 해결되었습니다.
- **Webhook**는 특정 조건에서 리스를 가져오지 못하고 제대로 작동하지 않습니다. 이 버그가 해결되었습니다.
- **v0.16.3**에서 생성된 **When** 표현식이 있는 파이프라인을 **v0.17.1** 이상에서 실행할 수 있습니다. 주석의 첫 글자로 대문자와 소문자가 모두 지원되므로 업그레이드 후 이전 버전에서 생성한 파이프라인 정의를 다시 적용할 필요가 없습니다.
- 기본적으로 고가용성을 위해 **leader-election-ha** 필드가 활성화됩니다. **disable-ha** 컨트롤러 플래그를 **true**로 설정하면 고가용성 지원이 비활성화됩니다.
- 중복된 클라우드 이벤트 문제가 수정되었습니다. 조건으로 인해 상태, 이유 또는 메시지가 변경될 때만 클라우드 이벤트가 전송됩니다.
- **PipelineRun** 또는 **TaskRun** 사양에 서비스 계정 이름이 없는 경우 컨트롤러는 **config-**

defaults 구성 맵의 서비스 계정 이름을 사용합니다. **config-defaults** 구성 맵에도 서비스 계정 이름이 없으면 컨트롤러에서 사양에 서비스 계정 이름을 **default**로 설정합니다.

- 동일한 영구 볼륨 클레임이 여러 작업 공간에 사용되지만 하위 경로가 다른 경우 유사성 도우미와의 호환성을 검증하는 기능이 지원됩니다.

4.1.5. Red Hat OpenShift Pipelines Technology Preview 1.2 릴리스 정보

4.1.5.1. 새로운 기능

이제 OpenShift Container Platform 4.6에서 Red Hat OpenShift Pipelines TP(Technology Preview) 1.2를 사용할 수 있습니다. 다음을 지원하도록 Red Hat OpenShift Pipelines TP 1.2가 업데이트되었습니다.

- **Tekton Pipelines 0.16.3**
- **Tekton tkn CLI 0.13.1**
- **Tekton Triggers 0.8.1**
- **Tekton Catalog 0.16 기반 클러스터 작업**
- **OpenShift Container Platform 4.6의 IBM Power Systems**
- **OpenShift Container Platform 4.6의 IBM Z 및 LinuxONE**

다음 섹션에서는 수정 및 안정성 개선 사항 외에 Red Hat OpenShift Pipelines 1.2의 새로운 기능도 소개합니다.

4.1.5.1.1. 파이프라인

- 이 Red Hat OpenShift Pipelines 릴리스에서는 오프라인 설치를 지원합니다.



참고

제한된 환경에서의 설치에 현재 IBM Power Systems, IBM Z, LinuxONE에서 지원되지 않습니다.

- 이제 **conditions** 리소스 대신 **when** 필드를 사용하여 특정 기준이 충족될 때만 작업을 실행할 수 있습니다. **WhenExpression** 리소스의 주요 구성 요소는 **Input, Operator, Values**입니다. 모든 표현식이 **True**로 평가되면 작업이 실행됩니다. **When** 표현식이 **False**로 평가되면 작업을 건너뜁니다.
- 작업 실행이 취소되거나 타임아웃되는 경우 단계 상태가 업데이트됩니다.
- **git-init**에서 사용하는 기본 이미지를 빌드하는 데 **Git LFS(Large File Storage)** 지원이 제공됩니다.
- **taskSpec** 필드를 사용하여 작업이 파이프라인에 포함된 경우 라벨 및 주석과 같은 메타데이터를 지정할 수 있습니다.
- 파이프라인 실행에서 클라우드 이벤트를 지원합니다. 클라우드 이벤트 파이프라인 리소스에서 보내는 클라우드 이벤트에 **backoff**를 통해 재시도할 수 있습니다.
- **Task** 리소스에서 선언해도 **TaskRun** 리소스에서 명시적으로 제공하지 않는 모든 작업 공간에 기본 **Workspace** 구성을 설정할 수 있습니다.
- **PipelineRun** 네임스페이스 및 **TaskRun** 네임스페이스에 대한 네임스페이스 변수 보간을 지원합니다.
- **TaskRun** 리소스가 유사성 도우미와 연결되어 있을 때 여러 개의 영구 볼륨 클레임 작업 공간이 사용되지 않는지 확인하기 위해 **TaskRun** 오브젝트에 대한 검증 작업이 추가되었습니다. 영구 볼륨 클레임 작업 공간이 두 개 이상 사용되면 **TaskRunValidationFailed** 조건이 포함된 작업 실행이 실패합니다. 기본적으로 유사성 도우미는 **Red Hat OpenShift Pipelines**에서 비활성화되어 있으므로 이 도우미를 사용하려면 활성화해야 합니다.

4.1.5.1.2. Pipeline CLI

- **tkn task describe, tkn taskrun describe, tkn clustertask describe, tkn pipeline describe, tkn pipelinerun describe** 명령에서 다음을 수행합니다.

- **Task, TaskRun, ClusterTask, Pipeline, PipelineRun** 리소스 중 하나만 있는 경우 각 리소스를 자동으로 선택합니다.
- **Task, TaskRun, ClusterTask, Pipeline, PipelineRun** 리소스의 결과를 출력에 각각 표시합니다.
- **Task, TaskRun, ClusterTask, Pipeline, PipelineRun** 리소스에 선언된 작업 공간을 각각 표시합니다.
- **tkn clustertask start** 명령에 **--prefix-name** 옵션을 사용하여 작업 실행 이름에 접두사를 지정할 수 있습니다.
- **tkn clustertask start** 명령에 대화형 모드가 지원됩니다.
- **TaskRun** 및 **PipelineRun** 오브젝트에 대한 로컬 또는 원격 파일 정의를 사용하여 파이프라인에서 지원하는 **PodTemplate** 속성을 지정할 수 있습니다.
- **tkn clustertask start** 명령에 **--use-params-defaults** 옵션을 사용하여 **ClusterTask** 구성에 설정된 기본값을 사용하고 작업 실행을 생성할 수 있습니다.
- 일부 매개변수에 기본값이 지정되지 않은 경우 **tkn pipeline start** 명령의 **--use-param-defaults** 플래그는 대화형 모드를 표시합니다.

4.1.5.1.3. Trigger

- **parseYAML**이라는 **CEL(Common Expression Language)** 함수가 추가되어 **YAML** 문자열을 문자열 맵으로 구문 분석합니다.
- **CEL** 표현식 구문 분석에 대한 오류 메시지가 개선되어 표현식을 평가하는 동안 그리고 평가 환경을 생성하기 위해 후크 본문을 구문 분석할 때 더 세부적인 내용이 표시됩니다.
- 부울 값 및 맵이 **CEL** 오버레이 메커니즘에서 표현식 값으로 사용되는 경우 부울 값 및 맵 마샬링이 지원됩니다.

- 다음 필드가 **EventListener** 오브젝트에 추가되었습니다.
 - **replicas** 필드를 사용하면 **YAML** 파일의 복제본 수를 지정하여 이벤트 리스너에서 여러 개의 **Pod**를 실행할 수 있습니다.
 - **NodeSelector** 필드를 사용하면 **EventListener** 오브젝트에서 이벤트 리스너 **Pod**를 특정 노드에 예약할 수 있습니다.
- **Webhook** 인터셉터에서 **EventListener-Request-URL** 헤더를 구문 분석하여 이벤트 리스너로 처리 중인 원래 요청 **URL**에서 매개변수를 추출할 수 있습니다.
- 이벤트 리스너에 있는 주석을 배포 **Pod**, 서비스 **Pod** 및 기타 **Pod**로 전파할 수 있습니다. 서비스 또는 배포에 대한 사용자 정의 주석을 덮어쓰므로 해당 주석을 전파하려면 이벤트 리스너 주석에 추가해야 합니다.
- 사용자가 **spec.replicas** 값을 **negative** 또는 **zero**로 지정하는 경우에도 **EventListener** 사양의 복제본을 올바르게 검증할 수 있습니다.
- **TriggerRef** 필드를 통해 **EventListener** 사양 내의 **TriggerCRD** 오브젝트를 참조로 지정하여 **TriggerCRD** 오브젝트를 별도로 생성한 다음 **EventListener** 사양 내에서 바인딩할 수 있습니다.
- **TriggerCRD** 오브젝트에 검증을 수행하고 기본값을 사용할 수 있습니다.

4.1.5.2. 사용되지 않는 기능

- 이제 **resourcetemplate**과 **triggertemplate** 리소스 매개변수를 혼동하지 않도록 **\$(params)** 매개변수가 **triggertemplate** 리소스에서 제거되고 **\$(tt.params)**로 대체되었습니다.
- 선택적 **EventListenerTrigger** 기반 인증 수준의 **ServiceAccount** 참조가 오브젝트 참조에서 **ServiceAccountName** 문자열로 변경되었습니다. 이로 인해 **ServiceAccount** 참조가 **EventListenerTrigger** 오브젝트와 동일한 네임스페이스에 있습니다.
- **Conditions CRD**(사용자 정의 리소스 정의)가 더 이상 사용되지 않습니다. 대신 **WhenExpressions CRD**를 사용합니다.

- PipelineRun.Spec.ServiceAccountNames** 오브젝트가 더 이상 사용되지 않고 **PipelineRun.Spec.TaskRunSpec[].ServiceAccountName** 오브젝트로 교체됩니다.

4.1.5.3. 확인된 문제

- 이 **Red Hat OpenShift Pipelines** 릴리스에서는 오프라인 설치를 지원합니다. 그러나 클러스터 작업에서 사용하는 일부 이미지는 연결이 끊긴 클러스터에서 작업하려면 미리링해야 합니다.
- openshift** 네임스페이스의 파이프라인은 **Red Hat OpenShift Pipelines Operator**를 설치 제거한 후에도 삭제되지 않습니다. 이 파이프라인을 삭제하려면 **oc delete pipelines -n openshift --all** 명령을 사용합니다.
- Red Hat OpenShift Pipelines Operator**를 설치 제거해도 이벤트 리스너는 제거되지 않습니다.

해결 방법은 **EventListener** 및 **Pod CRD**를 제거하는 것입니다.

- foregroundDeletion** 종료자를 사용하여 **EventListener** 오브젝트를 다음과 같이 편집합니다.

```
$ oc patch el/<eventlistener_name> -p '{"metadata":{"finalizers":["foregroundDeletion"]}}' --type=merge
```

예를 들면 다음과 같습니다.

```
$ oc patch el/github-listener-interceptor -p '{"metadata":{"finalizers":["foregroundDeletion"]}}' --type=merge
```

- EventListener CRD**를 삭제합니다.

```
$ oc patch crd/eventlisteners.triggers.tekton.dev -p '{"metadata":{"finalizers":[]}}' -type=merge
```

- IBM Power Systems(ppc64le)** 또는 **IBM Z(s390x)** 클러스터에서 명령 사양 없이 다중 아키텍처 컨테이너 이미지 작업을 실행하면 **TaskRun** 리소스가 실패하고 다음 오류 메시지가 표시됩니다.

Error executing command: fork/exec /bin/bash: exec format error

해결 방법은 아키텍처별 컨테이너 이미지를 사용하거나 **sha256** 다이제스트를 지정하여 올바른 아키텍처를 가리키는 것입니다. **sha256** 다이제스트를 가져오려면 다음을 입력합니다.

```
$ skopeo inspect --raw <image_name>| jq '.manifests[] | select(.platform.architecture == "<architecture>") | .digest'
```

4.1.5.4. 해결된 문제

- **CEL** 필터, **Webhook** 유효성 검증기의 오버레이, 인터셉터의 표현식을 확인하는 간단한 구문 검증 기능이 추가되었습니다.
- **Trigger**가 더 이상 기본 배포 및 서비스 오브젝트에 설정된 주석을 덮어쓰지 않습니다.
- 이전에는 이벤트 리스너에서 이벤트 수락을 중지했습니다. 이 수정에서는 이 문제를 해결하기 위해 **EventListener** 싱크에 120초의 유틸 상태 타임아웃이 추가되었습니다.
- 이전에는 **Failed(Canceled)** 상태로 파이프라인 실행을 취소하면 성공 메시지가 표시되었습니다. 이제 오류 메시지를 표시하도록 수정되었습니다.
- **tkn eventlistener list** 명령에서 나열된 이벤트 리스너의 상태를 제공하므로 사용 가능한 리스너를 쉽게 확인할 수 있습니다.
- 트리거가 설치되지 않았거나 리소스가 없는 경우 **triggers list** 및 **triggers describe** 명령에 대한 오류 메시지가 일관되게 표시됩니다.
- 이전에는 클라우드 이벤트를 제공하는 동안 다수의 유틸 연결이 빌드되었습니다. 이 문제를 해결하기 위해 **cloudeventclient** 구성에 **DisableKeepAlives: true** 매개변수가 추가되었습니다. 따라서 모든 클라우드 이벤트에 대해 새로운 연결이 설정됩니다.
- 이전에는 지정된 유형의 자격 증명이 제공되지 않은 경우에도 **creds-init** 코드에서 디스크에 빈 파일을 작성했습니다. 이번 수정에서는 올바른 주석이 있는 보안에서 실제로 마운트된 자격 증명에 있는 경우에만 파일을 작성하도록 **creds-init** 코드를 수정했습니다.

4.1.6. Red Hat OpenShift Pipelines Technology Preview 1.1 릴리스 정보

4.1.6.1. 새로운 기능

이제 OpenShift Container Platform 4.5에서 Red Hat OpenShift Pipelines TP(Technology Preview) 1.1을 사용할 수 있습니다. 다음을 지원하도록 Red Hat OpenShift Pipelines TP 1.1이 업데이트되었습니다.

- **Tekton Pipelines 0.14.3**
- **Tekton tkn CLI 0.11.0**
- **Tekton Triggers 0.6.1**
- **Tekton Catalog 0.14 기반 클러스터 작업**

다음 섹션에서는 수정 및 안정성 개선 사항 외에 Red Hat OpenShift Pipelines 1.1의 새로운 기능도 소개합니다.

4.1.6.1.1. 파이프라인

- 이제 파이프라인 리소스 대신 작업 공간을 사용할 수 있습니다. 파이프라인 리소스는 디버그하기 어렵고 범위가 제한되며 작업의 재사용 가능성을 낮추기 때문에 OpenShift Pipelines에서 작업 공간을 사용할 것을 권장합니다. 작업 공간에 대한 자세한 내용은 OpenShift Pipelines 이해 섹션을 참조하십시오.
- 볼륨 클레임 템플릿에 대한 작업 공간 지원이 추가되었습니다.
 - 이제 파이프 라인 실행 및 작업 실행에 대한 볼륨 클레임 템플릿을 작업 공간의 볼륨 소스로서 추가할 수 있습니다. 그런 다음 **tekton-controller**가 파이프라인의 모든 작업 실행에 대해 PVC로 표시되는 템플릿을 사용하여 **PVC(PersistentVolumeClaim)**를 생성합니다. 따라서 여러 **Task**에 걸쳐 있는 작업 공간을 바인드할 때마다 **PVC** 구성을 정의해야 합니다.
 - 볼륨 클레임 템플릿이 볼륨 소스로 사용될 때 **PVC**의 이름 검색 지원에서 이제 변수 대체를 사용할 수 있습니다.
- 감사 개선 지원:

- 이제 **PipelineRun.Status** 필드에 파이프라인의 모든 작업 실행 상태와 파이프라인 실행의 진행 상황을 모니터링하기 위해 파이프라인 실행을 인스턴스화하는 데 사용되는 파이프라인 사양이 포함됩니다.
- **Pipeline** 결과가 **Pipeline** 사양 및 **PipelineRun** 상태에 추가되었습니다.
- 이제 **TaskRun.Status** 필드에 **TaskRun** 리소스를 인스턴스화하는 데 사용되는 정확한 작업 사양이 포함됩니다.
- 조건에 기본 매개변수 적용을 지원합니다.
- 클러스터 작업을 참조하여 생성된 작업 실행이 이제 **tekton.dev/task** 레이블 대신 **tekton.dev/clusterTask** 레이블을 추가합니다.
- 이제 **kubeconfigwriter**가 리소스 구조에 **ClientKeyData** 및 **ClientCertificateData** 구성을 추가하여 파이프라인 리소스 유형 클러스터를 **kubeconfig-creator** 작업으로 교체할 수 있습니다.
- 이제 **feature-flags** 및 **config-defaults** 구성 맵의 이름을 이제 사용자 지정할 수 있습니다.
- 작업 실행에서 사용하는 **pod** 템플릿에서 호스트 네트워크에 대한 지원을 사용할 수 있습니다.
- 이제 **Affinity Assistant**를 사용하여 작업 공간 볼륨을 공유하는 작업 실행에서 노드 선호도를 지원할 수 있습니다. **OpenShift Pipelines**에서는 기본적으로 노드 선호도가 비활성화됩니다.
- **Pod** 템플릿이 **imagePullSecrets**를 지정하도록 업데이트되어 **Pod**를 시작할 때 컨테이너 런타임에서 컨테이너 이미지 가져오기를 승인하는 데 사용할 보안을 확인합니다.
- 컨트롤러가 작업 실행을 업데이트하지 못하는 경우 작업 실행 컨트롤러에서 경고 이벤트를 발송하도록 지원합니다.
- 애플리케이션 또는 구성 요소에 속하는 리소스를 식별하도록 표준 또는 권장 **k8s** 레이블이 모든 리소스에 추가되었습니다.

- 이제 **Entrypoint** 프로세스에 신호 알림이 전송되며, 이러한 신호는 **Entrypoint** 프로세스의 전용 **PID Group**을 사용하여 전파됩니다.
- 이제 작업 실행 사양을 사용하여 런타임에 작업 수준에서 **pod** 템플릿을 설정할 수 있습니다.
- **Kubernetes** 이벤트 발송 지원 :
 - 이제 컨트롤러가 추가 작업 실행 수명 주기 이벤트(**taskrun started** 및 **taskrun running**)에 대한 이벤트를 발송합니다.
 - 이제 파이프라인 실행 컨트롤러가 파이프라인이 시작될 때마다 이벤트를 발송합니다.
- 이제 기본 **Kubernetes** 이벤트 외에 작업 실행에 대한 클라우드 이벤트 지원도 제공됩니다. 생성, 시작 및 실패와 같은 작업 실행 이벤트를 클라우드 이벤트로서 발송하도록 컨트롤러를 구성할 수 있습니다.
- 파이프라인 실행 및 작업 실행에서 적절한 이름을 참조하도록 **\$context.<task|taskRun|pipeline|pipelineRun>.name** 변수 사용을 지원합니다.
- 이제 파이프라인 실행 매개변수에 대한 유효성 검사를 사용하여 파이프라인 실행에서 파이프라인에 필요한 모든 매개변수가 제공되는지 확인할 수 있습니다. 이를 통해 파이프라인 실행에서 필수 매개변수 외에 추가 매개변수도 제공할 수 있습니다.
- 이제 파이프라인 **YAML** 파일의 **finally** 필드를 사용하여 모든 작업을 성공적으로 완료한 후 또는 파이프라인의 작업 중 하나가 실패한 후 파이프라인이 종료되기 전에 항상 실행될 파이프라인 내 작업을 지정할 수 있습니다.
- 이제 **git-clone** 클러스터 작업을 사용할 수 있습니다.

4.1.6.1.2. Pipeline CLI

- 이제 포함된 트리거 바인딩 지원을 **tkn evenlistener describe** 명령에 사용할 수 있습니다.

- 하위 명령을 권장하고 잘못된 하위 명령을 사용할 때 의견을 제시하는 기능을 지원합니다.
- 이제 파이프라인에 작업이 한 개뿐인 경우 **tkn task describe** 명령에 의해 작업이 자동으로 선택됩니다.
- 이제 **tkn task start** 명령에 **--use-param-defaults** 플래그를 지정하여 기본 매개변수 값으로 작업을 시작할 수 있습니다.
- 이제 **tkn pipeline start** 또는 **tkn task start** 명령과 함께 **--workspace** 옵션을 사용하여 파이프라인 실행 또는 작업 실행에 대한 볼륨 클레임 템플릿을 지정할 수 있습니다.
- 이제 **tkn pipelinerun logs** 명령으로 **finally** 섹션에 나열된 최종 **Task**에 대한 로그를 표시할 수 있습니다.
- 이제 **tkn task start** 명령과 함께 **pipeline, pipelinerun, task, taskrun, clustertask, pipelineresource**와 같은 **tkn** 리소스에 대한 **describe** 하위 명령에 대화형 모드가 지원됩니다.
- 이제 **tkn version** 명령으로 클러스터에 설치된 트리거의 버전을 표시할 수 있습니다.
- 이제 **tkn pipeline describe** 명령으로 파이프라인에 사용된 작업에 대해 지정된 매개변수 값과 시간초과 사항을 표시할 수 있습니다.
- **tkn pipelinerun describe** 및 **tkn taskrun describe** 명령에 가장 최근 파이프라인 실행 또는 작업 실행을 각각 설명하는 **--last** 옵션에 대한 지원이 추가되었습니다.
- 이제 **tkn pipeline describe** 명령으로 파이프라인의 작업에 적용 가능한 조건을 표시할 수 있습니다.
- 이제 **tkn resource list** 명령과 함께 **--no-headers** 및 **--all-namespaces** 플래그를 사용할 수 있습니다.

4.1.6.1.3. Trigger

- 이제 다음과 같은 **CEL(Common Expression Language)** 기능을 사용할 수 있습니다.
 - **URL**의 일부를 구문 분석하고 추출하기 위한 **parseURL**
 - **deployment WebHook**의 **payload** 필드에 있는 문자열에 포함된 **JSON** 값 유형을 구문 분석하는 **parseJSON**
- **Bitbucket**의 **WebHook**에 대한 새로운 인터셉터가 추가되었습니다.
- 이제 이벤트 리스너가 **kubectl get** 명령으로 나열될 때 추가 필드로 **Address URL** 및 **Available status**를 표시합니다.
- 트리거 템플릿과 리소스 템플릿 매개변수 간의 혼동을 줄이기 위해 이제 트리거 템플릿 매개변수에 **\$(params.<paramName>)** 대신 **\$(tt.params.<paramName>)** 구문을 사용합니다.
- 보안 또는 관리 문제로 인해 모든 노드가 오염된 경우에도 이벤트 리스너가 동일한 구성으로 배포되도록 **EventListener CRD**에 **tolerations**를 추가할 수 있습니다.
- 이제 **URL/live**에서 이벤트 리스너 배포에 대한 준비 프로브를 추가할 수 있습니다.
- 이벤트 리스너 트리거에 **TriggerBinding** 사양을 포함하기 위한 지원이 추가되었습니다.
- 이제 권장 **app.kubernetes.io** 레이블을 사용하여 **Trigger** 리소스에 주석을 삽입할 수 있습니다.

4.1.6.2. 사용되지 않는 기능

이 릴리스에서는 더 이상 사용되지 않은 기능은 다음과 같습니다.

- **clustertask** 및 **clustertriggerbinding** 명령을 포함하여 모든 클러스터 단위 명령에 **--namespace** 또는 **-n** 플래그는 더 이상 사용되지 않습니다. 향후 릴리스에서 제거됩니다.
-

이벤트 리스너 내 **triggers.bindings**의 **name** 필드가 더 이상 사용되지 않고 향후 릴리스에서 제거될 것이며 **ref** 필드 사용을 권장합니다.

- 파이프라인 변수 보간 구문과 혼동을 줄이기 위해 **\$(params)**를 사용한 트리거 템플릿의 변수 보간은 더 이상 사용되지 않고, **\$(tt.params)** 사용을 권장합니다. **\$(params.<paramName>)** 구문은 향후 릴리스에서 제거됩니다.
- 클러스터 작업에서 **tekton.dev/task** 레이블이 더 이상 사용되지 않습니다.
- **TaskRun.Status.ResourceResults.ResourceRef** 필드가 더 이상 사용되지 않으며 제거됩니다.
- **tkn pipeline create**, **tkn task create** 및 **tkn resource create -f** 하위 명령이 제거되었습니다.
- **tkn** 명령에서 네임스페이스 유효성 검사가 제거되었습니다.
- 기본 시간 초과 **1h**와 **tkn ct start** 명령에 대한 **-t** 플래그가 제거되었습니다.
- **s2i** 클러스터 작업이 더 이상 사용되지 않습니다.

4.1.6.3. 확인된 문제

- 조건에서 작업 공간을 지원하지 않습니다.
- **tkn clustertask start** 명령에 **--workspace** 옵션과 대화형 모드가 지원되지 않습니다.
- **\$(params.<paramName>)** 구문의 역호환성 지원에 따라 파이프라인 특정 매개변수와 함께 트리거 템플릿을 사용하도록 수정되었습니다. 이는 트리거 **WebHook**에서 트리거 매개변수를 파이프라인 매개변수와 구별할 수 없기 때문입니다.
- **tekton_taskrun_count** 및 **tekton_taskrun_duration_seconds_count**에 대한 **promQL** 쿼리를 실행할 때 **Pipeline** 메트릭이 잘못된 값을 보고합니다.

- 작업 공간에 제공된 기존 PVC 이름이 없는 경우에도 파이프라인 실행 및 작업 실행이 **Running** 및 **Running(Pending)** 상태를 각각 유지합니다.

4.1.6.4. 해결된 문제

- 이전에는 작업과 클러스터 작업 이름이 동일할 때 `tkn task delete<name>--trs` 명령으로 작업과 클러스터 작업이 모두 삭제되었습니다. 이번 수정에서는 이 명령으로 작업 <name>에 의해 생성된 작업 실행만 삭제됩니다.
- 이전에는 `tkn pr delete -p<name>--keep 2` 명령을 `--keep` 플래그와 함께 사용할 때 `-p` 플래그가 무시되고 최근 두 개를 제외한 모든 파이프라인 실행이 삭제되었습니다. 이번 수정에서는 이 명령으로 최근 두 개를 제외하고 파이프라인 <name>에 의해 생성된 파이프라인 실행만 삭제됩니다.
- 이제 `tkn trigger template describe` 출력에 YAML 형식 대신 테이블 형식으로 리소스 템플릿이 표시됩니다.
- 전에는 컨테이너에 새 사용자를 추가할 때 `buildah` 클러스터 작업이 실패했습니다. 수정판에서는 이러한 문제가 해결되었습니다.

4.1.7. Red Hat OpenShift Pipelines Technology Preview 1.0 릴리스 정보

4.1.7.1. 새로운 기능

이제 OpenShift Container Platform 4.4에서 Red Hat OpenShift Pipelines TP(Technology Preview) 1.0을 사용할 수 있습니다. 다음을 지원하도록 Red Hat OpenShift Pipelines TP 1.0이 업데이트되었습니다.

- **Tekton Pipelines 0.11.3**
- **Tekton tkn CLI 0.9.0**
- **Tekton Triggers 0.4.0**
- **Tekton Catalog 0.11 기반 클러스터 작업**

다음 섹션에서는 수정 및 안정성 개선 사항 외에 **Red Hat OpenShift Pipelines 1.0**의 새로운 기능도 소개합니다.

4.1.7.1.1. 파이프라인

- **v1beta1 API** 버전을 지원합니다.
- 개선된 제한 범위를 지원합니다. 이전에는 작업 실행 및 파이프라인 실행에 대해서만 제한 범위를 지정했습니다. 이제 제한 범위를 명시적으로 지정할 필요가 없습니다. 네임스페이스의 최소 제한 범위가 사용됩니다.
- 작업 결과 및 작업 매개 변수를 사용하여 작업 간 데이터 공유를 지원합니다.
- 이제 **HOME** 환경 변수와 단계의 작업 디렉토리를 덮어쓰지 않도록 파이프라인을 구성할 수 있습니다.
- 작업 단계와 유사하게 **sidecars**가 이제 스크립트 모드를 지원합니다.
- 이제 작업 실행 **podTemplate** 리소스에서 다른 스케줄러 이름을 지정할 수 있습니다.
- **Star Array Notation**을 사용한 변수 대체를 지원합니다.
- 이제 개별 네임스페이스를 모니터링하도록 **Tekton** 컨트롤러를 구성할 수 있습니다.
- 이제 새로운 설명 필드가 파이프라인, 작업, 클러스터 작업, 리소스 및 조건의 사양에 추가되었습니다.
- **Git** 파이프라인 리소스에 프록시 매개변수를 추가합니다.

4.1.7.1.2. Pipeline CLI

- 이제 다음과 같은 **tkn** 리소스에 **describe** 하위 명령이 추가됩니다. **EventListener**, 조건, **TriggerTemplate**, **ClusterTask** 및 **TriggerSBinding**.

- **v1alpha1**에 대한 이전 버전과의 호환성과 함께 다음 리소스에 **v1beta 1** 지원이 추가되었습니다. **ClusterTask, Task, Pipeline, PipelineRun, TaskRun**.
- 이제 **tkn task list, tkn pipeline list, tkn taskrun list, tkn pipelinerun list**와 같은 **--all-namespaces** 플래그 옵션을 사용하여 모든 네임스페이스의 출력을 나열할 수 있습니다.
 - **--no-headers** 플래그 옵션을 사용하면 명령의 출력에 헤더 없이 정보가 표시되도록 향상되었습니다.
- 이제 **tkn pipelines start** 명령에서 **--use-param-defaults** 플래그를 지정하여 기본 매개변수 값을 사용하여 파이프라인을 시작할 수 있습니다.
- 이제 **tkn pipeline start** 및 **tkn task start** 명령에 작업 공간에 대한 지원이 추가되었습니다.
- **describe, delete, list** 하위 명령과 함께 이제 새로운 **clustertriggerbinding** 명령이 추가되었습니다.
- 이제 로컬 또는 원격 **yaml** 파일을 사용하여 **Pipeline Run**을 직접 시작할 수 있습니다.
- 이제 **describe** 하위 명령이 이제 보강되고 상세한 출력을 표시합니다. **description, timeout, param description** 및 **sidecar status**와 같은 새로운 필드가 추가되면서 특정 **tkn** 리소스에 대한 자세한 정보가 명령 출력에 제공됩니다.
- 네임스페이스에 있는 작업이 한 개뿐인 경우 **tkn task log** 명령으로 바로 로그를 표시할 수 있습니다.

4.1.7.1.3. Trigger

- 트리거 (Trigger)가 이제 **v1alpha1** 및 **v1beta1** 파이프라인 리소스를 모두 생성할 수 있습니다.
- 새로운 **CEL(Common Expression Language)** 인터셉터 기능 **-compareSecret** 지원 이 기능은 보안을 유지하면서 문자열을 **CEL** 표현식의 보안과 비교합니다.

- 이벤트 리스너 트리거 수준에서 인증 및 승인을 지원합니다.

4.1.7.2. 사용되지 않는 기능

이 릴리스에서는 더 이상 사용되지 않은 기능은 다음과 같습니다.

- Steps** 사양의 환경 변수 **\$HOME** 및 변수 **workingDir**은 더 이상 사용되지 않으며 향후 릴리스에서 변경될 수 있습니다. 현재 **Step** 컨테이너의 **HOME** 및 **workingDir** 매개 변수가 **/tekton/home**과 **/workspace**을 각각 덮어씁니다.

향후 릴리스에서 이 두 필드는 수정되지 않으며, 컨테이너 이미지 및 **Task YAML**에 정의된 값으로 설정될 것입니다. 이번 릴리스에서는 **disable-home-env-overwrite** 및 **disable-working-directory-overwrite** 플래그를 사용하여 **HOME** 및 **workingDir** 변수의 덮어쓰기 기능을 비활성화하십시오.

- 다음 명령은 더 이상 사용되지 않는 명령들이며 향후 릴리스에서 제거될 수 있습니다: **tkn pipeline create**, **tkn task create**

- tkn resource create** 명령과 함께 **-f** 플래그가 더 이상 사용되지 않습니다. 향후 릴리스에서 제거될 수 있습니다.

- tkn clustertask create** 명령에서 **-t** 플래그와 **--timeout** 플래그(초 형식)가 더 이상 사용되지 않습니다. 이제 지속 시간 초과 형식만 지원됩니다(예: **1h30s**). 더 이상 사용되지 않는 이러한 플래그는 향후 릴리스에서 제거될 수 있습니다.

4.1.7.3. 확인된 문제

- 이전 버전의 **Red Hat OpenShift Pipelines**에서 업그레이드하는 경우 **Red Hat OpenShift Pipelines** 버전 **1.0**으로 업그레이드하기 전에 기존 배포를 삭제해야 합니다. 기존 배포를 삭제하려면 먼저 사용자 정의 리소스를 삭제한 다음 **Red Hat OpenShift Pipelines Operator**를 설치 제거해야 합니다. 자세한 내용은 **Red Hat OpenShift Pipeline** 설치 제거 섹션을 참조하십시오.

- 동일한 **v1alpha1** 작업을 두 번 이상 제출하면 오류가 발생합니다. **v1alpha1** 작업을 다시 제출할 때 **oc apply** 대신 **oc replace** 명령을 사용하십시오.

- 컨테이너에 사용자가 새로 추가되면 **buildah** 클러스터 작업이 작동하지 않습니다.

Operator가 설치되면 **buildah** 클러스터 작업에 대한 **--storage-driver** 플래그가 지정되지 않으므로 플래그가 기본값으로 설정됩니다. 스토리지 드라이버가 잘못 설정되는 경우도 발생할 수 있습니다. 사용자가 새로 추가되면 잘못된 스토리지 드라이버로 인해 다음 오류가 발생되면서 **buildah** 클러스터 작업이 실패합니다.

```
useradd: /etc/passwd.8: lock file already used
useradd: cannot lock /etc/passwd; try again later.
```

이 문제를 해결하려면 **buildah-task.yaml** 파일에서 **--storage-driver** 플래그 값을 **overlay**로 직접 설정하십시오.

1. **cluster-admin** 권한으로 클러스터에 로그인합니다.

```
$ oc login -u <login> -p <password> https://openshift.example.com:6443
```

2. **oc edit** 명령을 사용하여 **buildah** 클러스터 작업을 편집합니다.

```
$ oc edit clustertask buildah
```

buildah clustertask YAML 파일의 현재 버전이 **EDITOR** 환경 변수에 의해 설정된 편집기에서 열립니다.

3. **Steps** 필드에서 다음 **command** 필드를 찾습니다.

```
command: ['buildah', 'bud', '--format=$(params.FORMAT)', '--tls-verify=$(params.TLSVERIFY)', '--layers', '-f', '$(params.DOCKERFILE)', '-t', '$(resources.outputs.image.url)', '$(params.CONTEXT)']
```

4. **command** 필드를 다음으로 변경합니다.

```
command: ['buildah', '--storage-driver=overlay', 'bud', '--format=$(params.FORMAT)', '--tls-verify=$(params.TLSVERIFY)', '--no-cache', '-f', '$(params.DOCKERFILE)', '-t', '$(params.IMAGE)', '$(params.CONTEXT)']
```

5. 파일을 저장하고 종료합니다.

또는 **Pipelines** → **Cluster Tasks** → **buildah**로 이동하여 웹 콘솔에서 직접 **buildah** 클러스터 작업 **YAML** 파일을 수정할 수도 있습니다. **Actions** 메뉴에서 **Edit Cluster Task**를 선택하고

이전 프로시저에서 안내한 대로 **command** 필드를 변경합니다.

4.1.7.4. 해결된 문제

- 이전에는 이미지 빌드가 이미 진행 중인 경우에도 **DeploymentConfig** 작업이 새 배포 빌드를 트리거했습니다. 이로 인해 파이프라인 배포가 실패로 끝납니다. 수정판에서는 진행 중인 배포를 마칠 때까지 대기하는 **oc rollout status** 명령으로 이제 **deploy task** 명령을 대체합니다.
- **APP_NAME** 매개변수에 대한 지원이 이제 파이프라인 템플릿에 추가됩니다.
- 전에는 **Java S2I**용 파이프라인 템플릿이 레지스트리에서 이미지를 찾지 못했습니다. 수정판에서는 사용자가 제공한 **IMAGE_NAME** 매개변수 대신 기존 이미지 파이프라인 리소스를 사용하여 이미지를 검색합니다.
- 이제 모든 **OpenShift Pipelines** 이미지가 **Red Hat UBI(Universal Base Images, 범용 기본 이미지)**를 기반으로 합니다.
- 전에는 **tekton-pipelines** 이외 네임스페이스에 파이프라인을 설치했을 때 **tkn version** 명령에서 파이프라인 버전을 **unknown**으로 표시했습니다. 수정판에서는 **tkn version** 명령으로 이제 모든 네임스페이스에 올바른 파이프라인 버전을 표시할 수 있습니다.
- **tkn version** 명령에 더 이상 **-c** 플래그가 지원되지 않습니다.
- 관리자 권한이 없는 사용자도 이제 클러스터 트리거 비인딩 목록을 볼 수 있습니다.
- **CEL** 인터셉터에 대한 이벤트 리스너 **CompareSecret** 기능이 수정되었습니다.
- 작업과 클러스터 작업의 이름이 같은 경우 작업 및 클러스터 작업에 대한 **list**, **describe** 및 **start** 하위 명령의 출력이 이제 올바르게 표시됩니다.
- 이전에는 **OpenShift Pipelines Operator**에서 권한이 필요한 **SCC(보안 컨텍스트 제약 조건)**를 수정하여 클러스터 업그레이드 도중 오류가 발생했습니다. 이 오류는 이제 수정되었습니다.
- **tekton-pipelines** 네임스페이스에서 모든 작업 실행 및 파이프라인 실행의 시간 초과 값이

이제 구성 맵을 사용하여 **default-timeout-minutes** 필드 값으로 설정됩니다.

