OpenShift Container Platform 4.15

Jenkins

Jenkins

Last Updated: 2024-10-02

OpenShift Container Platform 4.15 Jenkins

Jenkins

법적 공지

초록

OpenShift Container Platform용 Jenkins

차례

1장. JENKINS 이미지 구성

OpenShift Container Platform은 실행 중인 Jenkins에 대한 컨테이너 이미지를 제공합니다. 이 이미지는 Jenkins 서버 인스턴스를 제공하며, 지속적인 테스트, 통합 및 제공을 위한 기본 흐름을 설정하는 데 사용할 수 있습니다.

이미지는 Red Hat UBI(Universal Base Images)를 기반으로 합니다.

OpenShift Container Platform은 Jenkins의 LTS 릴리스를 따릅니다. OpenShift Container Platform은 Jenkins 2.x가 포함된 이미지를 제공합니다.

OpenShift Container Platform Jenkins 이미지는 Quay.io 또는 registry.redhat.io 에서 사용할 수 있습니다.

예를 들면 다음과 같습니다.

```
$ podman pull registry.redhat.io/ocp-tools-4/jenkins-rhel8:<image_tag>
```

이러한 이미지를 사용하려면 해당 레지스트리에서 직접 이미지에 액세스하거나 OpenShift Container Platform 컨테이너 이미지 레지스트리로 이미지를 푸시할 수 있습니다. 컨테이너 이미지 레지스트리 또는 외부 위치에서 이미지를 가리키는 이미지 스트림을 생성할 수도 있습니다. 그러면 OpenShift Container Platform 리소스에서 이미지 스트림을 참조할 수 있습니다.

하지만 편의를 위해 OpenShift Container Platform은 **openshift** 네임스페이스에서 핵심 Jenkins 이미지를 위한 이미지 스트림은 물론 Jenkins와 OpenShift Container Platform의 통합을 위한 에이전트 이미지 예제도 제공합니다.

1.1. 구성 및 사용자 정의

Jenkins 인증은 다음 두 가지 방법으로 관리할 수 있습니다.

- OpenShift Container Platform 로그인 플러그인에서 제공하는 OpenShift Container Platform OAuth 인증
- Jenkins에서 제공하는 표준 인증

1.1.1. OpenShift Container Platform OAuth 인증

OAuth 인증은 Jenkins UI의 **글로벌 보안 구성** 패널에서 옵션을 구성하거나 Jenkins **배포 구성**에서 **OPENSHIFT_ENABLE_OAUTH** 환경 변수를 **false** 이외의 값으로 설정하면 활성화됩니다. 이렇게 하면 OpenShift Container Platform 로그인 플러그인이 활성화되어 Pod 데이터에서 구성 정보를 검색하거나 OpenShift Container Platform API 서버와 상호 작용할 수 있습니다.

유효한 인증 정보는 OpenShift Container Platform ID 공급자가 제어합니다.

Jenkins는 브라우저 액세스 및 브라우저 이외의 액세스를 둘 다 지원합니다.

유효한 사용자는 로그인 시 Jenkins 권한 부여 매트릭스에 자동으로 추가되며 OpenShift Container Platform 역할에 따라 사용자가 가지는 특정 Jenkins 권한이 지정됩니다. 기본적으로 사용되는 역할은 사전 정의된 **admin**, **edit** 및 **view**입니다. 로그인 플러그인은 Jenkins가 실행되는 네임스페이스 또는 프로젝트에서 해당 역할에 대해 자체AR 요청을 실행합니다.

admin 역할의 사용자에게는 기존 Jenkins 관리자 권한이 있습니다. **edit** 또는 **view** 역할의 사용자는 점차 더 적은 권한을 가지게 됩니다.

기본 OpenShift Container Platform **admin**, **edit** 및 **view** 역할과 Jenkins 인스턴스에서 해당 역할이 할당된 Jenkins 권한은 구성할 수 있습니다.

OpenShift Container Platform Pod에서 Jenkins를 실행하는 경우 로그인 플러그인은 Jenkins가 실행되는 네임스페이스에서 **openshift-jenkins-login-plugin-config** 라는 구성 맵을 찾습니다.

이 플러그인이 해당 구성 맵을 찾아서 읽을 수 있는 경우 Jenkins 권한 매핑에 대해 역할을 정의할 수 있습니다. 특히 다음 내용에 유의하십시오.

- 로그인 플러그인은 구성 맵의 키 및 값 쌍을 OpenShift Container Platform 역할 매핑에 대한 Jenkins 권한으로 처리합니다.
- 키는 Jenkins 권한 그룹 짧은 ID와 Jenkins 권한 짧은 ID이며 두 개가 하이픈 문자로 구분되어 있습니다.
- OpenShift Container Platform 역할에 **Overall Jenkins Administer** 권한을 추가하려면 키가 **Overall-Administer**여야 합니다.
- 사용 가능한 권한 그룹 및 권한 ID를 파악하려면 Jenkins 콘솔 및 ID의 매트릭스 권한 부여 페이지로 이동하여 제공된 테이블의 그룹 및 개인 권한을 확인합니다.
- 키 및 값 쌍의 값은 권한이 적용되어야 하며 각 역할이 쉼표로 구분되어 있는 OpenShift Container Platform 역할의 목록입니다.
- 기본 **admin** 및 **edit** 역할과 생성한 새 jenkins 역할 모두에 **Overall Jenkins Administer** 권한을 추가하려면 키 **Overall-Administer**의 값이 **admin,edit,jenkins**가 됩니다.

참고

OpenShift Container Platform OAuth가 사용되는 경우 OpenShift Container Platform Jenkins 이미지에서 관리자 권한으로 미리 입력된 **admin** 사용자에게는 해당 권한이 부여되지 않습니다. 이러한 권한을 부여하려면 OpenShift Container Platform 클러스터 관리자가 OpenShift Container Platform ID 공급자에서 해당 사용자를 명시적으로 정의하고 **admin** 역할을 사용자에게 할당해야 합니다.

저장된 Jenkins 사용자 권한은 사용자가 처음 설정된 후 변경될 수 있습니다. OpenShift Container Platform 로그인 플러그인은 OpenShift Container Platform API 서버에서 권한을 폴링하고 Jenkins에 저장된 각 사용자의 권한을 OpenShift Container Platform에서 검색된 권한으로 업데이트합니다. Jenkins UI를 사용하여 Jenkins 사용자의 권한을 업데이트하는 경우 다음에 플러그인이 OpenShift Container Platform을 폴링할 때 권한 변경 사항을 덮어씁니다.

OPENSHIFT_PERMISSIONS_POLL_INTERVAL 환경 변수를 사용하여 폴링 발생 빈도를 제어할 수 있습니다. 기본 폴링 간격은 5분입니다.

OAuth 인증을 사용하여 새 Jenkins 서비스를 생성하는 가장 쉬운 방법은 템플릿을 사용하는 것입니다.

1.1.2. Jenkins 인증

템플릿을 사용하지 않고 이미지를 직접 실행하는 경우 기본적으로 Jenkins 인증이 사용됩니다.

Jenkins가 처음 시작되면 관리자 및 암호와 함께 구성이 생성됩니다. 기본 사용자 인증 정보는 **admin** 및 **password**입니다. 표준 Jenkins 인증을 사용하는 경우에만 **JENKINS_PASSWORD** 환경 변수를 설정하여 기본 암호를 구성합니다.

프로세스

- 다음 명령을 입력하여 표준 Jenkins 인증을 사용하는 Jenkins 애플리케이션을 생성합니다.

```
$ oc new-app -e \
    JENKINS_PASSWORD=<password> \
    ocp-tools-4/jenkins-rhel8
```

1.2. JENKINS 환경 변수

Jenkins 서버는 다음 환경 변수로 구성할 수 있습니다.

변수	정의	값 및 설정 예
OPENSHIFT_ENABLE_OAUTH	Jenkins에 로그인할 때 OpenShift Container Platform 로그인 플러그인이 인증을 관리하는지 여부를 결정합니다. 활성화하려면 **true**로 설정합니다.	기본값: **false**
JENKINS_PASSWORD	표준 Jenkins 인증을 사용하는 경우 **admin** 사용자의 암호입니다. **OPENSHIFT_ENABLE_OAUTH**가 **true**로 설정된 경우 해당되지 않습니다.	기본값: **password**
JAVA_MAX_HEAP_PARAM, **CONTAINER_HEAP_PERCENT**, **JENKINS_MAX_HEAP_UPPER_BOUND_MB**	이 값은 Jenkins JVM의 최대 힙 크기를 제어합니다. **JAVA_MAX_HEAP_PARAM**이 설정된 경우 해당 값이 우선합니다. 설정되지 않은 경우 최대 힙 크기는 컨테이너 메모리 제한의 **CONTAINER_HEAP_PERCENT**로 동적으로 계산되며, 선택적으로 **JENKINS_MAX_HEAP_UPPER_BOUND_MB**MiB로 제한될 수 있습니다. 기본적으로 Jenkins JVM의 최대 힙 크기는 제한 없이 컨테이너 메모리 제한의 50%로 설정됩니다.	**JAVA_MAX_HEAP_PARAM** 설정 예: **-Xmx512m** **CONTAINER_HEAP_PERCENT** 기본값: **0.5** 또는 50% **JENKINS_MAX_HEAP_UPPER_BOUND_MB** 설정 예: **512 MiB**
JAVA_INITIAL_HEAP_PARAM, **CONTAINER_INITIAL_PERCENT**	이 값은 Jenkins JVM의 초기 힙 크기를 제어합니다. **JAVA_INITIAL_HEAP_PARAM**이 설정된 경우 해당 값이 우선합니다. 설정되지 않은 경우 초기 힙 크기는 동적으로 계산된 최대 힙 크기의 **CONTAINER_INITIAL_PERCENT**로 동적으로 계산됩니다. 기본적으로 JVM은 초기 힙 크기를 설정합니다.	**JAVA_INITIAL_HEAP_PARAM** 설정 예: **-Xms32m** **CONTAINER_INITIAL_PERCENT** 설정 예: **0.1** 또는 10%

