W. Caban, *Architecting and Operating OpenShift Clusters*
https://doi.org/10.1007/978-1-4842-4985-7_8

8. Architecting OpenShift Jenkins Pipelines

William Caban [1]

(1) Columbia, MD, USA

The OpenShift platform provides multiple features to enhance the developer experience. These features are enabled and managed using the same RBAC and SCC options seen in Chapter 7. This chapter focuses on the OpenShift Jenkins Pipelines capabilities.

The *OpenShift Jenkins Pipelines* capabilities in *OpenShift Container Platform (OCP)* provide the ability to create advanced CI/CD pipelines that can be used to create new CI/CD processes, or to integrate with existing organizations CI/CD processes.

OpenShift Jenkins Pipelines provide support for using CI/CD pipelines to build, deploy, and promote applications on OpenShift. These Pipelines can use a combination of the *Jenkins Pipeline Build Strategy*, *Jenkinsfiles,* and the *OpenShift Jenkins Client Plugin*.

This chapter describes the basic configurations to start using the capabilities provided by the *OpenShift CI/CD* feature.

CI/CD Pipelines As a Service with OpenShift

When using the *Jenkins Pipeline Build Strategy* or using a *Jenkinsfile*, *OpenShift* CI/CD capabilities autoprovision a *Jenkins Master* for the *Project* and the *Jenkins Slaves* required to complete the stages.

This *Jenkins Master* will be used to execute all the *Jenkins Pipelines* defined at the *Project*.

By default, the *Jenkins Master* server uses the *OpenShift Jenkins-ephemeral* template to instantiate the server. To deploy a Jenkins server with persistent storage for the data and configuration stored in /var/lib/jenkins, the Project admin can manually deploy a Jenkins Master using the Jenkins-persistent template from the self-service catalog. To change the default Jenkins template, a cluster-admin can modify the *Master Nodes* configuration [1] to

set up the Jenkins-persistent template as the default template to use when autoprovisioning a Jenkins server (see Listing 8-1).

```
# Update /etc/origin/master/master-config.yaml to
include
jenkinsPipelineConfig:
  autoProvisionEnabled: true
  templateNamespace: openshift
  templateName: jenkins-persistent
  serviceName: jenkins-persistent-svc
```

Listing 8-1 Jenkins-persistent as default template for autoprovisioning of Jenkins servers

During the instantiation of the Jenkins Master, the process

- Deploys *Jenkins* into the *Project* using the official *OpenShift Jenkins* image
  - The Jenkins deployment can be done using ephemeral or persistent storage.
- Creates *Service* and *Route* resources for the *Jenkins Master*
- Creates a *jenkins Service Account (SA)* in the *Project*
  - Grant *Project*-level *edit* access to the new *jenkins Service Account*

When using an *OpenShift Pipeline* across *Projects*, the *jenkins* SA on the project hosting the *Jenkins Master* requires *edit* access level on the *Projects* it will manage.

```
# Option 1: Grant 'edit' access to 'jenkins' Service
Account on specific Projects
oc policy add-role-to-user edit system:serviceaccount:
<cicd-project>:jenkins -n <target-project>

# Option 2: Grant 'edit' access to 'jenkins' Service
Account on all Projects
oc adm policy add-cluster-role-to-user edit
system:serviceaccount:<cicd-project>:jenkins
```