- 전에는 관리자 권한이 없는 사용자에게는 웹 콘솔의 파이프라인 섹션이 표시되지 않았습니다. 이 문제는 이제 해결되었습니다.

4.2. OPENSIFT PIPELINES 이해

Red Hat OpenShift Pipelines는 **Kubernetes** 리소스 기반의 클라우드 네이티브 **CI/CD**(연속 통합 및 연속 제공) 솔루션입니다. **Tekton** 빌딩 블록을 사용하여 기본 구현 세부 사항을 요약함으로써 여러 플랫폼에서 배포를 자동화합니다. **Tekton**은 **Kubernetes** 배포 전반에서 이식 가능한 **CI/CD Pipeline**을 정의하는 데 사용되는 여러 가지 표준 **CRD(Custom Resource Definitions)**를 도입합니다.

4.2.1. 주요 기능

- **Red Hat OpenShift Pipelines**는 격리된 컨테이너에서 필요한 모든 종속 항목이 포함된 파이프라인을 실행하는 서버리스 **CI/CD** 시스템입니다.
- **Red Hat OpenShift Pipelines**는 마이크로 서비스 기반 아키텍처에서 작업하는 분산된 팀을 위해 설계되었습니다.
- **Red Hat OpenShift Pipelines**는 쉽게 확장하고 기존 **Kubernetes** 툴과 통합할 수 있는 표준 **CI/CD** 파이프라인 정의를 사용하므로 필요에 따라 스케일링할 수 있습니다.
- **Red Hat OpenShift Pipelines**를 사용하면 모든 **Kubernetes** 플랫폼에서 이식 가능한 **S2I(Source-to-Image)**, **Buildah**, **Buildpacks**, **Kaniko** 등의 **Kubernetes** 툴로 이미지를 빌드할 수 있습니다.
- **OpenShift Container Platform** 개발자 콘솔을 사용하여 **Tekton** 리소스를 생성하고, 파이프라인 실행 로그를 검토하고, **OpenShift Container Platform** 네임스페이스에서 파이프라인을 관리할 수 있습니다.

4.2.2. OpenShift Pipeline 개념

이 안내서에서는 다양한 파이프라인 개념을 소개합니다.

4.2.2.1. Task

Tasks는 Pipeline 구성 블록이며, 순차적으로 실행되는 단계들로 구성됩니다. 기본적으로 입력 및 출력의 기능입니다. 개별적으로 또는 파이프라인의 일부로 작업을 실행할 수 있습니다. 작업은 재사용이 가능하며 여러 파이프라인에서 사용할 수 있습니다.

Steps는 작업에 의해 순차적으로 실행되어 이미지 작성과 같은 특정 목표를 달성하는 일련의 명령입니다. 모든 작업은 Pod로 실행되고 각 단계는 해당 Pod 내에서 컨테이너로 실행됩니다. 여러 단계가 동일한 Pod 내에서 실행되기 때문에 이러한 단계에서 파일, 구성 맵, 보안을 캐싱하기 위해 동일한 볼륨에 액세스할 수 있습니다.

다음 예는 apply-manifests Task를 보여줍니다.

```

apiVersion: tekton.dev/v1beta1 1
kind: Task 2
metadata:
  name: apply-manifests 3
spec: 4
  workspaces:
  - name: source
  params:
  - name: manifest_dir
    description: The directory in source that contains yaml manifests
    type: string
    default: "k8s"
  steps:
  - name: apply
    image: image-registry.openshift-image-registry.svc:5000/openshift/cli:latest
    workingDir: /workspace/source
    command: ["/bin/bash", "-c"]
    args:
    - |-
      echo Applying manifests in $(params.manifest_dir) directory
      oc apply -f $(params.manifest_dir)
      echo -----

```

1

Task API 버전은 v1beta1입니다.

2

Kubernetes 오브젝트 유형은 Task입니다.

3

이 작업의 고유 이름입니다.

4

작업의 매개변수 및 단계 목록과 작업에서 사용하는 작업 공간입니다.

이 작업은 **Pod**를 시작하고 해당 **Pod** 내에서 지정된 이미지를 사용하는 컨테이너를 실행하여 지정된 명령을 실행합니다.

4.2.2.2. When 표현식

When 표현식은 파이프라인 내에서 작업 실행 기준을 설정하여 작업 실행을 보호합니다. 여기에는 특정 기준이 충족될 때만 작업을 실행할 수 있는 구성 요소 목록이 포함되어 있습니다. **when** 표현식은 파이프라인 **YAML** 파일의 **finally** 필드를 사용하여 지정된 최종 작업 세트에서도 지원됩니다.

when 표현식의 주요 구성 요소는 다음과 같습니다.

- **입력:** 매개 변수, 작업 결과 및 실행 상태와 같은 정적 입력 또는 변수를 지정합니다. 유효한 입력을 입력해야 합니다. 유효한 입력을 입력하지 않으면 기본값은 빈 문자열입니다.
- **operator:** 일련의 값에 대한 입력의 관계를 지정합니다. **Operator** 값으로 **in** 또는 **notin**을 입력합니다.
- **값:** 문자열 값의 배열을 지정합니다. 매개 변수, 결과 및 바인딩된 작업 영역 상태와 같은 정적 값 또는 변수의 비어 있지 않은 배열을 입력합니다.

선언된 **when** 표현식은 작업이 실행되기 전에 평가됩니다. **when** 표현식 값이 **True**이면 작업이 실행됩니다. **when** 표현식 값이 **False**이면 작업을 건너뜁니다.

다양한 사용 사례에서 **when** 표현식을 사용할 수 있습니다. 예를 들면 다음과 같은 경우입니다.

- 이전 작업의 결과는 예상대로 표시됩니다.
- **Git** 리포지토리의 파일이 이전 커밋에서 변경되었습니다.

- 레지스트리에 이미지가 있습니다.
- 선택적 작업 영역을 사용할 수 있습니다.

다음 예제에서는 파이프라인 실행에 대한 **when** 표현식을 보여줍니다. 파이프라인 실행은 다음 기준이 충족되는 경우에만 **create-file** 작업을 실행합니다. **path** 매개 변수는 **README.md**이고, **check-file** 작업의 **exists** 결과가 **yes**인 경우에만 **echo-file-exists** 작업이 실행됩니다.

```

apiVersion: tekton.dev/v1beta1
kind: PipelineRun ①
metadata:
  generateName: guarded-pr-
spec:
  serviceAccountName: 'pipeline'
  pipelineSpec:
    params:
      - name: path
        type: string
        description: The path of the file to be created
    workspaces:
      - name: source
        description: |
          This workspace is shared among all the pipeline tasks to read/write common resources
    tasks:
      - name: create-file ②
        when:
          - input: "${(params.path)}"
            operator: in
            values: ["README.md"]
        workspaces:
          - name: source
            workspace: source
        taskSpec:
          workspaces:
            - name: source
              description: The workspace to create the readme file in
          steps:
            - name: write-new-stuff
              image: ubuntu
              script: 'touch $(workspaces.source.path)/README.md'
      - name: check-file
        params:
          - name: path
            value: "${(params.path)}"
        workspaces:
          - name: source
            workspace: source
        runAfter:
          - create-file
        taskSpec:

```

```

params:
  - name: path
workspaces:
  - name: source
    description: The workspace to check for the file
results:
  - name: exists
    description: indicates whether the file exists or is missing
steps:
  - name: check-file
    image: alpine
    script: |
      if test -f ${workspaces.source.path}/${params.path}; then
        printf yes | tee /tekton/results/exists
      else
        printf no | tee /tekton/results/exists
      fi
- name: echo-file-exists
  when: 3
  - input: "${tasks.check-file.results.exists}"
    operator: in
    values: ["yes"]
  taskSpec:
    steps:
      - name: echo
        image: ubuntu
        script: 'echo file exists'
...
- name: task-should-be-skipped-1
  when: 4
  - input: "${params.path}"
    operator: notin
    values: ["README.md"]
  taskSpec:
    steps:
      - name: echo
        image: ubuntu
        script: exit 1
...
finally:
  - name: finally-task-should-be-executed
    when: 5
    - input: "${tasks.echo-file-exists.status}"
      operator: in
      values: ["Succeeded"]
    - input: "${tasks.status}"
      operator: in
      values: ["Succeeded"]
    - input: "${tasks.check-file.results.exists}"
      operator: in
      values: ["yes"]
    - input: "${params.path}"
      operator: in
      values: ["README.md"]
  taskSpec:
    steps:

```



```

- name: echo
  image: ubuntu
  script: 'echo finally done'
params:
- name: path
  value: README.md
workspaces:
- name: source
volumeClaimTemplate:
spec:
accessModes:
- ReadWriteOnce
resources:
requests:
storage: 16Mi

```

1

Kubernetes 개체의 유형을 지정합니다. 예에서는 **PipelineRun**입니다.

2

Pipeline에서 사용되는 작업 **create-file**입니다.

3

check-file 작업에서 **exists** 결과가 **yes**인 경우에만 **echo-file-exists** 작업을 실행하도록 지정하는 **when** 표현식입니다.

4

path 매개 변수가 **README.md**인 경우에만 **task-s shouldId-be-skipped-1** 작업을 건너뛰도록 지정하는 **When** 표현식입니다.

5

echo-file-exists 작업의 실행 상태와 작업 상태가 **Succeeded**인 경우에만, **finally-task-should-be-executed** 작업을 실행하도록 지정하는 **when** 표현식은 **check-file** 작업의 **exists** 결과는 **yes**이고 **path** 매개 변수는 **README.md**입니다.

OpenShift Container Platform 웹 콘솔의 Pipeline Run 세부 정보 페이지에 다음과 같이 작업의 상태와 **when** 표현식이 표시됩니다.

- 모든 기준이 충족됩니다. 유리한 형태로 표시되는 **when** 표현식 기호와 태스크는 녹색입니다.

•

다음 중 임의의 기준이 충족되지 않습니다. 작업을 건너뛵니다. 건너뛰기된 작업 및 **when** 표현식 기호는 회색입니다.

- 어떤 기준도 충족되지 않습니다. 작업을 건너뛵니다. 건너뛰기된 작업 및 **when** 표현식 기호는 회색입니다.
- 작업 실행에 실패합니다. 실패한 작업 및 **when** 표현식 기호는 빨간색입니다.

4.2.2.3. 마지막 작업

finally 작업은 파이프라인 YAML 파일의 **finally** 필드를 사용하여 지정된 최종 작업 집합입니다. **finally** 작업은 파이프라인 실행이 성공적으로 실행되는지 여부에 관계없이 항상 파이프라인 내의 작업을 실행합니다. **finally** 작업은 해당 파이프라인이 종료되기 전에 모든 파이프라인 작업이 실행된 후 병렬로 실행됩니다.

finally 작업은 동일한 파이프라인 내의 모든 작업 결과를 사용하도록 구성할 수 있습니다. 이 접근 방식은 이 최종 작업이 실행되는 순서를 변경하지 않습니다. 모든 최종 작업이 실행된 후 다른 최종 작업과 동시에 실행됩니다.

다음 예제에서는 **clone-cleanup-workspace** 파이프라인의 코드 스니펫을 보여줍니다. 이 코드는 리포지토리를 공유 작업 공간으로 복제하고 작업 영역을 정리합니다. 파이프라인 작업을 실행한 후 파이프라인 YAML 파일의 **finally** 섹션에 지정된 **cleanup** 작업이 작업 영역을 정리합니다.

```

apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: clone-cleanup-workspace 1
spec:
  workspaces:
    - name: git-source 2
  tasks:
    - name: clone-app-repo 3
      taskRef:
        name: git-clone-from-catalog
      params:
        - name: url
          value: https://github.com/tektoncd/community.git
        - name: subdirectory
          value: application
      workspaces:
        - name: output
          workspace: git-source
  finally:
    - name: cleanup 4

```

```

taskRef: 5
  name: cleanup-workspace
workspaces: 6
  - name: source
    workspace: git-source
- name: check-git-commit
params: 7
  - name: commit
    value: ${tasks.clone-app-repo.results.commit}
taskSpec: 8
  params:
    - name: commit
  steps:
    - name: check-commit-initialized
      image: alpine
      script: |
        if [[ ! $(params.commit) ]]; then
          exit 1
        fi

```

1

Pipeline의 고유한 이름입니다.

2

git 리포지토리가 복제되는 공유 작업 공간입니다.

3

애플리케이션 리포지토리를 공유 작업 영역에 복제하는 작업입니다.

4

공유 작업 영역을 정리하는 작업입니다.

5

TaskRun에서 실행할 작업에 대한 참조입니다.

6

파이프라인의 태스크가 런타임에 입력을 수신하거나 출력을 제공하는 데 필요한 공유 스토리지 볼륨입니다.

7

작업에 필요한 매개 변수 목록입니다. 매개 변수에 암시적 기본값이 없는 경우 해당 값을 명시적으로 설정해야 합니다.

8

4.2.2.4. TaskRun

TaskRun은 클러스터에서 특정 입력, 출력 및 실행 매개변수를 사용하여 실행할 **Task**를 인스턴스화합니다. 자체 또는 파이프라인의 각 작업에 대해 파이프라인 실행의 일부로 호출할 수 있습니다.

Task는 컨테이너 이미지를 실행하는 하나 이상의 단계(**Step**)로 구성되며, 각 컨테이너 이미지의 특정 빌드 작업을 수행합니다. **TaskRun**은 **Task**의 모든 단계(**Step**)를 지정된 순서로 실행하며, 모든 단계 (**Step**)가 성공적으로 실행되거나 실패하는 단계가 발생하면 실행을 멈춥니다. **TaskRun**은 **Pipeline**의 각 **Task**에 대한 **PipelineRun**에 의해 자동으로 생성되며,

다음 예는 관련 입력 매개변수를 사용하여 **apply-manifests Task**를 실행하는 **TaskRun**을 보여줍니다.

```
apiVersion: tekton.dev/v1beta1 1
kind: TaskRun 2
metadata:
  name: apply-manifests-taskrun 3
spec: 4
  serviceAccountName: pipeline
  taskRef: 5
    kind: Task
    name: apply-manifests
  workspaces: 6
  - name: source
  persistentVolumeClaim:
    claimName: source-pvc
```

1

TaskRun API 버전은 **v1beta1**입니다.

2

Kubernetes 개체의 유형을 지정합니다. 예에서는 **TaskRun**입니다.

3

이 **TaskRun**을 식별하는 고유한 이름입니다.

4

TaskRun의 정의입니다. 이 **TaskRun**에 대한 **Task** 및 필수 작업 **Workspace**가 지정됩니다.

5

이 **TaskRun**에 사용되는 **Task** 참조의 이름입니다. 이 **TaskRun**은 **apply-manifests Task**를 실행합니다.

6

TaskRun에서 사용하는 **Workspace**입니다.

4.2.2.5. 파이프라인

파이프라인은 특정 실행 순서대로 정렬된 **Task** 리소스 컬렉션입니다. 애플리케이션 빌드, 배포 및 제공 작업을 자동화하는 복잡한 워크플로를 구성하기 위해 실행됩니다. 하나 이상의 작업이 포함된 파이프라인을 사용하여 애플리케이션에 대한 **CI/CD** 워크플로를 정의할 수 있습니다.

Pipeline 리소스 정의는 함께 사용하면 파이프라인에서 특정 목표를 달성할 수 있는 여러 필드 또는 특성으로 구성됩니다. 각 **Pipeline** 리소스 정의에는 특정 입력을 수집하고 특정 출력을 생성하는 **Task**가 하나 이상 포함되어야 합니다. 또한 파이프라인 정의에는 애플리케이션 요구 사항에 따라 **Conditions**, **Workspaces**, **Parameters** 또는 **Resources**가 선택적으로 포함될 수 있습니다.

다음 예제에서는 **buildah ClusterTask** 리소스를 사용하여 **Git** 리포지토리에서 애플리케이션 이미지를 빌드하는 **build-and-deploy** 파이프라인을 보여줍니다.

```

apiVersion: tekton.dev/v1beta1 1
kind: Pipeline 2
metadata:
  name: build-and-deploy 3
spec: 4
  workspaces: 5
  - name: shared-workspace
  params: 6
  - name: deployment-name
    type: string
    description: name of the deployment to be patched
  - name: git-url
    type: string
    description: url of the git repo for the code of deployment
  - name: git-revision
    type: string
    description: revision to be used from repo of the code for deployment
    default: "pipelines-1.5"
  - name: IMAGE

```

```
type: string
description: image to be built from the code
tasks: 7
- name: fetch-repository
  taskRef:
    name: git-clone
    kind: ClusterTask
  workspaces:
    - name: output
      workspace: shared-workspace
  params:
    - name: url
      value: $(params.git-url)
    - name: subdirectory
      value: ""
    - name: deleteExisting
      value: "true"
    - name: revision
      value: $(params.git-revision)
- name: build-image 8
  taskRef:
    name: buildah
    kind: ClusterTask
  params:
    - name: TLSVERIFY
      value: "false"
    - name: IMAGE
      value: $(params.IMAGE)
  workspaces:
    - name: source
      workspace: shared-workspace
  runAfter:
    - fetch-repository
- name: apply-manifests 9
  taskRef:
    name: apply-manifests
  workspaces:
    - name: source
      workspace: shared-workspace
  runAfter: 10
    - build-image
- name: update-deployment
  taskRef:
    name: update-deployment
  workspaces:
    - name: source
      workspace: shared-workspace
  params:
    - name: deployment
      value: $(params.deployment-name)
    - name: IMAGE
      value: $(params.IMAGE)
  runAfter:
    - apply-manifests
```

1

Pipeline API 버전은 v1beta1입니다.

2

Kubernetes 개체의 유형을 지정합니다. 예에서는 Pipeline입니다.

3

이 Pipeline의 고유한 이름입니다.

4

Pipeline의 정의와 구조를 지정합니다.

5

Pipeline의 모든 Task에 사용되는 Workspace입니다.

6

Pipeline의 모든 Task에 사용되는 매개변수입니다.

7

Pipeline에서 사용되는 Task 목록을 지정합니다.

8

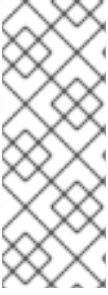
buildah ClusterTask를 사용하여 주어진 Git 리포지토리에서 애플리케이션 이미지를 빌드하는 Task build-image입니다.

9

동일한 이름의 사용자 지정 Task를 사용하는 Task apply-manifests입니다.

10

Pipeline에서 Task가 실행되는 순서를 지정합니다. 예에서는 apply-manifests Task는 build-image Task가 완료된 후에만 실행됩니다.



참고

Red Hat OpenShift Pipelines Operator는 **Buildah** 클러스터 작업을 설치하고 이미지를 빌드하고 푸시할 수 있는 충분한 권한이 있는 파이프라인 서비스 계정을 생성합니다. 권한이 충분하지 않은 다른 서비스 계정과 연결된 경우 **Buildah** 클러스터 작업이 실패할 수 있습니다.

4.2.2.6. PipelineRun

PipelineRun 은 **CI/CD** 워크플로를 실행하는 시나리오에 고유한 파이프라인, 작업 공간, 인증 정보 및 일련의 매개변수 값을 바인딩하는 리소스 유형입니다.

파이프라인 실행 은 파이프라인의 실행 중인 인스턴스입니다. 클러스터에서 특정 입력, 출력 및 실행 매개변수를 사용하여 실행할 파이프라인을 인스턴스화합니다. 또한 파이프라인 실행의 각 작업에 대한 작업 실행을 생성합니다.

파이프라인은 완료되거나 작업이 실패할 때까지 작업을 순차적으로 실행합니다. **status** 필드는 각 작업 실행의 진행 상황 및 모니터링 및 감사 목적으로 저장합니다.

다음 예는 관련 리소스 및 매개변수를 사용하여 **build-and-deploy Pipeline**을 실행하는 **PipelineRun**을 보여줍니다.

```

apiVersion: tekton.dev/v1beta1 1
kind: PipelineRun 2
metadata:
  name: build-deploy-api-pipelinerun 3
spec:
  pipelineRef:
    name: build-and-deploy 4
  params: 5
  - name: deployment-name
    value: vote-api
  - name: git-url
    value: https://github.com/openshift-pipelines/vote-api.git
  - name: IMAGE
    value: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/vote-api
  workspaces: 6
  - name: shared-workspace
    volumeClaimTemplate:
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 500Mi

```


1

파이프라인 실행 API 버전은 **v1beta1** 입니다.

2

Kubernetes 오브젝트의 유형입니다. 예에서는 **PipelineRun**입니다.

3

이 파이프라인 실행을 식별하는 고유한 이름입니다.

4

실행할 파이프라인의 이름입니다. 예에서는 **build-and-deploy**입니다.

5

파이프라인을 실행하는 데 필요한 매개변수 목록입니다.

6

파이프라인 실행에 사용되는 작업 공간입니다.

추가 리소스

•

[Git 보안을 사용하여 파이프라인 인증](#)

4.2.2.7. Workspace



참고

PipelineResources는 디버그하기 어렵고 범위가 제한되며 **Task**의 재사용 가능성을 낮추기 때문에 **OpenShift Pipelines**에서는 **PipelineResources** 대신 **Workspace**를 사용할 것을 권장합니다.

작업 공간은 입력 또는 출력을 제공하기 위해 런타임 시 파이프라인의 작업에 필요한 공유 스토리지 볼륨을 선언합니다. 볼륨의 실제 위치를 지정하는 대신 **Workspace**를 사용하여 런타임 시 필요한 파일 시스템 전체 또는 파일 시스템의 일부를 선언할 수 있습니다. 작업 또는 파이프라인은 작업 공간을 선언하고 볼륨의 특정 위치 세부 정보를 제공해야 합니다. 그런 다음 작업 실행 또는 파이프라인 실행에서 해당 작업 공간에 마운트됩니다. 이러한 방식으로 런타임 스토리지 볼륨에서 볼륨 선언을 분리하면 사용자 환경에 종속되지 않으며 유연성 높고 재사용 가능한 **Task**로 만들 수 있습니다.

다음과 같은 용도로 **Workspace**를 활용할 수 있습니다.

- **Task** 입력 및 출력 저장
- **Task** 간 데이터 공유
- 시크릿에 보관된 자격 증명의 마운트 지점으로 작업 공간 활용
- **ConfigMaps**에 보관된 구성의 마운트 지점으로 작업 공간 활용
- 조직에서 공유하는 공통 도구의 마운트 지점으로 작업 공간 활용
- 작업 속도를 높이는 빌드 아티팩트 캐시 생성

다음은 사용하여 **TaskRun** 또는 **PipelineRun**에서 **Workspace**를 지정할 수 있습니다.

- 읽기 전용 **ConfigMaps** 또는 **Secret**
- 다른 **Task**와 공유되는 기존 **PersistentVolumeClaim**
- 제공된 **VolumeClaimTemplate**의 **PersistentVolumeClaim**
- **TaskRun**이 완료되면 삭제되는 **emptyDir**

다음은 **Pipeline**에 정의된 대로 **build-image** 및 **apply-manifests** **Task**에 대한 **shared-workspace** **Workspace**를 선언하는 **build-and-deploy** **Pipeline**의 코드 스니펫 예입니다.

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: build-and-deploy
```

```

spec:
  workspaces: ❶
  - name: shared-workspace
  params:
  ...
  tasks: ❷
  - name: build-image
    taskRef:
      name: buildah
      kind: ClusterTask
    params:
      - name: TLSVERIFY
        value: "false"
      - name: IMAGE
        value: $(params.IMAGE)
    workspaces: ❸
      - name: source ❹
        workspace: shared-workspace ❺
    runAfter:
      - fetch-repository
      - name: apply-manifests
    taskRef:
      name: apply-manifests
    workspaces: ❻
      - name: source
        workspace: shared-workspace
    runAfter:
      - build-image
  ...

```

❶

Pipeline에 정의된 Task 사이에 공유되는 Workspace 목록입니다. Pipeline은 필요한 만큼 Workspace를 정의할 수 있습니다. 예에서는 **shared-workspace**라는 Workspace 한 개만 선언됩니다.

❷

Pipeline에서 사용되는 Task의 정의입니다. 이 스니펫은 공통 Workspace를 공유하는 두 개의 Task, **build-image**와 **apply-manifest**를 정의합니다.

❸

build-image Task에 사용되는 Workspace 목록입니다. Task 정의에 필요한 만큼의 Workspace를 포함할 수 있습니다. 하지만 Task에 사용되는 쓰기 가능한 Workspace를 한 개로 제한하는 것이 좋습니다.

❹

Task에서 사용되는 Workspace를 고유하게 식별하는 이름입니다. 이 Task는 **source**라는 Workspace 한 개를 사용합니다.

5

Task에서 사용하는 **Pipeline Workspace**의 이름입니다. 이어서 **source Workspace**는 **shared-workspace**라는 **Pipeline Workspace**를 사용한다는 점에 주목하십시오.

6

apply-manifests Task에 사용되는 **Workspace** 목록입니다. 이 **Task**는 **build-image Task**와 **source Workspace**를 공유한다는 점에 주목하십시오.

작업 공간을 사용하면 여러 작업에서 데이터를 공유하고 파이프라인의 각 작업을 실행하는 동안 필요한 하나 이상의 볼륨을 지정할 수 있습니다. 영구 볼륨 클레임을 생성하거나 사용자를 대신하여 영구 볼륨 클레임을 생성하는 볼륨 클레임 템플릿을 제공할 수 있습니다.

다음의 **build-deploy-api-pipelinerun PipelineRun** 코드 조각에서는 볼륨 클레임 템플릿을 사용하여 **build-and-deploy** 파이프라인에 사용된 **shared-workspace** 작업 공간의 스토리지 볼륨을 정의하는 영구 볼륨 클레임을 생성합니다.

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: build-deploy-api-pipelinerun
spec:
  pipelineRef:
    name: build-and-deploy
  params:
  ...

  workspaces: 1
  - name: shared-workspace 2
    volumeClaimTemplate: 3
      spec:
        accessModes:
          - ReadWriteOnce
        resources:
          requests:
            storage: 500Mi
```

1

PipelineRun에서 볼륨 바인딩이 제공될 **Pipeline Workspace** 목록을 지정합니다.

2

볼륨이 제공될 **Pipeline**의 **Workspace** 이름입니다.

3

4.2.2.8. Trigger

Kubernetes 리소스에서 전체 **CI/CD** 실행을 정의하는 완전한 **CI/CD** 시스템을 생성하려면 파이프라인과 함께 **트리거**를 사용합니다. 트리거는 **Git** 풀 요청과 같은 외부 이벤트를 캡처하고 처리하여 주요 정보를 추출합니다. 이 이벤트 데이터를 미리 정의된 매개변수 집합에 매핑하면 **Kubernetes** 리소스를 생성 및 배포하고 파이프라인을 인스턴스화할 수 있는 일련의 작업이 트리거됩니다.

애플리케이션에 **Red Hat OpenShift Pipeline**을 사용하여 **CI/CD** 워크플로를 정의하는 경우를 예로 들 수 있습니다. 새로운 변경 사항을 애플리케이션 리포지토리에 적용하려면 파이프라인을 시작해야 합니다. 트리거는 모든 변경 이벤트를 캡처하여 처리하고 최신 변경 사항이 적용된 새 이미지를 배포하는 파이프라인 실행을 트리거하는 방식으로 이 프로세스를 자동화합니다.

트리거는 함께 작동하여 재사용 가능하고 분리되고 자체 유지되는 **CI/CD** 시스템을 형성하는 다음과 같은 주요 리소스로 구성됩니다.

- **TriggerBinding** 리소스는 이벤트 페이로드에서 필드를 추출한 다음 해당 필드를 매개변수로 저장합니다.

다음은 수신된 이벤트 페이로드에서 **Git** 리포지토리 정보를 추출하는 **TriggerBinding** 리소스의 코드 조각 예입니다.

```
apiVersion: triggers.tekton.dev/v1alpha1 ①
kind: TriggerBinding ②
metadata:
  name: vote-app ③
spec:
  params: ④
  - name: git-repo-url
    value: $(body.repository.url)
  - name: git-repo-name
    value: $(body.repository.name)
  - name: git-revision
    value: $(body.head_commit.id)
```

①

TriggerBinding 리소스의 **API** 버전입니다. 이 예제에서는 **v1alpha1**입니다.

②

Kubernetes 개체의 유형을 지정합니다. 예에서는 **TriggerBinding**입니다.

3

TriggerBinding 리소스를 확인하는 고유한 이름입니다.

4

수신한 이벤트 페이로드에서 추출하여 **TriggerTemplate** 리소스로 전달할 매개변수 목록입니다. 예에서는 **Git** 리포지토리 **URL**, 이름 및 개정 정보가 이벤트 페이로드의 본문에서 추출됩니다.

•

TriggerTemplate 리소스는 리소스를 생성해야 하는 방법에 대해 표준 역할을 합니다. **TriggerBinding** 리소스에서 매개변수화된 데이터를 사용하는 방식을 지정합니다. 트리거 템플릿은 트리거 바인딩을 통해 입력을 수신한 다음 새 파이프라인 리소스를 생성하고 새 파이프라인 실행을 시작하는 일련의 작업을 수행합니다.

다음은 방금 생성한 **TriggerBinding** 리소스에서 수신한 **Git** 리포지토리 정보를 사용하여 애플리케이션을 생성하는 **TriggerTemplate** 리소스의 코드 조각입니다.

```

apiVersion: triggers.tekton.dev/v1alpha1 1
kind: TriggerTemplate 2
metadata:
  name: vote-app 3
spec:
  params: 4
  - name: git-repo-url
    description: The git repository url
  - name: git-revision
    description: The git revision
    default: pipelines-1.5
  - name: git-repo-name
    description: The name of the deployment to be created / patched

  resourcetemplates: 5
  - apiVersion: tekton.dev/v1beta1
    kind: PipelineRun
    metadata:
      name: build-deploy-${tt.params.git-repo-name}-${uid}
    spec:
      serviceAccountName: pipeline
      pipelineRef:
        name: build-and-deploy
      params:
      - name: deployment-name
        value: ${tt.params.git-repo-name}
      - name: git-url
        value: ${tt.params.git-repo-url}
      - name: git-revision
        value: ${tt.params.git-revision}
      - name: IMAGE

```

```

value: image-registry.openshift-image-registry.svc:5000/pipelines-
tutorial/$(tt.params.git-repo-name)
workspaces:
- name: shared-workspace
volumeClaimTemplate:
spec:
accessModes:
- ReadWriteOnce
resources:
requests:
storage: 500Mi

```

1

TriggerTemplate 리소스의 API 버전입니다. 이 예제에서는 v1alpha1입니다.

2

Kubernetes 개체의 유형을 지정합니다. 예에서는 **TriggerTemplate**입니다.

3

TriggerTemplate 리소스를 식별하는 고유 이름입니다.

4

TriggerBinding 리소스에서 제공되는 매개변수입니다.

5

TriggerBinding 또는 **EventListener** 리소스를 통해 수신한 매개변수를 사용하여 리소스를 생성해야 하는 방법을 지정하는 템플릿 목록입니다.

•

Trigger 리소스는 **TriggerBinding** 및 **TriggerTemplate** 리소스를 결합하고 선택적으로 **interceptors** 이벤트 프로세서를 결합합니다.

인터셉터는 **TriggerBinding** 리소스 전에 실행되는 특정 플랫폼의 모든 이벤트를 처리합니다. 인터셉터를 사용하여 페이로드를 필터링하고, 이벤트를 확인하고, 트리거 조건을 정의 및 테스트하고, 기타 유용한 처리를 구현할 수 있습니다. 인터셉터는 이벤트 확인을 위해 시크릿을 사용합니다. 이벤트 데이터가 인터셉터를 통과한 후 페이로드 데이터를 트리거 바인딩에 전달하기 전에 트리거로 이동합니다. 인터셉터를 사용하여 **EventListener** 사양에서 참조된 관련 트리거의 동작을 수정할 수도 있습니다.

다음 예제는 **TriggerBinding** 및 **TriggerTemplate** 리소스와 **interceptors** 이벤트 프로세서를 연결하는 **vote-trigger**라는 **Trigger** 리소스의 코드 조각을 보여줍니다.

-

```

apiVersion: triggers.tekton.dev/v1alpha1 ❶
kind: Trigger ❷
metadata:
  name: vote-trigger ❸
spec:
  serviceAccountName: pipeline ❹
  interceptors:
    - ref:
      name: "github" ❺
      params: ❻
        - name: "secretRef"
          value:
            secretName: github-secret
            secretKey: secretToken
        - name: "eventTypes"
          value: ["push"]
  bindings:
    - ref: vote-app ❼
  template: ❽
    ref: vote-app
---
apiVersion: v1
kind: Secret ❾
metadata:
  name: github-secret
type: Opaque
stringData:
  secretToken: "1234567"

```

❶

Trigger 리소스의 API 버전입니다. 이 예제에서는 **v1alpha1**입니다.

❷

Kubernetes 개체의 유형을 지정합니다. 이 예제에서는 **Trigger**입니다.

❸

Trigger 리소스를 식별하는 고유 이름입니다.

❹

사용할 서비스 계정 이름입니다.

❺

참조할 인터셉터 이름입니다. 예에서는 **github**입니다.

❻

7

TriggerTemplate 리소스에 연결할 TriggerBinding 리소스의 이름입니다.

8

TriggerBinding 리소스에 연결할 TriggerTemplate 리소스의 이름입니다.

9

이벤트를 확인하는 데 사용할 시크릿입니다.

•

EventListener 리소스는 JSON 페이로드와 함께 들어오는 HTTP 기반 이벤트를 수신 대기하는 끝점 또는 이벤트 싱크를 제공합니다. 각 TriggerBinding 리소스에서 이벤트 매개변수를 추출한 다음 이 데이터를 처리하여 해당 TriggerTemplate 리소스에서 지정하는 Kubernetes 리소스를 생성합니다. 또한 EventListener 리소스는 페이로드 유형을 확인하고 선택적으로 수정하는 이벤트 interceptors를 사용하여 페이로드에 대한 간단한 이벤트 처리 또는 기본 필터링 작업을 수행합니다. 현재 파이프라인 트리거는 5가지 유형의 인터셉터를 지원합니다. *Webhook 인터셉터*, *GitHub 인터셉터*, *GitLab 인터셉터*, *Bitbucket 인터셉터* 및 *CEL (Common Expression Language) 인터셉터*.

다음 예제에서는 vote-trigger라는 Trigger 리소스를 참조하는 EventListener 리소스를 보여줍니다.

```
apiVersion: triggers.tekton.dev/v1alpha1 1
kind: EventListener 2
metadata:
  name: vote-app 3
spec:
  serviceAccountName: pipeline 4
  triggers:
    - triggerRef: vote-trigger 5
```

1

EventListener 리소스의 API 버전입니다. 이 예제에서는 v1alpha1입니다.

2

Kubernetes 개체의 유형을 지정합니다. 예에서는 EventListener입니다.

3

EventListener 리소스를 식별하는 고유 이름입니다.

4

사용할 서비스 계정 이름입니다.

5

EventListener 리소스에서 참조하는 **Trigger** 리소스의 이름입니다.

4.2.3. 추가 리소스

- 파이프라인 설치에 대한 내용은 [OpenShift Pipelines 설치](#)를 참조하십시오.
- 사용자 정의 **CI/CD** 솔루션 생성에 대한 자세한 내용은 [CI/CD 파이프라인을 사용하여 애플리케이션 생성](#)을 참조하십시오.
- 재암호화 **TLS** 종료에 대한 자세한 내용은 [재암호화 종료](#)를 참조하십시오.
- 보안 경로에 대한 자세한 내용은 [보안 경로](#) 섹션을 참조하십시오.

4.3. OPENSIFT PIPELINES 설치

이 가이드에서는 클러스터 관리자에게 **Red Hat OpenShift Pipelines Operator**를 **OpenShift Container Platform** 클러스터에 설치하는 프로세스를 안내합니다.

사전 요구 사항

- **cluster-admin** 권한이 있는 계정을 사용하여 **OpenShift Container Platform** 클러스터에 액세스할 수 있습니다.
- **oc CLI**를 설치했습니다.
- 로컬 시스템에 **OpenShift Pipelines (tkn) CLI**를 설치했습니다.

4.3.1. 웹 콘솔에서 Red Hat OpenShift Pipelines Operator 설치

OpenShift Container Platform OperatorHub에 나열된 Operator를 사용하여 Red Hat OpenShift Pipelines를 설치할 수 있습니다. Red Hat OpenShift Pipelines Operator를 설치하면 파이프라인 구성에 필요한 CR(사용자 정의 리소스)이 Operator와 함께 자동으로 설치됩니다.

기본 Operator CRD(사용자 정의 리소스 정의) `config.operator.tekton.dev`가 `tektonconfigs.operator.tekton.dev`로 교체되었습니다. 또한 Operator에서 OpenShift Pipelines 구성 요소를 개별적으로 관리하기 위해 추가 CRD인 `tektonpipelines.operator.tekton.dev`, `tektontriggers.operator.tekton.dev`, `tektonaddons.operator.tekton.dev`를 제공합니다.

OpenShift Pipelines가 클러스터에 이미 설치되어 있는 경우 기존 설치가 원활하게 업그레이드됩니다. Operator는 필요에 따라 클러스터의 `config.operator.tekton.dev` 인스턴스를 `tektonconfigs.operator.tekton.dev` 인스턴스 및 기타 CRD의 추가 오브젝트로 교체합니다.