변수	정의	값 및 설정 예
CONTAINER_CORE_LIMIT	설정되는 경우 내부 JVM 스레드 수를 조정하는 데 사용되는 정수 코어 수를 지정합니다.	설정 예: **2**
JAVA_TOOL_OPTIONS	이 컨테이너에서 실행되는 모든 JVM에 적용할 옵션을 지정합니다. 이 값은 재정의하지 않는 것이 좋습니다.	Default: **-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true**
JAVA_GC_OPTS	Jenkins JVM 가비지 컬렉션 매개변수를 지정합니다. 이 값은 재정의하지 않는 것이 좋습니다.	Default: **-XX:+UseParallelGC -XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90**
JENKINS_JAVA_OVERRIDES	Jenkins JVM에 대한 추가 옵션을 지정합니다. 이러한 옵션은 위의 Java 옵션을 포함하여 다른 모든 옵션에 추가되며 필요한 경우 옵션을 재정의하는 데 사용될 수 있습니다. 각각의 추가 옵션을 공백으로 구분합니다. 옵션에 공백 문자가 포함되어 있으면 백슬래시로 이스케이프합니다.	설정 예: **-Dfoo -Dbar**; **-Dfoo=first\ value -Dbar=second\ value**
JENKINS_OPTS	Jenkins에 대한 인수를 지정합니다.	
INSTALL_PLUGINS	컨테이너를 처음 실행할 때 또는 **OVERRIDE_PV_PLUGINS_IMAGE_PLUGINS** 가 **true** 로 설정된 경우 설치할 추가 Jenkins 플러그인을 지정합니다. 플러그인은 쉼표로 구분된 이름:버전 쌍 목록으로 지정됩니다.	설정 예: **git:3.7.0,subversion:2.10.2**
OPENSHIFT_PERMISSIONS_POLL_INTERVAL	OpenShift Container Platform 로그인 플러그인이 Jenkins에서 정의된 각 사용자와 연결된 권한에 대해 OpenShift Container Platform을 폴링하는 간격(밀리초)을 지정합니다.	기본값: **300000** - 5분

변수	정의	값 및 설정 예
OVERRIDE_PV_CONFIG_WITH_IMAGE_CONFIG	Jenkins 구성 디렉토리에 대해 OpenShift Container Platform 영구 볼륨(PV)을 사용하여 이 이미지를 실행하는 경우 영구 볼륨 클레임(PVC) 이 생성되면 영구 볼륨이 할당되므로 이미지가 처음 시작될 때만 이미지에서 영구 볼륨으로 구성 전송을 수행합니다. 초기 시작 후 이 이미지를 확장하고 사용자 정의 이미지의 구성을 업데이트하는 사용자 정의 이미지를 생성하는 경우 이 환경 변수를 **true** 로 설정하지 않으면 구성이 복사되지 않습니다.	기본값: **false**
OVERRIDE_PV_PLUGINS_WITH_IMAGE_PLUGINS	Jenkins 구성 디렉토리에 대해 OpenShift Container Platform PV를 사용하여 이 이미지를 실행하는 경우 PVC가 생성될 때 PV가 할당되므로 이미지가 처음 시작될 때만 이미지에서 PV로 플러그인을 전송합니다. 초기 시작 후 이 이미지를 확장하고 사용자 정의 이미지의 플러그인을 업데이트하는 사용자 정의 이미지를 생성하는 경우 이 환경 변수를 **true** 로 설정하지 않으면 플러그인이 복사되지 않습니다.	기본값: **false**
ENABLE_FATAL_ERROR_LOG_FILE	Jenkins 구성 디렉토리에 대해 OpenShift Container Platform 영구 볼륨 클레임(PVC)으로 이 이미지를 실행하는 경우 이 환경 변수를 사용하면 치명적 오류가 발생할 때 치명적 오류 로그 파일을 유지할 수 있습니다. 치명적 오류 파일은 **/var/lib/jenkins/logs**에 저장됩니다.	기본값: **false**
AGENT_BASE_IMAGE	이 값을 설정하면 이 이미지와 함께 제공되는 샘플 Kubernetes 플러그인 pod 템플릿에서 **jnlp** 컨테이너에 사용되는 이미지가 재정의됩니다. 그렇지 않으면 **openshift** 네임스페이스에 있는 **jenkins-agent-base-rhel8:latest** 이미지 스트림 태그의 이미지가 사용됩니다.	기본값: **image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-rhel8:latest**

변수	정의	값 및 설정 예
JAVA_BUILDER_IMAGE	이 값을 설정하면 이 이미지와 함께 제공된 **java-builder** 샘플 Kubernetes 플러그인 pod 템플릿에서 **java-builder** 컨테이너에 사용되는 이미지가 재정의됩니다. 그렇지 않으면 **openshift** 네임스페이스에 있는 **java:latest** 이미지 스트림 태그의 이미지가 사용됩니다.	기본값: **image-registry.openshift-image-registry.svc:5000/openshift/java:latest**
JAVA_FIPS_OPTIONS	이 값을 설정하면 FIPS 노드에서 JVM이 작동하는 방식을 제어합니다. 자세한 내용은 FIPS 모드에서 OpenJDK 11의 Red Hat 빌드 구성을 참조하십시오.	기본값: **-Dcom.redhat.fips=false**

1.3. JENKINS에 프로젝트 간 액세스 권한 제공

동일한 프로젝트가 아닌 다른 위치에서 Jenkins를 실행하려는 경우 Jenkins에 액세스 토큰을 제공해야 프로젝트에 액세스할 수 있습니다.

프로세스

1. 다음 명령을 입력하여 Jenkins가 액세스해야 하는 프로젝트에 액세스할 수 있는 적절한 권한이 있는 서비스 계정의 시크릿을 식별합니다.

```
$ oc describe serviceaccount jenkins
```

출력 예

```
Name:       default
Labels:     <none>
Secrets:    { jenkins-token-uyswp    }
            { jenkins-dockercfg-xcr3d    }
Tokens:     jenkins-token-izv1u
            jenkins-token-uyswp
```

이 경우 시크릿 이름은 **jenkins-token-uyswp**로 지정됩니다.

2. 다음 명령을 입력하여 시크릿에서 토큰을 검색합니다.

```
$ oc describe secret <secret name from above>
```

출력 예

```
Name:       jenkins-token-uyswp
Labels:     <none>
Annotations:    kubernetes.io/service-account.name=jenkins,kubernetes.io/service-account.uid=32f5b661-2a8f-11e5-9528-3c970e3bf0b7
Type:   kubernetes.io/service-account-token
```

```
Data
====
ca.crt: 1066 bytes
token:  eyJhbGc..<content cut>....wRA
```

토큰 매개변수에는 Jenkins가 프로젝트에 액세스하는 데 필요한 토큰 값이 포함되어 있습니다.

1.4. JENKINS 볼륨 간 마운트 지점

구성을 위한 영구 스토리지가 활성화되도록 마운트된 볼륨을 사용하여 Jenkins 이미지를 실행할 수 있습니다.

- **/var/lib/jenkins** - Jenkins가 작업 정의를 비롯한 구성 파일을 저장하는 데이터 디렉토리입니다.

1.5. S2I(SOURCE-TO-IMAGE)를 통해 JENKINS 이미지 사용자 정의

공식 OpenShift Container Platform Jenkins 이미지를 사용자 정의하려면 이미지를 S2I(Source-to-Image) 빌더로 사용할 수 있습니다.

S2I를 사용하여 사용자 정의 Jenkins 작업 정의를 복사하거나 플러그인을 추가하거나 제공된 **config.xml** 파일을 자체 사용자 정의 구성으로 교체할 수 있습니다.

Jenkins 이미지에 수정 사항을 포함하려면 다음 디렉토리 구조의 Git 리포지터리가 있어야 합니다.

plugins

이 디렉터리에는 Jenkins로 복사하려는 바이너리 Jenkins 플러그인이 포함되어 있습니다.

plugins.txt

이 파일에는 다음 구문을 사용하여 설치할 플러그인이 나열되어 있습니다.

```
pluginId:pluginVersion
```

configuration/jobs

이 디렉토리에는 Jenkins 작업 정의가 있습니다.

configuration/config.xml

이 파일에는 사용자 정의 Jenkins 구성이 있습니다.

configuration/ 디렉토리의 콘텐츠는 **/var/lib/jenkins/** 디렉토리에 복사되므로 **credentials.xml** 같은 추가 파일도 포함할 수 있습니다.

OpenShift Container Platform에서 Jenkins 이미지를 사용자 지정하는 빌드 구성 샘플

```
apiVersion: build.openshift.io/v1
kind: BuildConfig
metadata:
  name: custom-jenkins-build
spec:
  source:                    1
    git:
      uri: https://github.com/custom/repository
    type: Git
  strategy:                  2
    sourceStrategy:
```

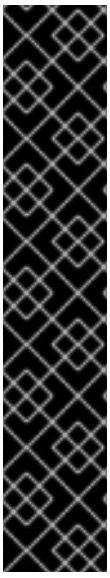
```
      from:
        kind: ImageStreamTag
        name: jenkins:2
        namespace: openshift
    type: Source
  output:                    3
    to:
      kind: ImageStreamTag
      name: custom-jenkins:latest
```

1. **source** 매개변수는 위에서 설명한 레이아웃으로 소스 Git 리포지터리를 정의합니다.
2. **strategy** 매개변수는 빌드의 소스 이미지로 사용할 원본 Jenkins 이미지를 정의합니다.
3. **output** 매개변수는 공식 Jenkins 이미지 대신 배포 구성에 사용할 수 있는 사용자 정의 Jenkins 이미지를 정의합니다.

1.6. JENKINS KUBERNETES 플러그인 구성

OpenShift Jenkins 이미지에는 Jenkins용 사전 설치된 Kubernetes 플러그인이 포함되어 Kubernetes 및 OpenShift Container Platform을 사용하여 여러 컨테이너 호스트에서 Jenkins 에이전트를 동적으로 프로비저닝할 수 있습니다.

Kubernetes 플러그인을 사용하기 위해 OpenShift Container Platform에서는 Jenkins 에이전트로 사용하기에 적합한 OpenShift 에이전트 기본 이미지를 제공합니다.