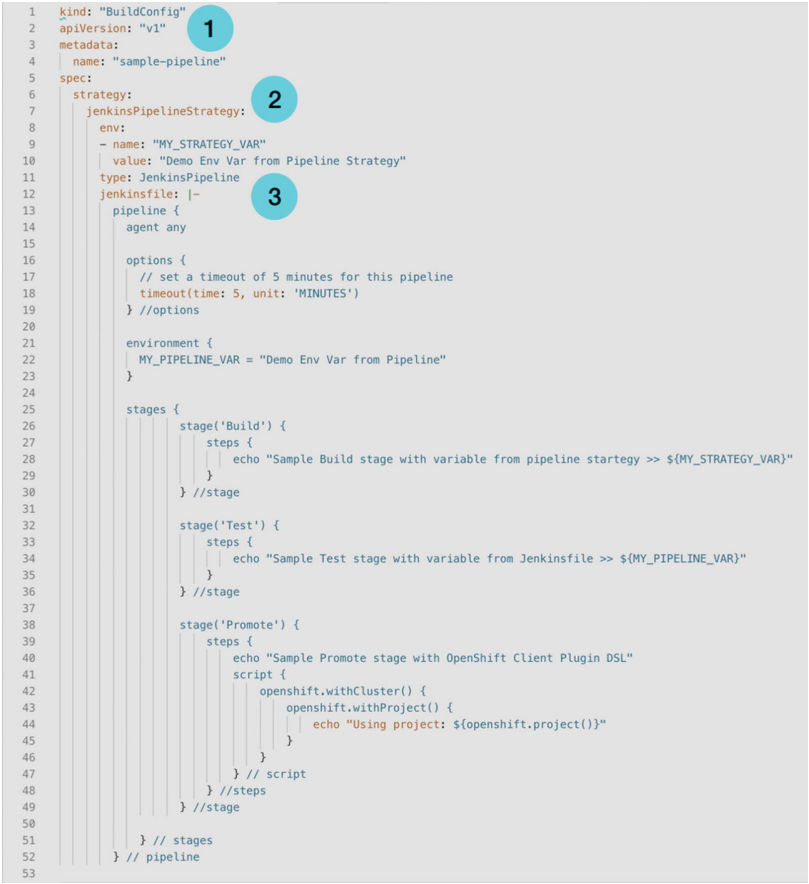
Listing 8-2 Grant 'edit' access to 'jenkins' Service Account

Jenkins Pipeline Build Strategy

OpenShift has the notion of *build configurations* or *BuildConfigs.* A *BuildConfig* is a configuration describing a single build definition. This includes information like the triggers that will provoke a new *build* and the *build strategy* to use. The *build strategy* determines the process to be used to execute a *build*. One of the build strategies is the *Pipeline Build Strategy.* [2]

The *Pipeline Build Strategy* is an *OpenShift Build* [3] type that enables developers to define *Jenkins* pipeline workflows which are executed inside the *OpenShift* platform.

To use this *Build Strategy*, the *Jenkins Pipeline* is defined in a *Jenkinsfile*. This can be embedded directly in the *BuildConfig* (see #3 on Figure 8-1) or provided on a *Git* repository (see #2 on Figure 8-2) referenced by the *BuildConfig* (see #3 on Figure 8-2).




```
kind: "BuildConfig"
apiVersion: "v1"
metadata:
  name: "sample-pipeline"
spec:
  strategy:
    jenkinsPipelineStrategy:
      env:
      - name: "MY_STRATEGY_VAR"
        value: "Demo Env Var from Pipeline Strategy"
      type: JenkinsPipeline
      jenkinsfile: |-
        pipeline {
          agent any

          options {
            // set a timeout of 5 minutes for this pipeline
            timeout(time: 5, unit: 'MINUTES')
          } //options

          environment {
            MY_PIPELINE_VAR = "Demo Env Var from Pipeline"
          }

          stages {
              stage('Build') {
                  steps {
                      echo "Sample Build stage with variable from pipeline startegy >> ${MY_STRATEGY_VAR}"
                  }
              } //stage

              stage('Test') {
                  steps {
                      echo "Sample Test stage with variable from Jenkinsfile >> ${MY_PIPELINE_VAR}"
                  }
              } //stage

              stage('Promote') {
                  steps {
                      echo "Sample Promote stage with OpenShift Client Plugin DSL"
                      script {
                          openshift.withCluster() {
                              openshift.withProject() {
                                  echo "Using project: ${openshift.project()}"
                              }
                          }
                      } // script
                  } //steps
              } //stage

          } // stages
        } // pipeline

```

Figure 8-1 OpenShift Pipeline Build Strategy with embedded Jenkinsfile definition

The *BuildConfig* with the embedded Jenkins pipeline definition is a YAML formatted configuration file specifying the *Jenkins Pipeline Strategy* (see #2 on Figure 8-1). The content of the *Jenkinsfile* is included as a multiline string block (see #3 on Figure 8-1) in the definition.