주의

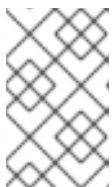
`resource name - cluster` 필드를 변경하여 `config.operator.tekton.dev` CRD 인스턴스의 타겟 네임스페이스를 변경하는 등 기존 설치를 수동으로 변경한 경우 업그레이드 경로가 제대로 작동하지 않습니다. 이러한 경우 권장되는 워크플로는 설치를 제거한 후 Red Hat OpenShift Pipelines Operator를 다시 설치하는 것입니다.

Red Hat OpenShift Pipelines Operator에서는 이제 TektonConfig CR의 일부로 프로필을 지정하여 설치할 구성 요소를 선택할 수 있는 옵션을 제공합니다. Operator가 설치되면 TektonConfig CR이 자동으로 설치됩니다. 지원되는 프로필은 다음과 같습니다.

- **Lite:** 그러면 Tekton 파이프라인만 설치됩니다.
- **기본:** 그러면 Tekton Pipelines 및 Tekton Trigger가 설치됩니다.
- **모두:** TektonConfig CR을 설치할 때 사용되는 기본 프로필입니다. 이 프로필은 모든 Tekton 구성 요소를 설치합니다. Tekton Pipelines, Tekton Triggers, Tekton Addons(ClusterTasks, ClusterTriggerBindings, ConsoleCLIDownload, ConsoleQuickStart 및 ConsoleYAMLSample 리소스 포함).

절차

1. 웹 콘솔의 관리자 화면에서 **Operator** → **OperatorHub**로 이동합니다.
2. 키워드로 필터링 박스를 사용하여 카탈로그에서 **Red Hat OpenShift Pipelines Operator**를 검색합니다. **Red Hat OpenShift Pipelines Operator** 타일을 클릭합니다.
3. **Red Hat OpenShift Pipelines Operator** 페이지에서 **Operator**에 대한 간략한 설명을 확인합니다. 설치를 클릭합니다.
4. **Operator** 설치 페이지에서 다음을 수행합니다.
 - a. **Installation Mode**로 **All namespaces on the cluster(default)**를 선택합니다. 이 모드에서는 기본 **openshift-operators** 네임스페이스에 **Operator**가 설치되므로 **Operator**가 클러스터의 모든 네임스페이스를 감시하고 사용 가능하게 만들 수 있습니다.
 - b. **Approval Strategy**으로 **Automatic**을 선택합니다. 그러면 **Operator**에 향후 지원되는 업그레이드가 **OLM(Operator Lifecycle Manager)**에 의해 자동으로 처리됩니다. **Manual** 승인 전략을 선택하면 **OLM**에서 업데이트 요청을 생성합니다. 클러스터 관리자는 **Operator**를 새 버전으로 업데이트하려면 **OLM** 업데이트 요청을 수동으로 승인해야 합니다.
 - c. **Update Channel**을 선택합니다.
 - **stable** 채널을 사용하면 **Red Hat OpenShift Pipelines Operator**의 안정적인 최신 릴리스를 설치할 수 있습니다.
 - **preview** 채널을 사용하면 **Red Hat OpenShift Pipelines Operator**의 최신 프리뷰 버전을 설치할 수 있습니다. 이 버전에는 **stable** 채널에서 아직 지원되지 않는 기능이 포함될 수 있습니다.
5. 설치를 클릭합니다. **Installed Operators** 페이지의 목록에 해당 **Operator**가 나타납니다.



참고

Operator는 **openshift-operators** 네임스페이스에 자동으로 설치됩니다.

6.

Red Hat OpenShift Pipelines Operator가 성공적으로 설치되었는지 확인하려면 상태가 최신 업데이트 완료로 설정되어 있는지 확인합니다.

4.3.2. CLI를 사용하여 OpenShift Pipelines Operator 설치

CLI를 사용하여 OperatorHub에서 Red Hat OpenShift Pipelines Operator를 설치할 수 있습니다.

프로세스

1.

서브스크립션 오브젝트 **YAML** 파일을 생성하여 Red Hat OpenShift Pipelines Operator에 네임스페이스를 서브스크립션합니다(예: **sub.yaml**).

Subscription의 예

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-pipelines-operator
  namespace: openshift-operators
spec:
  channel: <channel name> ①
  name: openshift-pipelines-operator-rh ②
  source: redhat-operators ③
  sourceNamespace: openshift-marketplace ④
```

①

Operator를 서브스크립션할 채널 이름을 지정합니다.

②

서브스크립션할 Operator의 이름입니다.

③

Operator를 제공하는 CatalogSource의 이름입니다.

④

2.

서브스크립션 오브젝트를 생성합니다.

```
$ oc apply -f sub.yaml
```

이제 **Red Hat OpenShift Pipelines Operator**가 기본 타겟 네임스페이스인 **openshift-operators**에 설치되었습니다.

4.3.3. 제한된 환경의 Red Hat OpenShift Pipelines Operator

Red Hat OpenShift Pipelines Operator는 제한된 네트워크 환경에서 파이프라인 설치를 지원합니다.

Operator는 **cluster** 프록시 오브젝트를 기반으로 **tekton-controller**에서 생성한 **Pod**의 컨테이너에 프록시 환경 변수를 설정하는 프록시 **Webhook**를 설치합니다. 또한 **TektonPipelines, TektonTriggers, Controllers, Webhooks, Operator Proxy Webhook** 리소스에서 프록시 환경 변수를 설정합니다.

기본적으로 프록시 **Webhook**는 **openshift-pipelines** 네임스페이스에 대해 비활성화되어 있습니다. 다른 네임스페이스에 대해 비활성화하려면 **namespace** 오브젝트에 **operator.tekton.dev/disable-proxy: true** 라벨을 추가하면 됩니다.

4.3.4. 추가 리소스

- **OpenShift Container Platform**에 **Operator**를 설치하는 방법은 [클러스터에 Operator 추가](#) 섹션에서 확인할 수 있습니다.
- 제한된 환경에서 파이프라인을 사용하는 방법에 대한 자세한 내용은 다음을 참조하십시오.
 - [제한된 환경에서 파이프라인을 실행하도록 이미지 미러링](#)
 - [제한된 클러스터에 대한 Samples Operator 구성](#)
 - [미러링된 레지스트리로 클러스터 생성](#)

4.4. OPENSIFT PIPELINES 설치 제거

Red Hat OpenShift Pipelines Operator 설치 제거는 2단계로 구성된 프로세스입니다.

1. **Red Hat OpenShift Pipelines Operator**를 설치할 때 기본적으로 추가된 **CR(Custom Resource)**을 삭제합니다.
2. **Red Hat OpenShift Pipelines Operator**를 설치 제거합니다.

Operator를 설치 제거하는 것만으로 설치 과정에서 기본적으로 생성된 **Red Hat OpenShift Pipelines** 구성 요소가 제거되지는 않습니다.

4.4.1. Red Hat OpenShift Pipelines 구성 요소 및 사용자 정의 리소스 삭제

Red Hat OpenShift Pipelines Operator 설치 과정에서 기본적으로 생성된 **CR(사용자 정의 리소스)**을 삭제합니다.

프로세스

1. 웹 콘솔의 관리자 화면에서 **Administration** → **Custom Resource Definition**로 이동합니다.
2. 이름으로 필터링 박스에 **config.operator.tekton.dev**를 입력하여 **Red Hat OpenShift Pipelines Operator CR**을 검색합니다.
3. **CRD Config**을 클릭하여 **Custom Resource Definition Details** 페이지를 엽니다.
4. **Actions** 드롭다운 메뉴를 클릭하고 **Delete Custom Resource Definition**를 선택합니다.



참고

CR을 삭제하면 **Red Hat OpenShift Pipelines** 구성 요소가 삭제되고 클러스터의 모든 작업과 파이프라인이 사라집니다.

5. **Delete**를 클릭하여 **CR** 삭제를 확인합니다.

4.4.2. Red Hat OpenShift Pipelines Operator 설치 제거

프로세스

1. **Operators** → **OperatorHub** 페이지에서 키워드로 필터링 박스를 사용하여 **Red Hat OpenShift Pipelines Operator**를 검색합니다.
2. **OpenShift Pipelines Operator** 타일을 클릭합니다. **Operator** 타일은 **Operator**가 설치되었음을 나타냅니다.
3. **OpenShift Pipelines Operator** 설명자 페이지에서 **Uninstall**를 클릭합니다.

추가 리소스

- **OpenShift Container Platform**에서 **Operator**를 설치 제거하는 방법에 대한 자세한 내용은 [클러스터에서 Operator 삭제](#) 섹션에서 확인할 수 있습니다.

4.5. OPENSIFT PIPELINES를 사용하여 애플리케이션용 CI/CD 솔루션 작성

Red Hat OpenShift Pipelines를 사용하면 애플리케이션을 빌드, 테스트, 배포하는 사용자 정의 **CI/CD** 솔루션을 생성할 수 있습니다.

애플리케이션에 사용할 완전한 셀프 서비스 **CI/CD** 파이프라인을 생성하려면 다음 작업을 수행합니다.

- 사용자 정의 작업을 생성하거나 재사용 가능한 기존 작업을 설치합니다.
- 애플리케이션용 제공 파이프라인을 생성하고 정의합니다.
- 다음 접근 방법 중 하나를 사용하여 파이프라인 실행을 위해 작업 공간에 연결된 스토리지 볼륨 또는 파일 시스템을 제공합니다.

- 영구 볼륨 클레임을 생성하는 볼륨 클레임 템플릿 지정
- 영구 볼륨 클레임 지정
- 파이프라인을 인스턴스화하고 호출할 **PipelineRun** 오브젝트를 생성합니다.
- 소스 리포지토리의 이벤트를 캡처하는 트리거를 추가합니다.

여기서는 **pipelines-tutorial** 예제를 사용하여 선행 **Task**들을 보여줍니다. 예에서는 다음으로 구성된 간단한 애플리케이션을 사용합니다.

- **pipelines-vote-ui** Git 리포지토리에 소스 코드가 있는 프런트 엔드 인터페이스 **pipelines-vote-ui**
- **pipelines-vote-api** Git 리포지토리에 소스 코드가 있는 백엔드 인터페이스 **pipelines-vote-api**
- **pipelines-tutorial** Git 리포지토리의 **apply-manifests** 및 **update-deployment** 작업

4.5.1. 사전 요구 사항

- **OpenShift Container Platform** 클러스터에 액세스 권한을 보유하고 있습니다.
- **OpenShift OperatorHub**에 나열된 **Red Hat OpenShift Pipelines Operator**를 사용하여 **OpenShift Pipelines**를 설치했습니다. 설치를 마친 후 전체 클러스터에 적용할 수 있습니다.
- **OpenShift Pipelines CLI**를 설치했습니다.
- **GitHub ID**를 사용하여 프런트 엔드 **pipelines-vote-u** 및 백엔드 **pipelines-vote-api** Git 리포지토리를 분기했으며, 이러한 리포지토리에 관리자 액세스 권한이 있습니다.

- 선택 사항: [pipelines-tutorial](#) Git 리포지토리를 복제했습니다.

4.5.2. 프로젝트 생성 및 파이프라인 서비스 계정 검사

프로세스

1. **OpenShift Container Platform** 클러스터에 로그인합니다.

```
$ oc login -u <login> -p <password> https://openshift.example.com:6443
```

2. 샘플 애플리케이션용 프로젝트를 생성합니다. 예시 워크플로에서는 **pipelines-tutorial** 프로젝트를 생성합니다.

```
$ oc new-project pipelines-tutorial
```



참고

다른 이름으로 프로젝트를 생성하는 경우, 예시에 사용된 리소스 URL을 사용자의 프로젝트 이름으로 업데이트하십시오.

3. **pipeline** 서비스 계정을 표시합니다.

Red Hat OpenShift Pipelines Operator는 이미지를 빌드하고 내보내기에 충분한 권한이 있는 **pipeline**이라는 서비스 계정을 추가하고 구성합니다. 이 서비스 계정은 **PipelineRun** 오브젝트에서 사용합니다.

```
$ oc get serviceaccount pipeline
```

4.5.3. 파이프라인 작업 생성

프로세스

1. 파이프라인의 재사용 가능한 작업 목록이 포함된 **pipelines-tutorial** 리포지토리에서 **apply-manifests** 및 **update-deployment** 작업을 설치합니다.

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.5/01_pipeline/01_apply_manifest_task.yaml
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.5/01_pipeline/02_update_deployment_task.yaml
```

2.

`tkn task list` 명령을 사용하여 생성한 작업 목록을 표시합니다.

```
$ tkn task list
```

`apply-manifest` 및 `update-deployment` 작업 리소스가 생성된 것이 출력에서 확인됩니다.

NAME	DESCRIPTION	AGE
<code>apply-manifests</code>		1 minute ago
<code>update-deployment</code>		48 seconds ago

3.

`tkn clustertasks list` 명령을 사용하여 **Operator**에서 설치한 추가 클러스터 작업 목록을 표시합니다(예: `buildah` 및 `s2i-python-3`).



참고

제한된 환경에서 `buildah` 클러스터 작업을 사용하려면 **Dockerfile**에서 내부 이미지 스트림을 기본 이미지로 사용해야 합니다.

```
$ tkn clustertasks list
```

Operator에서 설치한 **ClusterTask** 리소스가 출력에 나열됩니다.

NAME	DESCRIPTION	AGE
<code>buildah</code>		1 day ago
<code>git-clone</code>		1 day ago
<code>s2i-python</code>		1 day ago
<code>tkn</code>		1 day ago

4.5.4. 파이프라인 조립

파이프라인은 **CI/CD** 흐름을 나타내며 실행할 작업들로 정의됩니다. 여러 애플리케이션 및 환경에서 포괄적으로 적용되고 재사용 가능하도록 설계되었습니다.

파이프라인은 `from` 및 `runAfter` 매개변수를 사용하여 작업들이 상호 작용하는 방법과 실행 순서를 지정합니다. 그리고 `workspaces` 필드를 사용하여 파이프라인의 각 작업 실행 중 필요한 하나 이상의 볼륨을 지정합니다.

이 섹션에서는 **GitHub**에서 애플리케이션의 소스 코드를 가져와 **OpenShift Container Platform**에서 빌드 및 배포하는 파이프라인을 생성합니다.

파이프라인은 백엔드 애플리케이션 **pipelines-vote-api** 및 프론트 엔드 애플리케이션 **pipelines-vote-ui**에 대해 다음 작업을 수행합니다.

- **git-url** 및 **git-revision** 매개변수를 참조하여 **Git** 리포지토리에서 애플리케이션의 소스 코드를 복제합니다.
- **buildah** 클러스터 작업 사용하여 컨테이너 이미지를 빌드합니다.
- **image** 매개변수를 참조하여 내부 이미지 레지스트리로 이미지를 푸시합니다.
- **apply-manifests** 및 **update-deployment** 작업을 사용하여 **OpenShift Container Platform**에 새 이미지를 배포합니다.

프로세스

1. 다음 샘플 파이프라인 **YAML** 파일의 내용을 복사하여 저장합니다.

```

apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: build-and-deploy
spec:
  workspaces:
    - name: shared-workspace
  params:
    - name: deployment-name
      type: string
      description: name of the deployment to be patched
    - name: git-url
      type: string
      description: url of the git repo for the code of deployment
    - name: git-revision
      type: string
      description: revision to be used from repo of the code for deployment
      default: "pipelines-1.5"
    - name: IMAGE
      type: string
      description: image to be built from the code
  tasks:
    - name: fetch-repository

```

```

taskRef:
  name: git-clone
  kind: ClusterTask
workspaces:
- name: output
  workspace: shared-workspace
params:
- name: url
  value: $(params.git-url)
- name: subdirectory
  value: ""
- name: deleteExisting
  value: "true"
- name: revision
  value: $(params.git-revision)
- name: build-image
  taskRef:
    name: buildah
    kind: ClusterTask
  params:
    - name: IMAGE
      value: $(params.IMAGE)
  workspaces:
    - name: source
      workspace: shared-workspace
  runAfter:
    - fetch-repository
- name: apply-manifests
  taskRef:
    name: apply-manifests
  workspaces:
    - name: source
      workspace: shared-workspace
  runAfter:
    - build-image
- name: update-deployment
  taskRef:
    name: update-deployment
  params:
    - name: deployment
      value: $(params.deployment-name)
    - name: IMAGE
      value: $(params.IMAGE)
  runAfter:
    - apply-manifests

```

파이프라인 정의는 **Git** 소스 리포지토리 및 이미지 레지스트리의 세부 사항을 요약합니다. 이러한 세부 사항은 파이프라인이 트리거되고 실행될 때 **params**로 추가됩니다.

2.

파이프라인을 생성합니다.

```
$ oc create -f <pipeline-yaml-file-name.yaml>
```

또는 **Git** 리포지토리에서 직접 **YAML** 파일을 실행할 수도 있습니다.

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.5/01_pipeline/04_pipeline.yaml
```

3.

tkn pipeline list 명령을 사용하여 파이프라인이 애플리케이션에 추가되었는지 확인합니다.

```
$ tkn pipeline list
```

출력에서 **build-and-deploy** 파이프라인이 생성되었는지 확인합니다.

```
NAME          AGE          LAST RUN   STARTED   DURATION   STATUS
build-and-deploy 1 minute ago ---      ---      ---      ---
```

4.5.5. 제한된 환경에서 파이프라인을 실행하도록 이미지 미러링

연결이 끊긴 클러스터 또는 제한된 환경에서 프로비저닝된 클러스터에서 **OpenShift Pipelines**를 실행하려면 **Samples Operator**가 제한된 네트워크용으로 구성되었는지 또는 클러스터 관리자가 미러링된 레지스트리가 있는 클러스터를 생성했는지 확인해야 합니다.

다음 절차에서는 **pipelines-tutorial** 예제를 사용하여 미러링된 레지스트리가 있는 클러스터를 사용하여 제한된 환경에서 애플리케이션에 대한 파이프라인을 생성합니다. **pipelines-tutorial** 예제가 제한된 환경에서 작동하도록 하려면 프런트 엔드 인터페이스 **pipelines-vote-ui**, 백엔드 인터페이스 **pipelines-vote-api**, **cli**의 미러 레지스트리에서 해당 빌더 이미지를 미러링해야 합니다.

프로세스

1.

프런트 엔드 인터페이스 **pipelines-vote-ui**의 미러 레지스트리에서 빌더 이미지를 미러링합니다.

a.

필요한 이미지 태그를 가져오지 않았는지 확인합니다.

```
$ oc describe imagestream python -n openshift
```

출력 예

```
Name: python
```

```
Namespace: openshift
```

```
[...]
```

```
3.8-ubi8 (latest)
```

```
tagged from registry.redhat.io/ubi8/python-38:latest
prefer registry pullthrough when referencing this tag
```

```
Build and run Python 3.8 applications on UBI 8. For more information about using
this builder image, including OpenShift considerations, see
https://github.com/sclorg/s2i-python-container/blob/master/3.8/README.md.
```

```
Tags: builder, python
```

```
Supports: python:3.8, python
```

```
Example Repo: https://github.com/sclorg/django-ex.git
```

```
[...]
```

b.

지원되는 이미지 태그를 프라이빗 레지스트리로 미러링합니다.

```
$ oc image mirror registry.redhat.io/ubi8/python-38:latest <mirror-registry>:
<port>/ubi8/python-38
```

c.

이미지를 가져옵니다.

```
$ oc tag <mirror-registry>:<port>/ubi8/python-38 python:latest --scheduled -n
openshift
```

이미지는 정기적으로 다시 가져와야 합니다. **--scheduled** 플래그를 사용하면 자동으로 이미지를 다시 가져올 수 있습니다.

d.

지정된 태그가 있는 이미지를 가져왔는지 확인합니다.

```
$ oc describe imagestream python -n openshift
```

출력 예

```
Name: python
```

```
Namespace: openshift
```

```
[...]
```

```
latest
```

```
updates automatically from registry <mirror-registry>:<port>/ubi8/python-38
```

```
* <mirror-registry>:<port>/ubi8/python-38@sha256:3ee3c2e70251e75bfeac25c0c33356add9cc4abc9c51d858f39e4dc29c5f58
```

```
[...]
```

2.

백엔드 인터페이스 **pipelines-vote-api**의 미리 레지스트리에서 빌더 이미지를 미리링합니다.

a.

필요한 이미지 태그를 가져오지 않았는지 확인합니다.

```
$ oc describe imagestream golang -n openshift
```

출력 예

```
Name: golang
Namespace: openshift
[...]
```

```
1.14.7-ubi8 (latest)
tagged from registry.redhat.io/ubi8/go-toolset:1.14.7
prefer registry pullthrough when referencing this tag
```

```
Build and run Go applications on UBI 8. For more information about using this
builder image, including OpenShift considerations, see
https://github.com/sclorg/golang-container/blob/master/README.md.
```

```
Tags: builder, golang, go
Supports: golang
Example Repo: https://github.com/sclorg/golang-ex.git
```

```
[...]
```

b.

지원되는 이미지 태그를 프라이빗 레지스트리로 미리링합니다.

```
$ oc image mirror registry.redhat.io/ubi8/go-toolset:1.14.7 <mirror-registry>:
<port>/ubi8/go-toolset
```


- c. 이미지를 가져옵니다.

```
$ oc tag <mirror-registry>:<port>/ubi8/go-toolset golang:latest --scheduled -n openshift
```

이미지는 정기적으로 다시 가져와야 합니다. **--scheduled** 플래그를 사용하면 자동으로 이미지를 다시 가져올 수 있습니다.

- d. 지정된 태그가 있는 이미지를 가져왔는지 확인합니다.

```
$ oc describe imagestream golang -n openshift
```

출력 예

```
Name: golang
Namespace: openshift
[...]

latest
updates automatically from registry <mirror-registry>:<port>/ubi8/go-toolset

* <mirror-registry>:<port>/ubi8/go-toolset@sha256:59a74d581df3a2bd63ab55f7ac106677694bf612a1fe9e7e3e1487f55c421b37
[...]
```

3. cli의 미리 레지스트리에서 빌더 이미지를 미러링합니다.

- a. 필요한 이미지 태그를 가져오지 않았는지 확인합니다.

```
$ oc describe imagestream cli -n openshift
```

출력 예

```
Name:          cli
Namespace:     openshift
[...]
```

```
latest
```

```
updates automatically from registry quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143
551
```

```
* quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143
551
```

```
[...]
```

b.

지원되는 이미지 태그를 프라이빗 레지스트리로 미러링합니다.

```
$ oc image mirror quay.io/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143
551 <mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-dev:latest
```

c.

이미지를 가져옵니다.

```
$ oc tag <mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-dev cli:latest -
-scheduled -n openshift
```

이미지는 정기적으로 다시 가져와야 합니다. **--scheduled** 플래그를 사용하면 자동으로 이미지를 다시 가져올 수 있습니다.

d.

지정된 태그가 있는 이미지를 가져왔는지 확인합니다.

```
$ oc describe imagestream cli -n openshift
```

출력 예

```
Name:          cli
Namespace:     openshift
[...]
```

```
latest
```

```
updates automatically from registry <mirror-registry>:<port>/openshift-release-
dev/ocp-v4.0-art-dev
```

```
* <mirror-registry>:<port>/openshift-release-dev/ocp-v4.0-art-
dev@sha256:65c68e8c22487375c4c6ce6f18ed5485915f2bf612e41fef6d41cbfcdb143
551
```

```
[...]
```

추가 리소스

- [제한된 클러스터에 대한 Samples Operator 구성](#)
- [미러링된 레지스트리로 클러스터 생성](#)

4.5.6. 파이프라인 실행

PipelineRun 리소스는 파이프라인을 시작하고 특정 호출에 사용해야 하는 **Git** 및 이미지 리소스에 연결합니다. 그리고 파이프라인의 각 작업에 대해 **TaskRun** 리소스를 자동으로 생성하고 시작합니다.

프로세스

1. 백엔드 애플리케이션의 파이프라인을 시작합니다.

```
$ tkn pipeline start build-and-deploy \
  -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pi
pelines-tutorial/pipelines-1.5/01_pipeline/03_persistent_volume_claim.yaml \
  -p deployment-name=pipelines-vote-api \
  -p git-url=https://github.com/openshift/pipelines-vote-api.git \
  -p IMAGE=image-registry.openshift-image-registry.svc:5000/pipelines-
tutorial/pipelines-vote-api \
  --use-param-defaults
```

위 명령은 파이프라인 실행을 위한 영구 볼륨 클레임을 생성하는 볼륨 클레임 템플릿을 사용합니다.

2. 파이프라인 실행의 진행 상황을 추적하려면 다음 명령을 입력합니다.

```
$ tkn pipelinerun logs <pipelinerun_id> -f
```

위 명령의 <pipelinerun_id>는 이전 명령의 출력에서 반환된 PipelineRun의 ID입니다.

3.

프런트 엔드 애플리케이션의 파이프라인을 시작합니다.

```
$ tkn pipeline start build-and-deploy \
  -w name=shared-
workspace,volumeClaimTemplateFile=https://raw.githubusercontent.com/openshift/pi
pelines-tutorial/pipelines-1.5/01_pipeline/03_persistent_volume_claim.yaml \
  -p deployment-name=pipelines-vote-ui \
  -p git-url=https://github.com/openshift/pipelines-vote-ui.git \
  -p IMAGE=image-registry.openshift-image-registry.svc:5000/pipelines-
tutorial/pipelines-vote-ui \
  --use-param-defaults
```

4.

파이프라인 실행의 진행 상황을 추적하려면 다음 명령을 입력합니다.

```
$ tkn pipelinerun logs <pipelinerun_id> -f
```

위 명령의 <pipelinerun_id>는 이전 명령의 출력에서 반환된 PipelineRun의 ID입니다.

5.

몇 분 후에 `tkn pipelinerun list` 명령을 사용하여 모든 파이프라인 실행을 나열하여 파이프라인이 성공적으로 실행되었는지 확인합니다.

```
$ tkn pipelinerun list
```

파이프라인 실행 목록이 출력됩니다.

NAME	STARTED	DURATION	STATUS
build-and-deploy-run-xy7rw	1 hour ago	2 minutes	Succeeded
build-and-deploy-run-z2rz8	1 hour ago	19 minutes	Succeeded

6.

애플리케이션 경로를 가져옵니다.

```
$ oc get route pipelines-vote-ui --template='http://{{.spec.host}}'
```

이전 명령의 출력에 주목하십시오. 이 경로를 사용하여 애플리케이션에 액세스할 수 있습니다.

7.

이전 파이프라인의 파이프라인 리소스 및 서비스 계정을 사용하여 마지막 파이프라인 실행을 다시 실행하려면 다음을 실행합니다.

```
$ tkn pipeline start build-and-deploy --last
```

추가 리소스

•

[Git 보안을 사용하여 파이프라인 인증](#)

4.5.7. 파이프라인에 트리거 추가

트리거를 사용하면 파이프라인에서 내보내기 이벤트 및 가져오기 요청 등의 외부 **GitHub** 이벤트에 응답할 수 있습니다. 애플리케이션에 대한 파이프라인을 어셈블하고 시작한 후 **TriggerBinding**, **TriggerTemplate**, **Trigger**, **EventListener** 리소스를 추가하여 **GitHub** 이벤트를 캡처합니다.

프로세스

1.

다음 샘플 **TriggerBinding** **YAML** 파일의 내용을 복사하여 저장합니다.

```
apiVersion: triggers.tekton.dev/v1alpha1
kind: TriggerBinding
metadata:
  name: vote-app
spec:
  params:
    - name: git-repo-url
      value: $(body.repository.url)
    - name: git-repo-name
      value: $(body.repository.name)
    - name: git-revision
      value: $(body.head_commit.id)
```

2.

TriggerBinding 리소스를 생성합니다.

```
$ oc create -f <triggerbinding-yaml-file-name.yaml>
```

또는 **pipelines-tutorial** **Git** 리포지토리에서 직접 **TriggerBinding** 리소스를 생성할 수 있습니다.

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.5/03_triggers/01_binding.yaml
```

3.

다음 샘플 TriggerTemplate YAML 파일의 내용을 복사하여 저장합니다.

```
apiVersion: triggers.tekton.dev/v1alpha1
kind: TriggerTemplate
metadata:
  name: vote-app
spec:
  params:
    - name: git-repo-url
      description: The git repository url
    - name: git-revision
      description: The git revision
      default: pipelines-1.5
    - name: git-repo-name
      description: The name of the deployment to be created / patched

  resourcetemplates:
    - apiVersion: tekton.dev/v1beta1
      kind: PipelineRun
      metadata:
        generateName: build-deploy-$(tt.params.git-repo-name)-
      spec:
        serviceAccountName: pipeline
        pipelineRef:
          name: build-and-deploy
        params:
          - name: deployment-name
            value: $(tt.params.git-repo-name)
          - name: git-url
            value: $(tt.params.git-repo-url)
          - name: git-revision
            value: $(tt.params.git-revision)
          - name: IMAGE
            value: image-registry.openshift-image-registry.svc:5000/pipelines-tutorial/$(tt.params.git-repo-name)
        workspaces:
          - name: shared-workspace
            volumeClaimTemplate:
              spec:
                accessModes:
                  - ReadWriteOnce
                resources:
                  requests:
                    storage: 500Mi
```

템플릿은 작업 영역의 스토리지 볼륨을 정의하기 위해 영구 볼륨 클레임을 생성하는 볼륨 클레임 템플릿을 지정합니다. 따라서 데이터 스토리지를 제공하기 위해 영구 볼륨 클레임을 생성할 필요가 없습니다.

4. **TriggerTemplate** 리소스를 생성합니다.

```
$ oc create -f <triggertemplate-yaml-file-name.yaml>
```

또는 **pipelines-tutorial** Git 리포지토리에서 직접 **TriggerTemplate** 리소스를 생성할 수도 있습니다.

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.5/03_triggers/02_template.yaml
```

5. 다음 샘플 **Trigger** **YAML** 파일의 콘텐츠를 복사하여 저장합니다.

```
apiVersion: triggers.tekton.dev/v1alpha1
kind: Trigger
metadata:
  name: vote-trigger
spec:
  serviceAccountName: pipeline
  bindings:
  - ref: vote-app
  template:
    ref: vote-app
```

6. **Trigger** 리소스를 생성합니다.

```
$ oc create -f <trigger-yaml-file-name.yaml>
```

또는 **pipelines-tutorial** Git 리포지토리에서 직접 **Trigger** 리소스를 생성할 수도 있습니다.

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.5/03_triggers/03_trigger.yaml
```

7. 다음 샘플 **EventListener** **YAML** 파일의 내용을 복사하여 저장합니다.

```
apiVersion: triggers.tekton.dev/v1alpha1
kind: EventListener
metadata:
  name: vote-app
spec:
  serviceAccountName: pipeline
  triggers:
  - triggerRef: vote-trigger
```

또는 트리거 사용자 정의 리소스를 정의하지 않은 경우 트리거 이름을 참조하는 대신 바인딩 및 템플릿 사양을 **EventListener** YAML 파일에 추가합니다.

```
apiVersion: triggers.tekton.dev/v1alpha1
kind: EventListener
metadata:
  name: vote-app
spec:
  serviceAccountName: pipeline
  triggers:
  - bindings:
    - ref: vote-app
  template:
    ref: vote-app
```

8.

다음 단계를 수행하여 **EventListener** 리소스를 생성합니다.

-

보안 **HTTPS** 연결을 사용하여 **EventListener** 리소스를 생성하려면 다음을 수행합니다.

a.

EventListener 리소스에 대한 보안 **HTTPS** 연결을 활성화하려면 레이블을 추가합니다.

```
$ oc label namespace <ns-name> operator.tekton.dev/enable-annotation=enabled
```

b.

EventListener 리소스를 생성합니다.

```
$ oc create -f <eventlistener-yaml-file-name.yaml>
```

또는 **pipelines-tutorial** Git 리포지토리에서 직접 **EventListener** 리소스를 생성할 수도 있습니다.

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/pipelines-1.5/03_triggers/04_event_listener.yaml
```

c.

재암호화 **TLS** 종료로 경로를 생성합니다.

```
$ oc create route reencrypt --service=<svc-name> --cert=tls.crt --key=tls.key --ca-cert=ca.crt --hostname=<hostname>
```


또는 재암호화 TLS 종료 YAML 파일을 만들어 보안 경로를 만들 수도 있습니다.

보안 경로의 TLS 종료 YAML에 대한 재암호화의 예

```

apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-passthrough-secured ❶
spec:
  host: <hostname>
  to:
    kind: Service
    name: frontend ❷
  tls:
    termination: reencrypt ❸
    key: [as in edge termination]
    certificate: [as in edge termination]
    caCertificate: [as in edge termination]
    destinationCACertificate: |- ❹
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----

```

❶ ❷

63자로 제한되는 개체의 이름입니다.

❸

termination 필드는 **reencrypt**로 설정됩니다. 이 필드는 유일한 필수 **tls** 필드입니다.

❹

재암호화에 필요합니다. **destinationCACertificate** 는 엔드포인트 인증서의 유효성을 검사하고 라우터에서 대상 포드로의 연결을 보호하는 **CA** 인증서를 지정합니다. 서비스에서 서비스 서명 인증서를 사용 중이거나 관리자가 라우터의 기본 **CA** 인증서를 지정하고 서비스에 해당 **CA**에서 서명한 인증서가 있는 경우 이 필드를 생략할 수 있습니다.

자세한 옵션은 **oc create route reencrypt --help**를 참조하십시오.

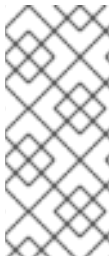
- 비보안 HTTP 연결을 사용하여 **EventListener** 리소스를 생성하려면 다음을 수행합니다.
 - a. **EventListener** 리소스를 생성합니다.
 - b. **EventListener** 서비스에 공개 액세스가 가능하도록 이 서비스를 **OpenShift Container Platform** 경로로 노출합니다.

```
$ oc expose svc el-vote-app
```

4.5.8. Webhook 생성

Webhooks는 리포지토리에 구성된 이벤트가 발생할 때마다 이벤트 리스너가 수신하는 **HTTP POST** 메시지입니다. 이어서 이벤트 페이로드가 트리거 바인딩에 매핑되고 트리거 템플릿에 의해 처리됩니다. 트리거 템플릿은 최종적으로 **Kubernetes** 리소스를 생성 및 배포를 수행할 하나 이상의 파이프라인 실행을 시작합니다.

여기서는 분기된 **Git** 리포지토리 **pipelines-vote-ui**와 **pipelines-vote-api**에 대한 **Webhook URL**을 구성합니다. 이 **URL**은 공개 액세스 가능한 **EventListener** 서비스 경로를 가리킵니다.



참고

Webhook를 추가하려면 리포지토리에 대한 관리자 권한이 필요합니다. 리포지토리에 대한 관리자 액세스 권한이 없으면 시스템 관리자에게 요청하여 **Webhook**를 추가하십시오.

프로세스

1. **Webhook URL**을 가져옵니다.

- 보안 **HTTPS** 연결의 경우 다음을 수행합니다.

```
$ echo "URL: $(oc get route el-vote-app --template='https://{{.spec.host}}')"
```

- **HTTP(비보안)** 연결의 경우 다음을 수행합니다.

```
$ echo "URL: $(oc get route el-vote-app --template='http://{{.spec.host}}')"
```

출력에서 가져온 **URL**을 기록해 둡니다.

2.
 - 프런트 엔드 리포지토리에서 수동으로 **Webhook**을 구성합니다.
 - a. 브라우저에서 프런트 엔드 **Git** 리포지토리 **pipelines-vote-ui**를 엽니다.
 - b. **Settings** → **Webhook** → **Webhook** 추가를 클릭합니다.
 - c. **Webhooks/Add Webhook** 페이지에서:
 - i. **Payload URL** 필드에 1단계의 **Webhook URL**을 입력합니다.
 - ii. **Content type**으로 **application/json**을 선택합니다.
 - iii. **Secret** 필드에 시크릿을 지정합니다.
 - iv. **Just the push event**이 선택되어 있는지 확인합니다.
 - v. **Active**를 선택하십시오.
 - vi. **Add Webhook**를 클릭합니다.
3.
 - 백엔드 리포지토리 **pipelines-vote-api**에 대해 2단계를 반복합니다.

4.5.9. 파이프라인 실행 트리거

Git 리포지토리에서 **push** 이벤트가 발생할 때마다 구성된 **Webhook**에서 공개 노출된 **EventListener** 서비스 경로로 이벤트 페이로드를 보냅니다. 애플리케이션의 **EventListener** 서비스는 페이로드를 처리 하여 관련 **TriggerBinding** 및 **TriggerTemplate** 쌍으로 전달합니다. **TriggerBinding** 리소스는 매개변수

를 추출하고 **TriggerTemplate** 리소스는 이러한 매개변수를 사용하여 리소스 생성 방식을 지정합니다. 그리고 애플리케이션을 다시 빌드 및 배포할 수도 있습니다.

이 섹션에서는 비어 있는 커밋을 프런트 엔드 **pipelines-vote-ui** 리포지토리로 내보냅니다. 그러면 파이프라인 실행이 트리거됩니다.

프로세스

1.

터미널에서 분기된 **Git** 리포지토리 **pipelines-vote-ui**를 복제합니다.

```
$ git clone git@github.com:<your GitHub ID>/pipelines-vote-ui.git -b pipelines-1.5
```

2.

비어 있는 커밋을 푸시합니다.

```
$ git commit -m "empty-commit" --allow-empty && git push origin pipelines-1.5
```

3.

파이프라인 실행이 트리거되었는지 확인합니다.