중요

OpenShift Container Platform 4.11은 OpenShift Jenkins 및 OpenShift Agent Base 이미지를 **registry.redhat.io** 의 **ocp-tools-4** 리포지토리로 이동하여 Red Hat이 OpenShift Container Platform 라이프사이클 외부에서 이미지를 생성하고 업데이트할 수 있습니다. 이전에는 이러한 이미지가 OpenShift Container Platform 설치 페이로드에 있고 **registry.redhat.io**의 **openshift4** 리포지토리에 있었습니다.

OpenShift Jenkins Maven 및 NodeJS 에이전트 이미지는 OpenShift Container Platform 4.11 페이로드에서 제거되었습니다. Red Hat은 더 이상 이러한 이미지를 생성하지 않으며 **registry.redhat.io**의 **ocp-tools-4** 리포지토리에서 사용할 수 없습니다. Red Hat은 OpenShift Container Platform 라이프사이클 정책에 따라 중요한 버그 수정 또는 보안 CVE에 대해 4.10 및 이전 버전의 이미지를 유지 관리합니다.

자세한 내용은 다음 "추가 리소스" 섹션의 "OpenShift Jenkins 이미지에 대한 중요한 변경 사항" 링크를 참조하십시오.

Maven 및 Node.js 에이전트 이미지는 Kubernetes 플러그인에 대한 OpenShift Container Platform Jenkins 이미지 구성에서 Kubernetes pod 템플릿 이미지로 자동 구성됩니다. 해당 구성에는 **Restrict where this project can be run** 아래의 모든 Jenkins 작업에 적용할 수 있는 각 이미지의 레이블이 포함됩니다. 레이블이 적용되면 해당 에이전트 이미지를 실행하는 OpenShift Container Platform pod에서 작업이 실행됩니다.

중요

OpenShift Container Platform 4.10 이상에서는 Kubernetes 플러그인을 사용하여 Jenkins 에이전트를 실행하는 데 권장되는 패턴은 **jnlp** 및 **sidecar** 컨테이너 모두에서 Pod 템플릿을 사용하는 것입니다. **jnlp** 컨테이너는 OpenShift Container Platform Jenkins Base 에이전트 이미지를 사용하여 빌드에 사용할 별도의 Pod를 쉽게 시작할 수 있습니다. **sidecar** 컨테이너 이미지에는 시작된 별도의 Pod 내에서 특정 언어로 빌드하는 데 필요한 툴이 있습니다. Red Hat Container Catalog의 많은 컨테이너 이미지는 **openshift** 네임스페이스의 샘플 이미지 스트림에서 참조됩니다. OpenShift Container Platform Jenkins 이미지에는 이 방법을 보여주는 사이드카 컨테이너가 있는 **java-build** 라는 Pod 템플릿이 있습니다. 이 pod 템플릿은 **openshift** 네임스페이스의 **java** 이미지 스트림에서 제공하는 최신 Java 버전을 사용합니다.

Jenkins 이미지는 Kubernetes 플러그인의 추가 에이전트 이미지 자동 검색 및 자동 구성 기능도 제공합니다.

Jenkins 시작 시 OpenShift Container Platform 동기화 플러그인을 사용하면 Jenkins 이미지가 실행 중인 프로젝트 또는 플러그인 구성에 나열된 프로젝트에서 다음 항목을 검색합니다.

- **role** 라벨이 **jenkins-agent** 로 설정된 이미지 스트림입니다.
- **role** 주석이 **jenkins-agent** 로 설정된 이미지 스트림 태그입니다.
- **role** 레이블이 **jenkins-agent** 로 설정된 구성 맵입니다.

Jenkins 이미지에서 적절한 레이블이 있는 이미지 스트림 또는 적절한 주석이 있는 이미지 스트림 태그를 찾으면 해당 Kubernetes 플러그인 구성이 생성됩니다. 이렇게 하면 이미지 스트림에서 제공하는 컨테이너 이미지를 실행하는 포드에서 Jenkins 작업을 실행하도록 할당할 수 있습니다.

이미지 스트림 또는 이미지 스트림 태그의 이름 및 이미지 참조는 Kubernetes 플러그인 pod 템플릿의 이름 및 이미지 필드에 매핑됩니다. **agent-label** 키로 이미지 스트림 또는 이미지 스트림 태그 오브젝트에 주석을 설정하여 Kubernetes 플러그인 pod 템플릿의 레이블 필드를 제어할 수 있습니다. 그러지 않으면 이름이 레이블로 사용됩니다.

참고

Jenkins 콘솔에 로그인하지 말고 pod 템플릿 구성을 변경합니다. Pod 템플릿이 생성된 후 이를 수행하고 OpenShift Container Platform 동기화 플러그인에서 이미지 스트림 또는 이미지 스트림 태그와 연결된 이미지가 변경되었음을 탐지하면 pod 템플릿을 교체하고 구성 변경 사항을 덮어씁니다. 새 구성은 기존 구성과 병합할 수 없습니다.

보다 복잡한 구성이 필요한 경우 구성 맵 접근 방법을 고려하십시오.

적절한 레이블이 있는 구성 맵을 찾으면 Jenkins 이미지는 구성 맵의 키-값 데이터 페이로드에 있는 값에 Jenkins 및 Kubernetes 플러그인 pod 템플릿의 구성 형식과 일치하는 XML(Extensible Markup Language)이 포함되어 있다고 가정합니다. 이미지 스트림 및 이미지 스트림 태그에 대한 구성 맵의 한 가지 주요 장점은 모든 Kubernetes 플러그인 pod 템플릿 매개변수를 제어할 수 있다는 것입니다.

jenkins-agent의 예제 구성 맵은 다음과 같습니다.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: jenkins-agent
  labels:
```

```
    role: jenkins-agent
data:
  template1: |-
    <org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
      <inheritFrom></inheritFrom>
      <name>template1</name>
      <instanceCap>2147483647</instanceCap>
      <idleMinutes>0</idleMinutes>
      <label>template1</label>
      <serviceAccount>jenkins</serviceAccount>
      <nodeSelector></nodeSelector>
      <volumes/>
      <containers>
        <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
          <name>jnlp</name>
          <image>openshift/jenkins-agent-maven-35-centos7:v3.10</image>
          <privileged>false</privileged>
          <alwaysPullImage>true</alwaysPullImage>
          <workingDir>/tmp</workingDir>
          <command></command>
          <args>${computer.jnlpmac} ${computer.name}</args>
          <ttyEnabled>false</ttyEnabled>
          <resourceRequestCpu></resourceRequestCpu>
          <resourceRequestMemory></resourceRequestMemory>
          <resourceLimitCpu></resourceLimitCpu>
          <resourceLimitMemory></resourceLimitMemory>
          <envVars/>
        </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
      </containers>
      <envVars/>
      <annotations/>
      <imagePullSecrets/>
      <nodeProperties/>
    </org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
```

다음 예제에서는 **openshift** 네임스페이스에서 이미지 스트림을 참조하는 두 개의 컨테이너를 보여줍니다. 하나의 컨테이너는 Jenkins 에이전트로 포드를 시작하기 위한 JNLP 계약을 처리합니다. 다른 컨테이너에서는 특정 코딩 언어로 코드를 빌드하는 데 툴이 포함된 이미지를 사용합니다.

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: jenkins-agent
  labels:
    role: jenkins-agent
data:
  template2: |-
      <org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
        <inheritFrom></inheritFrom>
        <name>template2</name>
        <instanceCap>2147483647</instanceCap>
        <idleMinutes>0</idleMinutes>
        <label>template2</label>
        <serviceAccount>jenkins</serviceAccount>
        <nodeSelector></nodeSelector>
        <volumes/>
```

```
        <containers>
          <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
            <name>jnlp</name>
            <image>image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-
rhel8:latest</image>
            <privileged>false</privileged>
            <alwaysPullImage>true</alwaysPullImage>
            <workingDir>/home/jenkins/agent</workingDir>
            <command></command>
            <args>\$(JENKINS_SECRET) \$(JENKINS_NAME)</args>
            <ttyEnabled>false</ttyEnabled>
            <resourceRequestCpu></resourceRequestCpu>
            <resourceRequestMemory></resourceRequestMemory>
            <resourceLimitCpu></resourceLimitCpu>
            <resourceLimitMemory></resourceLimitMemory>
            <envVars/>
          </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
          <org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
            <name>java</name>
            <image>image-registry.openshift-image-registry.svc:5000/openshift/java:latest</image>
            <privileged>false</privileged>
            <alwaysPullImage>true</alwaysPullImage>
            <workingDir>/home/jenkins/agent</workingDir>
            <command>cat</command>
            <args></args>
            <ttyEnabled>true</ttyEnabled>
            <resourceRequestCpu></resourceRequestCpu>
            <resourceRequestMemory></resourceRequestMemory>
            <resourceLimitCpu></resourceLimitCpu>
            <resourceLimitMemory></resourceLimitMemory>
            <envVars/>
          </org.csanchez.jenkins.plugins.kubernetes.ContainerTemplate>
        </containers>
        <envVars/>
        <annotations/>
        <imagePullSecrets/>
        <nodeProperties/>
      </org.csanchez.jenkins.plugins.kubernetes.PodTemplate>
```

참고

Jenkins 콘솔에 로그인하지 말고 pod 템플릿 구성을 변경합니다. Pod 템플릿이 생성된 후 이를 수행하고 OpenShift Container Platform 동기화 플러그인에서 이미지 스트림 또는 이미지 스트림 태그와 연결된 이미지가 변경되었음을 탐지하면 pod 템플릿을 교체하고 구성 변경 사항을 덮어씁니다. 새 구성은 기존 구성과 병합할 수 없습니다.

보다 복잡한 구성이 필요한 경우 구성 맵 접근 방법을 고려하십시오.

OpenShift Container Platform 동기화 플러그인이 설치되면 OpenShift Container Platform의 API 서버를 모니터링하여 이미지 스트림, 이미지 스트림 태그 및 구성 맵에 대한 업데이트를 확인하고 Kubernetes 플러그인의 구성을 조정합니다.