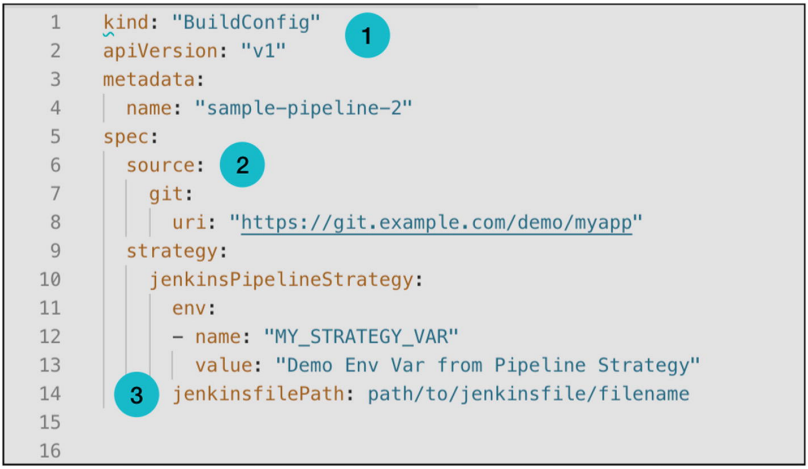


Figure 8-2 OpenShift Pipeline Build Strategy with Git Jenkinsfile definition

The other option for the Pipeline Strategy is the *BuildConfig* referencing a Jenkinsfile (see #3 on Figure 8-2) on a Git repository (see #2 on Figure 8-2). In this particular case, the Jenkinsfile can be in any directory of the referenced Git repository and can have any name as long as the full path and filename are specified in the corresponding *Jenkinsfile Path* variable. If this variable is not defined, it will retrieve a file named *Jenkinsfile* from the root directory of the Git repo.

CREATING THE PIPELINE BUILDCONFIG

The BuildConfig on Figure 8-1 is for a sample pipeline that defines environment variables at the pipeline strategy level (line 8 on Figure 8-1) and at the *Jenkinsfile* level (line 21 on Figure 8-1). The embedded *Jenkinsfile* defines a Jenkins Pipeline (line 13 on Figure 8-1) with some sample stages (line 25 on Figure 8-1). For the purpose of this example, there are three stages. To maintain a minimal structure to illustrate the use of the pipeline, in this example, each stage simply displays a message.

The YAML configuration for the *BuildConfig sample-pipeline* from Figure 8-1 is shown in Listing 8-3.

```
kind: "BuildConfig"
apiVersion: "v1"
metadata:
```

```
  name: "sample-pipeline"
spec:
  strategy:
    jenkinsPipelineStrategy:
      env:
      - name: "MY_STRATEGY_VAR"
        value: "Demo Env Var from Pipeline Strategy"
      type: JenkinsPipeline
      jenkinsfile: |-
        pipeline {
          agent any

          options {
            // set a timeout of 5 minutes for this
pipeline
            timeout(time: 5, unit: 'MINUTES')
          } //options

          environment {
            MY_PIPELINE_VAR = "Demo Env Var from
Pipeline"
          }

          stages {
                  stage('Build') {
                      steps {
                          echo "Sample Build stage
with variable from pipeline startegy >>
${MY_STRATEGY_VAR}"
                      }
                  } //stage

                  stage('Test') {
                      steps {
                          echo "Sample Test stage with
variable from Jenkinsfile >> ${MY_PIPELINE_VAR}"
                      }
                  } //stage

                  stage('Promote') {
                      steps {
```

```
                            echo "Sample Promote stage
with OpenShift Client Plugin DSL"
                            script {
                                openshift.withCluster()
{
                                    openshift.withProjec
t() {
                                        echo "Using
project: ${openshift.project()}"
                                    }
                                }
                            } // script
                        } //steps
                    } //stage

            } // stages
        } // pipeline
```

Listing 8-3 Sample Pipeline BuildConfig with embedded Jenkinsfile

The YAML configuration for the *BuildConfig sample-pipeline-2* from Figure 8-2 is shown in Listing 8-4.

```
kind: "BuildConfig"
apiVersion: "v1"
metadata:
  name: "sample-pipeline-2"
spec:
  source:
    git:
      uri: "https://git.example.com/demo/myapp"
  strategy:
    jenkinsPipelineStrategy:
      env:
      - name: "MY_STRATEGY_VAR"
        value: "Demo Env Var from Pipeline Strategy"
      jenkinsfilePath: path/to/jenkinsfile/filename
```

Listing 8-4 Sample Pipeline BuildConfig with Git referenced Jenkinsfile

DEPLOYING THE PIPELINE BUILDCONFIG

The BuildConfig is created at a Project level. It is up to the user to use a dedicated Project for the Pipeline and another for the application or use the same Project for the Pipeline and application.