```
$ tkn pipelinerun list
```

새로운 파이프라인 실행이 시작되었습니다.

4.5.10. 추가 리소스

-

개발자 화면의 파이프라인에 대한 자세한 내용은 [개발자 화면에서 파이프라인 작업](#) 섹션을 참조하십시오.

-

SCC(보안 컨텍스트 제약 조건)에 대한 자세한 내용은 [보안 컨텍스트 제약 조건 관리](#) 섹션을 참조하십시오.

-

재사용 가능 작업의 예를 더 보려면 [OpenShift 카탈로그](#) 리포지토리를 참조하십시오. **Tekton** 프로젝트의 **Tekton** 카탈로그도 참조할 수 있습니다.

-

재암호화 **TLS** 종료에 대한 자세한 내용은 [재암호화 종료](#)를 참조하십시오.

- 보안 경로에 대한 자세한 내용은 [보안 경로](#) 섹션을 참조하십시오.

4.6. 개발자 화면을 사용하여 RED HAT OPENSIFT PIPELINES 작업

OpenShift Container Platform 웹 콘솔의 개발자 화면을 사용하여 소프트웨어 제공 프로세스를 위한 **CI/CD Pipeline**을 생성할 수 있습니다.

개발자 화면에서:

- **Add** → **Pipeline** → **Pipeline Builder** 옵션을 사용하여 애플리케이션에 사용자 지정된 파이프라인을 생성합니다.
- **Add** → **From Git** 옵션을 사용하여 **OpenShift Container Platform**에서 애플리케이션을 생성하는 동안 **operator** 설치 파이프라인 템플릿과 리소스를 이용해 파이프라인을 생성합니다.

애플리케이션의 파이프라인을 생성한 후 **Pipelines** 보기에서 배포된 파이프라인을 보면서 시각적으로 상호 작용할 수 있습니다. **Topology** 보기에서도 **From Git** 옵션을 사용하여 생성된 파이프라인과 상호 작용할 수 있습니다. **Topology** 보기에서 볼 수 있으려면 **Pipeline Builder**를 사용하여 생성된 파이프라인에 사용자 지정 레이블을 적용해야 합니다.

사전 요구 사항

- **OpenShift Container Platform** 클러스터에 액세스하고 **개발자 화면으로 전환했습니다**.
- 클러스터에 **OpenShift Pipelines Operator**를 설치했습니다.
- 클러스터 관리자 또는 작성 및 편집 권한이 있는 사용자입니다.
- 프로젝트를 생성했습니다.

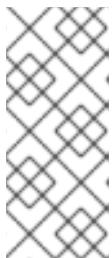
4.6.1. Pipeline 빌더를 사용하여 Pipeline 구성

콘솔의 개발자 화면에서 **+추가** → **파이프라인** → **파이프라인 빌더** 옵션을 사용하여 다음을 수행할 수 있습니다.

- 파이프라인 빌더 또는 **YAML** 보기를 사용하여 파이프라인을 구성합니다.
- 기존 작업 및 클러스터 작업을 사용하여 파이프라인 흐름을 구성합니다. **OpenShift Pipelines Operator**를 설치하면 재사용 가능한 파이프라인 클러스터 작업이 클러스터에 추가됩니다.
- 파이프라인 실행에 필요한 리소스 유형을 지정하고, 필요한 경우 파이프라인에 매개변수를 추가합니다.
- 파이프라인의 각 작업에서 이러한 파이프라인 리소스를 입력 및 출력 리소스로 참조합니다.
- 필요한 경우 작업의 파이프라인에 추가된 매개변수를 참조합니다. 작업 매개변수는 작업 사양에 따라 미리 채워집니다.
- **Operator**에서 설치한 재사용 가능 조각과 샘플을 사용하여 세부 파이프라인을 생성합니다.

프로세스

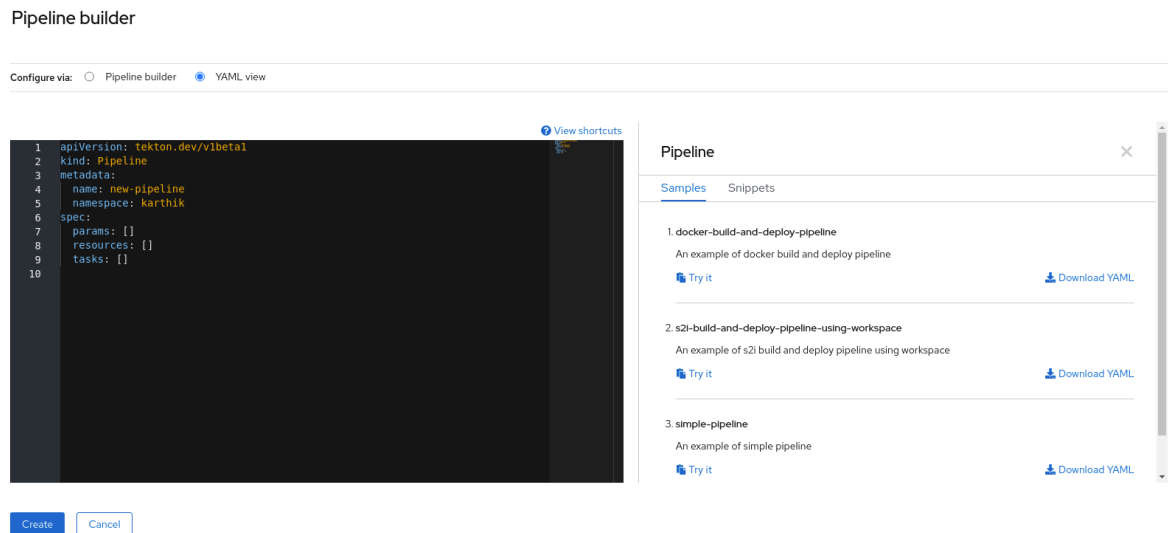
1. 개발자 화면의 +추가 보기에서 파이프라인 타일을 클릭하여 파이프라인 빌더 페이지를 표시합니다.
2. 파이프라인 빌더 보기 또는 **YAML** 보기를 사용하여 파이프라인을 구성합니다.



참고

파이프라인 빌더 보기에서는 제한된 수의 필드를 지원하는 반면 **YAML** 보기는 사용 가능한 모든 필드를 지원합니다. 필요한 경우 **Operator**에서 설치한 재사용 가능 조각과 샘플을 사용하여 세부 파이프라인을 생성할 수 있습니다.

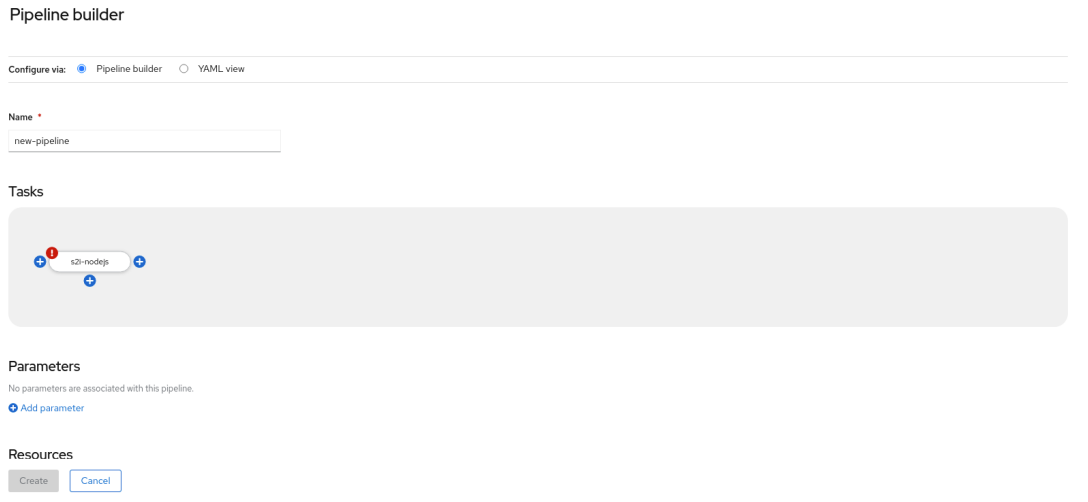
그림 4.1. YAML보기



파이프라인 빌더를 사용하여 파이프라인을 구성하려면 다음을 수행합니다.

- 이름 필드에 파이프라인의 고유 이름을 입력합니다.
- 작업 섹션의 **Select Task** 목록에서 작업을 선택하여 파이프라인에 작업을 추가합니다. 이 예제에서는 **s2i-nodejs** 작업을 사용합니다.
 - 파이프라인에 순차 작업을 추가하려면 작업 오른쪽 또는 왼쪽에 있는 더하기 아이콘을 클릭하고 작업 선택 목록에서 파이프라인에 추가할 작업을 선택합니다. 예를 들어 **openshift-client** 작업을 추가하려면 **s2i-nodejs** 작업 오른쪽에 있는 더하기 아이콘을 사용합니다.
 - 기존 작업에 병렬 작업을 추가하려면 작업 옆에 표시된 더하기 아이콘을 클릭하고 작업 선택 목록에서 파이프라인에 추가할 병렬 작업을 선택합니다.

그림 4.2. Pipeline 빌더



C.

리소스 섹션에서 리소스 추가를 클릭하여 파이프라인 실행에 사용할 리소스의 이름 및 유형을 지정합니다. 그러면 파이프라인의 작업에서 이러한 리소스를 입력 및 출력으로 사용합니다. 예시의 경우:

i.

입력 리소스를 추가합니다. 이름 필드에 **Source**를 입력하고 리소스 유형 드롭다운 목록에서 **Git**을 선택합니다.

ii.

출력 리소스를 추가합니다. 이름 필드에 **Img**를 입력하고 리소스 유형 드롭다운 목록에서 이미지를 선택합니다.



참고

리소스가 누락된 경우 작업 옆에 빨간색 아이콘이 표시됩니다.

d.

선택 사항: 작업의 매개 변수는 작업 사양에 따라 미리 채워집니다. 필요에 따라 매개 변수 섹션에 있는 매개 변수 추가 링크를 사용하여 매개 변수를 더 추가합니다.

e.

작업 공간 섹션에서 작업 공간 추가를 클릭하고 이름 필드에 고유한 작업 공간 이름을 입력합니다. 파이프라인에 여러 개의 작업 공간을 추가할 수 있습니다.

f.

작업 섹션에서 **s2i-nodejs** 작업을 클릭하여 작업 세부 정보가 있는 측면 패널을 확인합니다. 작업 측면 패널에서 **s2i-nodejs** 작업에 대한 리소스 및 매개 변수를 지정합니다.

i.

필요에 따라 매개 변수 섹션에서 $\$(params.<param-name>)$ 구문을 사용하여 기

본 매개변수에 매개변수를 더 추가합니다.

- ii. 이미지 섹션에서 리소스 섹션에 지정된 대로 **Img**를 입력합니다.
 - iii. 작업 공간 섹션의 소스 드롭다운에서 작업 공간을 선택합니다.
9. 리소스, 매개 변수 및 작업 공간을 **openshift-client** 작업에 추가합니다.
3. 생성을 클릭하여 파이프라인 세부 정보 페이지에서 파이프라인을 생성하고 봅니다.
 4. 작업 드롭다운 메뉴를 클릭한 다음 시작을 클릭하여 파이프라인 시작 페이지를 확인합니다.
 5. 작업 공간 섹션에는 이전에 생성한 작업 공간이 나열됩니다. 각 드롭다운을 사용하여 작업 공간의 볼륨 소스를 지정합니다. 다음과 같은 옵션이 있습니다. 빈 디렉터리, 구성 맵, 시크릿, **PersistentVolumeClaim** 또는 **VolumeClaimTemplate**.

4.6.2. OpenShift Pipelines를 사용하여 애플리케이션 작성

애플리케이션과 함께 파이프라인을 생성하려면 개발자 화면의 추가 보기에서 **From Git** 옵션을 사용합니다. 자세한 내용은 [개발자 화면을 사용하여 애플리케이션 생성](#)을 참조하십시오.

4.6.3. 개발자 화면을 사용하여 파이프라인과 상호 작용

개발자 화면의 파이프라인 보기에는 다음 세부 정보와 함께 프로젝트의 모든 파이프라인이 나열됩니다.

- 파이프라인이 생성된 네임스페이스
- 마지막 파이프라인 실행
- 파이프라인 실행 시 작업 상태

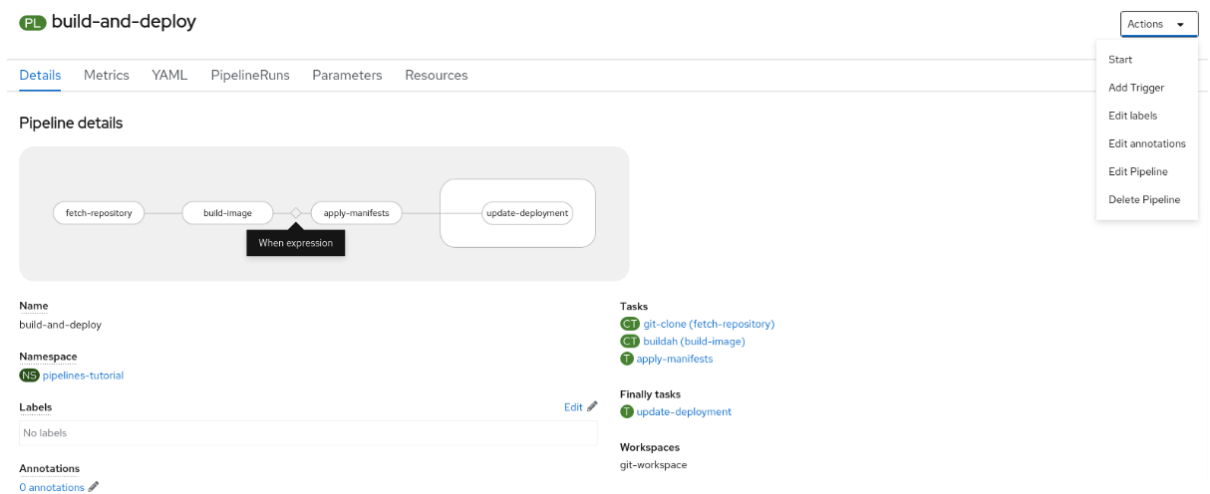
- 파이프라인 실행의 상태
- 마지막 파이프라인 실행 생성 시간

프로세스

1. 개발자 화면의 파이프라인 보기에서 프로젝트 드롭다운 목록에 있는 프로젝트를 선택하여 해당 프로젝트의 파이프라인을 확인합니다.
2. 필요한 파이프라인을 클릭하여 파이프라인 세부 정보 페이지를 확인합니다.

기본적으로 세부 정보 탭에는 모든 직렬 작업, 병렬 작업, **finally** 작업 및 파이프라인의 **when** 표현식이 모두 시각적으로 표시됩니다. 작업 및 **finally** 작업은 페이지 오른쪽 하단 목록에 표시됩니다. 목록의 **Tasks** 및 **Finally tasks**를 클릭하면 해당 작업의 세부 정보를 확인할 수 있습니다.

그림 4.3. 파이프 라인 세부 정보



3. 선택 사항: 파이프라인 세부 정보 페이지에서 지표 탭을 클릭하여 파이프라인에 대한 다음 정보를 확인합니다.

- 파이프 라인 성공률
- 파이프 라인 실행 수
- 파이프 라인 실행 기간

- **작업 실행 기간**

이 정보를 사용하여 파이프라인 라이프사이클 초기에 파이프라인 워크플로를 개선하고 문제를 제거할 수 있습니다.

4.

선택 사항: **YAML** 탭을 클릭하여 파이프라인의 **YAML** 파일을 편집합니다.

5.

선택 사항: 파이프라인 실행 탭을 클릭하여 파이프라인 실행 상태가 완료, 실행 중 또는 실패인지 확인합니다.

파이프라인 실행 탭에는 파이프라인 실행, 작업 상태 및 실패한 파이프라인 실행 디버그 링크에 대한 세부 정보가 있습니다. 옵션 메뉴



에서는 실행 중인 파이프라인 중지, 이전 파이프라인 실행과 동일한 매개변수 및 리소스를 사용하여 파이프라인 재실행 또는 파이프라인 실행 삭제 작업을 수행할 수 있습니다.

-

필요한 파이프라인 실행을 클릭하여 파이프라인 실행 세부 정보 페이지를 확인합니다. 기본적으로 세부 정보 탭에는 모든 직렬 작업, 병렬 작업, **finally** 작업 및 파이프라인 실행의 **When** 표현식의 시각적 표현이 표시됩니다. 성공적인 실행 결과는 페이지 하단의 파이프라인 실행 결과 창에 표시됩니다.



참고

파이프라인 실행 세부 정보 페이지의 세부 정보 섹션에는 실패한 파이프라인 실행에 대한 로그 조각이 표시됩니다. 로그 조각에는 일반적인 오류 메시지와 해당 로그의 조각이 있습니다. 로그 섹션 링크를 사용하면 실패한 실행에 대한 세부 정보에 빠르게 액세스할 수 있습니다.

-

파이프라인 실행 세부 정보 페이지에서 작업 실행 탭을 클릭하여 작업 상태가 완료, 실행 중 또는 실패인지 확인합니다.

작업 실행 탭은 해당 작업 및 **pod**에 대한 링크, 작업 실행 상태 및 기간과 함께 작업 실행에 대한 정보를 제공합니다. 옵션 메뉴



를 사용하여 작업 실행을 삭제합니다.

-

필요한 작업 실행을 클릭하여 작업 실행 세부 정보 페이지를 확인합니다. 성공적으로 실행된 결과는 페이지 하단에 있는 작업 실행 결과 창에 표시됩니다.



참고

작업 실행 세부 정보 페이지의 세부 정보 섹션에는 실패한 작업 실행에 대한 로그 조각이 표시됩니다. 로그 조각에는 일반적인 오류 메시지와 해당 로그의 조각이 있습니다. 로그 섹션 링크를 사용하면 실패한 작업 실행에 대한 세부 정보에 빠르게 액세스할 수 있습니다.

- 6. 매개변수 탭을 클릭하여 파이프라인에 정의된 매개변수를 확인합니다. 필요에 따라 매개변수를 추가하거나 편집할 수도 있습니다.
- 7. 리소스 탭을 클릭하여 파이프라인에 정의된 리소스를 확인합니다. 필요에 따라 리소스를 추가하거나 편집할 수도 있습니다.

4.6.4. Git 리포지토리에서 애플리케이션 생성 및 배포에 사용자 지정 파이프라인 템플릿 사용

클러스터 관리자는 **Git** 리포지토리에서 애플리케이션을 생성하고 배포하기 위해 **Red Hat OpenShift Pipelines 1.5** 이상에서 제공하는 기본 파이프라인 템플릿을 재정의하는 사용자 지정 파이프라인 템플릿을 사용할 수 있습니다.



참고

이 기능은 **Red Hat OpenShift Pipelines 1.4** 및 이전 버전에서 사용할 수 없습니다.

사전 요구 사항

Red Hat OpenShift Pipelines 1.5 이상이 설치되어 있고 모든 네임스페이스에서 사용할 수 있는지 확인합니다.

절차

- 1. **OpenShift Container Platform** 웹 콘솔에 클러스터 관리자로 로그인합니다.
- 2. 관리자 화면에서 왼쪽 탐색 패널을 사용하여 **Pipelines** 섹션으로 이동합니다.

- a. 프로젝트 드롭다운 메뉴에서 **openshift** 프로젝트를 선택합니다. 이렇게 하면 후속 단계가 **openshift** 네임스페이스에서 수행됩니다.
- b. 사용 가능한 파이프라인 목록에서 애플리케이션을 빌드하고 배포하는 데 적합한 파이프라인을 선택합니다. 예를 들어 애플리케이션에 **node.js** 런타임 환경이 필요한 경우 **s2i-nodejs** 파이프라인을 선택합니다.



참고

기본 파이프라인 템플릿을 편집하지 마십시오. UI 및 백엔드와 호환되지 않을 수 있습니다.

- c. 선택한 파이프라인의 **YAML** 탭에서 **YAML** 파일을 다운로드하여 로컬 시스템에 저장합니다. 사용자 지정 구성 파일에 오류가 발생하면 이 복사본을 사용하여 작동 중인 구성을 복원할 수 있습니다.
3. 기본 파이프라인 템플릿을 비활성화(삭제)합니다.
 - a. 왼쪽 탐색 패널을 사용하여 **Operator** → 설치된 **Operator**로 이동합니다.
 - b. **Red Hat OpenShift Pipelines** → **Tekton Configuration** 탭 → **config** → **YAML** 탭을 클릭합니다.
 - c. **openshift** 네임스페이스에서 기본 파이프라인 템플릿을 비활성화(삭제)하려면 **TektonConfig** 사용자 지정 리소스 **YAML**에서 **pipelineTemplates** 매개변수를 **false**로 설정하고 저장합니다.

```
apiVersion: operator.tekton.dev/v1alpha1
kind: TektonConfig
metadata:
  name: config
spec:
  profile: all
targetNamespace: openshift-pipelines
addon:
  params:
    - name: clusterTasks
      value: "true"
```

- name: pipelineTemplates
value: "false"

...



참고

기본 파이프라인 템플릿을 수동으로 삭제하면 **Operator**가 업그레이드 중에 기본값을 복원합니다.



주의

클러스터 관리자는 **operator** 구성에서 기본 파이프라인 템플릿 설치를 비활성화할 수 있습니다. 그러나 이러한 구성은 사용자 지정하려는 템플릿뿐만 아니라 모든 기본 파이프라인 템플릿을 비활성화(삭제)합니다.

4.

사용자 정의 파이프라인 템플릿을 생성합니다.

a.

왼쪽 탐색 패널을 사용하여 *파이프라인* 섹션으로 이동합니다.

b.

생성 드롭다운 메뉴에서 파이프라인을 선택합니다.

c.

openshift 네임스페이스에 필요한 파이프라인을 생성합니다. 기본 이름(예: **custom-nodejs**)과는 다른 이름을 지정합니다. 다운로드한 기본 파이프라인 템플릿을 시작점으로 사용하고 사용자 지정할 수 있습니다.



참고

openshift는 **Operator**가 설치한 파이프라인 템플릿에서 사용하는 기본 네임스페이스이므로 **openshift** 네임스페이스에서 사용자 지정 파이프라인 템플릿을 생성해야 합니다. 애플리케이션에서 파이프라인 템플릿을 사용하면 템플릿이 해당 프로젝트의 네임스페이스에 자동으로 복사됩니다.

d.

생성된 파이프라인의 세부 정보 탭에서 사용자 지정 템플릿의 레이블이 기본 파이프라인의 레이블과 일치하는지 확인합니다. 사용자 지정 파이프라인 템플릿에는 애플리케이션의

런타임, 유형 및 전략에 대한 올바른 레이블이 있어야 합니다. 예를 들어 **OpenShift Container Platform**에 배포된 **node.js** 애플리케이션의 필수 레이블은 다음과 같습니다.

```
...
pipeline.openshift.io/runtime: nodejs
pipeline.openshift.io/type: openshift
...
```



참고

런타임 환경 및 배포 유형의 각 조합에 대해 하나의 파이프라인 템플릿만 사용할 수 있습니다.

5.

개발자 화면에서 +추가 → **Git** 리포지토리 → **Git**에서 옵션을 사용하여 생성 및 배포할 애플리케이션 유형을 선택합니다. 애플리케이션의 필수 런타임 및 유형에 따라 사용자 지정 템플릿이 자동으로 선택됩니다.

4.6.5. 파이프라인 시작

파이프라인을 생성한 후 포함된 작업을 정의된 순서로 실행하려면 파이프라인을 시작해야 합니다. 파이프라인 보기, 파이프라인 세부 정보 페이지 또는 토폴로지 보기에서 파이프라인을 시작할 수 있습니다.

프로세스

파이프라인 보기를 사용하여 파이프라인을 시작하려면 다음을 수행합니다.

1.

개발자 화면의 파이프라인 보기에서 파이프라인 옆에 있는 옵션 메뉴를 클릭하고 시작을 선택합니다.

2.

파이프라인 정의에 따라 파이프라인 시작 대화 상자에 **Git** 리소스 및 이미지 리소스가 표시됩니다.



참고

Git에서 옵션을 사용하여 생성한 파이프라인의 경우 파이프라인 시작 대화 상자의 매개변수 섹션에 **APP_NAME** 필드도 표시되며, 대화 상자의 모든 필드가 파이프라인 템플릿에 의해 미리 채워집니다.

- a. 네임스페이스에 리소스가 있는 경우 **Git Resources** 및 **Image Resources** 필드에 해당 리소스가 미리 채워집니다. 필요한 경우 드롭다운 목록을 사용하여 필요한 리소스를 선택하거나 생성한 다음 파이프라인 실행 인스턴스를 사용자 정의합니다.
3. 선택 사항: **Advanced Options**(고급 옵션)를 수정하여 지정된 개인 **Git** 서버 또는 이미지 레지스트리를 인증하는 자격 증명을 추가합니다.
- a. **Advanced Options**에서 **Show Credentials Options**를 클릭하고 **Add Secret**을 선택합니다.
 - b. **Create Source Secret** 섹션에서 다음 사항을 지정합니다.
 - i. 보안에 대한 고유한 보안 이름입니다.
 - ii. **Designated provider to be authenticated** 섹션에서 **Access to** 필드에 인증할 공급자를 지정하고 기본 **Server URL**을 지정합니다.
 - iii. **Authentication Type**을 선택하고 자격 증명을 제공합니다.
 - 인증 유형 **Image Registry Credentials**의 경우 인증할 레지스트리 서버 주소를 지정하고 사용자 이름, 암호, 이메일 필드에 자격 증명을 제공합니다.
추가 **Registry Server Address**를 지정하려면 **Add Credentials**를 선택하십시오.
 - **Authentication Type Basic Authentication**의 경우 **UserName** 및 **Password or Token** 필드 값을 지정합니다.
 - 인증 유형 **SSH Keys**의 경우 **SSH 개인 키** 필드 값을 지정합니다.



참고

기본 인증 및 SSH 인증의 경우 다음과 같은 주석을 사용할 수 있습니다.

○

tekton.dev/git-0: <https://github.com>

○

tekton.dev/git-1: <https://gitlab.com>.

iv.

확인 표시를 선택하여 보안을 추가합니다.

파이프라인의 리소스 수에 따라 여러 개의 보안을 추가할 수 있습니다.

4.

시작을 클릭하여 파이프라인을 시작합니다.

5.

파이프라인 실행 세부 정보 페이지에 실행 중인 파이프라인이 표시됩니다. 파이프라인이 시작된 후 작업과 각 작업 내 단계가 실행됩니다. 다음을 수행할 수 있습니다.

●

작업 위로 커서를 이동하여 각 단계를 실행하는 데 걸리는 시간을 확인합니다.

●

작업을 클릭하여 각 작업 단계에 대한 로그를 확인합니다.

●

로그 탭을 클릭하여 작업 실행 순서와 관련된 로그를 확인합니다. 관련 버튼을 사용하여 창을 확장하고 로그를 개별적 또는 일괄적으로 다운로드할 수도 있습니다.

●

이벤트 탭을 클릭하여 파이프라인 실행으로 생성된 이벤트 스트림을 확인합니다.

작업 실행, 로그, 이벤트 탭을 사용하면 실패한 파이프라인 실행 또는 실패한 작업 실행을 디버깅하는 데 도움이 될 수 있습니다.

그림 4.4. '파이프 라인 실행' 세부 정보

Project: pipelines-tutorial ▾

[Pipeline Runs](#) > Pipeline Run details

PLR build-and-deploy-tcy5g4 Running

[Details](#) | [YAML](#) | [Task Runs](#) | [Logs](#) | [Events](#)

Pipeline Run details

Name
build-and-deploy-...

Namespace
pipelines-tutorial

Labels
tekton.dev/pipeline=build-and-deploy

Status
Running

Pipeline
build-and-deploy

Triggered by:
kube:admin

6.

Git에서 옵션을 사용하여 생성한 파이프라인의 경우 토폴로지 보기를 사용하여 파이프라인을 시작한 후 상호 작용할 수 있습니다.



참고

토폴로지 보기에서 파이프라인 빌더를 사용하여 생성한 파이프라인을 보려면 파이프라인 라벨을 사용자 정의하여 파이프라인을 애플리케이션 워크로드에 연결합니다.

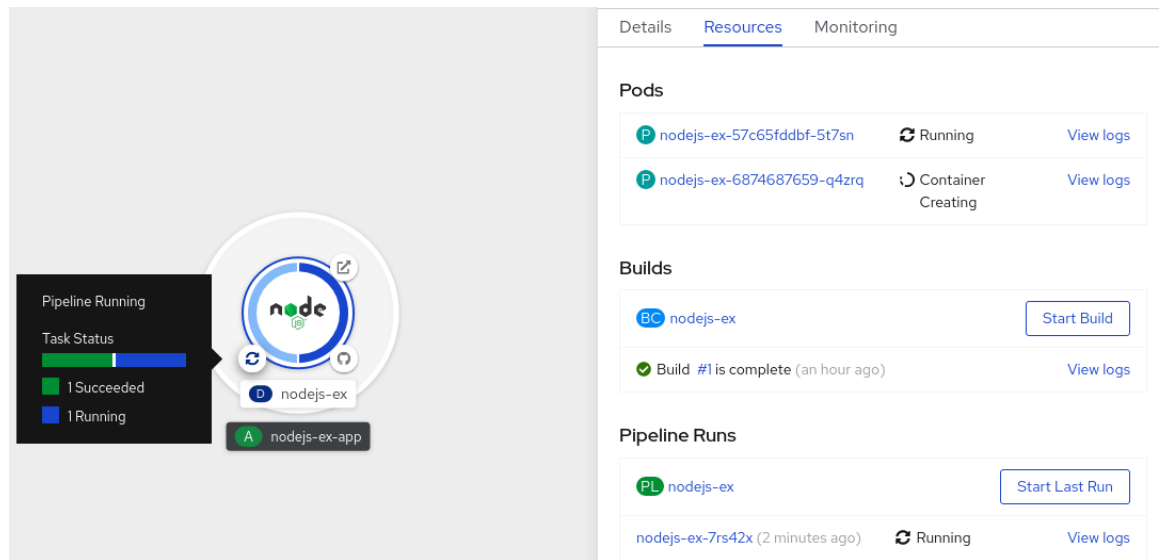
a.

왼쪽 탐색 패널에서 토폴로지를 클릭하고 애플리케이션을 클릭하여 사이드 패널에서 파이프라인 실행 목록을 확인합니다.

b.

파이프라인 실행 섹션에서 마지막 실행 시작을 클릭하여 이전 파이프라인과 동일한 매개변수 및 리소스로 새 파이프라인 실행을 시작합니다. 파이프라인 실행이 시작되지 않은 경우 이 옵션이 비활성화되어 있습니다.

그림 4.5. 토폴로지 보기의 파이프라인



C.

토폴로지 페이지에서 애플리케이션 왼쪽으로 커서를 이동하여 애플리케이션의 파이프라인 실행 상태를 확인합니다.



참고

토폴로지 페이지의 애플리케이션 노드 사이드 패널에는 파이프라인 실행이 특정 작업 실행에서 실패할 때 로그 조각이 표시됩니다. 로그 조각은 리소스 탭의 파이프라인 실행 섹션에서 확인할 수 있습니다. 로그 조각에는 일반적인 오류 메시지와 해당 로그의 조각이 있습니다. 로그 섹션 링크를 사용하면 실패한 실행에 대한 세부 정보에 빠르게 액세스할 수 있습니다.

4.6.6. Pipeline 편집

웹 콘솔의 개발자 화면을 사용하여 클러스터의 **Pipeline**을 편집할 수 있습니다.

프로세스

1.

개발자 화면의 **Pipelines** 보기에서 편집할 **Pipeline**을 선택하여 **Pipeline**의 세부 사항을 표시합니다. **Pipeline Details** 페이지에서 **Actions**를 클릭하고 **Edit Pipelin**을 선택합니다.

2.

Pipeline Builder 페이지에서:

•

추가 **Task**, 매개변수 또는 리소스를 **Pipeline**에 추가할 수 있습니다.

•

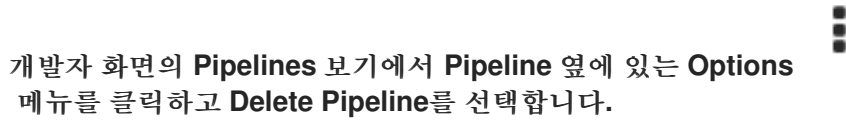
수정하려는 **Task**를 클릭하면 측면 패널에 **Task** 세부 사항이 표시됩니다. 여기서 표시 이름, 매개변수 및 리소스와 같이 필요한 **Task** 세부 사항을 수정할 수 있습니다.

- 또는 **Task**를 클릭하고 측면 패널에서 **Actions**를 클릭하고 **Remove Task**를 선택하여 **Task**를 삭제할 수도 있습니다.
3. **Save**를 클릭하여 수정된 **Pipeline**을 저장합니다.

4.6.7. Pipeline 삭제

웹 콘솔의 개발자 화면을 사용하여 클러스터의 **Pipeline**을 삭제할 수 있습니다.

프로세스

1. 개발자 화면의 **Pipelines** 보기에서 **Pipeline** 옆에 있는 **Options** 메뉴를 클릭하고 **Delete Pipeline**를 선택합니다.
 
2. **Delete Pipeline** 확인 프롬프트에서 **Delete**를 클릭하여 삭제를 확인합니다.

4.7. 파이프라인의 리소스 사용량 감소

멀티 테넌트 환경에서 클러스터를 사용하는 경우 각 프로젝트 및 **Kubernetes** 오브젝트에 대한 **CPU**, 메모리 및 스토리지 리소스의 사용을 제어해야 합니다. 따라서 하나의 애플리케이션이 너무 많은 리소스를 소비하고 다른 애플리케이션에 영향을 주지 않도록 방지할 수 있습니다.

결과 **pod**에 설정된 최종 리소스 제한을 정의하기 위해 **Red Hat OpenShift Pipelines**는 리소스 할당량 제한 및 해당 **Pod**가 실행되는 프로젝트의 제한 범위를 사용합니다.

프로젝트의 리소스 사용을 제한하려면 다음을 수행할 수 있습니다.

- 리소스 할당량을 설정하고 관리하여 집계 리소스 사용을 제한합니다.
- **pod**, 이미지, 이미지 스트림, 영구 볼륨 클레임과 같은 특정 오브젝트에 대한 제한 범위를 사

용하여 리소스 사용을 제한합니다.

4.7.1. 파이프라인에서 리소스 사용 이해

각 작업은 **Task** 리소스의 **steps** 필드에 정의된 특정 순서로 실행하는 데 필요한 여러 단계로 구성됩니다. 모든 작업은 **Pod**로 실행되고 각 단계는 해당 **Pod** 내에서 컨테이너로 실행됩니다.

단계는 한 번에 하나씩 실행됩니다. 작업을 실행하는 **Pod**는 한 번에 작업에서 단일 컨테이너 이미지 (단계)를 실행하기에 충분한 리소스만 요청하므로 작업의 모든 단계에 대한 리소스를 저장하지 않습니다.

spec 단계의 **Resources** 필드는 리소스 소비에 대한 제한을 지정합니다. 기본적으로 **CPU**, 메모리, 임시 스토리지에 대한 리소스 요청은 **BestEffort (zero)** 값으로 설정되거나 해당 프로젝트에서 제한 범위를 통해 설정된 최소값으로 설정됩니다.

리소스 요청 구성 및 단계 제한의 예

```
spec:
  steps:
  - name: <step_name>
    resources:
      requests:
        memory: 2Gi
        cpu: 600m
      limits:
        memory: 4Gi
        cpu: 900m
```

LimitRange 매개변수 및 컨테이너 리소스 요청에 대한 최소 값이 **Pipeline** 및 작업을 실행하는 프로젝트에 지정되면 **Red Hat OpenShift Pipelines**는 프로젝트의 모든 **LimitRange** 값을 살펴보고 0 대신 최소 값을 사용합니다.

프로젝트 수준에서 제한 범위 매개변수 구성 예

```
apiVersion: v1
kind: LimitRange
metadata:
  name: <limit_container_resource>
```

```
spec:
  limits:
  - max:
    cpu: "600m"
    memory: "2Gi"
  min:
    cpu: "200m"
    memory: "100Mi"
  default:
    cpu: "500m"
    memory: "800Mi"
  defaultRequest:
    cpu: "100m"
    memory: "100Mi"
  type: Container
...
```

4.7.2. 파이프라인에서 추가 리소스 소비 완화

Pod의 컨테이너에 리소스 제한이 설정된 경우 **OpenShift Container Platform**은 모든 컨테이너가 동시에 실행될 때 요청된 리소스 제한을 합계합니다.

호출된 작업에서 한 번에 한 단계를 실행하는 데 필요한 최소 리소스 양을 사용하기 위해 **Red Hat OpenShift Pipelines**는 가장 많은 리소스 양이 필요한 단계에 지정된 대로 최대 **CPU**, 메모리 및 임시 스토리지를 요청합니다. 이렇게 하면 모든 단계의 리소스 요구 사항이 충족됩니다. 최대 값 이외의 요청은 **0**으로 설정됩니다.

그러나 이 동작으로 인해 리소스 사용량이 필요 이상으로 증가할 수 있습니다. 리소스 할당량을 사용하는 경우 **Pod**를 예약할 수 없게 될 수도 있습니다.

예를 들어 스크립트를 사용하고 리소스 제한과 요청을 정의하지 않는 두 단계로 이루어진 작업을 살펴 보겠습니다. 결과 **pod**에는 두 개의 **init** 컨테이너(한 개는 진입점 복사용, 다른 하나는 스크립트 작성용)와 두 개의 컨테이너(단계 당 하나씩)가 있습니다.

OpenShift Container Platform은 프로젝트에 필요한 리소스 요청 및 제한을 계산하기 위해 설정된 제한 범위를 사용합니다. 이 예에서는 프로젝트에서 다음 제한 범위를 설정합니다.