적용되는 규칙은 다음과 같습니다.

- 구성 맵, 이미지 스트림 또는 이미지 스트림 태그에서 레이블 또는 주석을 제거하면 Kubernetes 플러그인 구성에서 기존 **PodTemplate** 이 삭제됩니다.
- 이러한 오브젝트가 제거되면 Kubernetes 플러그인에서 해당 구성이 제거됩니다.
- 레이블 또는 주석이 적절히 지정된 **ConfigMap**,**ImageStream** 또는 **ImageStreamTag** 오브젝트를 생성하거나 초기 생성 후 라벨을 추가하면 Kubernetes-plugin 구성에 **PodTemplate** 이 생성됩니다.
- 구성 맵 형식별 **PodTemplate**의 경우 **PodTemplate**의 구성 맵 데이터에 대한 변경 사항이 Kubernetes 플러그인 구성의 **PodTemplate** 설정에 적용됩니다. 변경 사항은 구성 맵 변경 사이에 Jenkins UI를 통해 **PodTemplate** 의 변경 사항도 덮어씁니다.

컨테이너 이미지를 Jenkins 에이전트로 사용하려면 이미지가 에이전트를 진입점으로 실행해야 합니다. 자세한 내용은 공식 Jenkins 설명서를 참조하십시오.

추가 리소스

- OpenShift Jenkins 이미지에 대한 중요한 변경 사항

1.7. JENKINS 권한

구성 맵에서 pod 템플릿 XML의 **<serviceAccount>** 요소가 결과 pod에 사용된 OpenShift Container Platform 서비스 계정인 경우 서비스 계정 인증 정보가 해당 pod에 마운트됩니다. 권한은 이 서비스 계정과 연결되며 pod에서 허용되는 OpenShift Container Platform 마스터 작업을 제어합니다.

Pod에 사용된 서비스 계정을 사용하는 다음 시나리오를 고려해 보십시오. 이 pod는 OpenShift Container Platform Jenkins 이미지에서 실행되는 Kubernetes 플러그인에 의해 시작됩니다.

OpenShift Container Platform에서 제공하는 Jenkins에 대한 템플릿 예를 사용하는 경우 Jenkins가 실행되는 프로젝트에 대해 **jenkins** 서비스 계정이 **edit** 역할로 정의되고 마스터 Jenkins Pod에 해당 서비스 계정이 마운트됩니다.

Jenkins 구성에 삽입된 두 개의 기본 Maven 및 NodeJS pod 템플릿도 Jenkins 마스터와 동일한 서비스 계정을 사용하도록 설정됩니다.

- 이미지 스트림 또는 이미지 스트림 태그에 필요한 레이블 또는 주석이 있기 때문에 OpenShift Container Platform 동기화 플러그인에서 자동으로 검색되는 pod 템플릿은 서비스 계정으로 Jenkins 마스터 서비스 계정을 사용하도록 구성됩니다.
- Jenkins 및 Kubernetes 플러그인에 pod 템플릿 정의를 제공할 수 있는 다른 방법의 경우 사용할 서비스 계정을 명시적으로 지정해야 합니다. 다른 방법으로는 Jenkins 콘솔, Kubernetes 플러그인에서 제공하는 **podTemplate** 파이프라인 DSL 또는 pod 템플릿의 XML 구성 데이터가 있는 구성 맵에 레이블을 지정하는 방법이 있습니다.
- 서비스 계정의 값을 지정하지 않으면 **default** 서비스 계정이 사용됩니다.
- 사용할 서비스 계정이 OpenShift Container Platform 내에 정의된 필요한 권한, 역할 등을 가지고 있어서 pod에서 선택한 모든 프로젝트를 조작할 수 있는지 확인하십시오.

1.8. 템플릿에서 JENKINS 서비스 생성

템플릿은 모든 환경 변수를 사전 정의된 기본값으로 정의하는 매개변수 필드를 제공합니다. OpenShift Container Platform은 새로운 Jenkins 서비스 생성을 용이하게 해주는 템플릿을 제공합니다. Jenkins 템플릿은 초기 클러스터를 설정하는 중에 클러스터 관리자에 의해 기본 **openshift** 프로젝트에서 등록되어야 합니다.

사용 가능한 두 템플릿은 둘 다 배포 구성 및 서비스를 정의합니다. 스토리지 전략에서는 pod를 다시 시작하면 Jenkins 콘텐츠가 지속되는지 여부에 영향을 미칩니다.

참고

pod가 다른 노드로 이동되거나 배포 구성 업데이트에 따라 재배포가 트리거되는 경우 pod가 재시작될 수 있습니다.

- **jenkins-ephemeral**에서는 ephemeral 스토리지를 사용합니다. pod를 다시 시작하면 모든 데이터가 손실됩니다. 이 템플릿은 개발 또는 테스트에만 유용합니다.
- **jenkins-persistent**에서는 영구 볼륨 (PV) 저장소를 사용합니다. pod를 다시 시작해도 데이터가 유지됩니다.

영구 볼륨 (PV) 저장소를 사용하려면 클러스터 관리자가 OpenShift Container Platform 배포에서 영구 볼륨 (PV) 풀을 정의해야 합니다.

원하는 템플릿을 선택한 후 Jenkins를 사용할 수 있도록 템플릿을 인스턴스화해야 합니다.

프로세스

- 다음 방법 중 하나를 사용하여 새 Jenkins 애플리케이션을 생성합니다.
  - A PV:

```
$ oc new-app jenkins-persistent
```

  - pod를 다시 시작하면 구성이 유지되지 않는 **emptyDir** 유형 볼륨:

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

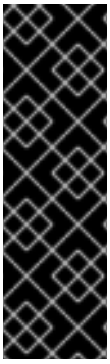
두 템플릿 모두 **oc describe** 를 실행하여 재정의에 사용할 수 있는 모든 매개변수를 확인할 수 있습니다.

예를 들면 다음과 같습니다.

```
$ oc describe jenkins-ephemeral
```

1.9. JENKINS KUBERNETES 플러그인 사용

다음 예에서는 **openshift-jee-sample BuildConfig** 오브젝트로 인해 Jenkins Maven 에이전트 pod가 동적으로 프로비저닝됩니다. pod는 일부 Java 소스 코드를 복제하고 WAR 파일을 빌드하며 두 번째 **BuildConfig**인 **openshift-jee-sample-docker**가 실행되도록 합니다. 두 번째 **BuildConfig**는 컨테이너 이미지에 새 WAR 파일의 계층을 지정합니다.



중요

OpenShift Container Platform 4.11은 페이로드에서 OpenShift Jenkins Maven 및 NodeJS 에이전트 이미지를 제거했습니다. Red Hat은 더 이상 이러한 이미지를 생성하지 않으며 **registry.redhat.io**의 **ocp-tools-4** 리포지토리에서 사용할 수 없습니다. Red Hat은 OpenShift Container Platform 라이프사이클 정책에 따라 중요한 버그 수정 또는 보안 CVE에 대해 4.10 및 이전 버전의 이미지를 유지 관리합니다.

자세한 내용은 다음 "추가 리소스" 섹션의 "OpenShift Jenkins 이미지에 대한 중요한 변경 사항" 링크를 참조하십시오.

Jenkins Kubernetes 플러그인을 사용하는 BuildConfig 예

```
kind: List
apiVersion: v1
items:
- kind: ImageStream
  apiVersion: image.openshift.io/v1
  metadata:
    name: openshift-jee-sample
- kind: BuildConfig
  apiVersion: build.openshift.io/v1
  metadata:
    name: openshift-jee-sample-docker
  spec:
    strategy:
      type: Docker
    source:
      type: Docker
      dockerfile: |-
        FROM openshift/wildfly-101-centos7:latest
        COPY ROOT.war /wildfly/standalone/deployments/ROOT.war
        CMD $STI_SCRIPTS_PATH/run
      binary:
        asFile: ROOT.war
    output:
      to:
        kind: ImageStreamTag
        name: openshift-jee-sample:latest
- kind: BuildConfig
  apiVersion: build.openshift.io/v1
  metadata:
    name: openshift-jee-sample
  spec:
    strategy:
      type: JenkinsPipeline
      jenkinsPipelineStrategy:
        jenkinsfile: |-
          node("maven") {
            sh "git clone https://github.com/openshift/openshift-jee-sample.git ."
            sh "mvn -B -Popenshift package"
            sh "oc start-build -F openshift-jee-sample-docker --from-file=target/ROOT.war"
          }
    triggers:
    - type: ConfigChange
```

동적으로 생성된 Jenkins 에이전트 pod의 사양을 재정의할 수도 있습니다. 다음은 이전 예를 수정하여 컨테이너 메모리를 재정의하고 환경 변수를 지정했습니다.

Jenkins Kubernetes 플러그인을 사용하여 메모리 제한 및 환경 변수를 지정하는 BuildConfig 샘플

```
kind: BuildConfig
apiVersion: build.openshift.io/v1
metadata:
  name: openshift-jee-sample
spec:
  strategy:
    type: JenkinsPipeline
    jenkinsPipelineStrategy:
      jenkinsfile: |-
        podTemplate(label: "mypod", 1
                cloud: "openshift", 2
                inheritFrom: "maven", 3
                containers: [
          containerTemplate(name: "jnlp", 4
                  image: "openshift/jenkins-agent-maven-35-centos7:v3.10", 5
                  resourceRequestMemory: "512Mi", 6
                  resourceLimitMemory: "512Mi", 7
                  envVars: [
            envVar(key: "CONTAINER_HEAP_PERCENT", value: "0.25") 8
          ])
        ]) {
          node("mypod") { 9
            sh "git clone https://github.com/openshift/openshift-jee-sample.git ."
            sh "mvn -B -Popenshift package"
            sh "oc start-build -F openshift-jee-sample-docker --from-file=target/ROOT.war"
          }
        }
  triggers:
  - type: ConfigChange
```

1. **mypod**라는 새로운 pod 템플릿이 동적으로 정의됩니다. 새 pod 템플릿 이름이 노드 스탠자에서 참조됩니다.
2. **cloud** 값은 **openshift**로 설정되어야 합니다.
3. 새 pod 템플릿은 기존 pod 템플릿의 구성을 상속할 수 있습니다. 이 경우, OpenShift Container Platform에 의해 미리 정의된 Maven pod 템플릿에서 상속됩니다.
4. 이 예에서는 기존 컨테이너의 값을 재정의하며, 이름별로 지정해야 합니다. OpenShift Container Platform과 함께 제공되는 모든 Jenkins 에이전트 이미지는 컨테이너 이름으로 **jnlp**를 사용합니다.
5. 컨테이너 이미지 이름을 다시 지정하십시오. 이것은 확인된 문제입니다.
6. **512 Mi** 메모리 요청이 지정되었습니다.
7. **512 Mi** 메모리 제한이 지정되었습니다.
8. 값이 **0.25**인 환경 변수 **CONTAINER_HEAP_PERCENT**가 지정되었습니다.