From the OpenShift Application Console, import the YAML for the BuildConfig (see #1 on Figure 8-3).

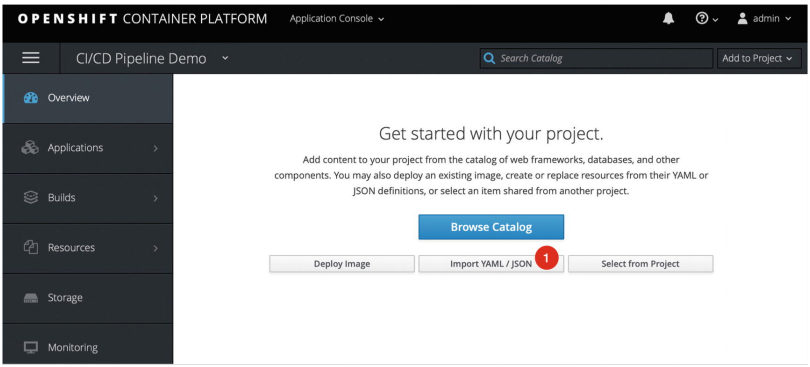


Figure 8-3 Import BuildConfig YAML definition

The *Import YAML* window allows for uploading a YAML file from the local machine or for the copy and paste of the *BuildConfig* at the editor window (see #1 on Figure 8-4). On the successful upload or definition of the *BuildConfig,* a new Pipeline is created.

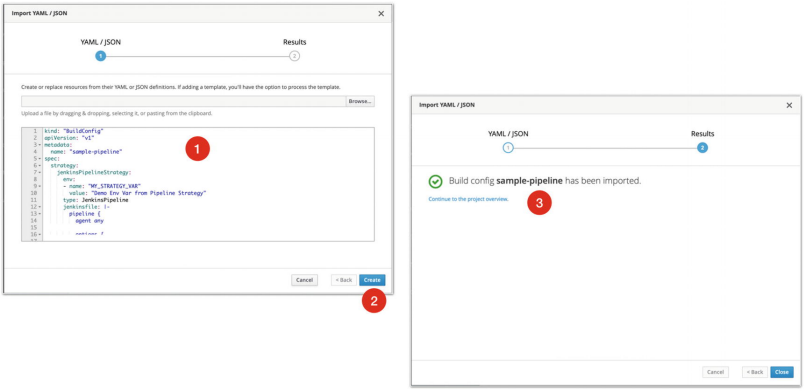


Figure 8-4 Importing the BuildConfig and creating the Pipeline

The first time a *Pipeline* strategy is defined for a *Project*, *OpenShift* instantiates a *Jenkins Master* server in that *Project* (see Figure 8-5). This

Jenkins server is used to execute the Pipeline definition from the BuildConfig.

> **NOTE** Additional *Pipeline Build* configurations or *BuildConfigs*, in the same project, will share the same Jenkins server.

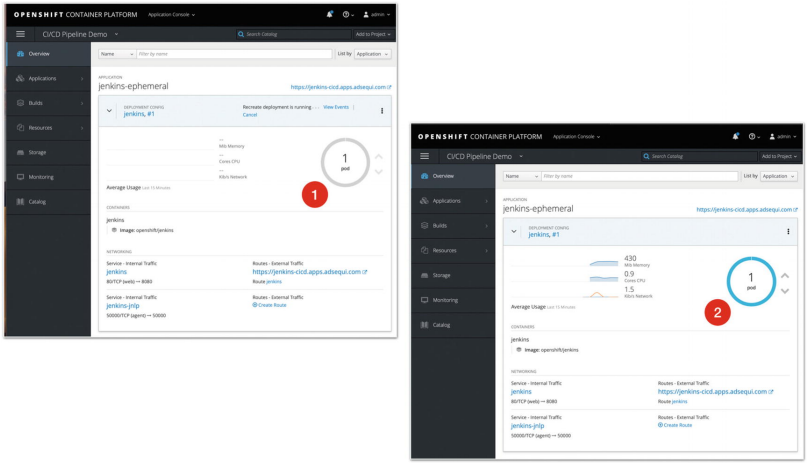


Figure 8-5 Instantiation of an embedded Jenkins server

> **NOTE** The instantiation of the initial Jenkins server takes some time to complete. After about 10 minutes after the instantiation, the system will be ready to receive triggers to execute the Pipeline.