```
apiVersion: v1
kind: LimitRange
metadata:
  name: mem-min-max-demo-lr
spec:
```

```
limits:
- max:
  memory: 1Gi
  min:
  memory: 500Mi
  type: Container
```

이 시나리오에서 각 **init** 컨테이너는 **1Gi**의 요청 메모리 (제한 범위의 최대 제한)를 사용하고 각 컨테이너는 **500Mi**의 요청 메모리를 사용합니다. 따라서 **Pod**의 총 메모리 요청은 **2Gi**입니다.

10단계의 작업에 동일한 제한 범위를 사용하는 경우 최종 메모리 요청은 **5Gi**로 각 단계에서 실제로 필요한 것보다 높은 **500Mi**입니다 (각 단계가 차례로 실행되기 때문).

따라서 리소스의 리소스 사용량을 줄이기 위해 다음을 수행할 수 있습니다.

- 스크립트 기능 및 동일한 이미지를 사용하여 서로 다른 단계를 한 단계로 그룹화하여 지정된 작업의 단계 수를 줄입니다. 이렇게 하면 요청된 최소 리소스가 줄어듭니다.
- 서로 상대적으로 독립적이며 자체적으로 실행할 수 있는 단계를 단일 작업 대신 여러 작업에 분산합니다. 이렇게 하면 각 작업의 단계 수가 줄어들어 각 작업에 대한 요청이 줄어들며, 리소스가 사용 가능할 때 스케줄러가 해당 단계를 실행할 수 있습니다.

4.7.3. 추가 리소스

- [리소스 할당량](#)
- [제한 범위를 사용하여 리소스 사용 제한](#)
- [Kubernetes에서 리소스 요청 및 제한](#)

4.8. 권한 있는 보안 컨텍스트에서 POD 사용

OpenShift Pipelines 1.3.x 이상 버전의 기본 구성에서는 파이프라인 실행 또는 작업 실행으로 인해 **pod**가 실행된 경우 권한 있는 보안 컨텍스트로 **Pod**를 실행할 수 없습니다. 이러한 **Pod**의 기본 서비스 계정은 **pipeline**이며 **pipelines** 서비스 계정과 연결된 **SCC**(보안 컨텍스트 제약 조건)는 **pipelines-scc**입니다. **pipelines-scc** **SCC**는 **anyuid** **SCC**와 유사하지만 파이프라인 **SCC**의 **YAML** 파일에 정의된 것과 약간의 차이가 있습니다.

SecurityContextConstraints 오브젝트 예

```

apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
...
fsGroup:
  type: MustRunAs
...

```

또한 **OpenShift Pipelines**의 일부로 제공되는 **Buildah** 클러스터 작업은 **vfs**를 기본 스토리지 드라이버로 사용합니다.

4.8.1. 파이프라인 실행 및 작업 실행 권한이 있는 보안 컨텍스트에서 Pod 실행

절차

privileged 있는 보안 컨텍스트를 사용하여 **Pod**(파이프라인 실행 또는 작업 실행)를 실행하려면 다음 수정 작업을 수행합니다.

- 명시적 **SCC**를 갖도록 연결된 사용자 계정 또는 서비스 계정을 구성합니다. 다음 방법을 사용하여 구성을 수행할 수 있습니다.
 - 다음 명령을 실행합니다.


```
$ oc adm policy add-scc-to-user <scc-name> -z <service-account-name>
```
 - 또는 **RoleBinding** 및 **Role** 또는 **ClusterRole**에 대한 **YAML** 파일을 수정합니다.

RoleBinding 오브젝트의 예

```

apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: service-account-name 1
  namespace: default

```



```

roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: pipelines-scc-clusterrole 2
subjects:
- kind: ServiceAccount
  name: pipeline
  namespace: default

```

1

적절한 서비스 계정 이름으로 바꿉니다.

2

사용하는 역할 바인딩에 따라 적절한 클러스터 역할로 바꿉니다.

ClusterRole 오브젝트의 예

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: pipelines-scc-clusterrole 1
rules:
- apiGroups:
  - security.openshift.io
  resourceNames:
  - nonroot
  resources:
  - securitycontextconstraints
  verbs:
  - use

```

1

사용하는 역할 바인딩에 따라 적절한 클러스터 역할로 바꿉니다.



참고

기본 **YAML** 파일의 복사본을 만들고 중복 파일을 변경하는 것이 좋습니다.

-

vfs 스토리지 드라이버를 사용하지 않는 경우 작업 실행 또는 파이프라인 실행과 연결된 서비스 계정을 구성하여 권한 있는 **SCC**를 구성하고 보안 컨텍스트를 **privileged: true**로 설정합니다.

4.8.2. 사용자 정의 SCC 및 사용자 정의 서비스 계정을 사용하여 파이프라인 실행 및 작업 실행

기본 **pipelines** 서비스 계정과 연결된 **pipelines-scc SCC**(보안 컨텍스트 제약 조건)를 사용하는 경우 파이프라인 실행 및 작업 실행 **Pod**가 시간 초과에 발생할 수 있습니다. 이 문제는 기본 **pipelines-scc SCC**에서 **fsGroup.type** 매개변수가 **MustRunAs** 로 설정되어 있기 때문에 발생합니다.



참고

Pod 시간 초과에 대한 자세한 내용은 [BZ#1995779](#) 에서 참조하십시오.

Pod의 시간 초과를 방지하려면 **fsGroup.type** 매개 변수를 **RunAsAny**로 설정하여 사용자 지정 **SCC**를 만들고 사용자 지정 서비스 계정과 연결할 수 있습니다.



참고

파이프라인 실행 및 작업 실행에 사용자 정의 **SCC** 및 사용자 지정 서비스 계정을 사용하는 것이 좋습니다. 이 접근 방식을 사용하면 유연성이 향상되고 업그레이드 중에 기본값을 수정할 때 실행을 중단하지 않습니다.

절차

- 1.

fsGroup.type 매개변수를 **RunAsAny** 로 설정하여 사용자 지정 **SCC**를 정의합니다.

예제: 사용자 지정 **SCC**

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  annotations:
    kubernetes.io/description: my-scc is a close replica of anyuid scc. pipelines-scc has
```

```

fsGroup - RunAsAny.
  name: my-scc
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities: null
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
groups:
- system:cluster-admins
priority: 10
readOnlyRootFilesystem: false
requiredDropCapabilities:
- MKNOD
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret

```

2.

사용자 정의 SCC를 생성합니다.

예제: my-scc SCC 생성

```
$ oc create -f my-scc.yaml
```

3.

사용자 정의 서비스 계정을 생성합니다.

예제: fsgruip-runasanv 서비스 계정 만들기

예제: `fsgroup-runasany` 서비스 계정 생성

```
$ oc create serviceaccount fsgroup-runasany
```

4.

사용자 지정 SCC를 사용자 지정 서비스 계정과 연결합니다.

예제: `my-scc` SCC를 `fsgroup-runasany` 서비스 계정과 연결합니다.

```
$ oc adm policy add-scc-to-user my-scc -z fsgroup-runasany
```

권한 있는 작업에 사용자 지정 서비스 계정을 사용하려면 다음 명령을 실행하여 권한 있는 SCC를 사용자 지정 서비스 계정과 연결할 수 있습니다.

예제: 권한이 있는 SCC를 `fsgroup-runasany` 서비스 계정과 연결합니다.

```
$ oc adm policy add-scc-to-user privileged -z fsgroup-runasany
```

5.

파이프라인 실행 및 작업 실행에서 사용자 정의 서비스 계정을 사용합니다.

예제: 파이프라인은 `fsgroup-runasany` 사용자 정의 서비스 계정을 사용하여 YAML을 실행합니다.

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: <pipeline-run-name>
spec:
```

```

pipelineRef:
  name: <pipeline-cluster-task-name>
  serviceAccountName: 'fsgroup-runasany'

```

예제: 작업 실행 **fsgroup-runasany** 사용자 정의 서비스 계정

```

apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: <task-run-name>
spec:
  taskRef:
    name: <cluster-task-name>
    serviceAccountName: 'fsgroup-runasany'

```

4.8.3. 추가 리소스

- SCC 관리에 대한 자세한 내용은 [보안 컨텍스트 제약 조건 관리](#)를 참조하십시오.

4.9. 이벤트 리스너로 WEBHOOK 보안

관리자는 이벤트 리스너를 사용하여 Webhook를 보호할 수 있습니다. 네임스페이스를 생성한 후 네임스페이스에 `operator.tekton.dev/enable-annotation=enabled` 레이블을 추가하여 Eventlistener 리소스의 HTTPS를 활성화합니다. 그런 다음 재암호화 TLS 종료를 사용하여 Trigger 리소스 및 보안 경로를 생성합니다.

Red Hat OpenShift Pipelines에서 트리거는 Eventlistener 리소스에 대한 비보안 HTTP 및 보안 HTTPS 연결을 모두 지원합니다. HTTPS는 클러스터 내부 및 외부의 연결을 보호합니다.

Red Hat OpenShift Pipelines는 네임스페이스의 레이블을 감시하는 `tekton-operator-proxy-webhook` Pod를 실행합니다. 네임스페이스에 라벨을 추가하면 Webhook에서 EventListener 오브젝트에 `service.beta.openshift.io/serving-cert-secret-name=<secret_name>` 주석을 설정합니다. 이를 통해 시크릿과 필수 인증서를 생성합니다.

```

service.beta.openshift.io/serving-cert-secret-name=<secret_name>

```

또한 생성된 시크릿을 **EventListener pod** 마운트하여 요청을 보호할 수 있습니다.

4.9.1. OpenShift 경로와의 보안 연결 제공

재암호화 TLS 종료로 경로를 생성합니다.

```
$ oc create route reencrypt --service=<svc-name> --cert=tls.crt --key=tls.key --ca-cert=ca.crt --hostname=<hostname>
```

또는 재암호화된 TLS 종료 YAML 파일을 만들어 보안 경로를 만들 수도 있습니다.

보안 경로를 생성하기 위해 TLS 종료 YAML 을 재암호화하는 예

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: route-passthrough-secured 1
spec:
  host: <hostname>
  to:
    kind: Service
    name: frontend 2
  tls:
    termination: reencrypt 3
    key: [as in edge termination]
    certificate: [as in edge termination]
    caCertificate: [as in edge termination]
    destinationCACertificate: |- 4
      -----BEGIN CERTIFICATE-----
      [...]
      -----END CERTIFICATE-----
```

1 2

63자로 제한되는 개체의 이름입니다.

3

종료 필드는 **reencrypt**로 설정됩니다. 이 필드는 유일한 필수 TLS 필드입니다.

4

이는 재암호화에 필요합니다. **destinationCACertificate** 필드는 엔드포인트 인증서의 유효성을 검사하고 라우터에서 대상 pod로의 연결을 보호합니다. 다음 시나리오 중 하나에서 이 필드를 생략할 수 있습니다.

- 서비스는 서비스 서명 인증서를 사용합니다.
- 관리자는 라우터의 기본 CA 인증서를 지정하고 서비스에는 해당 CA에서 서명한 인증서가 있습니다.

oc create route reencrypt --help 명령을 실행하여 더 많은 옵션을 표시할 수 있습니다.

4.9.2. 보안 HTTPS 연결을 사용하여 샘플 EventListener 리소스 생성

이 섹션에서는 **pipelines-tutorial** 예제를 사용하여 보안 HTTPS 연결을 사용하여 샘플 EventListener 리소스 생성을 만드는 방법을 보여줍니다.

절차

1. **pipelines-tutorial** 리포지토리에서 사용 가능한 YAML 파일에서 **TriggerBinding** 리소스를 생성합니다.

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/master/03_triggers/01_binding.yaml
```

2. **pipelines-tutorial** 리포지토리에서 직접 **TriggerTemplate** 리소스를 생성합니다.

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/master/03_triggers/02_template.yaml
```

3. **pipelines-tutorial** 리포지토리에서 직접 **Trigger** 리소스를 생성합니다.

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/master/03_triggers/03_trigger.yaml
```

4. 보안 HTTPS 연결을 사용하여 **EventListener** 리소스를 생성합니다.

a.

EventListener 리소스에 대한 보안 **HTTPS** 연결을 활성화하려면 레이블을 추가합니다.

```
$ oc label namespace <ns-name> operator.tekton.dev/enable-annotation=enabled
```

b.

pipelines-tutorial 리포지토리에서 사용 가능한 **YAML** 파일에서 **EventListener** 리소스를 생성합니다.

```
$ oc create -f https://raw.githubusercontent.com/openshift/pipelines-tutorial/master/03_triggers/04_event_listener.yaml
```

c.

재암호화 **TLS** 종료로 경로를 생성합니다.

```
$ oc create route reencrypt --service=<svc-name> --cert=tls.crt --key=tls.key --ca-cert=ca.crt --hostname=<hostname>
```

4.10. GIT 보안을 사용하여 파이프라인 인증

Git 시크릿은 **Git** 리포지토리와 안전하게 상호 작용하는 자격 증명으로 구성되며, 종종 인증을 자동화하는 데 사용됩니다. **Red Hat OpenShift Pipelines**에서는 **Git** 시크릿을 사용하여 실행 중에 **Git** 리포지토리와 상호 작용하는 파이프라인 실행 및 작업 실행을 인증할 수 있습니다.

파이프라인 실행 또는 작업 실행은 연결된 서비스 계정을 통해 시크릿에 액세스할 수 있습니다. 파이프라인은 기본 인증 및 **SSH** 기반 인증을 위해 주석(키-값 쌍)으로 **Git** 보안을 사용할 수 있도록 지원합니다.

4.10.1. 인증 정보 선택

파이프라인 실행 또는 작업 실행에는 다른 **Git** 리포지토리에 액세스하려면 여러 인증이 필요할 수 있습니다. **Pipeline**에서 인증 정보를 사용할 수 있는 도메인에 각 시크릿에 주석을 담니다.

Git 보안에 대한 인증 정보 주석 키는 **tekton.dev/git-**로 시작해야 하며 해당 값은 **Pipeline**에서 해당 인증 정보를 사용할 호스트의 **URL**입니다.

다음 예제에서 파이프라인은 사용자 이름과 암호를 사용하는 **basic-auth** 보안을 사용하여 **github.com** 및 **gitlab.com** 의 리포지토리에 액세스합니다.

예제: 기본 인증을 위한 여러 인증 정보


```

apiVersion: v1
kind: Secret
metadata:
  annotations:
    tekton.dev/git-0: github.com
    tekton.dev/git-1: gitlab.com
type: kubernetes.io/basic-auth
stringData:
  username: 1
  password: 2

```

1

리포지토리의 사용자 이름

2

리포지토리의 암호 또는 개인 액세스 토큰

ssh-auth 시크릿(개인 키)을 사용하여 **Git** 리포지토리에 액세스할 수도 있습니다.

예제: **SSH** 기반 인증을 위한 개인 키

```

apiVersion: v1
kind: Secret
metadata:
  annotations:
    tekton.dev/git-0: https://github.com
type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: 1

```

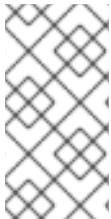
1

SSH 개인 키 문자열이 포함된 파일의 이름입니다.

4.10.2. Git에 대한 기본 인증 구성

암호로 보호된 리포지토리에서 리소스를 검색하려면 파이프라인에 대한 기본 인증을 구성해야 합니다.

파이프라인에 대한 기본 인증을 구성하려면 지정된 리포지토리의 **Git** 시크릿에서 인증 정보로 **secret .yaml, serviceaccount.yaml**을 업데이트하고 **.yaml** 파일을 실행합니다. 이 프로세스를 완료하면 **Pipeline**에서 해당 정보를 사용하여 지정된 파이프라인 리소스를 검색할 수 있습니다.



참고

GitHub의 경우 일반 암호를 사용한 인증이 더 이상 사용되지 않습니다. 대신 **개인 액세스 토큰**을 사용합니다.

절차

1. **secret.yaml** 파일에서 사용자 이름 및 암호 또는 **GitHub 개인 액세스 토큰**을 지정하여 대상 **Git** 리포지토리에 액세스합니다.

```

apiVersion: v1
kind: Secret
metadata:
  name: basic-user-pass 1
  annotations:
    tekton.dev/git-0: https://github.com
  type: kubernetes.io/basic-auth
stringData:
  username: 2
  password: 3

```

1

보안의 이름입니다. 예에서는 **basic-user-pass**입니다.

2

Git 리포지토리의 사용자 이름.

3

Git 리포지토리의 암호입니다.

2.

`serviceaccount.yaml` 파일에서 시크릿을 적절한 서비스 계정과 연결합니다.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: build-bot ①
secrets:
  - name: basic-user-pass ②
```

①

서비스 계정 이름입니다. 예에서는 **build-bot** 입니다.

②

보안의 이름입니다. 예에서는 **basic-user-pass** 입니다.

3.

`run.yaml` 파일에서 서비스 계정을 작업 실행 또는 파이프라인 실행과 연결합니다.

•

서비스 계정을 작업 실행과 연결합니다.

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: build-push-task-run-2 ①
spec:
  serviceAccountName: build-bot ②
  taskRef:
    name: build-push ③
```

①

작업 실행의 이름입니다. 예에서는 **build-push-task-run-2** 입니다.

②

서비스 계정 이름입니다. 예에서는 **build-bot** 입니다.

③

작업 이름입니다. 예에서는 **build-push** 입니다.

- 서비스 계정을 **PipelineRun** 리소스와 연결합니다.

```

apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  name: demo-pipeline 1
  namespace: default
spec:
  serviceAccountName: build-bot 2
  pipelineRef:
    name: demo-pipeline 3

```

1

파이프라인 실행의 이름입니다. 예에서는 **demo-pipeline** 입니다.

2

서비스 계정 이름입니다. 예에서는 **build-bot** 입니다.

3

파이프라인의 이름입니다. 예에서는 **demo-pipeline** 입니다.

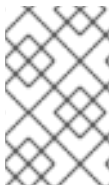
4. 변경 사항을 적용합니다.

```
$ kubectl apply --filename secret.yaml,serviceaccount.yaml,run.yaml
```

4.10.3. Git에 대한 SSH 인증 구성

SSH 키로 구성된 리포지토리에서 리소스를 검색하려면 해당 파이프라인에 대한 SSH 기반 인증을 구성해야 합니다.

파이프라인에 대한 SSH 기반 인증을 구성하려면 지정된 리포지토리의 SSH 개인 키에서 인증 정보를 사용하여 **secret.yaml,serviceaccount.yaml**을 업데이트하고 **.yaml** 파일을 실행합니다. 이 프로세스를 완료하면 Pipeline에서 해당 정보를 사용하여 지정된 파이프라인 리소스를 검색할 수 있습니다.



참고

기본 인증이 아닌 SSH 기반 인증을 사용하는 것이 좋습니다.

절차

1. **SSH 개인 키**를 생성하거나 일반적으로 `~/.ssh/id_rsa` 파일에서 사용할 수 있는 기존 개인 키를 복사합니다.
2. `secret.yaml` 파일에서 `ssh-privatekey` 값을 SSH 개인 키 파일의 이름으로 설정하고 `known_hosts` 값을 알려진 호스트 파일의 이름으로 설정합니다.

```

apiVersion: v1
kind: Secret
metadata:
  name: ssh-key ①
  annotations:
    tekton.dev/git-0: github.com
  type: kubernetes.io/ssh-auth
stringData:
  ssh-privatekey: ②
  known_hosts: ③

```

①

SSH 개인 키가 포함된 시크릿의 이름입니다. 예에서는 `ssh-key` 입니다.

②

SSH 개인 키 문자열이 포함된 파일의 이름입니다.

③

알려진 호스트 목록을 포함하는 파일의 이름입니다.

경고

개인 키를 생략하면 Pipeline에서 모든 서버의 공개 키를 허용합니다.

3. 선택 사항: 사용자 지정 SSH 포트를 지정하려면 주석 값 끝에 `:<port number>` 를 추가합니다. 예를 들면 `tekton.dev/git-0: github.com:2222` 입니다.
4. `serviceaccount.yaml` 파일에서 `ssh-key` 시크릿을 `build-bot` 서비스 계정과 연결합니다.

```

apiVersion: v1

```

```
kind: ServiceAccount
metadata:
  name: build-bot 1
secrets:
  - name: ssh-key 2
```

1

서비스 계정 이름입니다. 예에서는 **build-bot** 입니다.

2

SSH 개인 키가 포함된 시크릿의 이름입니다. 예에서는 **ssh-key** 입니다.

5.

run.yaml 파일에서 서비스 계정을 작업 실행 또는 파이프라인 실행과 연결합니다.

-

서비스 계정을 작업 실행과 연결합니다.

```
apiVersion: tekton.dev/v1beta1
kind: TaskRun
metadata:
  name: build-push-task-run-2 1
spec:
  serviceAccountName: build-bot 2
  taskRef:
    name: build-push 3
```

1

작업 실행의 이름입니다. 예에서는 **build-push-task-run-2** 입니다.

2

서비스 계정 이름입니다. 예에서는 **build-bot** 입니다.

3

작업 이름입니다. 예에서는 **build-push** 입니다.

-

서비스 계정을 파이프라인 실행과 연결합니다.

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
```

```

metadata:
  name: demo-pipeline ❶
  namespace: default
spec:
  serviceAccountName: build-bot ❷
  pipelineRef:
    name: demo-pipeline ❸

```

❶

파이프라인 실행의 이름입니다. 예에서는 **demo-pipeline** 입니다.

❷

서비스 계정 이름입니다. 예에서는 **build-bot** 입니다.

❸

파이프라인의 이름입니다. 예에서는 **demo-pipeline** 입니다.

6.

변경 사항을 적용합니다.

```
$ kubectl apply --filename secret.yaml,serviceaccount.yaml,run.yaml
```

4.10.4. Git 유형 작업에서 SSH 인증 사용

Git 명령을 호출할 때 작업 단계에서 직접 SSH 인증을 사용할 수 있습니다. SSH 인증은 \$HOME 변수를 무시하고 /etc/passwd 파일에 지정된 사용자의 홈 디렉토리만 사용합니다. 따라서 작업의 각 단계에서 /tekton/home/.ssh 디렉토리를 연결된 사용자의 홈 디렉토리로 심볼릭해야 합니다.

그러나 git 유형의 파이프라인 리소스를 사용하거나 Tekton 카탈로그에서 사용할 수 있는 git-clone 작업을 사용하는 경우에는 명시적 심볼릭 링크가 필요하지 않습니다.

git 유형 작업에서 SSH 인증을 사용하는 예제는 [authenticateing-git-commands.yaml](#) 을 참조하십시오.

4.10.5. root가 아닌 사용자로 보안 사용

다음과 같이 특정 시나리오에서 루트가 아닌 사용자로 시크릿을 사용해야 할 수 있습니다.

- 컨테이너가 실행을 실행하는 데 사용하는 사용자 및 그룹은 플랫폼에 의해 무작위화됩니다.
- 작업의 단계에서는 루트가 아닌 보안 컨텍스트를 정의합니다.
- 작업은 작업의 모든 단계에 적용되는 글로벌 루트가 아닌 보안 컨텍스트를 지정합니다.

이러한 시나리오에서는 작업 실행 및 파이프라인 실행에 대한 다음 측면을 루트가 아닌 사용자로 고려합니다.

- **Git**에 대한 **SSH** 인증에는 사용자에게 `/etc/passwd` 디렉터리에 구성된 유효한 홈 디렉터리가 있어야 합니다. 유효한 홈 디렉터리가 없는 **UID**를 지정하면 인증에 실패합니다.
- **SSH** 인증은 `$HOME` 환경 변수를 무시합니다. 따라서 **Pipeline(/tekton/home)**에서 루트가 아닌 사용자의 유효한 홈 디렉터리로 정의된 `$HOME` 디렉토리에서 적절한 시크릿 파일을 가져와야 합니다.

또한 루트가 아닌 보안 컨텍스트에서 **SSH** 인증을 구성하려면 [git 명령을 인증하는 예를 참조하십시오](#).

4.10.6. 특정 단계에 대한 보안 액세스 제한

기본적으로 파이프라인의 시크릿은 `$HOME/tekton/home` 디렉터리에 저장되며 작업의 모든 단계에서 사용할 수 있습니다.

특정 단계로 보안을 제한하려면 **secret** 정의를 사용하여 볼륨을 지정하고 특정 단계에서 볼륨을 마운트합니다.

4.11. OPENSIFT LOGGING OPERATOR를 사용하여 파이프라인 로그 보기

파이프라인 실행, 작업 실행, 이벤트 리스너로 생성된 로그는 해당 **Pod**에 저장됩니다. 문제 해결 및 감사를 위해 로그를 검토하고 분석하는 것이 유용합니다.

그러나 **Pod**를 무기한 유지하면 불필요한 리소스 소비 및 복잡한 네임스페이스가 발생합니다.

Pod에 대한 종속성을 제거하여 파이프라인 로그를 볼 수 있습니다. **OpenShift Elasticsearch Operator** 및 **OpenShift Logging Operator**를 사용하면 됩니다. 이러한 **Operator**는 로그가 포함된 Pod를 삭제한 후에도 **Elasticsearch Kibana** 스택을 사용하여 파이프라인 로그를 확인하는 데 도움이 됩니다.

4.11.1. 사전 요구 사항

Kibana 대시보드에서 파이프라인 로그를 보기 전에 다음을 확인하십시오.

- 단계는 클러스터 관리자가 수행합니다.
- 파이프라인 실행 및 작업 실행에 대한 로그를 사용할 수 있습니다.
- **OpenShift Elasticsearch Operator** 및 **OpenShift Logging Operator**가 설치되어 있습니다.

4.11.2. Kibana에서 파이프라인 로그 보기

Kibana 웹 콘솔에서 파이프라인 로그를 보려면 다음을 수행합니다.

절차

1. 클러스터 관리자로 **OpenShift Container Platform** 웹 콘솔에 로그인합니다.
2. 메뉴 표시줄의 오른쪽 상단에서 **grid** 아이콘 → **Observability** → **Logging** 을 클릭합니다. **Kibana** 웹 콘솔이 표시됩니다.
3. 인덱스 패턴을 생성합니다.
 - a. **Kibana** 웹 콘솔의 왼쪽 탐색 패널에서 관리를 클릭합니다.
 - b. 인덱스 패턴 생성을 클릭합니다.
 - c. 2단계 중 1단계에서 2단계 인덱스 패턴 → 인덱스 패턴을 정의하고 * 패턴을 입력하고

다음 단계를 클릭합니다.

- d. 다음 2단계: 설정 구성 → 시간 필터 필드 이름, 드롭다운 메뉴에서 @timestamp 를 선택한 다음 인덱스 패턴 생성을 클릭합니다.

4. 필터를 추가합니다.

- a. Kibana 웹 콘솔의 왼쪽 탐색 패널에서 Discover(검색)를 클릭합니다.

- b. 필터 추가 → 쿼리 DSL 편집을 클릭합니다.



참고

- 다음 예제 필터 각각에 대해 쿼리를 편집하고 Save(저장)를 클릭합니다.
- 필터는 차례로 적용됩니다.

- i. 파이프라인과 관련된 컨테이너를 필터링합니다.

파이프라인 컨테이너를 필터링하는 쿼리의 예

```
{
  "query": {
    "match": {
      "kubernetes.flat_labels": {
        "query": "app_kubernetes_io/managed-by=tekton-pipelines",
        "type": "phrase"
      }
    }
  }
}
```

- ii.

place-tools 컨테이너가 아닌 모든 컨테이너를 필터링합니다. 쿼리 DSL을 편집하는 대신 그래픽 드롭다운 메뉴를 사용하는 의 그림으로 다음 접근 방식을 고려하십시오.

그림 4.6. 드롭다운 필드를 사용한 필터링 예

iii.

강조 표시를 위해 라벨에서 **pipelinerun** 을 필터링합니다.

강조 표시를 위한 라벨에서 **pipelinerun** 을 필터링하는 쿼리의 예

```
{
  "query": {
    "match": {
      "kubernetes.flat_labels": {
        "query": "tekton_dev/pipelineRun=",
        "type": "phrase"
      }
    }
  }
}
```

iv.

강조 표시를 위해 라벨에서 파이프라인을 필터링합니다.

강조 표시를 위한 라벨에서 파이프라인을 필터링하는 쿼리의 예

```
{
  "query": {
    "match": {
      "kubernetes.flat_labels": {
        "query": "tekton_dev/pipeline=",
        "type": "phrase"
      }
    }
  }
}
```

c.

Available field(사용 가능한 필드) 목록에서 다음 필드를 선택합니다.

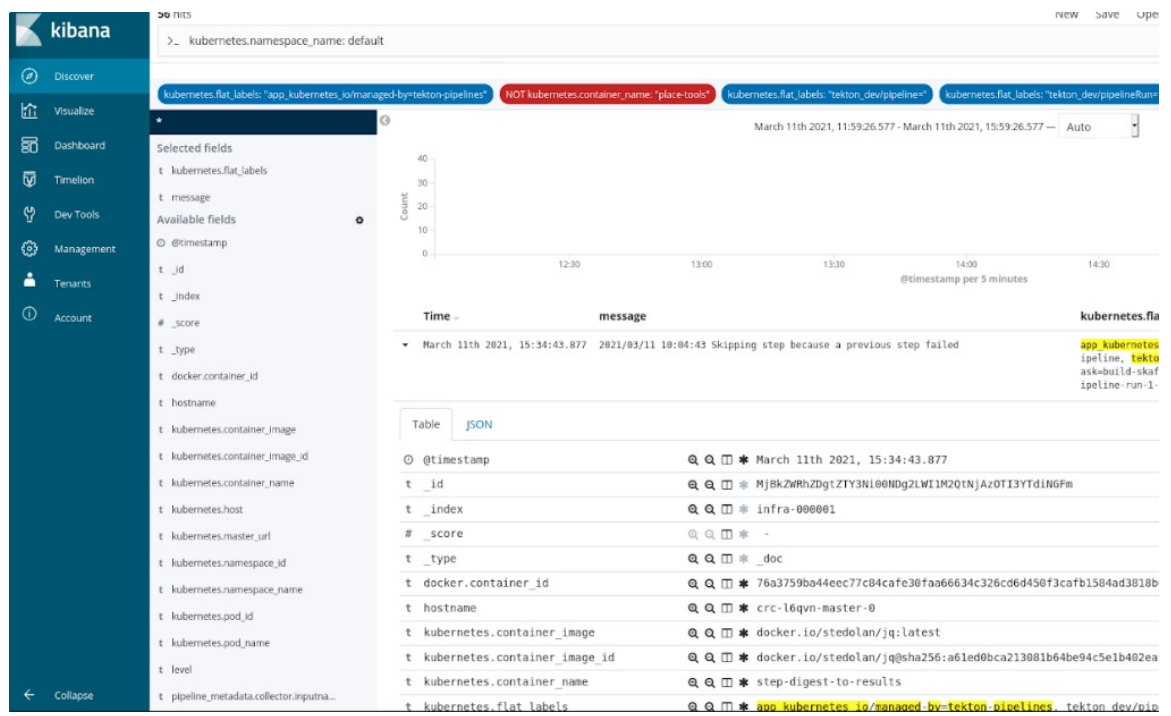
- **kubernetes.flat_labels**
- **message**

선택한 필드가 **Selected field**(선택한 필드) 목록에 표시되는지 확인합니다.

d.

로그는 메시지 필드에 표시됩니다.

그림 4.7. 필터링된 메시지



4.11.3. 추가 리소스

- [OpenShift Logging 설치](#)
- [리소스의 로그 보기](#)
- [Kibana를 사용하여 클러스터 로그 보기](#)

5장. GITOPS

5.1. RED HAT OPENSIFT GITOPS 릴리스 정보

Red Hat OpenShift GitOps는 클라우드 네이티브 애플리케이션에 대한 연속 배포를 구현하는 선언적 방법입니다. Red Hat OpenShift GitOps를 사용하면 개발, 스테이징, 프로덕션과 같은 다양한 환경의 다양한 클러스터에 애플리케이션을 배포할 때 애플리케이션의 일관성을 유지할 수 있습니다. Red Hat OpenShift GitOps는 다음 작업을 자동화하는 데 도움이 됩니다.

- 클러스터의 구성, 모니터링, 스토리지 상태가 비슷한지 확인
- 알려진 상태에서 클러스터 복구 또는 재생성
- 여러 OpenShift Container Platform 클러스터에 구성 변경 사항 적용 또는 되돌리기
- 템플릿 구성을 다른 환경과 연결
- 스테이징에서 프로덕션까지 클러스터 전체에서 애플리케이션 승격

Red Hat OpenShift GitOps 개요는 [OpenShift GitOps 이해](#)를 참조하십시오.

5.1.1. 호환성 및 지원 매트릭스

이 릴리스의 일부 기능은 현재 [기술 프리뷰](#) 단계에 있습니다. 이러한 실험적 기능은 프로덕션용이 아닙니다.

아래 표에서 기능은 다음과 같은 상태로 표시되어 있습니다.

- **TP:** 기술 프리뷰
- **GA:** 정식 출시일 (GA)

OpenShift GitOps							OpenShift Versions	지원 상태
구성 요소 버전	OpenShift	GitOps	OpenShift	GitOps	OpenShift	GitOps	OpenShift	GitOps
버전	kam	Helm	Kustomize	Argo CD	Applicat ionSet	DEX		
1.6.0	0.0.46 TP	3.8.1 GA	4.4.1 GA	2.4.5 GA	2.4.5 GA	2.30.3 GA	4.8-4.11	GA
1.5.0	0.0.42 TP	3.8.0 GA	4.4.1 GA	2.3.3 GA	0.4.1 TP	2.30.3 GA	4.8-4.11	GA
1.4.0	0.0.41 TP	3.7.1 GA	4.2.0 GA	2.2.2 GA	0.2.0 TP	2.30.0 GA	4.7-4.11	GA
1.3.0	0.0.40	3.6.0	4.2.0	2.1.2	0.2.0	2.28.0	4.7-4.9	GA
1.2.0	0.0.38	3.5.0	3.9.4	2.0.5	0.1.0	해당 없 음	4.8	GA
1.1.0	0.0.32	3.5.0	3.9.4	2.0.0	해당 없 음	해당 없 음	4.7	GA



참고

OpenShift Container Platform 4.6에서 Red Hat OpenShift GitOps v1.3을 사용하려면 Red Hat OpenShift GitOps를 새로 설치하십시오. OpenShift 4.6의 Red Hat OpenShift GitOps 1.0(TP)의 업그레이드 경로는 없습니다.

5.1.2. 보다 포괄적 수용을 위한 오픈 소스 용어 교체

Red Hat은 코드, 문서, 웹 속성에서 문제가 있는 용어를 교체하기 위해 최선을 다하고 있습니다. 먼저 마스터(master), 슬레이브(slave), 블랙리스트(blacklist), 화이트리스트(whitelist) 등 네 가지 용어를 교체하고 있습니다. 이러한 변경 작업은 작업 범위가 크므로 향후 여러 릴리스에 걸쳐 점차 구현할 예정입니다. 자세한 내용은 [CTO Chris Wright의 메시지](#)를 참조하십시오.

5.1.3. Red Hat OpenShift GitOps 1.6.6 릴리스 노트

Red Hat OpenShift GitOps 1.6.6은 이제 OpenShift Container Platform 4.8, 4.9, 4.10 및 4.11에서 사용할 수 있습니다.

5.1.3.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 **v0.5.0**부터 모든 **Argo CD Operator** 버전이 정보 공개 취약점에 취약했습니다. 결과적으로 권한이 없는 사용자가 **API** 오류 메시지를 검사하여 애플리케이션 이름을 열거하고 검색된 애플리케이션 이름을 다른 공격의 시작점으로 사용할 수 있습니다. 예를 들어 공격자는 애플리케이션 이름에 대한 지식을 사용하여 관리자에게 더 높은 권한을 부여하도록 유도할 수 있습니다. 이번 업데이트에서는 **CVE-2022-41354** 오류가 수정되었습니다. [GITOPS-2635](#), [CVE-2022-41354](#)

5.1.4. Red Hat OpenShift GitOps 1.6.7 릴리스 노트

Red Hat OpenShift GitOps 1.6.7은 이제 **OpenShift Container Platform 4.8, 4.9, 4.10** 및 **4.11**에서 사용할 수 있습니다.

5.1.4.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 **v0.5.0**부터 모든 **Argo CD Operator** 버전이 정보 공개 취약점에 취약했습니다. 결과적으로 권한이 없는 사용자가 **API** 오류 메시지를 검사하여 애플리케이션 이름을 열거하고 검색된 애플리케이션 이름을 다른 공격의 시작점으로 사용할 수 있습니다. 예를 들어 공격자는 애플리케이션 이름에 대한 지식을 사용하여 관리자에게 더 높은 권한을 부여하도록 유도할 수 있습니다. 이번 업데이트에서는 **CVE-2022-41354** 오류가 수정되었습니다. [GITOPS-2635](#), [CVE-2022-41354](#)

5.1.5. Red Hat OpenShift GitOps 1.6.2 릴리스 노트

Red Hat OpenShift GitOps 1.6.2는 이제 **OpenShift Container Platform 4.8, 4.9, 4.10** 및 **4.11**에서 사용할 수 있습니다.

5.1.5.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 프로젝트에 **Operator 5개 이상**을 설치할 때 누락된 **InstallPlan**에 대한 서브스크립션 상태 점검이 성능이 저하 되었습니다. 이번 업데이트에서는 이 문제가 해결되었습니다. [GITOPS-2018](#)
-

이번 업데이트 이전에는 **Red Hat OpenShift GitOps Operator**에서 **Argo CD** 인스턴스가 더 이상 사용되지 않는 필드를 사용했음을 탐지할 때마다 사용 중단 알림 경고와 함께 클러스터를 스캔했습니다. 이번 업데이트에서는 이 문제를 해결했으며 필드를 감지하는 각 인스턴스에 대해 하나의 경고 이벤트만 표시합니다. [GITOPS-2230](#)

•

OpenShift Container Platform 4.12에서는 콘솔을 설치하는 것이 선택 사항입니다. 이번 수정을 통해 콘솔이 설치되지 않은 경우 **Operator**에 오류가 발생하지 않도록 **Red Hat OpenShift GitOps Operator**가 업데이트되었습니다. [GITOPS-2352](#)

5.1.6. Red Hat OpenShift GitOps 1.6.1 릴리스 노트

Red Hat OpenShift GitOps 1.6.1은 이제 **OpenShift Container Platform 4.8** 이상에서 사용할 수 있습니다.

5.1.6.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

•

이번 업데이트 이전에는 다수의 애플리케이션이 활성 프로브의 응답하지 않음으로 인해 애플리케이션 컨트롤러가 여러 번 다시 시작되었습니다. 이번 업데이트에서는 애플리케이션 컨트롤러 **StatefulSet** 오브젝트에서 활성 프로브를 제거하여 문제를 해결합니다. [GITOPS-2153](#)

•

이번 업데이트 이전에는 인증 기관에서 서명하지 않은 인증서로 설정되는 경우 **RHSSO** 인증서를 검증할 수 없습니다. 이번 업데이트에서는 이 문제를 수정하고 이제 통신할 때 **Keycloak**의 **TLS** 인증서를 확인하는 데 사용할 사용자 정의 인증서를 제공할 수 있습니다. **rootCA**를 **Argo CD** 사용자 정의 리소스 **.spec.keycloak.rootCA** 필드에 추가할 수 있습니다. **Operator**는 이 변경 사항을 조정하고 **argocd-cm ConfigMap**의 **oidc.config** 필드를 **PEM** 인코딩 루트 인증서로 업데이트합니다. [GITOPS-2214](#)



참고

.spec.keycloak.rootCA 필드를 업데이트한 후 **Argo CD** 서버 **Pod**를 다시 시작하십시오.

예를 들면 다음과 같습니다.

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
```

```

name: example-argocd
labels:
  example: basic
spec:
  sso:
    provider: keycloak
    keycloak:
      rootCA: |
        ---- BEGIN CERTIFICATE ----
        This is a dummy certificate
        Please place this section with appropriate rootCA
        ---- END CERTIFICATE ----
  server:
    route:
      enabled: true

```

- 이번 업데이트 이전에는 **Argo CD**에서 관리하는 종료 네임스페이스가 다른 관리 네임스페이스의 역할 및 기타 구성 생성을 차단합니다. 이번 업데이트에서는 이 문제가 해결되었습니다. [GITOPS-2277](#)
- 이번 업데이트 이전에는 **anyuid**의 **SCC**가 **Dex ServiceAccount** 리소스에 할당되면 **Dex Pod**가 **CreateContainerConfigError**로 시작하지 못했습니다. 이번 업데이트에서는 기본 사용자 **ID**를 **Dex** 컨테이너에 할당하여 이 문제를 해결합니다. [GITOPS-2235](#)

5.1.7. Red Hat OpenShift GitOps 1.6.0 릴리스 노트

Red Hat OpenShift GitOps 1.6.0은 이제 OpenShift Container Platform 4.8 이상에서 사용할 수 있습니다.