9 노드 스탠자는 정의된 pod 템플릿의 이름을 참조합니다.

기본적으로 pod는 빌드가 완료되면 삭제됩니다. 이 동작은 플러그인 또는 파이프라인 Jenkinsfile 내에서 수정할 수 있습니다.

업스트림 Jenkins는 최근에 파이프라인과 함께 **podTemplate** 파이프라인 DSL을 정의하는 YAML 선언 형식을 도입했습니다. 다음은 OpenShift Container Platform Jenkins 이미지에 정의된 샘플 **java-builder** pod 템플릿을 사용하여 다음 형식의 예입니다.

```
def nodeLabel = 'java-buidler'

pipeline {
  agent {
    kubernetes {
      cloud 'openshift'
      label nodeLabel
      yaml """
apiVersion: v1
kind: Pod
metadata:
  labels:
    worker: ${nodeLabel}
spec:
  containers:
  - name: jnlp
    image: image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-rhel8:latest
    args: ['\$(JENKINS_SECRET)', '\$(JENKINS_NAME)']
  - name: java
    image: image-registry.openshift-image-registry.svc:5000/openshift/java:latest
    command:
    - cat
    tty: true
"""
    }
  }

  options {
    timeout(time: 20, unit: 'MINUTES')
  }

  stages {
    stage('Build App') {
      steps {
        container("java") {
          sh "mvn --version"
        }
      }
    }
  }
}
```

추가 리소스

- OpenShift Jenkins 이미지에 대한 중요한 변경 사항

1.10. JENKINS 메모리 요구 사항

제공된 Jenkins Ephemeral 또는 Jenkins Persistent 템플릿으로 배포하는 경우 기본 메모리 제한은 **1Gi**입니다.

기본적으로 Jenkins 컨테이너에서 실행되는 다른 모든 프로세스에서는 총 **512 MiB** 이상의 메모리를 사용할 수 없습니다. 더 많은 메모리가 필요한 경우 컨테이너가 중지됩니다. 따라서 가능한 경우 Pipeline은 에이전트 컨테이너에서 외부 명령을 실행하는 것이 좋습니다.

Project 할당량이 허용하는 경우 메모리 관점에서 Jenkins 마스터가 갖추어야 할 사항에 대한 Jenkins 문서의 권장 사항을 참조하십시오. 이러한 권장 사항에서는 Jenkins 마스터에 더 많은 메모리를 할당하지 않도록 합니다.

Jenkins Kubernetes 플러그인에 의해 생성된 에이전트 컨테이너에서 메모리 요청 및 제한 값을 지정하는 것이 좋습니다. 관리자는 Jenkins 구성을 통해 개별 에이전트 이미지를 기반으로 기본값을 설정할 수 있습니다. 메모리 요청 및 제한 매개변수는 개별 컨테이너를 기반으로 재정의할 수도 있습니다.

imagestreamtagJenkins Ephemeral 또는 Jenkins Persistent 템플릿을 인스턴스화하는 경우 **MEMORY_LIMIT** 매개변수를 재정의하여 Jenkins에서 사용할 수 있는 메모리 크기를 늘릴 수 있습니다.

1.11. 추가 리소스

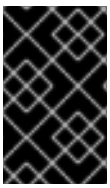
- Red Hat UBI(Universal Base Images)에 대한 자세한 내용은 기본 이미지 옵션을 참조하십시오.
- OpenShift Jenkins 이미지에 대한 중요한 변경 사항

2장. JENKINS 에이전트

OpenShift Container Platform에서는 Jenkins 에이전트로 사용할 기본 이미지를 제공합니다.

Jenkins 에이전트의 기본 이미지는 다음을 수행합니다.

- 필요한 도구, 헤드리스 Java, Jenkins JNLP 클라이언트 및 **git**,**tar**,**zip** 및 **nss** 를 포함한 유용한 툴을 모두 가져옵니다.
- JNLP 에이전트를 진입점으로 설정합니다.
- Jenkins 작업 내에서 명령줄 작업을 호출하는 **oc** 클라이언트 도구가 포함되어 있습니다.
- RHEL(Red Hat Enterprise Linux) 및 **localdev** 이미지 모두에 Dockerfile을 제공합니다.



중요

OpenShift Container Platform 릴리스 버전에 적합한 에이전트 이미지 버전을 사용합니다. OpenShift Container Platform 버전과 호환되지 않는 **oc** 클라이언트 버전을 포함하면 예기치 않은 동작이 발생할 수 있습니다.

OpenShift Container Platform Jenkins 이미지는 Jenkins Kubernetes 플러그인에서 에이전트 이미지를 사용하는 방법을 설명하기 위해 다음 샘플 **java-builder** pod 템플릿도 정의합니다.

java-builder pod 템플릿은 다음 두 개의 컨테이너를 사용합니다.

- OpenShift Container Platform 기본 에이전트 이미지를 사용하고 Jenkins 에이전트를 시작하고 중지하기 위한 JNLP 계약을 처리하는 **jnlp** 컨테이너입니다.
- 코드 빌드를 위해 Maven 바이너리 **mvn** 을 포함하여 다양한 Java 바이너리가 포함된 **java** 컨테이너는 java OpenShift Container Platform 샘플 ImageStream을 사용합니다.

2.1. JENKINS 에이전트 이미지

OpenShift Container Platform Jenkins 에이전트 이미지는 Quay.io 또는 registry.redhat.io 에서 사용할 수 있습니다.

Jenkins 이미지는 Red Hat Registry를 통해 사용할 수 있습니다.

```
$ docker pull registry.redhat.io/ocp-tools-4/jenkins-rhel8:<image_tag>
```

```
$ docker pull registry.redhat.io/ocp-tools-4/jenkins-agent-base-rhel8:<image_tag>
```

이러한 이미지를 사용하려면 Quay.io 또는 registry.redhat.io 에서 직접 이미지에 액세스하거나 OpenShift Container Platform 컨테이너 이미지 레지스트리로 푸시할 수 있습니다.

2.2. JENKINS 에이전트 환경 변수

각 Jenkins 에이전트 컨테이너는 다음 환경 변수를 사용하여 구성할 수 있습니다.

변수	정의	값 및 설정 예
JAVA_MAX_HEAP_PARAM, **CONTAINER_HEAP_PERCENT**, **JENKINS_MAX_HEAP_UPPER_BOUND_MB**	이 값은 Jenkins JVM의 최대 힙 크기를 제어합니다. **JAVA_MAX_HEAP_PARAM** 이 설정된 경우 해당 값이 우선합니다. 설정되지 않은 경우 최대 힙 크기는 컨테이너 메모리 제한의 **CONTAINER_HEAP_PERCENT**로 동적으로 계산되며, 선택적으로 **JENKINS_MAX_HEAP_UPPER_BOUND_MB**MiB로 제한될 수 있습니다. 기본적으로 Jenkins JVM의 최대 힙 크기는 제한 없이 컨테이너 메모리 제한의 50%로 설정됩니다.	**JAVA_MAX_HEAP_PARAM** 설정 예: **-Xmx512m** **CONTAINER_HEAP_PERCENT** 기본값: **0.5** 또는 50% **JENKINS_MAX_HEAP_UPPER_BOUND_MB** 설정 예: **512 MiB**
JAVA_INITIAL_HEAP_PARAM, **CONTAINER_INITIAL_PERCENT**	이 값은 Jenkins JVM의 초기 힙 크기를 제어합니다. **JAVA_INITIAL_HEAP_PARAM**이 설정된 경우 해당 값이 우선합니다. 설정되지 않은 경우 초기 힙 크기는 동적으로 계산된 최대 힙 크기의 **CONTAINER_INITIAL_PERCENT**로 동적으로 계산됩니다. 기본적으로 JVM은 초기 힙 크기를 설정합니다.	**JAVA_INITIAL_HEAP_PARAM** 설정 예: **-Xms32m** **CONTAINER_INITIAL_PERCENT** 설정 예: **0.1** 또는 10%
CONTAINER_CORE_LIMIT	설정되는 경우 내부 JVM 스레드 수를 조정하는 데 사용되는 정수 코어 수를 지정합니다.	설정 예: **2**
JAVA_TOOL_OPTIONS	이 컨테이너에서 실행되는 모든 JVM에 적용할 옵션을 지정합니다. 이 값은 재정의하지 않는 것이 좋습니다.	Default: **-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap -Dsun.zip.disableMemoryMapping=true**
JAVA_GC_OPTS	Jenkins JVM 가비지 컬렉션 매개변수를 지정합니다. 이 값은 재정의하지 않는 것이 좋습니다.	Default: **-XX:+UseParallelGC -XX:MinHeapFreeRatio=5 -XX:MaxHeapFreeRatio=10 -XX:GCTimeRatio=4 -XX:AdaptiveSizePolicyWeight=90**

변수	정의	값 및 설정 예
JENKINS_JAVA_OVERRIDES	Jenkins JVM에 대한 추가 옵션을 지정합니다. 이러한 옵션은 위의 Java 옵션을 포함하여 다른 모든 옵션에 추가되며 필요한 경우 옵션을 재정의하는 데 사용될 수 있습니다. 각각의 추가 옵션을 공백으로 구분하고 옵션에 공백 문자가 포함되어 있으면 백슬래시로 이스케이프합니다.	설정 예: **-Dfoo -Dbar**; **-Dfoo=first\ value -Dbar=second\ value**
USE_JAVA_VERSION	컨테이너에서 에이전트를 실행하는 데 사용할 Java 버전의 버전을 지정합니다. 컨테이너 기본 이미지에는 java의 두 가지 버전이 설치되어 있습니다. **java-11** 및 **java-1.8.0**. 컨테이너 기본 이미지를 확장하는 경우 관련 접미사를 사용하여 대체 버전의 java를 지정할 수 있습니다.	기본값은 **java-11** 입니다. 설정 예: **java-1.8.0**

2.3. JENKINS 에이전트 메모리 요구 사항

JVM은 모든 Jenkins 에이전트에서 Jenkins JNLP 에이전트를 호스트하고 **javac**, Maven 또는 Gradle 같은 Java 애플리케이션을 실행하는 데 사용됩니다.