The *Pipeline* can be triggered by a *Webhook, Image Change, Configuration Change, or Manually*. To execute a manual trigger from GUI, at the *Application Console*, go to *Builds* ➤ *Pipelines* (see #1 and #2 on Figure 8-6) or from CLI (see #3 on Figure 8-6).

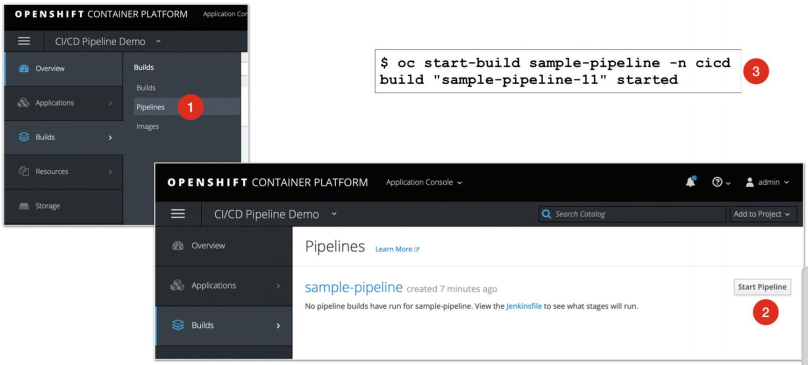


Figure 8-6 Executing the Pipeline Build Strategy

A visual representation of the pipeline will be highlighting the step that is executing (see #2 on Figure 8-7). As stages are successful, the stage representation will be colored green.

After several execution of the Pipeline, the *History* tab of the Pipeline pane will show a histogram of the time it took to complete an execution and a color-coded view showing failed and successful attempts (see #3 on Figure 8-7).

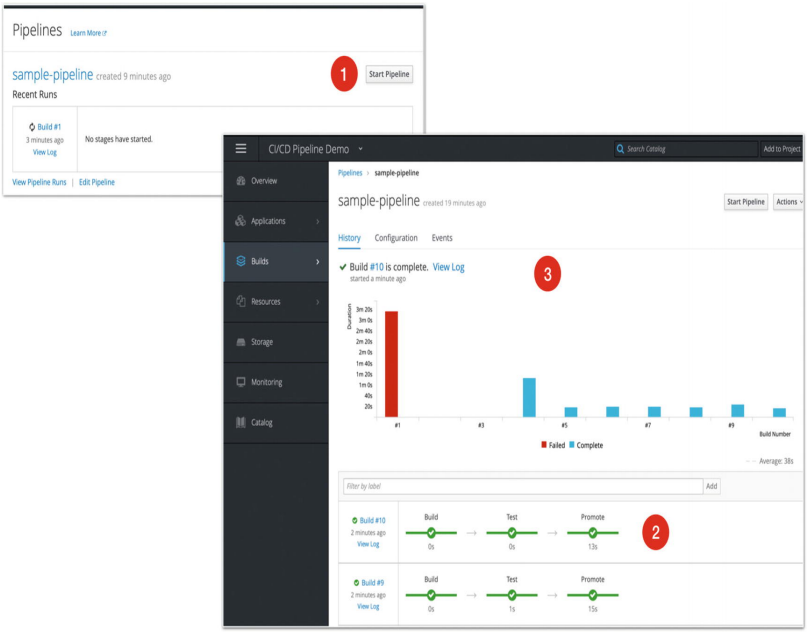


Figure 8-7 Pipeline Build History

The *OpenShift* integration with the Jenkins instance allows access to the logs generated during the execution of the *Pipeline Build*.