5.1.7.1. 새로운 기능

현재 릴리스에서는 다음과 같은 개선 사항이 추가되었습니다.

- 이전에는 **Argo CD ApplicationSet** 컨트롤러가 **TP(기술 프리뷰)** 기능이었습니다. 이번 업데이트를 통해 **GA(General Availability)** 기능입니다. [GITOPS-1958](#)
- 이번 업데이트를 통해 **Red Hat OpenShift GitOps**의 최신 릴리스는 최신 버전 기반 채널에서 사용할 수 있습니다. 이러한 업그레이드를 가져오려면 **Subscription** 오브젝트 **YAML** 파일에서 **channel** 매개변수를 업데이트합니다. 값을 **stable**에서 최신 또는 **gitops-1.6**과 같은 버전 기반 채널로 변경합니다. [GITOPS-1791](#)
- 이번 업데이트를 통해 **keycloak** 구성을 제어하는 **spec.sso** 필드의 매개변수가

`.spec.sso.keycloak` 로 이동합니다. `.spec.dex` 필드의 매개변수가 `.spec.sso.dex` 에 추가되었습니다. `.spec.sso.provider` 를 사용하여 `Dex`를 활성화하거나 비활성화합니다. `.spec.dex` 매개변수는 더 이상 사용되지 않으며 `keycloak` 구성에 대한 `DISABLE_DEX` 및 `.spec.sso` 필드와 함께 버전 1.9에서 제거될 예정입니다. [GITOPS-1983](#)

이번 업데이트를 통해 `Argo CD` 알림 컨트롤러는 `Argo CD` 사용자 정의 리소스 정의에서 `.spec.notifications.enabled` 매개변수를 사용하여 활성화하거나 비활성화할 수 있는 선택적 워크로드입니다. `Argo CD` 알림 컨트롤러는 기술 프리뷰 기능입니다. [GITOPS-1917](#)

중요

`Argo CD` 알림 컨트롤러는 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 <https://access.redhat.com/support/offerings/techpreview/>를 참조하십시오.

이번 업데이트를 통해 `Tekton` 파이프라인 실행 및 작업 실행에 대한 리소스 제외가 기본적으로 추가됩니다. `Argo CD`는 기본적으로 이러한 리소스를 정리합니다. 이러한 리소스 제외는 `OpenShift Container Platform`에서 생성된 새로운 `Argo CD` 인스턴스에 추가됩니다. `CLI`에서 인스턴스가 생성되는 경우 리소스가 추가되지 않습니다. [GITOPS-1876](#)

이번 업데이트를 통해 `Argo CD Operand`의 사용자 정의 리소스 정의에서 `resourcetracingMethod` 매개변수를 설정하여 `Argo CD CD`에서 사용하는 추적 방법을 선택할 수 있습니다. [GITOPS-1862](#)

이번 업데이트를 통해 `Red Hat OpenShift GitOps Argo CD` 사용자 정의 리소스의 `extraConfig` 필드를 사용하여 `argocd-cm configMap`에 항목을 추가할 수 있습니다. 지정된 항목은 검증 없이 라이브 `config-cm configMap`으로 조정됩니다. [GITOPS-1964](#)

이번 업데이트를 통해 `OpenShift Container Platform 4.11` 이상에서는 `Red Hat OpenShift GitOps` 개발자 화면의 `Red Hat OpenShift GitOps` 환경 세부 정보 페이지에 각 배포 버전에 대한 링크와 함께 애플리케이션 환경 배포 기록이 표시됩니다. [GITOPS-1269](#)

이번 업데이트를 통해 `Operator`에서 템플릿 리소스 또는 "소스"로 사용되는 `Argo CD`를 사용하여 리소스를 관리할 수 있습니다. [GITOPS-982](#)

- 이번 업데이트를 통해 **Operator**는 **Kubernetes 1.24**에서 활성화된 **Pod** 보안 승인을 충족하기 위해 올바른 권한으로 **Argo CD** 워크로드를 구성합니다. [GITOPS-2026](#)
- 이번 업데이트를 통해 **Config Management Plugins 2.0**이 지원됩니다. **Argo CD** 사용자 정의 리소스를 사용하여 리포지토리 서버의 사이드바 컨테이너를 지정할 수 있습니다. [GITOPS-776](#)
- 이번 업데이트를 통해 **Argo CD** 구성 요소와 **Redis** 캐시 간의 모든 통신이 최신 **TLS** 암호화를 사용하여 보호됩니다. [GITOPS-720](#)
- 이번 **Red Hat OpenShift GitOps** 릴리스에는 **IBM Z** 및 **IBM Power on OpenShift Container Platform 4.10** 지원이 추가되었습니다. 제한된 환경에서의 설치에 **IBM Z** 및 **IBM Power**에서 지원되지 않습니다.

5.1.7.2. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 **south :serviceaccount:argocd:gitops-argocd-application-controller** 컨트롤러가 네임스페이스의 **monitoring.coreos.com** API 그룹에 **"prometheusrules"** 리소스를 생성하지 않았습니다. 이번 업데이트에서는 이 문제가 해결되어 **Red Hat OpenShift GitOps**가 **monitoring.coreos.com** API 그룹의 모든 리소스를 관리할 수 있습니다. [GITOPS-1638](#)
- 이번 업데이트 이전에는 클러스터 권한을 재조정하는 동안 시크릿이 클러스터 구성 인스턴스에 속하는 경우 삭제되었습니다. 이번 업데이트에서는 이 문제가 해결되었습니다. 이제 시크릿 대신 시크릿의 **namespaces** 필드가 삭제됩니다. [GITOPS-1777](#)
- 이번 업데이트 이전에는 **Operator**를 통해 **Argo CD**의 **HA** 변형을 설치한 경우 **Operator**에서 **podAntiAffinity** 규칙 대신 **podAffinity** 규칙을 사용하여 **Redis StatefulSet** 오브젝트를 생성했습니다. 이번 업데이트에서는 이 문제가 해결되었습니다. 이제 **Operator**에서 **podAntiAffinity** 규칙과 **Redis StatefulSet**를 생성합니다. [GITOPS-1645](#)
- 이번 업데이트 이전에는 **Argo CD ApplicationSet**에 너무 많은 **ssh Zombie** 프로세스가 있었습니다. 이번 업데이트에서는 이 문제가 해결되었습니다. 이는 프로세스를 생성하고 좀비를 **ApplicationSet** 컨트롤러에 생성하는 간단한 **init** 데몬인 **tini**를 추가합니다. 이렇게 하면 **SIGTERM** 신호가 실행 중인 프로세스에 올바르게 전달되어 좀비 프로세스가 되지 않습니다. [GITOPS-2108](#)

5.1.7.3. 확인된 문제

- **Red Hat OpenShift GitOps Operator**는 **Dex** 외에도 **OIDC**를 통해 **RHSSO(Keycloak)**를 사용할 수 있습니다. 그러나 최근 보안 수정 사항이 적용된 경우 일부 시나리오에서는 **RHSSO** 인증서를 검증할 수 없습니다. [GITOPS-2214](#)

이 문제를 해결하려면 **ArgoCD** 사양에서 **OIDC(Keycloak/RHSSO)** 엔드포인트에 대한 **TLS** 검증을 비활성화합니다.

```
spec:
  extraConfig:
    oidc.tls.insecure.skip.verify: "true"
  ...
```

5.1.8. Red Hat OpenShift GitOps 1.5.7 릴리스 노트

Red Hat OpenShift GitOps 1.5.7은 이제 **OpenShift Container Platform 4.8, 4.9, 4.10** 및 **4.11**에서 사용할 수 있습니다.

5.1.8.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- **OpenShift Container Platform 4.12**에서는 콘솔을 설치하는 것이 선택 사항입니다. 이번 수정을 통해 콘솔이 설치되지 않은 경우 **Operator**에 오류가 발생하지 않도록 **Red Hat OpenShift GitOps Operator**가 업데이트되었습니다. [GITOPS-2353](#)

5.1.9. Red Hat OpenShift GitOps 1.5.6 릴리스 노트

Red Hat OpenShift GitOps 1.5.6은 이제 **OpenShift Container Platform 4.8** 이상에서 사용할 수 있습니다.

5.1.9.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 다수의 애플리케이션이 활성 프로브의 응답하지 않음으로 인해 애플리케이션 컨트롤러가 여러 번 다시 시작되었습니다. 이번 업데이트에서는 애플리케이션 컨트롤러 **StatefulSet** 오브젝트에서 활성 프로브를 제거하여 문제를 해결합니다. [GITOPS-2153](#)

- 이번 업데이트 이전에는 인증 기관에서 서명하지 않은 인증서로 설정되는 경우 **RHSSO** 인증서를 검증할 수 없습니다. 이번 업데이트에서는 이 문제를 수정하고 이제 통신할 때 **Keycloak**의 **TLS** 인증서를 확인하는 데 사용할 사용자 정의 인증서를 제공할 수 있습니다. **rootCA**를 **Argo CD** 사용자 정의 리소스 **.spec.keycloak.rootCA** 필드에 추가할 수 있습니다. **Operator**는 이 변경 사항을 조정하고 **argocd-cm ConfigMap**의 **oidc.config** 필드를 **PEM** 인코딩 루트 인증서로 업데이트합니다. [GITOPS-2214](#)



참고

.spec.keycloak.rootCA 필드를 업데이트한 후 **Argo CD** 서버 **Pod**를 다시 시작하십시오.

예를 들면 다음과 같습니다.

```

apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  sso:
    provider: keycloak
    keycloak:
      rootCA: |
        ---- BEGIN CERTIFICATE ----
        This is a dummy certificate
        Please place this section with appropriate rootCA
        ---- END CERTIFICATE ----
  server:
    route:
      enabled: true

```

- 이번 업데이트 이전에는 **Argo CD**에서 관리하는 종료 네임스페이스가 다른 관리 네임스페이스의 역할 및 기타 구성 생성을 차단합니다. 이번 업데이트에서는 이 문제가 해결되었습니다. [GITOPS-2277](#)

- 이번 업데이트 이전에는 **anyuid**의 **SCC**가 **Dex ServiceAccount** 리소스에 할당되면 **Dex Pod**가 **CreateContainerConfigError**로 시작하지 못했습니다. 이번 업데이트에서는 기본 사용자 **ID**를 **Dex** 컨테이너에 할당하여 이 문제를 해결합니다. [GITOPS-2235](#)

5.1.10. Red Hat OpenShift GitOps 1.5.5 릴리스 노트

Red Hat OpenShift GitOps 1.5.5는 이제 OpenShift Container Platform 4.8 이상에서 사용할 수 있습니다.

5.1.10.1. 새로운 기능

현재 릴리스에서는 다음과 같은 개선 사항이 추가되었습니다.

- 이번 업데이트를 통해 번들된 **Argo CD**가 **2.3.7** 버전으로 업데이트되었습니다.

5.1.10.2. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 클러스터에 더 제한적인 **SCC**가 있는 경우 **ArgoCD** 인스턴스의 **redis-haproxy Pod**가 실패했습니다. 이번 업데이트에서는 워크로드의 보안 컨텍스트를 업데이트하여 문제를 해결합니다. [GITOPS-2034](#)

5.1.10.3. 확인된 문제

- **Red Hat OpenShift GitOps Operator**는 **OIDC** 및 **Dex**와 함께 **RHSSO(KeyCloak)**를 사용할 수 있습니다. 그러나 최근 보안 수정 사항이 적용된 경우 **Operator**는 일부 시나리오에서 **RHSSO** 인증서를 검증할 수 없습니다. [GITOPS-2214](#)

이 문제를 해결하려면 **ArgoCD** 사양에서 **OIDC(Keycloak/RHSSO)** 엔드포인트에 대한 **TLS** 검증을 비활성화합니다.

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
spec:
  extraConfig:
    "admin.enabled": "true"
  ...
```

5.1.11. Red Hat OpenShift GitOps 1.5.4 릴리스 노트

Red Hat OpenShift GitOps 1.5.4는 이제 OpenShift Container Platform 4.8 이상에서 사용할 수 있습니다.

5.1.11.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 **Red Hat OpenShift GitOps**에서 이전 버전의 **REDIS 5** 이미지 태그를 사용하고 있었습니다. 이번 업데이트에서는 문제를 해결하고 **rhel8/redis-5** 이미지 태그를 업그레이드합니다. [GITOPS-2037](#)

5.1.12. Red Hat OpenShift GitOps 1.5.3 릴리스 노트

Red Hat OpenShift GitOps 1.5.3은 이제 **OpenShift Container Platform 4.8** 이상에서 사용할 수 있습니다.

5.1.12.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 **Argo CD v1.0.0** 이상의 패치되지 않은 모든 버전이 교차 사이트 스크립팅 버그에 취약합니다. 결과적으로 권한이 없는 사용자는 **UI**에 **javascript** 링크를 삽입할 수 있습니다. 이제 이 문제가 해결되었습니다. [CVE-2022-31035](#)
- 이번 업데이트 이전에는 **Argo CD v0.11.0** 이상의 모든 버전이 **Argo CD CLI** 또는 **UI**에서 **SSO** 로그인을 시작할 때 여러 공격에 취약합니다. 이제 이 문제가 해결되었습니다. [CVE-2022-31034](#)
- 이번 업데이트 이전에는 **Argo CD v1.0.0** 이상의 패치되지 않은 모든 버전이 교차 사이트 스크립팅 버그에 취약합니다. 그 결과 권한이 없는 사용자는 **UI**에 **JavaScript** 링크를 삽입할 수 있었습니다. 이제 이 문제가 해결되었습니다. [CVE-2022-31016](#)
- 이번 업데이트 이전에는 **Argo CD v1.3.0**의 패치되지 않은 모든 버전이 **symlink-following** 버그에 취약합니다. 결과적으로 리포지토리 쓰기 액세스 권한이 있는 인증되지 않은 사용자는 **Argo CD**의 **repo-server**에서 중요한 **YAML** 파일을 유출할 수 있습니다. 이제 이 문제가 해결되었습니다. [CVE-2022-31036](#)

5.1.13. Red Hat OpenShift GitOps 1.5.2 릴리스 노트

Red Hat OpenShift GitOps 1.5.2는 이제 **OpenShift Container Platform 4.8** 이상에서 사용할 수 있습니다.

5.1.13.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 **redhat-operator-index** 에서 참조하는 이미지가 누락되었습니다. 이제 이 문제가 해결되었습니다. [GITOPS-2036](#)

5.1.14. Red Hat OpenShift GitOps 1.5.1 릴리스 노트

Red Hat OpenShift GitOps 1.5.1은 이제 OpenShift Container Platform 4.8 이상에서 사용할 수 있습니다.

5.1.14.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 **Argo CD**의 익명 액세스가 활성화된 경우 인증되지 않은 사용자가 **JWT** 토큰을 작성하고 **Argo CD** 인스턴스에 대한 전체 액세스 권한을 얻을 수 있었습니다. 이 문제는 이제 해결되었습니다. [CVE-2022-29165](#)
- 이번 업데이트 이전에는 인증되지 않은 사용자가 **SSO**가 활성화된 동안 로그인 화면에 오류 메시지를 표시할 수 있었습니다. 이제 이 문제가 해결되었습니다. [CVE-2022-24905](#)
- 이번 업데이트 이전에는 **Argo CD v0.7.0** 이상의 패치되지 않은 모든 버전이 **symlink-following** 버그에 취약합니다. 결과적으로 리포지터리 쓰기 액세스 권한이 있는 인증되지 않은 사용자는 **Argo CD**의 **repo-server**에서 중요한 파일을 유출할 수 있습니다. 이제 이 문제가 해결되었습니다. [CVE-2022-24904](#)

5.1.15. Red Hat OpenShift GitOps 1.5.0 릴리스 노트

Red Hat OpenShift GitOps 1.5.0은 이제 OpenShift Container Platform 4.8 이상에서 사용할 수 있습니다.

5.1.15.1. 새로운 기능

현재 릴리스에서는 다음과 같은 개선 사항이 추가되었습니다.

- 이번 개선된 기능을 통해 **Argo CD**가 **2.3.3** 버전으로 업그레이드되었습니다. [GITOPS-1708](#)
- 이번 개선된 기능을 통해 **Dex**를 버전 **2.30.3** 으로 업그레이드했습니다. [GITOPS-1850](#)
- 이번 개선된 기능을 통해 **Helm**이 버전 **3.8.0** 으로 업그레이드되었습니다. [GITOPS-1709](#)
- 이번 개선된 기능에는 **Kustomize**가 버전 **4.4.1** 로 업그레이드되었습니다. [GITOPS-1710](#)
- 이번 개선된 기능을 통해 **Application Set to version 0.4.1** 로 업그레이드했습니다.
- 이번 업데이트를 통해 **latest** 라는 새 채널이 추가되었습니다. 이 채널은 **Red Hat OpenShift GitOps**의 최신 릴리스를 제공합니다. **GitOps v1.5.0**의 경우 **Operator**가 **gitops-1.5**, 최신 채널 및 기존 **stable** 채널로 푸시됩니다. **GitOps v1.6**에서 모든 최신 릴리스는 **stable** 채널이 아닌 최신 채널로만 푸시됩니다. [GITOPS-1791](#)
- 이번 업데이트를 통해 새 **CSV**는 **olm.skipRange: '>=1.0.0 <1.5.0'** 주석을 추가합니다. 결과적으로 이전 릴리스 버전을 모두 건너뜀니다. **Operator**가 **v1.5.0**으로 직접 업그레이드합니다. [GITOPS-1787](#)
- 이번 업데이트를 통해 **Operator**는 다음과 같은 향상된 기능을 포함하여 **RH-SSO**(Red Hat Single Sign-On)를 버전 **vtekton.1**로 업데이트합니다.
 - **kube:admin** 인증 정보를 포함하여 **OpenShift Container Platform** 인증 정보를 사용하여 **Argo CD**에 로그인할 수 있습니다.
 - **RH-SSO**는 **OpenShift Container Platform** 그룹을 사용하여 **RBAC**(역할 기반 액세스 제어)에 대한 **Argo CD** 인스턴스를 지원하고 구성합니다.
 - **RH-SSO**는 **HTTP_Proxy** 환경 변수를 따릅니다. 프록시 뒤에서 실행되는 **Argo CD**의 **SSO**로 **RH-SSO**를 사용할 수 있습니다.

GITOPS-1330

- 이번 업데이트를 통해 **Argo CD** 피연산자의 **.status** 필드에 새 **.host URL** 필드가 추가됩니다. 경로에 지정된 우선 순위를 사용하여 경로 또는 수신을 활성화하면 새 **URL** 필드에 경로가 표시됩니다. 경로 또는 인그레스에서 **URL**을 제공하지 않으면 **.host** 필드가 표시되지 않습니다.

경로 또는 수신이 구성되어 있지만 해당 컨트롤러가 올바르게 설정되지 않았으며 준비 상태에 있지 않거나 **URL**을 전파하지 않는 경우 피연산자의 **.status.host** 필드의 값은 **URL**을 표시하는 대신 **Pending** 으로 표시됩니다. 이는 사용 가능한 대신 보류 상태를 설정하여 피연산자의 전반적인 상태에 영향을 미칩니다. [GITOPS-654](#)

5.1.15.2. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 **AppProjects** 와 관련된 **RBAC** 규칙에서 역할의 **subject** 필드에 쉼표를 사용하지 않아 **LDAP** 계정에 대한 바인딩이 발생하지 않았습니다. 이번 업데이트에서는 이 문제가 해결되어 이제 **AppProject** 특정 **RBAC** 규칙에 복잡한 역할 바인딩을 지정할 수 있습니다. [GITOPS-1771](#)
- 이번 업데이트 이전에는 **DeploymentConfig** 리소스가 **0** 으로 조정될 때 **Argo CD**가 **Pod**를 실행할 때까지 기다리는 동안 상태 상태 메시지가 있는 상태가 진행 중으로 표시되었습니다. 이번 업데이트에서는 옛지 케이스를 수정하고 상태 점검에서 **DeploymentConfig** 리소스의 올바른 상태를 보고합니다. [GITOPS-1738](#)
- 이번 업데이트 이전에는 **ArgoCD CR** 사양 **tls.initialCerts** 필드에 인증서가 구성되지 않은 경우 **argocd-tls-certs-cm** 구성 맵의 **TLS** 인증서가 **Red Hat OpenShift GitOps**에서 삭제되었습니다. 이 문제는 이제 해결되었습니다. [GITOPS-1725](#)
- 이번 업데이트 이전에는 **managed-by** 레이블이 있는 네임스페이스를 생성할 때 새 네임스페이스에 많은 **RoleBinding** 리소스가 생성되었습니다. 이번 업데이트에서는 이 문제를 해결했으며 이제 **Red Hat OpenShift GitOps**가 이전 버전에서 생성한 관련이 없는 역할 및 **Role Binding** 리소스를 제거합니다. [GITOPS-1550](#)
- 이번 업데이트 이전에는 패스스루 모드에서 경로의 **TLS** 인증서에 **CA** 이름이 없었습니다. 결과적으로 **Firefox 94** 이상에서는 오류 코드 **SEC_ERROR_BAD_DER** 를 사용하여 **Argo CD UI**에 연결하지 못했습니다. 이번 업데이트에서는 이 문제가 해결되었습니다. < **openshift-gitops-ca**> 보안을 삭제하고 다시 생성하도록 해야 합니다. 그런 다음 < **openshift-gitops-tls**> 시크릿을 삭제해야 합니다. **Red Hat OpenShift GitOps**가 이를 다시 생성한 후 **Firefox**에서 **Argo CD UI**에 다시 액세스할 수 있습니다. [GITOPS-1548](#)

5.1.15.3. 확인된 문제

- OpenShift Container Platform 클러스터에서 **Route** 리소스 대신 **Ingress** 리소스가 사용 중인 경우 **Argo CD** `.status.host` 필드가 업데이트되지 않습니다. [GITOPS-1920](#)

5.1.16. Red Hat OpenShift GitOps 1.4.13 릴리스 노트

Red Hat OpenShift GitOps 1.4.13은 이제 OpenShift Container Platform 4.7, 4.8, 4.9, 4.10 및 4.11에서 사용할 수 있습니다.

5.1.16.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- OpenShift Container Platform 4.12에서는 콘솔을 설치하는 것이 선택 사항입니다. 이번 수정을 통해 콘솔이 설치되지 않은 경우 **Operator**에 오류가 발생하지 않도록 **Red Hat OpenShift GitOps Operator**가 업데이트되었습니다. [GITOPS-2354](#)

5.1.17. Red Hat OpenShift GitOps 1.4.12 릴리스 노트

Red Hat OpenShift GitOps 1.4.12는 이제 OpenShift Container Platform 4.7 이상에서 사용할 수 있습니다.

5.1.17.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 다수의 애플리케이션이 활성 프로브의 응답하지 않음으로 인해 애플리케이션 컨트롤러가 여러 번 다시 시작되었습니다. 이번 업데이트에서는 애플리케이션 컨트롤러 **StatefulSet** 오브젝트에서 활성 프로브를 제거하여 문제를 해결합니다. [GITOPS-2153](#)
- 이번 업데이트 이전에는 인증 기관에서 서명하지 않은 인증서로 설정되는 경우 **RHSSO** 인증서를 검증할 수 없습니다. 이번 업데이트에서는 이 문제를 수정하고 이제 통신할 때 **Keycloak**의 **TLS** 인증서를 확인하는 데 사용할 사용자 정의 인증서를 제공할 수 있습니다. **rootCA**를 **Argo CD** 사용자 정의 리소스 `.spec.keycloak.rootCA` 필드에 추가할 수 있습니다. **Operator**는 이 변경 사항을 조정하고 **argocd-cm ConfigMap**의 `oidc.config` 필드를 **PEM** 인코딩 루트 인증서로 업데이트합니다. [GITOPS-2214](#)



참고

`.spec.keycloak.rootCA` 필드를 업데이트한 후 **Argo CD** 서버 **Pod**를 다시 시작하십시오.

예를 들면 다음과 같습니다.

```

apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  sso:
    provider: keycloak
    keycloak:
      rootCA: |
        ---- BEGIN CERTIFICATE ----
        This is a dummy certificate
        Please place this section with appropriate rootCA
        ---- END CERTIFICATE ----
  server:
    route:
      enabled: true

```

- 이번 업데이트 이전에는 **Argo CD**에서 관리하는 종료 네임스페이스가 다른 관리 네임스페이스의 역할 및 기타 구성 생성을 차단합니다. 이번 업데이트에서는 이 문제가 해결되었습니다. [GITOPS-2277](#)
- 이번 업데이트 이전에는 `anyuid`의 **SCC**가 **Dex ServiceAccount** 리소스에 할당되면 **Dex Pod**가 **CreateContainerConfigError**로 시작하지 못했습니다. 이번 업데이트에서는 기본 사용자 **ID**를 **Dex** 컨테이너에 할당하여 이 문제를 해결합니다. [GITOPS-2235](#)

5.1.18. Red Hat OpenShift GitOps 1.4.5 릴리스 노트

Red Hat OpenShift GitOps 1.4.5는 이제 OpenShift Container Platform 4.7 이상에서 사용할 수 있습니다.

5.1.18.1. 해결된 문제



주의

Red Hat OpenShift GitOps v1.4.3에서 Red Hat OpenShift GitOps v1.4.5로 직접 업그레이드해야 합니다. 프로덕션 환경에서는 Red Hat OpenShift GitOps v1.4.4를 사용하지 마십시오. Red Hat OpenShift GitOps v1.4.4에 영향을 받는 주요 문제는 Red Hat OpenShift GitOps 1.4.5에서 수정되었습니다.

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 **Argo CD Pod가 ErrImagePullBackOff** 상태에 있었습니다. 다음과 같은 오류 메시지가 표시되었습니다.