기본적으로 Jenkins JNLP 에이전트 JVM은 힙에 대해 컨테이너 메모리 제한의 50%를 사용합니다. 이 값은 **CONTAINER_HEAP_PERCENT** 환경 변수를 통해 수정할 수 있습니다. 상한값으로 제한하거나 완전히 재정의할 수도 있습니다.

파이프라인에서 실행되는 쉘 스크립트나 **oc** 명령과 같이 Jenkins 에이전트 컨테이너에서 실행되는 다른 모든 프로세스는 기본적으로 OOM을 종료하지 않고는 나머지 50%의 메모리 제한을 초과하여 사용할 수 없습니다.

기본적으로 Jenkins 에이전트 컨테이너에서 실행되는 각각의 추가 JVM 프로세스는 힙에 대해 컨테이너 메모리 제한의 최대 25%를 사용합니다. 많은 빌드 워크로드에 대해 이 제한을 튜닝해야 할 수도 있습니다.

2.4. JENKINS 에이전트 GRADLE 빌드

OpenShift Container Platform의 Jenkins 에이전트에서 Gradle 빌드를 호스트하면 Jenkins JNLP 에이전트 및 Gradle JVM 외에도 여러 빌드가 지정된 경우 테스트를 실행하는 세 번째 JVM을 Gradle이 생성하므로 복잡성이 더욱 가중됩니다.

다음은 OpenShift Container Platform의 메모리가 제한된 Jenkins 에이전트에서 Gradle 빌드를 실행하기 위한 시작점으로 제안된 설정입니다. 필요에 따라 이러한 설정을 수정할 수 있습니다.

- **org.gradle.daemon=false**를 **gradle.properties** 파일에 추가하여 오래된 Gradle 데몬이 비활성화되었는지 확인합니다.
- **org.gradle.parallel=true**가 **gradle.properties** 파일에서 설정되지 않았고 **--parallel**이 명령줄 인수로 설정되지 않았음을 확인하여 병렬 빌드 실행을 비활성화합니다.

- Java 컴파일이 프로세스 외부에서 실행되지 않도록 하려면 **build.gradle** 파일에서 **java { options.fork = false }**를 설정합니다.
- **build.gradle** 파일에서 **test { maxParallelForks = 1 }**가 설정되었는지 확인하여 여러 추가 테스트 프로세스를 비활성화합니다.
- **GRADLE_OPTS**, **JAVA_OPTS** 또는 **JAVA_TOOL_OPTIONS** 환경 변수를 통해 Gradle JVM 메모리 매개변수를 재정의합니다.
- **maxHeapSize** 및 **jvmArgs** 설정을 **build.gradle**에서 정의하거나 **-Dorg.gradle.jvmargs** 명령줄 인수를 통해 Gradle 테스트 JVM에 대한 최대 힙 크기 및 JVM 인수를 설정합니다.

2.5. JENKINS 에이전트 POD 보존

Jenkins 에이전트 pod는 빌드가 완료되거나 중지되면 기본적으로 삭제됩니다. 이 동작은 Kubernetes 플러그인 pod 보존 설정을 통해 변경할 수 있습니다. pod 보존은 각 pod 템플릿에 대한 재정의를 통해 모든 Jenkins 빌드에 대해 설정할 수 있습니다. 지원되는 동작은 다음과 같습니다.

- **Always**는 빌드 결과에 관계없이 빌드 pod를 유지합니다.
- **Default** 는 플러그인 값을 사용합니다. 이 값은 pod 템플릿만 사용합니다.
- **Never**는 pod를 항상 삭제합니다.
- **On Failure**는 빌드 중 실패하면 pod를 유지합니다.

pod 보존은 Pipeline Jenkinsfile에서 재정의할 수 있습니다.

```
podTemplate(label: "mypod",
  cloud: "openshift",
  inheritFrom: "maven",
  podRetention: onFailure(), 1
  containers: [
    ...
  ]) {
  node("mypod") {
    ...
  }
}
```

1 **podRetention**에 대해 허용되는 값은 **never()**, **onFailure()**, **always()** 및 **default()**입니다.

주의

보존된 pod를 계속 실행하면서 리소스 할당량에 대한 계산에 반영할 수 있습니다.

3장. JENKINS에서 OPENSHIFT PIPELINES 또는 TEKTON으로 마이그레이션

Tekton 프로젝트를 기반으로 하는 클라우드 네이티브 CI/CD 환경인 Jenkins에서 Red Hat OpenShift Pipelines 로 CI/CD 워크플로를 마이그레이션할 수 있습니다.

3.1. JENKINS 및 OPENSHIFT PIPELINES 개념 비교

Jenkins 및 OpenShift Pipelines에서 사용되는 다음과 동등한 용어를 검토하고 비교할 수 있습니다.

3.1.1. Jenkins 용어

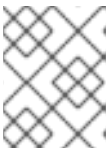
Jenkins는 공유 라이브러리 및 플러그인을 사용하여 확장할 수 있는 선언적 및 스크립팅된 파이프라인을 제공합니다. Jenkins의 몇 가지 기본 용어는 다음과 같습니다.

- **pipeline**: Groovy 구문을 사용하여 애플리케이션을 빌드, 테스트 및 배포하는 전체 프로세스를 자동화합니다.
- **Node**: 스크립팅된 파이프라인을 오케스트레이션하거나 실행할 수 있는 시스템입니다.
- **Stage**: 파이프라인에서 수행되는 작업의 개념적으로 구별되는 하위 집합입니다. 플러그인 또는 사용자 인터페이스는 종종 이 블록을 사용하여 작업의 상태 또는 진행 상황을 표시합니다.
- **Step**: 명령 또는 스크립트를 사용하여 수행할 정확한 작업을 지정하는 단일 작업입니다.

3.1.2. OpenShift Pipelines 용어

OpenShift Pipelines는 선언적 파이프라인에 YAML 구문을 사용하고 작업으로 구성됩니다. OpenShift Pipelines의 몇 가지 기본 용어는 다음과 같습니다.

- **Pipeline**: 일련의 직렬, 병렬 또는 둘 다에 있는 작업 세트입니다.
- **Task**: 명령, 바이너리 또는 스크립트로 구성된 일련의 단계입니다.
- **PipelineRun**: 하나 이상의 작업이 있는 파이프라인 실행입니다.
- **TaskRun**: 하나 이상의 단계로 작업을 실행합니다.



참고

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

- **Workspace**: OpenShift Pipelines에서 작업 공간은 다음과 같은 목적을 제공하는 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 OpenShift Pipelines 작업 공간에 직접적으로 동일한 작업 공간이 없습니다. 복제된 코드 리포지토리를 저장하고 기록 및 아티팩트를 빌드하므로 컨트롤 노드를 작업 영역으로 간주할 수 있습니다. 작업이 다른 노드에 할당되면 복제된 코드와 생성된 아티팩트가 해당 노드에 저장되지만 제어 노드는 빌드 내역을 유지합니다.

3.1.3. 개념 매핑

Jenkins 및 OpenShift Pipelines의 구성 요소는 동일하지 않으며 특정 비교에서는 기술적으로 정확한 매핑을 제공하지 않습니다. Jenkins 및 OpenShift Pipelines의 다음 용어 및 개념은 일반적으로 상관 관계가 있습니다.

표 3.1. Jenkins 및 OpenShift Pipelines - 기본 비교

Jenkins	OpenShift Pipelines
Pipeline	Pipeline 및 PipelineRun
Stage	Task
Step	작업의 단계

3.2. JENKINS에서 OPENSHIFT PIPELINES로 샘플 파이프라인 마이그레이션

다음 동등한 예제를 사용하여 Jenkins에서 OpenShift Pipelines로 파이프라인을 마이그레이션, 테스트 및 배포할 수 있습니다.

3.2.1. Jenkins 파이프라인

빌드, 테스트 및 배포를 위해 Groovy로 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'
            }
```

```
      }
    }
}
```

3.2.2. OpenShift Pipelines 파이프라인

이전 Jenkins 파이프라인과 동일한 OpenShift Pipelines에서 파이프라인을 생성하려면 다음 세 가지 작업을 생성합니다.

build 작업 YAML 정의 파일 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 YAML 정의 파일 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 task YAML 정의 파일 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 OpenShift Pipelines에서 파이프라인을 구성할 수 있습니다.

예: 빌드, 테스트 및 배포를 위한 OpenShift Pipelines 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
```

3.3. JENKINS 플러그인에서 TEKTON HUB 작업으로 마이그레이션

플러그인을 사용하여 Jenkins의 기능을 확장할 수 있습니다. OpenShift Pipelines에서 유사한 확장성을 얻으려면 Tekton Hub 에서 사용할 수 있는 작업을 사용합니다.

예를 들어 Jenkins의 git plugin 에 해당하는 Tekton Hub의 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
```

3.4. 사용자 정의 작업 및 스크립트를 사용하여 OPENSHIFT PIPELINES 기능 확장

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

예: maven test 명령을 실행하기 위한 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)
```

예: 경로를 제공하여 사용자 정의 쉘 스크립트 실행

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

예: YAML 파일에 작성하여 사용자 정의 Python 스크립트 실행

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

3.5. JENKINS 및 OPENSHIFT PIPELINES 실행 모델 비교

Jenkins 및 OpenShift Pipelines는 유사한 기능을 제공하지만 아키텍처 및 실행에서는 다릅니다.