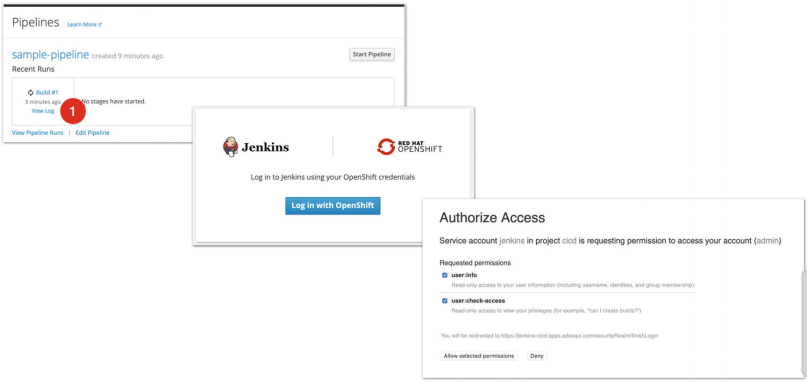


Figure 8-8 View Logs from Jenkins Console

To access the Logs for a particular *Build* execution, select the *View Log* link under the execution number (see #1 on Figure 8-8). This will redirect to the *Jenkins Console* where *OpenShift* credentials can be used to log in to the Jenkins server and see the logs (see Figure 8-9).

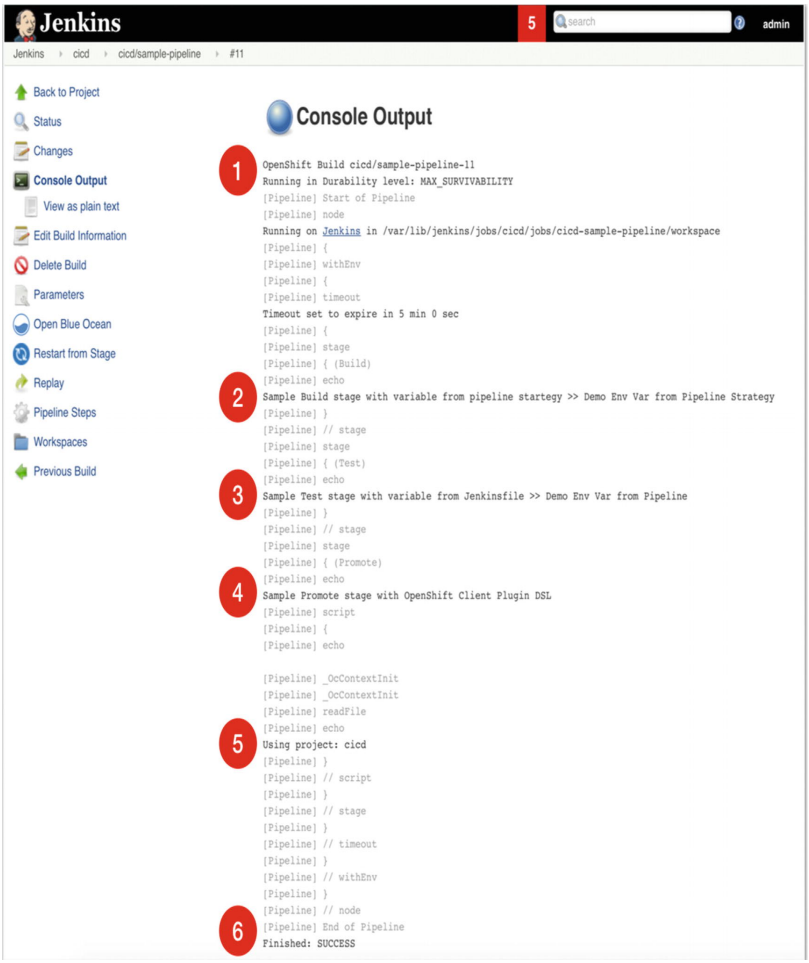


Figure 8-9 Build Logs at Jenkins Console

The logs for a particular *Pipeline* will include the actions and output from those actions (see #2, #3, #4, and #5 on Figure 8-9), for each one of the *Pipeline Stages* defined by the *Jenkinsfile*.

NOTE The Jenkins server must be manually deleted by the user. It will not be automatically removed, even after deleting all Pipeline build configurations.

Jenkinsfile with Source Code

When using this option, the *Jenkinsfile* must be included with the application source code at the root of the repository or at the root of the *contextDir* of the repository. When deploying an application and referencing a repository containing a *Jenkinsfile*

- If there is not an existing Jenkins instance in the Project, *OpenShift* creates a *DeploymentConfig* and deploys a Jenkins instance.