```
reason: ErrImagePull
message: >-
  rpc error: code = Unknown desc = reading manifest
  sha256:ff4ad30752cf0d321cd6c2c6fd4490b716607ea2960558347440f2f370a586a8
  in registry.redhat.io/openshift-gitops-1/argocd-rhel8: StatusCode:
  404, <HTML><HEAD><TITLE>Error</TITLE></HEAD><BODY>
```

이제 이 문제가 해결되었습니다. [GITOPS-1848](#)

5.1.19. Red Hat OpenShift GitOps 1.4.3 릴리스 노트

Red Hat OpenShift GitOps 1.4.3은 이제 OpenShift Container Platform 4.7, 4.8 및 4.9에서 사용할 수 있습니다.

5.1.19.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 **ArgoCD CR 사양 tls.initialCerts** 필드에 인증서가 구성되지 않은 **Red Hat OpenShift GitOps**에서 **argocd-tls-certs-cm** 구성 맵의 **TLS** 인증서가 삭제되었습니다. 이번 업데이트에서는 이 문제가 해결되었습니다. [GITOPS-1725](#)

5.1.20. Red Hat OpenShift GitOps 1.4.2 릴리스 노트

Red Hat OpenShift GitOps 1.4.2는 이제 OpenShift Container Platform 4.7, 4.8 및 4.9에서 사용할 수 있습니다.

5.1.20.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- **Argo CD**의 모든 버전은 **Helm** 차트에서 임의의 값을 사용할 수 있는 경로 순방향 버그에 취약합니다. 이번 업데이트에서는 **Helm** 값 파일을 전달할 때 **CVE-2022-24348 gitops** 오류, 경로 트랜잭션 및 역참조가 수정되었습니다. [GITOPS-1756](#)
- 이번 업데이트 이전에는 하나 이상의 **Ingress** 가 경로에 연결된 경우 **Route** 리소스가 **Progressing Health** 상태가 되었습니다. 이번 업데이트에서는 상태 점검을 수정하고 **Route** 리소스의 올바른 상태를 보고합니다. [GITOPS-1751](#)

5.1.21. Red Hat OpenShift GitOps 1.4.1 릴리스 노트

Red Hat OpenShift GitOps 1.4.1은 이제 OpenShift Container Platform 4.7, 4.8 및 4.9에서 사용할 수 있습니다.

5.1.21.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- **Red Hat OpenShift GitOps Operator v1.4.0**에서는 다음 **CRD**에 대한 **spec** 에서 **description** 필드를 제거하는 회귀 문제가 도입되었습니다.
 - `argoproj.io_applications.yaml`
 - `argoproj.io_appprojects.yaml`
 - `argoproj.io_argocds.yaml`

이번 업데이트 이전에는 `kubectl create` 를 사용하여 **AppProject** 리소스를 생성한 경우 동기화하지 못했습니다. 이번 업데이트에서는 **CustomResourceDefinitions**에서 누락된 설명 필드를 복원합니다.

ArgoCD, AppProject, Application 로 인해 kubectl create 가 실패했습니다. [GITOPS-1721](#)

5.1.22. Red Hat OpenShift GitOps 1.4.0 릴리스 노트

Red Hat OpenShift GitOps 1.4.0은 이제 OpenShift Container Platform 4.7, 4.8 및 4.9에서 사용할 수 있습니다.

5.1.22.1. 새로운 기능

현재 릴리스에서는 다음과 같은 개선 사항이 추가되었습니다.

- 이번 개선된 기능을 통해 Red Hat OpenShift GitOps Application Manager (kam)를 버전 0.0.41 로 업그레이드합니다. [GITOPS-1669](#)
- 이번 개선된 기능은 Argo CD를 버전 2.2.2 로 업그레이드합니다. [GITOPS-1532](#)
- 이번 개선된 기능은 Helm을 버전 3.7.1 로 업그레이드합니다. [GITOPS-1530](#)
- 이번 개선된 기능에는 DeploymentConfig, Route, OLM Operator 항목의 상태가 Argo CD 대시보드 및 OpenShift Container Platform 웹 콘솔에 추가되었습니다. 이 정보는 애플리케이션의 전반적인 상태를 모니터링하는 데 도움이 됩니다. [GITOPS-655](#), [GITOPS-915](#), [GITOPS-916](#), [GITOPS-1110](#)
- 이번 업데이트를 통해 Argo CD 사용자 정의 리소스에 .spec.server.replicas 및 .spec.repo.replicas 특성을 설정하여 argocd-server 및 argocd-repo-server 구성 요소에 대해 원하는 복제본 수를 지정할 수 있습니다. argocd-server 구성 요소에 HPA(수평 Pod 자동 스케일링)를 구성하는 경우 Argo CD 사용자 정의 리소스 속성보다 우선합니다. [GITOPS-1245](#)
- 관리 사용자로 argocd.argoproj.io/managed-by 레이블을 사용하여 Argo CD 액세스 권한을 부여하면 namespace-admin 권한을 가정합니다. 이러한 권한은 관리자가 아닌 사용자가 네트워크 정책과 같은 오브젝트를 수정할 수 있기 때문에 개발 팀과 같은 관리자가 아닌 관리자에게 네임스페이스를 제공하는 관리자에게 문제가 됩니다.

이번 업데이트를 통해 관리자는 모든 관리 네임스페이스에 대한 공통 클러스터 역할을 구성할 수 있습니다. Argo CD 애플리케이션 컨트롤러의 역할 바인딩에서 Operator는 CONTROLLER_CLUSTER_ROLE 환경 변수를 참조합니다. Argo CD 서버의 역할 바인딩에서 Operator는 SERVER_CLUSTER_ROLE 환경 변수를 참조합니다. 이러한 환경 변수에 사용자 정

의 역할이 포함된 경우 **Operator**는 기본 **admin** 역할을 생성하지 않습니다. 대신 모든 관리 네임스페이스에 기존 사용자 지정 역할을 사용합니다. [GITOPS-1290](#)

- 이번 업데이트를 통해 **OpenShift Container Platform** 개발자 콘솔의 환경 페이지에 성능 저하된 리소스가 표시되고 상태가 **Progressing, Missing, Unknown**인 리소스를 제외하고 손상되는 아이콘이 표시됩니다. 콘솔에는 동기화되지 않은 리소스를 나타내는 노란색 유효 기호 아이콘이 표시됩니다. [GITOPS-1307](#)

5.1.22.2. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 **URL**에 경로를 지정하지 않고 **Red Hat OpenShift GitOps Application Manager(kam)**로의 경로가 액세스되면 사용자에게 유용한 정보가 표시되지 않은 기본 페이지가 표시됩니다. 이번 업데이트에서는 문제가 해결되어 기본 페이지에 **kam**에 대한 다운로드 링크가 표시되도록 합니다. [GITOPS-923](#)
- 이번 업데이트 이전에는 **Argo CD** 사용자 정의 리소스의 네임스페이스에 리소스 할당량을 설정하면 **RH SSO(Red Hat SSO)** 인스턴스 설정이 실패할 수 있습니다. 이번 업데이트에서는 **RH SSO** 배포 포드에 대한 최소 리소스 요청을 설정하여 이 문제를 해결합니다. [GITOPS-1297](#)
- 이번 업데이트 이전에는 **argocd-repo-server** 워크로드의 로그 수준을 변경한 경우 **Operator**가 이 설정을 조정하지 않았습니다. 해결방법은 **Operator**가 새 로그 수준으로 다시 생성되도록 배포 리소스를 삭제하는 것이었습니다. 이번 업데이트를 통해 기존 **argocd-repo-server** 워크로드에 대한 로그 수준이 올바르게 조정됩니다. [GITOPS-1387](#)
- 이번 업데이트 이전에는 **Operator**에서 **argocd-secret Secret .data** 필드가 없는 **Argo CD** 인스턴스를 관리하면 해당 인스턴스의 **Operator**가 충돌했습니다. 이번 업데이트에서는 **.data** 필드가 누락될 때 **Operator**가 충돌하지 않도록 문제를 해결합니다. 대신 보안이 다시 생성되고 **gitops-operator-controller-manager** 리소스가 재배포됩니다. [GITOPS-1402](#)
- 이번 업데이트 이전에는 **gitopsservice** 서비스에 내부 오브젝트로 주석이 달렸습니다. 이번 업데이트에서는 주석을 제거하므로 기본 **Argo CD** 인스턴스를 업데이트하거나 삭제하고 **UI**를 사용하여 인프라 노드에서 **GitOps** 워크로드를 실행할 수 있습니다. [GITOPS-1429](#)

5.1.22.3. 확인된 문제

다음은 현재 릴리스에서 알려진 문제입니다.

- **Dex** 인증 공급자에서 **Keycloak** 공급자로 마이그레이션하는 경우 **Keycloak**에 로그인 문제가 발생할 수 있습니다.

이 문제를 방지하려면 **Argo CD** 사용자 정의 리소스에서 **.spec.dex** 섹션을 제거하여 **Dex**를 제거합니다. **Dex**가 완전히 제거될 때까지 몇 분 정도 기다립니다. 그런 다음 **Argo CD** 사용자 정의 리소스에 **.spec.sso.provider: keycloak** 을 추가하여 **Keycloak**을 설치합니다.

해결 방법으로 **.spec.sso.provider: keycloak** 을 제거하여 **Keycloak**을 제거합니다. 그런 다음 다시 설치하십시오. [GITOPS-1450](#), [GITOPS-1331](#)

5.1.23. Red Hat OpenShift GitOps 1.3.7 릴리스 노트

Red Hat OpenShift GitOps 1.3.7은 이제 **OpenShift Container Platform 4.7** 이상에서 사용할 수 있습니다.

5.1.23.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 **OpenSSL**에서 취약점이 발견되었습니다. 이번 업데이트에서는 보안 취약점을 방지하기 위해 **OpenSSL**의 기본 이미지를 최신 버전으로 업데이트하여 문제를 해결합니다. ([CVE-2022-0778](#)).



참고

현재 **Red Hat OpenShift GitOps 1.3** 릴리스를 설치하고 제품 라이프 사이클 동안 추가 업데이트를 받으려면 **GitOps-1.3** 채널로 전환합니다.

5.1.24. Red Hat OpenShift GitOps 1.3.6 릴리스 노트

Red Hat OpenShift GitOps 1.3.6은 이제 **OpenShift Container Platform 4.7** 이상에서 사용할 수 있습니다.

5.1.24.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- **Red Hat OpenShift GitOps**에서 부적절한 액세스 제어는 관리자 권한 상승 (**CVE-2022-1025**) 을 허용합니다. 이번 업데이트에서는 이 문제가 해결되었습니다.
- 경로 문제 취약점으로 인해 아웃 오브 바운드 파일 (**CVE-2022-24731**) 이 유출될 수 있습니다. 이번 업데이트에서는 이 문제가 해결되었습니다.
- 경로 문제 취약점 및 부적절한 액세스 제어로 인해 아웃 오브 바운드 파일 (**CVE-2022-24730**) 이 유출될 수 있습니다. 이번 업데이트에서는 이 문제가 해결되었습니다.

5.1.25. Red Hat OpenShift GitOps 1.3.2 릴리스 노트

Red Hat OpenShift GitOps 1.3.2는 이제 **OpenShift Container Platform 4.6, 4.7, 4.8** 및 **4.9**에서 사용할 수 있습니다.

5.1.25.1. 새로운 기능

다음 섹션에서는 수정 및 안정성 개선 사항 외에 **Red Hat OpenShift GitOps 1.3.2**의 새로운 기능도 소개합니다.

- **Argo CD**를 버전 **2.1.8**로 업그레이드
- 버전 **2.30.0**으로 업그레이드 **Dex**

5.1.25.2. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이번 업데이트 이전에는 연결 해제 로 필터링하면 **Red Hat OpenShift GitOps Operator**가 **CSV** 파일에 관련 주석이 설정되지 않았기 때문에 인프라 기능 섹션의 **OperatorHub UI**에서 **Red Hat OpenShift GitOps Operator**가 검색 결과에 표시되지 않았습니다. 이번 업데이트를 통해 **Red Hat OpenShift GitOps Operator**에 인프라 기능으로 연결되지 않은 **Cluster** 주석이 추가되었습니다. **GITOPS-1539**
- 네임스페이스 범위의 **Argo CD** 인스턴스를 사용하는 경우 예를 들어 클러스터에서 모든 **Namepaces** 로 지정되지 않은 **Argo CD** 인스턴스인 **Red Hat OpenShift GitOps**는 관리 네임스페이스 목록을 동적으로 유지합니다. 이러한 네임스페이스에는

`argocd.argoproj.io/managed-by` 레이블이 포함됩니다. 이 네임스페이스 목록은 **Argo CD** → 설정 → 클러스터 → "클러스터" → **NAMESPACES** 의 캐시에 저장됩니다. 이번 업데이트 이전에는 이러한 네임스페이스 중 하나를 삭제한 경우 **Operator**에서 해당 작업을 무시하고 네임스페이스가 목록에 남아 있었습니다. 이로 인해 클러스터 구성에서 **CONNECTION STATE** 가 중단되고 모든 동기화 시도에 오류가 발생했습니다. 예를 들면 다음과 같습니다.

Argo service account does not have <random_verb> on <random_resource_type> in namespace <the_namespace_you_deleted>.

이 버그가 수정되었습니다. [GITOPS-1521](#)

- 이번 업데이트를 통해 **Red Hat OpenShift GitOps Operator**에 **Deep Insights** 기능 수준이 주석이 추가되었습니다. [GITOPS-1519](#)
- 이전에는 **Argo CD Operator**가 `resource.exclusion` 필드를 자체적으로 관리했지만 `resource.inclusion` 필드를 무시했습니다. 이로 인해 **Argo CD CR**에 구성된 `resource.inclusion` 필드가 `argocd-cm` 구성 맵에 생성되지 않았습니다. 이 문제는 이번 릴리스에서 해결되었습니다. [GITOPS-1518](#)

5.1.26. Red Hat OpenShift GitOps 1.3.1 릴리스 노트

이제 **OpenShift Container Platform 4.6, 4.7, 4.8** 및 **4.9**에서 **Red Hat OpenShift GitOps 1.3.1**을 사용할 수 있습니다.

5.1.26.1. 해결된 문제

- **v1.3.0**으로 업그레이드하는 경우 **Operator**는 순서가 지정된 환경 변수 슬라이스를 반환하지 않습니다. 결과적으로 조정기가 실패하여 프록시 뒤에서 실행되는 **OpenShift Container Platform** 클러스터에서 **Argo CD Pod**가 자주 복제됩니다. 이번 업데이트에서는 **Argo CD Pod**가 다시 생성되지 않도록 문제가 해결되었습니다. [GITOPS-1489](#)

5.1.27. Red Hat OpenShift GitOps 1.3 릴리스 노트

이제 **OpenShift Container Platform 4.7, 4.8, 4.9**에서 **Red Hat OpenShift GitOps 1.3**을 사용할 수 있습니다.

5.1.27.1. 새로운 기능

다음 섹션에서는 수정 및 안정성 개선 사항 외에 **Red Hat OpenShift GitOps 1.3.0**의 새로운 기능도 소개합니다.

- **v1.3.0**을 새로 설치하는 경우 **Dex**가 자동으로 구성됩니다. **OpenShift** 또는 **kubeadmin** 인증 정보를 사용하여 **openshift-gitops** 네임스페이스에서 기본 **Argo CD** 인스턴스에 로그인할 수 있습니다. 관리자는 **Operator**를 설치한 후 **openshift-gitops** 네임스페이스에서 **Dex** 배포를 제거한 후 **Dex** 설치를 비활성화할 수 있습니다.
- **Operator**에서 설치한 기본 **Argo CD** 인스턴스와 함께 컨트롤러는 이제 간단한 구성 토글을 설정하여 클러스터의 인프라 노드에서 실행할 수 있습니다.
- **Argo CD**의 내부 통신은 이제 **TLS** 및 **OpenShift** 클러스터 인증서를 사용하여 보안을 설정할 수 있습니다. **Argo CD** 경로는 이제 **cert-manager**와 같은 외부 인증서 관리자를 사용하는 것 외에도 **OpenShift** 클러스터 인증서를 활용할 수 있습니다.
- 콘솔 **4.9**의 개발자 관점에서 향상된 환경 세부 정보 보기를 사용하여 **GitOps** 환경에 대한 정보를 얻습니다.
- **OLM**을 사용하여 설치된 **DeploymentConfig** 리소스, 경로 리소스 및 **Operator**의 **Argo CD**에서 사용자 정의 상태 점검에 액세스할 수 있습니다.
- 이제 **GitOps Operator**가 최신 **Operator-SDK**에서 권장하는 명명 규칙을 따릅니다.
 - **gitops-operator-** 접두사가 모든 리소스에 추가되었습니다.
 - 서비스 계정의 이름이 **gitops-operator-controller-manager**로 변경되었습니다.

5.1.27.2. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이전 버전에서는 **Argo CD**의 새 인스턴스에서 관리하도록 새 네임스페이스를 설정하는 경우 **Operator**에서 새 네임스페이스를 관리하기 위해 생성하는 새 역할 및 바인딩으로 인해 즉시 동기화되지 않았습니다. 이 동작은 수정되었습니다. [GITOPS-1384](#)

5.1.27.3. 확인된 문제

- **Dex** 인증 공급자에서 **Keycloak** 공급자로 마이그레이션하는 동안 **Keycloak**에 로그인 문제가 발생할 수 있습니다. [GITOPS-1450](#)

위의 문제를 방지하려면 **Argo CD** 사용자 정의 리소스에 있는 **.spec.dex** 섹션을 제거하여 **Dex**를 제거하십시오. **Dex**가 완전히 제거될 때까지 몇 분 동안 기다린 다음 **Argo CD** 사용자 정의 리소스에 **.spec.sso.provider: keycloak** 을 추가하여 **Keycloak**을 설치합니다.

이 문제를 해결하려면 **.spec.sso.provider: keycloak** 을 제거한 다음 다시 설치하여 **Keycloak**을 제거합니다.

5.1.28. Red Hat OpenShift GitOps 1.2.2 릴리스 노트

Red Hat OpenShift GitOps 1.2.2는 이제 **OpenShift Container Platform 4.8**에서 사용할 수 있습니다.

5.1.28.1. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- **Argo CD**의 모든 버전은 **Helm** 차트에서 임의의 값을 사용할 수 있는 경로 순방향 버그에 취약합니다. 이번 업데이트에서는 **Helm** 값 파일을 전달할 때 **CVE-2022-24348 gitops** 오류, **path traversal** 및 역참조가 수정되었습니다. [GITOPS-1756](#)

5.1.29. Red Hat OpenShift GitOps 1.2.1 릴리스 노트

이제 **OpenShift Container Platform 4.7** 및 **4.8**에서 **Red Hat OpenShift GitOps 1.2.1**을 사용할 수 있습니다.

5.1.29.1. 지원 매트릭스

이 릴리스의 일부 기능은 현재 기술 프리뷰 단계에 있습니다. 이러한 실험적 기능은 프로덕션용이 아닙니다.

기술 프리뷰 기능 지원 범위

아래 표에서 기능은 다음 상태로 표시됩니다.

- **TP: 기술 프리뷰**
- **GA: 정식 출시일 (GA)**

해당 기능은 **Red Hat Customer Portal**의 지원 범위를 참조하십시오.

표 5.1. 지원 매트릭스

기능	Red Hat OpenShift GitOps 1.2.1
Argo CD	GA
Argo CD ApplicationSet	TP
Red Hat OpenShift GitOps Application Manager (kam)	TP

5.1.29.2. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이전에는 시작 시 애플리케이션 컨트롤러에서 대규모 메모리 스파이크를 관찰했습니다. 애플리케이션 컨트롤러의 `--kubectl-parallelism-limit` 플래그는 기본적으로 10으로 설정되어 있지만 **Argo CD CR** 사양에 `.spec.controller.kubeParallelismLimit`의 번호를 지정하여 이 값을 재정의할 수 있습니다. [GITOPS-1255](#)
- 최신 **Triggers API**로 인해 **kam** 부트스트랩 명령을 사용할 때 `kustomization.yaml`의 중복 항목으로 인해 **Kubernetes** 빌드가 실패했습니다. **Pipelines** 및 **Tekton** 트리거 구성 요소가 이제 이 문제를 해결하기 위해 각각 **v0.24.2** 및 **v0.14.2**로 업데이트되었습니다. [GITOPS-1273](#)
- 소스 네임스페이스의 **Argo CD** 인스턴스가 삭제될 때 대상 네임스페이스에서 **RBAC** 역할 및 바인딩을 유지하면 대상 네임스페이스에서 자동으로 제거됩니다. [GITOPS-1228](#)
- 이전 버전에서는 **Argo CD** 인스턴스를 네임스페이스에 배포할 때 **Argo CD** 인스턴스가 **"managed-by"** 라벨이 자체 네임스페이스로 변경되었습니다. 이번 수정을 통해 네임스페이스에 필요한 **RBAC** 역할 및 바인딩도 생성 및 삭제되는 동안 네임스페이스에 라벨이 지정되지 않았습니다. [GITOPS-1247](#)

- 이전에는 **repo-server** 및 애플리케이션 컨트롤러를 위한 **Argo CD** 워크로드에 대한 기본 리소스 요청 제한이 매우 제한적이었습니다. 이제 기존 리소스 할당량이 제거되었으며 리포지토리 서버에서 기본 메모리 제한이 **1024M**으로 증가했습니다. 이 변경 사항은 새 설치에만 영향을 미칩니다. 기존 **Argo CD** 인스턴스 워크로드는 영향을 받지 않습니다. [GITOPS-1274](#)

5.1.30. Red Hat OpenShift GitOps 1.2 릴리스 노트

Red Hat OpenShift GitOps 1.2는 이제 **OpenShift Container Platform 4.7** 및 **4.8**에서 사용할 수 있습니다.

5.1.30.1. 지원 매트릭스

이 릴리스의 일부 기능은 현재 기술 프리뷰 단계에 있습니다. 이러한 실험적 기능은 프로덕션용이 아닙니다.

기술 프리뷰 기능 지원 범위

아래 표에서 기능은 다음 상태로 표시됩니다.

- TP: 기술 프리뷰**
- GA: 정식 출시일 (GA)**

해당 기능은 **Red Hat Customer Portal**의 지원 범위를 참조하십시오.

표 5.2. 지원 매트릭스

기능	Red Hat OpenShift GitOps 1.2
Argo CD	GA
Argo CD ApplicationSet	TP
Red Hat OpenShift GitOps Application Manager (kam)	TP

5.1.30.2. 새로운 기능

다음 섹션에서는 수정 및 안정성 개선 사항 외에 **Red Hat OpenShift GitOps 1.2**의 새로운 기능도 소개합니다.

- openshift-gitops** 네임스페이스에 대한 읽기 또는 쓰기 권한이 없는 경우 이제 **GitOps Operator**에서 **DISABLE_DEFAULT_ARGOCD_INSTANCE** 환경 변수를 사용하고 기본 **Argo CD** 인스턴스가 **openshift-gitops** 네임스페이스에서 시작되지 않도록 **TRUE**로 설정할 수 있습니다.
- 이제 리소스 요청 및 제한이 **Argo CD** 워크로드에서 구성됩니다. **openshift-gitops** 네임스페이스에서 리소스 할당량이 활성화됩니다. 결과적으로 **openshift-gitops** 네임스페이스에 수동으로 배포된 대역 외 워크로드는 리소스 요청 및 제한을 사용하여 구성해야 하며 리소스 할당량을 늘려야 할 수 있습니다.
- Argo CD** 인증은 이제 **Red Hat SSO**와 통합되며 클러스터의 **OpenShift 4 ID** 공급자로 자동으로 구성됩니다. 이 기능은 기본적으로 비활성화되어 있습니다. **Red Hat SSO**를 활성화하려면 다음과 같이 **ArgoCD CR**에 **SSO** 구성을 추가합니다. 현재 **keycloak**은 지원되는 유일한 공급자입니다.

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  sso:
    provider: keycloak
  server:
    route:
      enabled: true
```

- 라우터 샤딩을 지원하기 위해 경로 레이블을 사용하여 호스트 이름을 정의할 수 있습니다. 이제 **server (argocd server)**, **grafana**, **prometheus** 경로에서 레이블 설정 지원을 사용할 수 있습니다. 경로에 레이블을 설정하려면 **ArgoCD CR**의 서버에 대한 경로 구성 아래에 **labels**를 추가합니다.

argocd 서버에 라벨을 설정하는 **ArgoCD CR YAML**의 예

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
```

```

example: basic
spec:
  server:
    route:
      enabled: true
    labels:
      key1: value1
      key2: value2
    
```

- GitOps Operator**는 레이블을 적용하여 대상 네임스페이스의 리소스를 관리할 수 있도록 **Argo CD** 인스턴스에 권한을 자동으로 부여합니다. 사용자는 `argocd.argoproj.io/managed-by:<source-namespace>` 라벨을 사용하여 대상 네임스페이스에 레이블을 지정할 수 있습니다. 여기서 `source-namespace`는 **argocd** 인스턴스가 배포된 네임스페이스입니다.

5.1.30.3. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이전 버전에서는 사용자가 **openshift-gitops** 네임스페이스에서 기본 클러스터 인스턴스에서 관리하는 **Argo CD**의 추가 인스턴스를 생성한 경우 새 **Argo CD** 인스턴스를 담당하는 애플리케이션이 **OutOfSync** 상태로 중단되었습니다. 이 문제는 클러스터 시크릿에 소유자 참조를 추가하여 해결되었습니다. [GITOPS-1025](#)

5.1.30.4. 확인된 문제

이는 **Red Hat OpenShift GitOps 1.2**에서 알려진 문제입니다.

- 소스 네임스페이스에서 **Argo CD** 인스턴스가 삭제되면 대상 네임스페이스의 `argocd.argoproj.io/managed-by` 레이블이 제거되지 않습니다. [GITOPS-1228](#)
- Red Hat OpenShift GitOps 1.2**의 **openshift-gitops** 네임스페이스에서 리소스 할당량이 활성화되었습니다. 이는 수동으로 배포된 대역 외 워크로드 및 **openshift-gitops** 네임스페이스의 기본 **Argo CD** 인스턴스에서 배포한 워크로드에 영향을 미칠 수 있습니다. **Red Hat OpenShift GitOps v1.1.2**에서 **v1.2**로 업그레이드하는 경우 리소스 요청 및 제한을 사용하여 워크로드를 구성해야 합니다. 추가 워크로드가 있는 경우 **openshift-gitops** 네임스페이스의 리소스 할당량을 늘려야 합니다.

openshift-gitops 네임스페이스의 현재 리소스 할당량입니다.

리소스	요구 사항	제한
CPU	6688m	13750m
메모리	4544Mi	9070Mi

아래 명령을 사용하여 **CPU** 제한을 업데이트할 수 있습니다.

```
$ oc patch resourcequota openshift-gitops-compute-resources -n openshift-gitops --type=json -p='[{"op": "replace", "path": "/spec/hard/limits.cpu", "value": "9000m"}]'
```

아래 명령을 사용하여 **CPU** 요청을 업데이트할 수 있습니다.

```
$ oc patch resourcequota openshift-gitops-compute-resources -n openshift-gitops --type=json -p='[{"op": "replace", "path": "/spec/hard/cpu", "value": "7000m"}]'
```

위의 명령의 경로를 **cpu**에서 **memory**로 교체하여 메모리를 업데이트할 수 있습니다.

5.1.31. Red Hat OpenShift GitOps 1.1 릴리스 노트

Red Hat OpenShift GitOps 1.1은 이제 OpenShift 컨테이너 플랫폼 4.7에서 사용할 수 있습니다.

5.1.31.1. 지원 매트릭스

이 릴리스의 일부 기능은 현재 기술 프리뷰 단계에 있습니다. 이러한 실험적 기능은 프로덕션용이 아닙니다.

기술 프리뷰 기능 지원 범위

아래 표에서 기능은 다음 상태로 표시됩니다.

- **TP: 기술 프리뷰**
- **GA: 정식 출시일 (GA)**

해당 기능은 **Red Hat Customer Portal**의 지원 범위를 참조하십시오.

표 5.3. 지원 매트릭스

기능	Red Hat OpenShift GitOps 1.1
Argo CD	GA
Argo CD ApplicationSet	TP
Red Hat OpenShift GitOps Application Manager (kam)	TP

5.1.31.2. 새로운 기능

다음 섹션에서는 수정 및 안정성 개선 사항 외에 **Red Hat OpenShift GitOps 1.1**의 새로운 기능도 소개합니다.

- 이제 **ApplicationSet** 기능이 추가되었습니다(기술 프리뷰). **ApplicationSet** 기능을 사용하면 **Argo CD** 애플리케이션을 다수의 클러스터와 **monorepo** 내에서 관리할 때 자동화와 유연성을 모두 사용할 수 있습니다. 또한 멀티테넌트 **Kubernetes** 클러스터에서 셀프 서비스를 사용할 수 있습니다.
- Argo CD**는 이제 클러스터 로깅 스택 및 **OpenShift Container Platform** 모니터링 및 경고 기능과 통합되었습니다.
- Argo CD** 인증이 **OpenShift Container Platform**과 통합되었습니다.
- Argo CD** 애플리케이션 컨트롤러는 이제 수평 크기 조정을 지원합니다.
- Argo CD Redis** 서버는 이제 **HA(고가용성)**를 지원합니다.

5.1.31.3. 해결된 문제

현재 릴리스에서 다음 문제가 해결되었습니다.

- 이전에는 활성 글로벌 프록시 설정을 사용하여 프록시 서버 설정에서 **Red Hat OpenShift**

GitOps가 예상대로 작동하지 않았습니다. 이 문제는 해결되었으며 이제 Red Hat OpenShift GitOps Operator가 Argo CD는 Pod에 FQDN(정규화된 도메인 이름)을 사용하여 구성 요소 간 통신을 활성화하도록 구성되어 있습니다. [GITOPS-703](#)

- **Red Hat OpenShift GitOps 백엔드는 Red Hat OpenShift GitOps URL의 ?ref= 쿼리 매개 변수를 사용하여 API를 호출합니다. 이전에는 이 매개 변수를 URL에서 읽지 않아 백엔드에서 항상 기본 참조를 고려했습니다. 이 문제는 해결되어 Red Hat OpenShift GitOps 백엔드에서 Red Hat OpenShift GitOps URL에서 참조 쿼리 매개 변수를 추출하고 입력 참조가 제공되지 않은 경우에만 기본 참조를 사용합니다. [GITOPS-817](#)**
- **이전에는 Red Hat OpenShift GitOps 백엔드에서 유효한 GitLab 리포지토리를 찾지 못했습니다. 이는 Red Hat OpenShift GitOps 백엔드가 GitLab 리포지토리의 master 대신 main 분기 참조로 확인되었기 때문입니다. 이 문제는 이제 해결되었습니다. [GITOPS-768](#)**
- **OpenShift Container Platform 웹 콘솔의 개발자 화면에 있는 환경 페이지에 애플리케이션 목록과 환경 수가 표시됩니다. 이 페이지에는 모든 애플리케이션을 나열하는 Argo CD 애플리케이션 페이지로 이동하는 Argo CD 링크도 표시됩니다. Argo CD 애플리케이션 페이지에는 선택한 애플리케이션만 필터링할 수 있는 LABELS (예: app.kubernetes.io/name=appName)가 있습니다. [GITOPS-544](#)**

5.1.31.4. 확인된 문제

이는 Red Hat OpenShift GitOps 1.1에서 알려진 문제입니다.

- **Red Hat OpenShift GitOps는 Helm v2 및 ksonnet을 지원하지 않습니다.**
- **RH SSO(Red Hat SSO) Operator는 연결이 끊긴 클러스터에서 지원되지 않습니다. 결과적으로 연결이 끊긴 클러스터에서 Red Hat OpenShift GitOps Operator 및 RH SSO 통합이 지원되지 않습니다.**
- **OpenShift Container Platform 웹 콘솔에서 Argo CD 애플리케이션을 삭제하면 사용자 인터페이스에서 Argo CD 애플리케이션이 삭제되지만 배포는 여전히 클러스터에 있습니다. 해결 방법으로 Argo CD 콘솔에서 Argo CD 애플리케이션을 삭제합니다. [GITOPS-830](#)**

5.1.31.5. 변경 사항 중단

5.1.31.5.1. Red Hat OpenShift GitOps v1.0.1에서 업그레이드

Red Hat OpenShift GitOps v1.0.1에서 v1.1으로 업그레이드하는 경우 Red Hat OpenShift GitOps

Operator는 `openshift-gitops` 네임스페이스에 생성된 기본 **Argo CD** 인스턴스 이름을 `argocd-cluster`에서 `openshift-gitops`로 변경합니다.

이는 변경 사항이 중단되어 업그레이드 전에 수동으로 다음 단계를 수행해야 합니다.

1.

OpenShift Container Platform 웹 콘솔로 이동하여 `openshift-gitops` 네임스페이스에 있는 `argocd-cm.yml` 구성 맵 파일의 콘텐츠를 로컬 파일에 복사합니다. 내용은 다음 예와 같을 수 있습니다.

`argocd` 구성 맵 **YAML**의 예

```
kind: ConfigMap
apiVersion: v1
metadata:
  selfLink: /api/v1/namespaces/openshift-gitops/configmaps/argocd-cm
  resourceVersion: '112532'
  name: argocd-cm
  uid: f5226fbc-883d-47db-8b53-b5e363f007af
  creationTimestamp: '2021-04-16T19:24:08Z'
  managedFields:
  ...
  namespace: openshift-gitops
  labels:
    app.kubernetes.io/managed-by: argocd-cluster
    app.kubernetes.io/name: argocd-cm
    app.kubernetes.io/part-of: argocd
  data: "" 1
  admin.enabled: 'true'
  statusbadge.enabled: 'false'
  resource.exclusions: |
    - apiGroups:
      - tekton.dev
      clusters:
      - '*'
      kinds:
      - TaskRun
      - PipelineRun
  ga.trackingid: ""
  repositories: |
    - type: git
      url: https://github.com/user-name/argocd-example-apps
  ga.anonymouseusers: 'false'
  help.chatUrl: ""
  url: >-
    https://argocd-cluster-server-openshift-gitops.apps.dev-svc-4.7-041614.devcluster.openshift.com "" 2
  help.chatText: ""
  kustomize.buildOptions: ""
```

```
resource.inclusions: "
repository.credentials: "
users.anonymous.enabled: 'false'
configManagementPlugins: "
application.instanceLabelKey: "
```

1

`argocd-cm.yml` 구성 맵 파일의 `data` 섹션만 수동으로 복원합니다.

2

구성 맵 항목의 `URL` 값을 새 인스턴스 이름 `openshift-gitops`로 바꿉니다.

2.

기본 `argocd-cluster` 인스턴스를 삭제합니다.

3.

새 `argocd-cm.yml` 구성 맵 파일을 편집하여 전체 `data` 섹션을 수동으로 복원합니다.

4.

구성 맵 항목의 `URL` 값을 새 인스턴스 이름 `openshift-gitops`로 바꿉니다. 예를 들어 위 예제에서 `URL` 값을 다음 `URL` 값으로 바꿉니다.

```
url: >-
https://openshift-gitops-server-openshift-gitops.apps.dev-svc-4.7-
041614.devcluster.openshift.com
```

5.

Argo CD 클러스터에 로그인하고 이전 구성이 있는지 확인합니다.

5.2. OPENSIFT GITOPS 이해

5.2.1. GitOps 정보

GitOps는 클라우드 네이티브 애플리케이션에 대한 연속 배포를 구현하는 선언적 방법입니다. **GitOps**를 사용하여 다중 클러스터 **Kubernetes** 환경에서 **OpenShift Container Platform** 클러스터 및 애플리케이션을 관리하기 위해 반복 가능한 프로세스를 생성할 수 있습니다. **GitOps**는 복잡한 배포를 빠른 속도로 처리하고 자동화하여 배포 및 릴리스 주기 동안 시간을 절약합니다.

GitOps 워크플로는 개발, 테스트, 스테이징, 프로덕션 단계를 통해 애플리케이션을 내보냅니다.

GitOps는 새 애플리케이션을 배포하거나 기존 애플리케이션을 업데이트하므로 리포지토리만 업데이트 하면 됩니다. 기타 모든 작업은 **GitOps**에서 자동으로 처리합니다.

GitOps는 **Git** 가져오기 요청을 사용하여 인프라 및 애플리케이션 구성을 관리하는 일련의 관행입니다. **GitOps**의 **Git** 리포지토리는 시스템 및 애플리케이션 구성에 사용하는 단일 정보 소스입니다. 이 **Git** 리포지토리에는 지정된 환경에서 필요한 인프라에 대한 선언적 설명과 환경을 설명된 상태에 맞게 조정하는 자동화된 프로세스가 포함되어 있습니다. 또한 시스템의 전체 상태가 포함되므로 시스템 상태에 대한 변경 내역을 보고 감사할 수 있습니다. **GitOps**를 사용하면 인프라 및 애플리케이션 구성 확산 문제를 해결할 수 있습니다.

GitOps는 인프라 및 애플리케이션 정의를 코드로 정의합니다. 그런 다음 이 코드를 사용하여 여러 작업 공간과 클러스터를 관리하여 인프라 및 애플리케이션 구성 생성 작업을 단순화합니다. 코드 원칙을 따라 **Git** 리포지토리에 클러스터 및 애플리케이션 구성을 저장한 다음 **Git** 워크플로를 따라 이러한 리포지토리를 선택한 클러스터에 적용할 수 있습니다. **Git** 리포지토리에서 소프트웨어 개발 및 유지보수의 핵심 원칙을 클러스터 및 애플리케이션 구성 파일의 생성 및 관리에 적용할 수 있습니다.

5.2.2. Red Hat OpenShift GitOps 정보

Red Hat OpenShift GitOps를 사용하면 개발, 스테이징, 프로덕션과 같은 다양한 환경의 다양한 클러스터에 애플리케이션을 배포할 때 애플리케이션의 일관성을 유지할 수 있습니다. **Red Hat OpenShift GitOps**는 구성 리포지토리를 중심으로 배포 프로세스를 구성한 후 이 프로세스를 중심 요소로 만듭니다. 항상 두 개 이상의 리포지토리가 있습니다.

1. 소스 코드가 있는 애플리케이션 리포지토리
2. 원하는 애플리케이션 상태를 정의하는 환경 구성 리포지토리

이러한 리포지토리에는 지정된 환경에서 필요한 인프라에 대한 선언적 설명이 포함되어 있습니다. 또한 환경을 설명된 상태에 맞게 조정하는 자동화된 프로세스가 포함되어 있습니다.

Red Hat OpenShift GitOps는 **Argo CD**를 사용하여 클러스터 리소스를 유지합니다. **Argo CD**는 애플리케이션의 **CI/CD**(연속 통합 및 연속 배포)에 사용되는 오픈 소스 선언 도구입니다. **Red Hat OpenShift GitOps**는 **Argo CD**를 컨트롤러로 구현하여 **Git** 리포지토리에 정의된 애플리케이션 정의 및 구성을 지속적으로 모니터링합니다. 그러면 **Argo CD**에서 이러한 구성의 지정된 상태를 클러스터의 라이브 상태와 비교합니다.

Argo CD는 지정된 상태에서 벗어난 모든 구성을 보고합니다. 이러한 보고서를 통해 관리자는 자동 또는 수동으로 구성을 정의된 상태로 다시 동기화할 수 있습니다 따라서 **Argo CD**를 사용하면 **OpenShift**

Container Platform 클러스터를 구성하는 데 사용하는 리소스와 같이 글로벌 사용자 정의 리소스를 제공할 수 있습니다.

5.2.2.1. 주요 기능

Red Hat OpenShift GitOps는 다음 작업을 자동화하는 데 도움이 됩니다.

- 클러스터의 구성, 모니터링, 스토리지 상태가 비슷한지 확인
- 여러 **OpenShift Container Platform** 클러스터에 구성 변경 사항 적용 또는 되돌리기
- 템플릿 구성을 다른 환경과 연결
- 스테이징에서 프로덕션까지 클러스터 전체에서 애플리케이션 승격

5.3. OPENSIFT GITOPS 설치

Red Hat OpenShift GitOps는 **Argo CD**를 사용하여 클러스터 **Operator**, 선택적 **OLM(Operator Lifecycle Manager) Operator** 및 사용자 관리를 포함한 특정 클러스터 범위 리소스를 관리합니다.

이 가이드에서는 **Red Hat OpenShift GitOps Operator**를 **OpenShift Container Platform** 클러스터에 설치하고 **Argo CD** 인스턴스에 로그인하는 방법을 설명합니다.

5.3.1. 웹 콘솔에서 OpenShift GitOps Operator 설치

사전 요구 사항

- **OpenShift Container Platform** 웹 콘솔에 액세스합니다.
- **cluster-admin** 역할이 있는 계정.
- **OpenShift Container Platform** 클러스터에 관리자로 로그인되어 있습니다.



주의

Argo CD Operator의 커뮤니티 버전을 이미 설치한 경우 **Red Hat OpenShift GitOps Operator**를 설치하기 전에 **Argo CD Community Operator**를 제거하십시오.

절차

1. 왼쪽 메뉴에 있는 웹 콘솔의 관리자 화면을 열고 **Operator** → **OperatorHub**로 이동합니다.
2. **OpenShift GitOps**를 검색하고 **Red Hat OpenShift GitOps** 타일을 클릭한 다음 설치를 클릭합니다.

Red Hat OpenShift GitOps는 클러스터의 모든 네임스페이스에 설치됩니다.

Red Hat OpenShift GitOps Operator를 설치한 후 **openshift-gitops** 네임스페이스에서 제공되는 즉시 사용 가능한 **Argo CD** 인스턴스가 자동으로 설정되고 콘솔 도구 모음에 **Argo CD** 아이콘이 표시됩니다. 프로젝트에서 애플리케이션에 대한 후속 **Argo CD** 인스턴스를 생성할 수 있습니다.

5.3.2. Argo CD 관리자 계정을 사용하여 Argo CD 인스턴스에 로그인

Red Hat OpenShift GitOps Operator는 **openshift-gitops** 네임스페이스에서 사용할 수 있는 즉시 사용 가능한 **Argo CD** 인스턴스를 자동으로 생성합니다.

사전 요구 사항

- 클러스터에 **Red Hat OpenShift GitOps Operator**가 설치되어 있습니다.

절차

1. 웹 콘솔의 관리자 화면에서 **Operator** → 설치된 **Operator**로 이동하여 **Red Hat OpenShift GitOps Operator**가 설치되어 있는지 확인합니다.
- 2.



메뉴 → **OpenShift GitOps** → 클러스터 **Argo CD** 로 이동합니다. **Argo CD UI**의 로그인 페이지가 새 창에 표시됩니다.

3.

Argo CD 인스턴스의 암호를 가져옵니다.

- a. 콘솔의 왼쪽 패널에서 모드 전환기를 사용하여 개발자 화면으로 전환합니다.
- b. 프로젝트 드롭다운 목록을 사용하고 **openshift-gitops** 프로젝트를 선택합니다.
- c. 왼쪽 탐색 패널을 사용하여 시크릿 페이지로 이동합니다.
- d. 암호를 표시할 **argocd-cluster-cluster** 인스턴스를 선택합니다.
- e. 암호를 복사합니다.



참고

OpenShift Container Platform 인증 정보로 로그인하려면 **Argo CD** 사용자 인터페이스에서 **LOG IN VIA OPENSIFT** 옵션을 선택합니다.

1.

이 암호와 **admin**을 사용자 이름으로 사용하여 새 창에서 **Argo CD UI**에 로그인합니다.



참고

동일한 네임스페이스에 두 개의 **Argo CD CR**을 생성할 수 없습니다.

5.4. OPENSIFT GITOPS 설치 제거

Red Hat OpenShift GitOps Operator 설치 제거는 2단계 프로세스입니다.

1.

Red Hat OpenShift GitOps Operator의 기본 네임스페이스 아래에 추가된 Argo CD 인스턴스를 삭제합니다.

2.

Red Hat OpenShift GitOps Operator를 설치 제거합니다.

Operator를 설치 제거하는 것만으로는 생성된 Argo CD 인스턴스가 제거되지 않습니다.

5.4.1. Argo CD 인스턴스 삭제

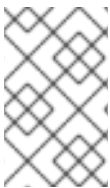
GitOps Operator의 네임스페이스에 추가된 Argo CD 인스턴스를 삭제합니다.

프로세스

1.

터미널 유형에서 다음 명령을 입력합니다.

```
$ oc delete gitopsservice cluster -n openshift-gitops
```



참고

웹 콘솔 UI에서 Argo CD 클러스터를 삭제할 수 없습니다.

명령이 성공적으로 실행되면 모든 Argo CD 인스턴스가 openshift-gitops 네임스페이스에서 삭제됩니다.

동일한 명령을 사용하여 다른 네임스페이스에서 다른 Argo CD 인스턴스를 삭제합니다.