표 3.2. Jenkins 및 OpenShift Pipelines의 실행 모델 비교

Jenkins	OpenShift Pipelines
Jenkins에는 컨트롤러 노드가 있습니다. Jenkins는 파이프라인을 실행하고 중앙 집중적으로 단계를 실행하거나 다른 노드에서 실행되는 작업을 오케스트레이션합니다.	OpenShift Pipelines는 서버리스 및 배포되며 실행을 위한 중앙 종속성이 없습니다.
컨테이너는 파이프라인을 통해 Jenkins 컨트롤러 노드에서 시작됩니다.	OpenShift Pipelines는 모든 단계가 Pod에서 컨테이너로 실행되는 '컨테이너 우선' 접근 방식을 채택합니다(Jenkins의 노드와 동일).
확장성은 플러그인을 사용하여 달성합니다.	확장성은 Tekton Hub의 작업을 사용하거나 사용자 지정 작업 및 스크립트를 생성하여 얻을 수 있습니다.

3.6. 일반적인 사용 사례 예

Jenkins 및 OpenShift Pipelines 모두 다음과 같은 일반적인 CI/CD 사용 사례에 대한 기능을 제공합니다.

- Apache Maven을 사용하여 이미지 컴파일, 빌드 및 배포
- 플러그인을 사용하여 코어 기능 확장
- 공유 가능한 라이브러리 및 사용자 지정 스크립트 사용

3.6.1. Jenkins 및 OpenShift Pipelines에서 Maven 파이프라인 실행

Jenkins 및 OpenShift Pipelines 워크플로우 모두에서 Maven을 사용하여 이미지를 컴파일, 빌드 및 배포할 수 있습니다. 기존 Jenkins 워크플로를 OpenShift Pipelines에 매핑하려면 다음 예제를 고려하십시오.

예: 이미지를 컴파일 및 빌드하고 Jenkins에서 Maven을 사용하여 OpenShift에 배포합니다.

```
#!/usr/bin/groovy
node('maven') {
   stage 'Checkout'
   checkout scm

   stage 'Build'
   sh 'cd helloworld && mvn clean'
   sh 'cd helloworld && mvn compile'

   stage 'Run Unit Tests'
   sh 'cd helloworld && mvn test'

   stage 'Package'
   sh 'cd helloworld && mvn package'

   stage 'Archive artifact'
```

```
    sh 'mkdir -p artifacts/deployments && cp helloworld/target/*.war artifacts/deployments'
    archive 'helloworld/target/*.war'

    stage 'Create Image'
    sh 'oc login https://kubernetes.default -u admin -p admin --insecure-skip-tls-verify=true'
    sh 'oc new-project helloworldproject'
    sh 'oc project helloworldproject'
    sh 'oc process -f helloworld/jboss-eap70-binary-build.json | oc create -f -'
    sh 'oc start-build eap-helloworld-app --from-dir=artifacts/'

    stage 'Deploy'
    sh 'oc new-app helloworld/jboss-eap70-deploy.json' }
```

예: 이미지를 컴파일 및 빌드하고 OpenShift Pipelines의 Maven을 사용하여 OpenShift에 배포합니다.

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: maven-pipeline
spec:
  workspaces:
    - name: shared-workspace
    - name: maven-settings
    - name: kubeconfig-dir
      optional: true
  params:
    - name: repo-url
    - name: revision
    - name: context-path
  tasks:
    - name: fetch-repo
      taskRef:
        name: git-clone
      workspaces:
        - name: output
          workspace: shared-workspace
      params:
        - name: url
          value: "$(params.repo-url)"
        - name: subdirectory
          value: ""
        - name: deleteExisting
          value: "true"
        - name: revision
          value: $(params.revision)
    - name: mvn-build
      taskRef:
        name: maven
      runAfter:
        - fetch-repo
      workspaces:
        - name: source
          workspace: shared-workspace
        - name: maven-settings
```

```
      workspace: maven-settings
  params:
    - name: CONTEXT_DIR
      value: "$(params.context-path)"
    - name: GOALS
      value: ["-DskipTests", "clean", "compile"]
- name: mvn-tests
  taskRef:
    name: maven
  runAfter:
    - mvn-build
  workspaces:
    - name: source
      workspace: shared-workspace
    - name: maven-settings
      workspace: maven-settings
  params:
    - name: CONTEXT_DIR
      value: "$(params.context-path)"
    - name: GOALS
      value: ["test"]
- name: mvn-package
  taskRef:
    name: maven
  runAfter:
    - mvn-tests
  workspaces:
    - name: source
      workspace: shared-workspace
    - name: maven-settings
      workspace: maven-settings
  params:
    - name: CONTEXT_DIR
      value: "$(params.context-path)"
    - name: GOALS
      value: ["package"]
- name: create-image-and-deploy
  taskRef:
    name: openshift-client
  runAfter:
    - mvn-package
  workspaces:
    - name: manifest-dir
      workspace: shared-workspace
    - name: kubeconfig-dir
      workspace: kubeconfig-dir
  params:
    - name: SCRIPT
      value: |
        cd "$(params.context-path)"
        mkdir -p ./artifacts/deployments && cp ./target/*.war ./artifacts/deployments
        oc new-project helloworldproject
        oc project helloworldproject
        oc process -f jboss-eap70-binary-build.json | oc create -f -
        oc start-build eap-helloworld-app --from-dir=artifacts/
        oc new-app jboss-eap70-deploy.json
```

3.6.2. 플러그인을 사용하여 Jenkins 및 OpenShift Pipelines의 핵심 기능 확장

Jenkins는 광범위한 사용자 기반에 의해 수년 동안 개발 된 수많은 플러그인의 대규모 에코 시스템의 이점을 가지고 있습니다. Jenkins 플러그인 색인 에서 플러그인을 검색하고 검색할 수 있습니다.

OpenShift Pipelines에는 커뮤니티 및 엔터프라이즈 사용자가 개발하고 기여한 많은 작업이 있습니다. 재사용 가능한 OpenShift Pipelines 작업의 공개적으로 카탈로그는 Tekton Hub 에서 사용할 수 있습니다.

또한 OpenShift Pipelines는 핵심 기능 내에 Jenkins 에코 시스템의 많은 플러그인을 통합합니다. 예를 들어 권한 부여는 Jenkins 및 OpenShift Pipelines 모두에서 중요한 기능입니다. Jenkins는 역할 기반 권한 부여 전략 플러그인을 사용하여 권한 부여를 수행하는 반면, OpenShift Pipelines는 OpenShift의 기본 제공 역할 기반 액세스 제어 시스템을 사용합니다.

3.6.3. Jenkins 및 OpenShift Pipelines에서 재사용 가능한 코드 공유

Jenkins 공유 라이브러리는 Jenkins 파이프라인의 일부에 재사용 가능한 코드를 제공합니다. 라이브러리는 코드 반복 없이 고도로 모듈식 파이프라인을 생성하기 위해 Jenkinsfiles 간에 공유됩니다.

OpenShift Pipelines에는 Jenkins 공유 라이브러리가 직접적으로 동일하지는 않지만 사용자 지정 작업 및 스크립트와 함께 Tekton Hub 의 작업을 사용하여 유사한 워크플로를 수행할 수 있습니다.

3.7. 추가 리소스

- OpenShift Pipelines 이해
- 역할 기반 액세스 제어

4장. OPENSHIFT JENKINS 이미지에 대한 중요한 변경 사항

OpenShift Container Platform 4.11은 OpenShift Jenkins 및 OpenShift 에이전트 기본 이미지를 **registry.redhat.io** 의 **ocp-tools-4** 리포지토리로 이동합니다. 또한 페이로드에서 OpenShift Jenkins Maven 및 NodeJS 에이전트 이미지도 제거합니다.

- OpenShift Container Platform 4.11은 OpenShift Jenkins 및 OpenShift Agent Base 이미지를 **registry.redhat.io** 의 **ocp-tools-4** 리포지토리로 이동하여 Red Hat이 OpenShift Container Platform 라이프사이클 외부에서 이미지를 생성하고 업데이트할 수 있습니다. 이전에는 이러한 이미지가 OpenShift Container Platform 설치 페이로드에 있고 **registry.redhat.io**의 **openshift4** 리포지토리에 있었습니다.

- OpenShift Container Platform 4.10은 OpenShift Jenkins Maven 및 NodeJS 에이전트 이미지를 더 이상 사용되지 않습니다. OpenShift Container Platform 4.11은 페이로드에서 이러한 이미지를 제거합니다. Red Hat은 더 이상 이러한 이미지를 생성하지 않으며 **registry.redhat.io**의 **ocp-tools-4** 리포지토리에서 사용할 수 없습니다. Red Hat은 OpenShift Container Platform 라이프사이클 정책에 따라 중요한 버그 수정 또는 보안 CVE에 대해 4.10 및 이전 버전의 이미지를 유지 관리합니다.

이러한 변경 사항은 Jenkins Kubernetes 플러그인에서 여러 컨테이너 포드 템플릿을 사용하도록 OpenShift Container Platform 4.10 권장 사항을 지원합니다.

4.1. OPENSHIFT JENKINS 이미지 재배치

OpenShift Container Platform 4.11은 특정 OpenShift Jenkins 이미지의 위치와 가용성을 크게 변경합니다. 또한 이러한 이미지를 업데이트할 시기와 방법을 구성할 수 있습니다.

OpenShift Jenkins 이미지와 동일하게 유지되는 것은 무엇입니까?

- Cluster Samples Operator는 OpenShift Jenkins 이미지 작동을 위한 **ImageStream** 및 **Template** 오브젝트를 관리합니다.

- 기본적으로 Jenkins Pod 템플릿의 Jenkins **DeploymentConfig** 오브젝트는 Jenkins 이미지가 변경될 때 재배포를 트리거합니다. 기본적으로 이 이미지는 Samples Operator 페이로드의 **ImageStream** YAML 파일에 있는 **openshift** 네임스페이스에 있는 Jenkins 이미지 스트림 태그의 **jenkins:2** 이미지 스트림 태그에서 참조합니다.

- OpenShift Container Platform 4.10 및 이전 버전에서 4.11로 업그레이드하는 경우 더 이상 사용되지 않는 **maven** 및 **nodejs** pod 템플릿은 여전히 기본 이미지 구성에 있습니다.

- OpenShift Container Platform 4.10 및 이전 버전에서 4.11로 업그레이드하는 경우 **jenkins-agent-maven** 및 **jenkins-agent-nodejs** 이미지 스트림이 여전히 클러스터에 있습니다. 이러한 이미지 스트림을 유지하려면 "**openshift** 네임스페이스의 **jenkins-agent-maven** 및 **jenkins-agent-nodejs** 이미지 스트림에서는 어떻게 됩니까?"를 참조하십시오.

OpenShift Jenkins 이미지의 지원 매트릭스는 어떻게 됩니까?

registry.redhat.io 레지스트리의 **ocp-tools-4** 리포지토리의 각 새 이미지는 여러 버전의 OpenShift Container Platform을 지원합니다. Red Hat은 이러한 새 이미지 중 하나를 업데이트하면 모든 버전에서 동시에 사용할 수 있습니다. 이 가용성은 Red Hat이 보안 권고에 대응하여 이미지를 업데이트할 때 이상적입니다. 처음에는 이 변경 사항이 OpenShift Container Platform 4.11 이상에 적용됩니다. 이러한 변경 사항은 결국 OpenShift Container Platform 4.9 이상에 적용될 예정입니다.

이전에는 각 Jenkins 이미지가 하나의 OpenShift Container Platform 버전만 지원했으며 Red Hat은 시간이 지남에 따라 해당 이미지를 순차적으로 업데이트할 수 있었습니다.

OpenShift Jenkins 및 Jenkins 에이전트 기본 ImageStream 및 ImageStreamTag 오브젝트에는 어떤 추가 기능이 있습니까?

in-knativeload 이미지 스트림에서 비지침 로드 이미지를 참조하는 이미지 스트림으로 이동하면 OpenShift Container Platform에서 추가 이미지 스트림 태그를 정의할 수 있습니다. Red Hat은 OpenShift Container Platform 4.10 및 이전 버전에 존재하는 **"value": "jenkins:2"** 및 **"value": "image-registry.openshift-image-registry.svc:5000/openshift/jenkins-agent-base-rhel8:latest"** 와 함께 사용할 수 있도록 일련의 새 이미지 스트림 태그를 생성했습니다. 이러한 새 이미지 스트림 태그는 Jenkins 관련 이미지 스트림을 유지하는 방법을 개선하기 위해 일부 요청을 처리합니다.

새 이미지 스트림 태그 정보:

ocp-upgrade-redeploy

OpenShift Container Platform을 업그레이드할 때 Jenkins 이미지를 업데이트하려면 Jenkins 배포 구성에서 이 이미지 스트림 태그를 사용하십시오. 이 이미지 스트림 태그는 **jenkins** 이미지 스트림의 기존 **2** 이미지 스트림 태그와 **jenkins-agent-base-rhel8** 이미지 스트림의 **latest** 이미지 스트림 태그에 해당합니다. 하나의 SHA 또는 이미지 다이제스트에만 이미지 태그를 사용합니다. Jenkins 보안 권고와 같이 **ocp-tools-4** 이미지가 변경되면 Red Hat Engineering에서 Cluster Samples Operator 페이로드를 업데이트합니다.

user-maintained-upgrade-redeploy

OpenShift Container Platform을 업그레이드한 후 Jenkins를 수동으로 재배포하려면 Jenkins 배포 구성에서 이 이미지 스트림 태그를 사용하십시오. 이 이미지 스트림 태그는 사용 가능한 가장 구체적인 이미지 버전 표시기를 사용합니다. Jenkins를 재배포하는 경우 **$ oc import-image jenkins:user-maintained-upgrade-redeploy -n openshift** 를 실행합니다. 이 명령을 실행하면 OpenShift Container Platform **ImageStream** 컨트롤러에서 **registry.redhat.io** 이미지 레지스트리에 액세스하고 해당 Jenkins **ImageStreamTag** 오브젝트의 OpenShift 이미지 레지스트리 슬롯에 업데이트된 이미지를 저장합니다. 그렇지 않으면 이 명령을 실행하지 않으면 Jenkins 배포 구성에서 재배포를 트리거하지 않습니다.

scheduled-upgrade-redeploy

최신 버전의 Jenkins 이미지가 릴리스될 때 자동으로 재배포하려면 Jenkins 배포 구성에서 이 이미지 스트림 태그를 사용하십시오. 이 이미지 스트림 태그는 OpenShift Container Platform 이미지 스트림 컨트롤러의 이미지 스트림 태그 기능을 주기적으로 가져와서 백업 이미지 변경 사항을 확인합니다. 예를 들어 최근 Jenkins 보안 권고로 인해 이미지가 변경되면 OpenShift Container Platform에서 Jenkins 배포 구성의 재배포를 트리거합니다. 다음 "ECDHE 리소스"의 "이미지 스트림 태그 구성"을 참조하십시오.

openshift 네임스페이스의 jenkins-agent-maven 및 jenkins-agent-nodejs 이미지 스트림은 어떻게 됩니까?

OpenShift Container Platform의 OpenShift Jenkins Maven 및 NodeJS 에이전트 이미지는 4.10에서 더 이상 사용되지 않으며 4.11의 OpenShift Container Platform 설치 페이로드에서 제거됩니다. 이러한 대안은 **ocp-tools-4** 리포지토리에 정의되어 있지 않습니다. 그러나 다음 "Jenkins 에이전트" 섹션에 설명된 사이드카 패턴을 사용하여 이 문제를 해결할 수 있습니다.

그러나 Cluster Samples Operator는 이전 릴리스에서 생성한 **jenkins-agent-maven** 및 **jenkins-agent-nodejs** 이미지 스트림을 삭제하지 않으므로 **registry.redhat.io** 의 각 OpenShift Container Platform 페이로드 이미지의 태그를 가리킵니다. 따라서 다음 명령을 실행하여 이러한 이미지에 대한 업데이트를 가져올 수 있습니다.

```
$ oc import-image jenkins-agent-nodejs -n openshift
```

```
$ oc import-image jenkins-agent-maven -n openshift
```

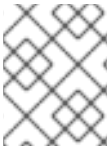
4.2. JENKINS 이미지 스트림 태그 사용자 정의

기본 업그레이드 동작을 재정의하고 Jenkins 이미지 업그레이드 방법을 제어하려면 Jenkins 배포 구성에서 사용하는 이미지 스트림 태그 값을 설정합니다.

기본 업그레이드 동작은 Jenkins 이미지가 설치 페이로드의 일부일 때 존재하는 동작입니다. **jenkins-rhel.json** 이미지 스트림 파일의 이미지 스트림 태그 이름 **2** 및 **ocp-upgrade-redeploy** 에는 SHA 특정 이미지 참조를 사용합니다. 따라서 이러한 태그가 새 SHA로 업데이트되면 OpenShift Container Platform 이미지 변경 컨트롤러는 **jenkins-ephemeral.json** 또는 **jenkins-persistent.json** 과 같은 관련 템플릿에서 Jenkins 배포 구성을 자동으로 재배포합니다.

새 배포의 경우 해당 기본값을 재정의하려면 **jenkins-ephemeral.json** Jenkins 템플릿에서 **JENKINS_IMAGE_STREAM_TAG** 의 값을 변경합니다. 예를 들어 **"value": "jenkins:2"**의 **2** 를 다음 이미지 스트림 태그 중 하나로 바꿉니다.

- **OCP-upgrade-redeploy**, 기본값은 OpenShift Container Platform을 업그레이드할 때 Jenkins 이미지를 업데이트합니다.
- **user-maintained-upgrade-redeploy** 를 사용하려면 OpenShift Container Platform을 업그레이드한 후 **$ oc import-image jenkins:user-maintained-upgrade-redeploy -n openshift** 를 실행하여 Jenkins를 수동으로 재배포해야 합니다.
- **scheduled-upgrade-redeploy** 는 지정된 **<image>:<tag>** 조합에서 변경 사항이 있는지 주기적으로 확인하고 변경 시 이미지를 업그레이드합니다. 이미지 변경 컨트롤러는 변경된 이미지를 가져와서 템플릿에서 프로비저닝한 Jenkins 배포 구성을 재배포합니다. 이 예약된 가져오기 정책에 대한 자세한 내용은 다음 "ECDHE 리소스"의 "이미지 스트림에 태그 추가"를 참조하십시오.



참고

기존 배포의 현재 업그레이드 값을 재정의하려면 해당 템플릿 매개변수에 해당하는 환경 변수의 값을 변경합니다.

사전 요구 사항

- OpenShift Container Platform 4.15에서 OpenShift Jenkins를 실행 중입니다.
- OpenShift Jenkins가 배포된 네임스페이스를 알고 있습니다.

프로세스

- 이미지 스트림 태그 값을 설정하고 **<namespace>**를 OpenShift Jenkins가 배포된 네임스페이스로 바꾸고 **<image_stream_tag>**를 이미지 스트림 태그로 바꿉니다.

  예제

  ```
  $ oc patch dc jenkins -p '{"spec":{"triggers":[{"type":"ImageChange","imageChangeParams":
  {"automatic":true,"containerNames":["jenkins"],"from":
  {"kind":"ImageStreamTag","namespace":"<namespace>","name":"jenkins:
  <image_stream_tag>"}}}]}}'
  ```

  작은 정보

  또는 Jenkins 배포 구성 YAML을 편집하려면 **$ oc edit dc/jenkins -n <namespace>**를 입력하고 **value: 'jenkins:<image_stream_tag>'** 행을 업데이트합니다.

4.3. 추가 리소스

- 이미지 스트림에 태그 추가
- 주기적으로 이미지 스트림 태그 가져오기 구성
- Jenkins 에이전트
- 인증된 **jenkins** 이미지
- 인증된 **jenkins-agent-base** 이미지
- 인증된 **jenkins-agent-maven** 이미지
- 인증된 **jenkins-agent-nodejs** 이미지