```
$ oc delete gitopsservice cluster -n <namespace>
```

5.4.2. GitOps Operator 설치 제거

프로세스

1.

Operators → OperatorHub 페이지에서 키워드로 필터링 상자를 사용하여 Red Hat OpenShift GitOps Operator 타일을 검색합니다.

2.

Red Hat OpenShift GitOps Operator 타일을 클릭합니다. **Operator** 타일은 **Operator**가 설치되었음을 나타냅니다.

3.

Red Hat OpenShift GitOps Operator 설명자 페이지에서 설치 제거를 클릭합니다.

추가 리소스

- **OpenShift Container Platform**에서 **Operator**를 제거하는 방법에 대한 자세한 내용은 [클러스터에서 Operator 삭제](#) 섹션에서 확인할 수 있습니다.

5.5. 클러스터 구성으로 애플리케이션을 배포하여 OPENSIFT 클러스터 구성

Red Hat OpenShift GitOps를 사용하면 **Argo CD**를 구성하여 **Git** 디렉터리의 콘텐츠를 클러스터의 사용자 지정 구성이 포함된 애플리케이션과 반복적으로 동기화할 수 있습니다.

사전 요구 사항

- 관리자로 제품 인타이틀먼트 클러스터에 로그인했습니다.
- 클러스터에 **gitops-title Operator**가 설치되어 있습니다.
- **Argo CD** 인스턴스에 로그인했습니다.

5.5.1. Argo CD 인스턴스를 사용하여 클러스터 범위 리소스 관리

클러스터 범위 리소스를 관리하려면 **gitops-title Operator**의 기존 **Subscription** 오브젝트를 업데이트하고 **spec** 섹션의 **ARGOCD_CLUSTER_CONFIG_NAMESPACES** 환경 변수에 **Argo CD** 인스턴스의 네임스페이스를 추가합니다.

절차

1. 웹 콘솔의 관리자 화면에서 **Operator** → 설치된 **Operator** → **Red Hat OpenShift GitOps** → 서브스크립션으로 이동합니다.
2. 작업 드롭다운 메뉴를 클릭한 다음 서브스크립션 편집 을 클릭합니다.

3.

openshift-gitops-operator Subscription 세부 정보 페이지의 **YAML** 탭에서 **Argo CD** 인스턴스의 네임스페이스를 **spec** 섹션의 **ARGOCD_CLUSTER_CONFIG_NAMESPACES** 환경 변수에 추가하여 서브스크립션 **YAML** 파일을 편집합니다.

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-gitops-operator
  namespace: openshift-operators
...
spec:
  config:
    env:
      - name: ARGOCD_CLUSTER_CONFIG_NAMESPACES
        value: openshift-gitops, <list of namespaces of cluster-scoped Argo CD instances>
...
```

4.

Argo 인스턴스가 클러스터 역할로 구성되어 클러스터 범위 리소스를 관리하는지 확인하려면 다음 단계를 수행합니다.

a.

사용자 관리 → 역할로 이동하고 필터 드롭다운 메뉴에서 클러스터 전체 역할을 선택합니다.

b.

이름으로 검색 필드를 사용하여 **argocd-application-controller** 를 검색합니다.

Roles (역할) 페이지에 생성된 클러스터 역할이 표시됩니다.

작은 정보

또는 **OpenShift CLI**에서 다음 명령을 실행합니다.

```
oc auth can-i create oauth -n openshift-gitops --as
system:serviceaccount:openshift-gitops:openshift-gitops-argocd-application-
controller
```

출력 **yes** 는 클러스터 범위 리소스를 관리하기 위해 **Argo** 인스턴스가 클러스터 역할로 구성되었는지 확인합니다. 구성을 확인하고 필요에 따라 필요한 단계를 수행합니다.

5.5.2. Argo cd 인스턴스의 기본 권한

기본적으로 **Argo CD** 인스턴스에는 다음 권한이 있습니다.

- **Argo CD** 인스턴스에는 배포된 네임스페이스에서만 리소스를 관리할 수 있는 관리자 권한이 있습니다. 예를 들어 **foo** 네임스페이스에 배포된 **Argo CD** 인스턴스에는 해당 네임스페이스에 대해서만 리소스를 관리할 수 있는 관리자 권한이 있습니다.
- **Argo CD**에는 리소스에 대한 클러스터 전체 읽기 권한이 있어야 제대로 작동하기 때문에 **Argo CD**에는 다음과 같은 클러스터 범위 권한이 있습니다.

```
- verbs:
  - get
  - list
  - watch
apiGroups:
  - '*'
resources:
  - '*'
- verbs:
  - get
  - list
nonResourceURLs:
  - '*'
```

참고

- **Argo CD**가 관리하려는 네임스페이스와 리소스로만 쓰기 권한이 제한되도록 **Argo CD** 및 **argocd-server** 및 **argocd-application-controller** 구성 요소에서 사용하는 클러스터 역할을 편집할 수 있습니다.

```
$ oc edit clusterrole argocd-server
$ oc edit clusterrole argocd-application-controller
```

5.5.3. 클러스터 수준에서 **Argo CD** 인스턴스 실행

Red Hat OpenShift GitOps Operator에서 설치한 기본 **Argo CD** 인스턴스 및 관련 컨트롤러는 이제 간단한 구성 토글을 설정하여 클러스터의 인프라 노드에서 실행할 수 있습니다.

절차

1. 기존 노드에 레이블을 지정합니다.

```
$ oc label node <node-name> node-role.kubernetes.io/infra=""
```

2.

선택 사항: 필요한 경우 테인트를 적용하고 인프라 노드에 워크로드를 격리하고 다른 워크로드가 이러한 노드에서 예약되지 않도록 할 수도 있습니다.

```
$ oc adm taint nodes -l node-role.kubernetes.io/infra \
infra=reserved:NoSchedule infra=reserved:NoExecute
```

3.

GitOpsService 사용자 정의 리소스에 runOnInfra 토글을 추가합니다.

```
apiVersion: pipelines.openshift.io/v1alpha1
kind: GitopsService
metadata:
  name: cluster
spec:
  runOnInfra: true
```

4.

선택 사항: 노드에 테인트를 추가한 경우 GitOpsService 사용자 정의 리소스에 허용 오차를 추가합니다. 예를 들면 다음과 같습니다.

```
spec:
  runOnInfra: true
  tolerations:
  - effect: NoSchedule
    key: infra
    value: reserved
  - effect: NoExecute
    key: infra
    value: reserved
```

5.

콘솔 UI에서 Pod → Pod 세부 정보를 확인하여 openshift-gitops 네임스페이스의 워크로드가 인프라 노드에서 예약되었는지 확인합니다.



참고

기본 Argo CD 사용자 정의 리소스에 수동으로 추가된 nodeSelector 및 허용 오차는 GitOpsService 사용자 정의 리소스의 토글 및 허용 오차를 덮어씁니다.

5.5.4. Argo CD 대시보드를 사용하여 애플리케이션 생성

Argo CD는 애플리케이션을 만들 수 있는 대시보드를 제공합니다.

이 샘플 워크플로에서는 Argo CD를 구성하여 cluster 디렉터리의 콘텐츠를 cluster-configs 애플리

케이션과 반복적으로 동기화하는 프로세스를 보여줍니다. 디렉터리는 웹 콘솔의



메뉴에 있는 **Red Hat** 개발자 블로그에 링크를 추가하는 **OpenShift Container Platform** 웹 콘솔 클러스터 구성을 정의하고 클러스터에 **spring-petclinic** 네임스페이스를 정의합니다.

절차

1. **Argo CD** 대시보드에서 **NEW APP** (새 앱)을 클릭하여 새 **Argo CD** 애플리케이션을 추가합니다.
2. 이 워크플로의 경우 다음 구성을 사용하여 **cluster-configs** 애플리케이션을 생성합니다.

애플리케이션 이름

cluster-configs

프로젝트

default

동기화 정책

Manual

리포지터리 URL

<https://github.com/redhat-developer/openshift-gitops-getting-started>

버전

HEAD

경로

cluster

대상

<https://kubernetes.default.svc>

네임스페이스

spring-petclinic

디렉토리 반복

checked

3. **CREATE** (생성)를 클릭하여 애플리케이션을 생성합니다.
4. 웹 콘솔의 관리자 화면을 열고 왼쪽 메뉴에 있는 관리 → 네임스페이스로 이동합니다.
5. 네임스페이스를 검색하고 선택한 다음 **openshift -gitops** 네임스페이스의 **Argo CD** 인스턴스가 네임스페이스를 관리할 수 있도록 **Label(레이블)** 필드에 **argocd.argoproj.io/managed-by=openshift -gitops** 를 입력합니다.

5.5.5. oc 툴을 사용하여 애플리케이션 생성

oc 툴을 사용하여 터미널에서 **Argo CD** 애플리케이션을 생성할 수 있습니다.

프로세스

1. [샘플 애플리케이션](#)을 다운로드합니다.

```
$ git clone git@github.com:redhat-developer/openshift-gitops-getting-started.git
```

2. 애플리케이션을 생성합니다.

```
$ oc create -f openshift-gitops-getting-started/argo/cluster.yaml
```

3. **oc get** 명령을 실행하여 생성된 애플리케이션을 검토합니다.


```
$ oc get application -n openshift-gitops
```

4. **openshift-gitops** 네임스페이스의 **Argo CD** 인스턴스가 이를 관리할 수 있도록 애플리케이션이 배포된 네임스페이스에 레이블을 추가합니다.

```
$ oc label namespace spring-petclinic argocd.argoproj.io/managed-by=openshift-gitops
```

5.5.6. Git 리포지토리와 애플리케이션 동기화

프로세스

1. **Argo CD** 대시보드에서 **cluster-configs Argo CD** 애플리케이션은 **Missing** 및 **OutOfSync** 상태입니다. 애플리케이션이 수동 동기화 정책으로 구성되었으므로 **Argo CD**는 자동으로 동기화되지 않습니다.
2. **cluster-configs** 타일에서 **SYNC** 를 클릭하고 변경 사항을 검토한 다음 **SYNCHRONIZE** 를 클릭합니다. **Argo CD**는 **Git** 리포지토리의 모든 변경 사항을 자동으로 감지합니다. 구성이 변경되면 **Argo CD**는 **cluster-configs**의 상태를 **OutOfSync**로 변경합니다. **Argo CD**의 동기화 정책을 수정하여 **Git** 리포지토리에서 클러스터에 변경 사항을 자동으로 적용할 수 있습니다.
3. **cluster-configs Argo CD** 애플리케이션이 이제 **Healthy** 및 **Synced** 상태가 됩니다. **cluster-configs** 타일을 클릭하여 동기화된 리소스의 세부 정보와 클러스터의 상태를 확인합니다.
4. **OpenShift Container Platform** 웹 콘솔로 이동하여  을 클릭하여 **Red Hat** 개발자 블로그 - **Kubernetes** 에 대한 링크가 있는지 확인합니다.
5. 프로젝트 페이지로 이동하여 **spring-petclinic** 네임스페이스를 검색하여 클러스터에 추가되었는지 확인합니다.

클러스터 구성이 클러스터에 성공적으로 동기화됩니다.

5.5.7. 클러스터 구성에 대한 내장 권한

기본적으로 **Argo CD** 인스턴스에는 클러스터 **Operator**, 선택적 **OLM Operator** 및 사용자 관리와 같은 특정 클러스터 범위 리소스를 관리할 수 있는 권한이 있습니다.



참고

Argo CD에는 **cluster-admin** 권한이 없습니다.

Argo CD 인스턴스에 대한 권한:

Resources	설명
리소스 그룹	사용자 또는 관리자 구성
operators.coreos.com	OLM에서 관리하는 선택적 Operator
user.openshift.io , rbac.authorization.k8s.io	그룹, 사용자 및 권한
config.openshift.io	클러스터 전체 빌드 구성, 레지스트리 구성 및 스케줄러 정책을 구성하는 데 사용되는 CVO에서 관리하는 컨트롤 플레인 Operator
storage.k8s.io	스토리지
console.openshift.io	콘솔 사용자 지정

5.5.8. 클러스터 구성에 대한 권한 추가

Argo CD 인스턴스에 대한 권한을 부여하여 클러스터 구성을 관리할 수 있습니다. 추가 권한으로 클러스터 역할을 생성한 다음 새 클러스터 역할 바인딩을 생성하여 클러스터 역할을 서비스 계정과 연결합니다.

프로세스

1. **OpenShift Container Platform** 웹 콘솔에 관리자로 로그인합니다.
2. 웹 콘솔에서 사용자 관리 → 역할 → 역할 생성을 선택합니다. 다음 **ClusterRole** YAML 템플릿을 사용하여 추가 권한을 지정하는 규칙을 추가합니다.

```

apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secrets-cluster-role
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: [""]
    
```

3. 생성을 클릭하여 클러스터 역할을 추가합니다.
4. 이제 클러스터 역할 바인딩을 생성합니다. 웹 콘솔에서 사용자 관리 → 역할 바인딩 → 바인딩 생성을 선택합니다.

5. 프로젝트 드롭다운에서 모든 프로젝트를 선택합니다.
6. 바인딩 생성을 클릭합니다.
7. 클러스터 전체 역할 바인딩(**ClusterRoleBinding**)으로 바인딩 유형을 선택합니다.
8. **RoleBinding** 이름에 고유한 값을 입력합니다.
9. 드롭다운 목록에서 새로 생성된 클러스터 역할 또는 기존 클러스터 역할을 선택합니다.
10. **Subject as ServiceAccount** 를 선택하고 주체 네임스페이스 와 이름을 입력합니다.
 - a. 제목 네임스페이스:**openshift-gitops**
 - b. 제목 이름:**openshift-gitops-argocd-application-controller**
11. 생성을 클릭합니다. **ClusterRoleBinding** 오브젝트의 **YAML** 파일은 다음과 같습니다.

```

kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: cluster-role-binding
subjects:
  - kind: ServiceAccount
    name: openshift-gitops-argocd-application-controller
    namespace: openshift-gitops
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: admin

```

5.5.9. Red Hat OpenShift GitOps를 사용하여 OLM Operator 설치

클러스터 구성을 사용하는 **Red Hat OpenShift GitOps**는 특정 클러스터 범위 리소스를 관리하고 클러스터 **Operator** 또는 네임스페이스 범위 **OLM Operator**를 설치합니다.

클러스터 관리자로서 **Tekton**과 같은 **OLM Operator**를 설치해야 하는 경우를 고려하십시오. **OpenShift Container Platform** 웹 콘솔을 사용하여 **Tekton Operator** 또는 **OpenShift CLI**를 수동으로 설치하여 클러스터에 **Tekton** 서브스크립션 및 **Tekton Operator group**을 수동으로 설치합니다.

Red Hat OpenShift GitOps는 **Kubernetes** 리소스를 **Git** 리포지토리에 배치합니다. 클러스터 관리자는 **Red Hat OpenShift GitOps**를 사용하여 수동 절차 없이 다른 **OLM Operator**의 설치를 관리하고 자동화합니다. 예를 들어 **Red Hat OpenShift GitOps**를 사용하여 **Git** 리포지토리에 **Tekton** 서브스크립션을 배치한 후 **Red Hat OpenShift GitOps**는 **Git** 리포지토리에서 이 **Tekton** 서브스크립션을 자동으로 가져와서 클러스터에 **Tekton Operator**를 설치합니다.

5.5.9.1. 클러스터 범위 Operator 설치

OLM(Operator Lifecycle Manager)은 클러스터 범위 **Operator**에 **openshift-operators** 네임스페이스에서 기본 **global-operators Operator group**을 사용합니다. 따라서 **Gitops** 리포지토리에서 **OperatorGroup** 리소스를 관리할 필요가 없습니다. 그러나 네임스페이스 범위 **Operator**의 경우 해당 네임스페이스의 **OperatorGroup** 리소스를 관리해야 합니다.

클러스터 범위 **Operator**를 설치하려면 **Git** 리포지토리에 필요한 **Operator**의 **Subscription** 리소스를 생성하고 배치합니다.

예제: **Grafana Operator** 서브스크립션

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: grafana
spec:
  channel: v4
  installPlanApproval: Automatic
  name: grafana-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

5.5.9.2. namespace-scoped Operator 설치

네임스페이스 범위 **Operator**를 설치하려면 **Git** 리포지토리에 필요한 **Operator**의 **Subscription** 및 **OperatorGroup** 리소스를 생성하고 배치합니다.

예제: Ansible Automation Platform Resource Operator

```

...
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
  name: ansible-automation-platform
...
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: ansible-automation-platform-operator
  namespace: ansible-automation-platform
spec:
  targetNamespaces:
    - ansible-automation-platform
...
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ansible-automation-platform
  namespace: ansible-automation-platform
spec:
  channel: patch-me
  installPlanApproval: Automatic
  name: ansible-automation-platform-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
...

```

중요

Red Hat OpenShift GitOps를 사용하여 여러 Operator를 배포하는 경우 해당 네임스페이스에 단일 Operator 그룹만 생성해야 합니다. 단일 네임스페이스에 두 개 이상의 Operator group이 있는 경우 해당 네임스페이스에서 생성된 모든 CSV는 TooManyOperatorGroups 이유와 함께 실패 상태로 전환됩니다. 해당 네임스페이스의 Operator groups 수가 1에 도달하면 이전 실패 상태 CSV가 모두 pending 상태로 전환됩니다. Operator 설치를 완료하려면 보류 중인 설치 계획을 수동으로 승인해야 합니다.

5.6. ARGO CD로 SPRING BOOT 애플리케이션 배포

Argo CD를 사용하면 Argo CD 대시보드를 사용하거나 oc 툴을 사용하여 애플리케이션을 OpenShift 클러스터에 배포할 수 있습니다.

사전 요구 사항

- **Red Hat OpenShift GitOps가 클러스터에 설치되어 있습니다.**
- **Argo CD 인스턴스에 로그인합니다.**

5.6.1. Argo CD 대시보드를 사용하여 애플리케이션 생성

Argo CD는 애플리케이션을 만들 수 있는 대시보드를 제공합니다.

이 샘플 워크플로에서는 **Argo CD**를 구성하여 **cluster** 디렉터리의 콘텐츠를 **cluster-configs** 애플리케이션과 반복적으로 동기화하는 프로세스를 보여줍니다. 디렉터리는 웹 콘솔의



메뉴에 있는 **Red Hat** 개발자 블로그에 링크를 추가하는 **OpenShift Container Platform** 웹 콘솔 클러스터 구성을 정의하고 클러스터에 **spring-petclinic** 네임스페이스를 정의합니다.

프로세스

1. **Argo CD** 대시보드에서 **NEW APP** (새 앱)을 클릭하여 새 **Argo CD** 애플리케이션을 추가합니다.
2. 이 워크플로의 경우 다음 구성을 사용하여 **cluster-configs** 애플리케이션을 생성합니다.

애플리케이션 이름

cluster-configs

프로젝트

default

동기화 정책

Manual

리포지터리 URL

<https://github.com/redhat-developer/openshift-gitops-getting-started>

버전

HEAD

경로

cluster

대상

<https://kubernetes.default.svc>

네임스페이스

spring-petclinic

디렉토리 반복

checked

3.

이 워크플로의 경우 다음 구성을 사용하여 **Spring-petclinic** 애플리케이션을 생성합니다.

애플리케이션 이름

spring-petclinic

프로젝트

default

동기화 정책

자동

리포지터리 URL

<https://github.com/redhat-developer/openshift-gitops-getting-started>

버전

HEAD

경로

app

대상

<https://kubernetes.default.svc>

네임스페이스

spring-petclinic

4. **CREATE** (생성)를 클릭하여 애플리케이션을 생성합니다.
5. 웹 콘솔의 관리자 화면을 열고 왼쪽 메뉴에 있는 관리 → 네임스페이스 로 이동합니다.
6. 네임스페이스를 검색하고 선택한 다음 **openshift -gitops** 네임스페이스의 **Argo CD** 인스턴스가 네임스페이스를 관리할 수 있도록 **Label(레이블)** 필드에 **argocd.argoproj.io/managed-by=openshift -gitops** 를 입력합니다.

5.6.2. oc 툴을 사용하여 애플리케이션 생성

oc 툴을 사용하여 터미널에서 **Argo CD** 애플리케이션을 생성할 수 있습니다.

프로세스

1. [샘플 애플리케이션](#)을 다운로드합니다.

```
$ git clone git@github.com:redhat-developer/openshift-gitops-getting-started.git
```

2. 애플리케이션을 생성합니다.

```
$ oc create -f openshift-gitops-getting-started/argo/app.yaml
```

```
$ oc create -f openshift-gitops-getting-started/argo/cluster.yaml
```

3. **oc get** 명령을 실행하여 생성된 애플리케이션을 검토합니다.

```
$ oc get application -n openshift-gitops
```

4.

openshift-gitops 네임스페이스의 **Argo CD** 인스턴스가 이를 관리할 수 있도록 애플리케이션이 배포된 네임스페이스에 레이블을 추가합니다.

```
$ oc label namespace spring-petclinic argocd.argoproj.io/managed-by=openshift-gitops
```

```
$ oc label namespace spring-petclinic argocd.argoproj.io/managed-by=openshift-gitops
```

5.6.3. Argo CD 자동 복구 동작 확인

Argo CD는 배포된 애플리케이션의 상태를 지속적으로 모니터링하고, **Git**에서 지정된 매니페스트와 클러스터의 실시간 변경 사항 간의 차이점을 감지한 다음 자동으로 수정합니다. 이 동작을 자동 복구라고 합니다.

Argo CD에서 자동 복구 동작을 테스트하고 관찰할 수 있습니다.

사전 요구 사항

•

샘플 **app-spring-petclinic** 애플리케이션이 배포 및 구성되어 있습니다.

프로세스

1.

Argo CD 대시보드에서 애플리케이션에 **Synced** 상태가 있는지 확인합니다.

2.

Argo CD 대시보드에서 **app-spring-petclinic** 타일을 클릭하여 클러스터에 배포된 애플리케이션 리소스를 확인합니다.

3.

OpenShift Container Platform 웹 콘솔에서 개발자 화면으로 이동합니다.

4.

Spring PetClinic 배포를 수정하고 **Git** 리포지토리의 **app/** 디렉터리에 대한 변경 사항을 커밋합니다. **Argo CD**는 클러스터에 변경 사항을 자동으로 배포합니다.

a.

OpenShift GitOps를 시작하기 리포지토리를 분기합니다.

- b. **deployment.yaml** 파일에서 **failureThreshold** 값을 5로 변경합니다.
- c. 배포 클러스터에서 다음 명령을 실행하여 **failureThreshold** 필드의 변경된 값을 확인합니다.

```
$ oc edit deployment spring-petclinic -n spring-petclinic
```

- 5. OpenShift Container Platform 웹 콘솔에서 애플리케이션을 모니터링하면서 클러스터에서 배포를 수정하고 2개의 pod로 확장하여 자동 복구 동작을 테스트합니다.

- a. 다음 명령을 실행하여 배포 상태를 확인합니다.

```
$ oc scale deployment spring-petclinic --replicas 2 -n spring-petclinic
```

- b. OpenShift Container Platform 웹 콘솔에서 배포는 두 개의 pod로 확장되었다가 즉시 하나의 pod로 축소됩니다. Argo CD는 Git 리포지토리와 차이점을 감지하고 OpenShift Container Platform 클러스터에서 애플리케이션을 자동 복구했습니다.

- 6. Argo CD 대시보드에서 **app-spring-petclinic** 타일 → **APP DETAILS** → **EVENTS**를 클릭합니다. **KialiEVENTS(S)** 탭에는 다음 이벤트가 표시됩니다. Argo CD는 클러스터에서 배포 리소스를 동기화하지 않은 다음 Git 리포지토리를 다시 동기화하여 수정합니다.

5.7. DEX를 사용하여 ARGO CD에 대한 SSO 구성

Red Hat OpenShift GitOps Operator가 설치되면 Argo CD에서 admin 권한이 있는 사용자를 자동으로 생성합니다. 클러스터 관리자는 여러 사용자를 관리하기 위해 Argo CD를 사용하여 SSO(Single Sign-On)를 구성할 수 있습니다.

5.7.1. Dex OpenShift OAuth 커넥터 활성화

DEX는 플랫폼에서 제공하는 OAuth 서버를 확인하여 OpenShift 내에 정의된 사용자와 그룹을 사용합니다. 다음 예제에서는 예제 구성과 함께 Dex의 속성을 보여줍니다.

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
```

```

example: openshift-oauth
spec:
  dex:
    openShiftOAuth: true ①
    groups: ②
    - default
  rbac: ③
  defaultPolicy: 'role:readonly'
  policy: |
    g, cluster-admins, role:admin
  scopes: '[groups]'

```

①

`openShiftOAuth` 속성은 값이 `true` 로 설정된 경우 기본 제공 **OpenShift OAuth** 서버를 자동으로 구성하도록 **Operator**를 트리거합니다.

②

`groups` 속성은 `groups` 목록에 있는 한 그룹 또는 모든 그룹에 사용자를 할당합니다.

③

RBAC 정책 속성은 **Argo CD** 클러스터의 **admin** 역할을 **OpenShift cluster-admins** 그룹의 사용자에게 할당합니다.

5.7.1.1. 특정 역할에 사용자 매핑

Argo CD는 직접 **ClusterRoleBinding** 역할이 있는 경우 사용자를 특정 역할에 매핑할 수 없습니다. **OpenShift**를 통해 **SSO**에서 `role:admin`으로 역할을 수동으로 변경할 수 있습니다.

절차

1. `cluster-admins` 라는 그룹을 만듭니다.

```
$ oc adm groups new cluster-admins
```

2. 사용자를 그룹에 추가합니다.

```
$ oc adm groups add-users cluster-admins USER
```

3. `cluster-admin ClusterRole` 을 그룹에 적용합니다.

-

```
$ oc adm policy add-cluster-role-to-group cluster-admin cluster-admins
```

5.7.2. Dex 비활성화

DEX는 **Operator**에서 생성한 모든 **Argo CD** 인스턴스에 기본적으로 설치됩니다. **Dex**를 비활성화할 수 있습니다.

절차

- **Operator**의 **YAML** 리소스에서 환경 변수 **DISABLE_DEX** 를 **true**로 설정합니다.

```
spec:
  config:
    env:
      - name: DISABLE_DEX
        value: "true"
```

5.8. KEYCLOAK을 사용하여 ARGO CD에 대한 SSO 구성

Red Hat OpenShift GitOps Operator가 설치되면 **Argo CD**에서 **admin** 권한이 있는 사용자를 자동으로 생성합니다. 클러스터 관리자는 여러 사용자를 관리하기 위해 **Argo CD**를 사용하여 **SSO(Single Sign-On)**를 구성할 수 있습니다.

사전 요구 사항

- **Red Hat SSO**가 클러스터에 설치되어 있습니다.
- **Argo CD**가 클러스터에 설치되어 있습니다.

5.8.1. Keycloak에서 새 클라이언트 구성

DEX는 **Operator**에서 생성한 모든 **Argo CD** 인스턴스에 기본적으로 설치됩니다. 그러나 **Dex** 구성을 삭제하고 대신 **OpenShift** 인증 정보를 사용하여 **Argo CD**에 로그인하도록 **Keycloak**을 추가할 수 있습니다. **Keycloak**은 **Argo CD**와 **OpenShift** 간의 **ID** 브로커 역할을 합니다.

프로세스

Keycloak을 구성하려면 다음 단계를 따르십시오.

1.

Argo CD CR(사용자 정의 리소스)에서 다음 섹션을 제거하여 **Dex** 구성을 삭제하고 **CR**을 저장합니다.

```
dex:
  openShiftOAuth: true
resources:
  limits:
    cpu:
    memory:
  requests:
    cpu:
    memory:
```

2.

Argo CD CR을 편집하고 **provider** 매개 변수의 값을 **keycloak** 로 업데이트하여 **Keycloak** 을 구성합니다. 예를 들면 다음과 같습니다.

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
  labels:
    example: basic
spec:
  sso:
    provider: keycloak
  server:
    route:
      enabled: true
```



참고

Keycloak 인스턴스를 설치하고 실행하는 데 **2-3분**이 걸립니다.

5.8.2. Keycloak에 로그인

Keycloak 콘솔에 로그인하여 **ID** 또는 역할을 관리하고 다양한 역할에 할당된 권한을 정의합니다.

사전 요구 사항

- **Dex**의 기본 구성이 제거됩니다.
- **Argo CD CR**은 **Keycloak SSO** 공급자를 사용하도록 구성해야 합니다.

프로세스

- 1. 로그인을 위한 **Keycloak** 경로 **URL**을 가져옵니다.

```
$ oc -n argocd get route keycloak
```

NAME	HOST/PORT	PATH	SERVICES	PORT
keycloak	keycloak-default.apps.ci-ln-*****.origin-ci-int-aws.dev.**.com			keycloak
<all>	reencrypt	None		

- 2. 사용자 이름과 암호를 환경 변수로 저장하는 **Keycloak Pod** 이름을 가져옵니다.

```
$ oc -n argocd get pods
```

NAME	READY	STATUS	RESTARTS	AGE
keycloak-1-2sjcl	1/1	Running	0	45m

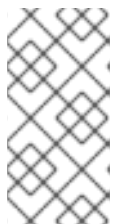
- a. **Keycloak** 사용자 이름을 가져옵니다.

```
$ oc -n argocd exec keycloak-1-2sjcl -- "env" | grep SSO_ADMIN_USERNAME
SSO_ADMIN_USERNAME=Cqid54lh
```

- b. **Keycloak** 암호를 가져옵니다.

```
$ oc -n argocd exec keycloak-1-2sjcl -- "env" | grep SSO_ADMIN_PASSWORD
SSO_ADMIN_PASSWORD=GVXxHifH
```

- 3. 로그인 페이지에서 **LOG IN VIA KEYCLOAK(VIA KEYCLOAK 로그인)**를 클릭합니다.



참고

Keycloak 인스턴스가 준비된 후 **LOGIN VIA KEYCLOAK** 옵션만 표시됩니다.

- 4. **Login with OpenShift (OpenShift로 로그인)**를 클릭합니다.



참고

kubeadmin 을 사용한 로그인은 지원되지 않습니다.

5. 로그인할 **OpenShift** 자격 증명을 입력합니다.
6. 선택 사항: 기본적으로 **Argo CD**에 로그인한 모든 사용자에게는 읽기 전용 액세스 권한이 있습니다. **argocd-rbac-cm** 구성 맵을 업데이트하여 사용자 수준 액세스를 관리할 수 있습니다.

```
policy.csv:
<name>, <email>, role:admin
```

5.8.3. 연결이 끊긴 클러스터에서 OpenShift OAuth 서버와 Keycloak 통합

연결이 끊긴 클러스터에서 **Keycloak**은 프록시를 통해 **OpenShift OAuth** 서버와 통신합니다.

프로세스

다음 단계에 따라 **Keycloak**을 **OpenShift OAuth** 서버와 통합하십시오.

1. **Keycloak Pod**에 로그인합니다.

```
$ oc exec -it dc/keycloak -n argocd -- /bin/bash
```

2. **JBoss CLI** 도구를 시작하여 프록시 매핑을 설정합니다.

```
/opt/eap/bin/jboss-cli.sh
```

3. **JBoss CLI** 틀에서 다음 명령을 실행하여 포함된 독립 실행형 서버를 시작합니다.

```
embed-server --server-config=standalone-openshift.xml
```

4. **OpenShift OAuth** 서버 호스트에 대한 프록시 매핑을 설정합니다.

```
/subsystem=keycloak-server/spi=connectionsHttpClient/provider=default:write-attribute(name=properties.proxy-mappings,value=["<oauth-server-hostname>;http://<proxy-server-host>:<proxy-server-port>"])
```

5.

포함된 서버를 중지합니다.

```
quit
```

6.

JBoss CLI 도구를 다시 로드하여 프록시 매핑을 적용합니다.

```
/opt/eap/bin/jboss-cli.sh --connect --command=:reload
```

5.8.4. Keycloak 제거

Argo CD CR(사용자 정의 리소스) 파일에서 **SSO** 필드를 제거하여 **Keycloak** 리소스 및 관련 구성을 삭제할 수 있습니다. **SSO** 필드를 제거한 후 파일의 값은 다음과 유사합니다.

```
apiVersion: argoproj.io/v1alpha1
kind: ArgoCD
metadata:
  name: example-argocd
labels:
  example: basic
spec:
  server:
    route:
      enabled: true
```

참고

이 방법을 사용하여 생성한 **Keycloak** 애플리케이션은 현재 영구적이지 않습니다. **Argo CD Keycloak** 영역에서 생성된 추가 구성은 서버를 다시 시작할 때 삭제됩니다.

5.9. 인프라 노드에서 GITOPS 컨트롤 플레인 워크로드 실행

인프라 노드를 사용하여 서브스크립션 수에 대한 추가 청구 비용을 방지할 수 있습니다.

OpenShift Container Platform을 사용하여 **Red Hat OpenShift GitOps Operator**가 설치한 인프라 노드에서 특정 워크로드를 실행할 수 있습니다. 이는 해당 네임스페이스의 기본 **Argo CD** 인스턴스를 포

함하여 **openshift-gitops** 네임스페이스에 기본적으로 **Red Hat OpenShift GitOps Operator**가 설치한 워크로드를 구성됩니다.



참고

사용자 네임스페이스에 설치된 다른 **Argo CD** 인스턴스는 인프라 노드에서 실행할 수 없습니다.

5.9.1. GitOps 워크로드를 인프라 노드로 이동

Red Hat OpenShift GitOps에서 설치한 기본 워크로드를 인프라 노드로 이동할 수 있습니다. 이동할 수 있는 워크로드는 다음과 같습니다.

- **Kam deployment**
- **클러스터 배포 (backend service)**
- **openshift-gitops-applicationset-controller deployment**
- **openshift-gitops-dex-server deployment**
- **openshift-gitops-redis deployment**
- **openshift-gitops-redis-ha-haproxy deployment**
- **openshift-gitops-repo-sever deployment**
- **openshift-gitops-server deployment**
- **openshift-gitops-application-controller statefulset**

- **openshift-gitops-redis-server statefulset**

프로세스

1. 다음 명령을 실행하여 기존 노드를 인프라로 지정합니다.

```
$ oc label node <node-name> node-role.kubernetes.io/infra=
```

2. **GitOpsService CR**(사용자 정의 리소스)을 편집하여 인프라 노드 선택기를 추가합니다.

```
$ oc edit gitopsservice -n openshift-gitops
```

3. **GitOpsService CR** 파일에서 **runOnInfra** 필드를 **spec** 섹션에 추가하고 **true** 로 설정합니다. 이 필드는 **openshift-gitops** 네임스페이스의 워크로드를 인프라 노드로 이동합니다.

```
apiVersion: pipelines.openshift.io/v1alpha1
kind: GitopsService
metadata:
  name: cluster
spec:
  runOnInfra: true
```

4. 선택 사항: 테인트를 적용하고 인프라 노드에 워크로드를 분리하고 다른 워크로드가 이러한 노드에 예약되지 않도록 합니다.

```
$ oc adm taint nodes -l node-role.kubernetes.io/infra
infra=reserved:NoSchedule infra=reserved:NoExecute
```

5. 선택 사항: 노드에 테인트를 적용하는 경우 **GitOpsService CR**에 허용 오차를 추가할 수 있습니다.

```
spec:
  runOnInfra: true
  tolerations:
    - effect: NoSchedule
      key: infra
      value: reserved
    - effect: NoExecute
      key: infra
      value: reserved
```

Red Hat OpenShift GitOps 네임스페이스의 인프라 노드에 워크로드가 예약되었는지 확인하려면 포드 이름을 클릭하고 노드 선택기 및 허용 오차가 추가되었는지 확인합니다.



참고

기본 **Argo CD CR**에 수동으로 추가된 노드 선택기 및 허용 오차는 **GitOpsService CR**의 토글과 허용 오차를 통해 덮어씁니다.

5.10. GITOPS OPERATOR의 크기 조정 요구 사항

크기 조정 요구 사항 페이지에는 **OpenShift Container Platform**에 **Red Hat OpenShift GitOps**를 설치하기 위한 크기 조정 요구 사항이 표시됩니다. 또한 **GitOps Operator**에서 인스턴스화한 기본 **ArgoCD** 인스턴스의 크기 조정 세부 정보도 제공합니다.

5.10.1. GitOps의 크기 조정 요구 사항

Red Hat OpenShift GitOps는 클라우드 네이티브 애플리케이션에 대한 연속 배포를 구현하는 선언적 방법입니다. **GitOps**를 통해 애플리케이션의 **CPU** 및 메모리 요구 사항을 정의하고 구성할 수 있습니다.

Red Hat OpenShift GitOps Operator를 설치할 때마다 네임스페이스의 리소스가 정의된 제한 내에 설치됩니다. 기본 설치에서 제한 또는 요청을 설정하지 않으면 **Operator**가 할당량이 있는 네임스페이스 내에서 실패합니다. 리소스가 충분하지 않으면 클러스터에서 **ArgoCD** 관련 **Pod**를 예약할 수 없습니다. 다음 표에서는 기본 워크로드에 대한 리소스 요청 및 제한에 대해 자세히 설명합니다.

워크로드	CPU 요청	CPU 제한	메모리 요청	메모리 제한
argocd-application-controller	1	2	1024M	2048M
applicationset-controller	1	2	512M	1024M
argocd-server	0.125	0.5	128M	256M
argocd-repo-server	0.5	1	256M	1024M
argocd-redis	0.25	0.5	128M	256M
argocd-dex	0.25	0.5	128M	256M

워크로드	CPU 요청	CPU 제한	메모리 요청	메모리 제한
HAProxy	0.25	0.5	128M	256M

필요한 경우 **oc** 명령과 함께 **ArgoCD** 사용자 정의 리소스를 사용하여 특정 항목을 확인하고 수정할 수도 있습니다.

```
oc edit argocd <name of argo cd> -n namespace
```