- OpenShift creates a *BuildConfig* (see #1 on Figure 8-10) with

  - A *jenkinsPipelineStrategy* (see #5 on Figure 8-10) referring the *Jenkinsfile* in the Git repository (see #5 on Figure 8-10)

  - A set of Webhook triggers: GitHub and Generic (see #6 and #7 on Figure 8-10)

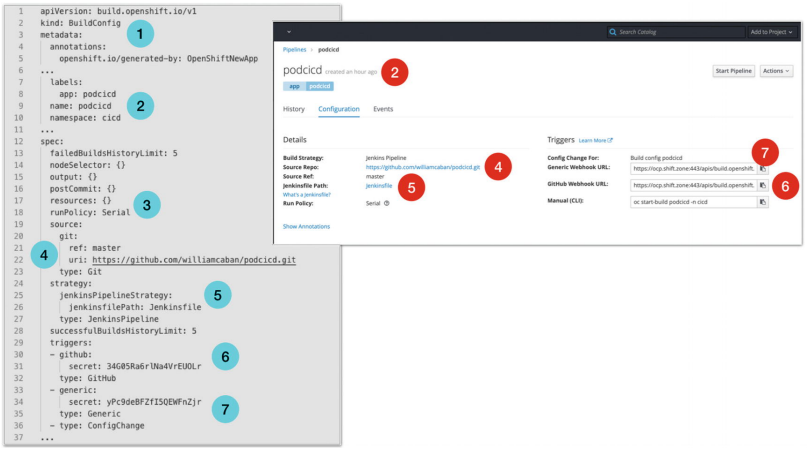


Figure 8-10 Pipeline BuildConfig from Jenkinsfile on Git repository

The URL for the Webhook triggers will follow the format:

`https://`**`<ocp-cluster-fqdn>`**`/apis/build.openshift.io/v1/namespaces/`**`<name-of-project>`**`/buildconfigs/`**`<name-of-buildconfig>`**`/webhooks/`**`<trigger-token>`**`/`**`<trigger-type>`**

These Webhook triggers [4] enable external tools to initiate a new pipeline execution. Figure 8-11 shows a Webhook call (#1 on Figure 8-11) triggering a new Build for the pipeline (#3 on Figure 8-11). #4 on Figure 8-11 clearly shows the CI/CD pipeline was triggered by a *Generic Webhook* call.

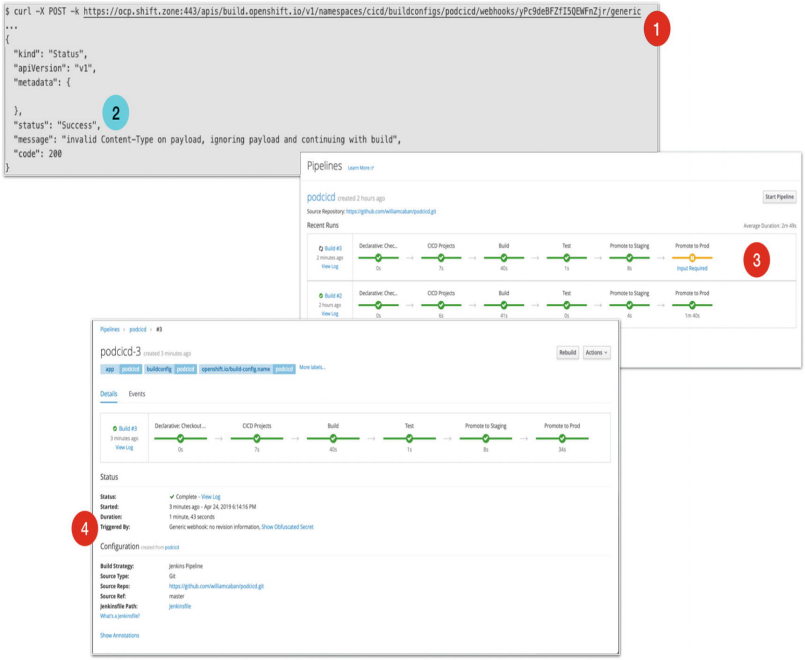


Figure 8-11 Using Webhook triggers to start a Pipeline execution

MULTIPROJECT PIPELINES

When using an OpenShift Jenkins Pipelines to promote an application build across multiple projects, the *jenkins* Service Account must have *edit* access privileges on each of the target Projects as shown in Listing 8-2.

The implementation of a *CI/CD Pipeline* like the one shown in Figure 8-12 involves four different projects. In this example, the *Jenkins Master* is instantiated in the "cicd" project (#2 on Figure 8-12) where it may be used by multiple Pipelines in the same Project.

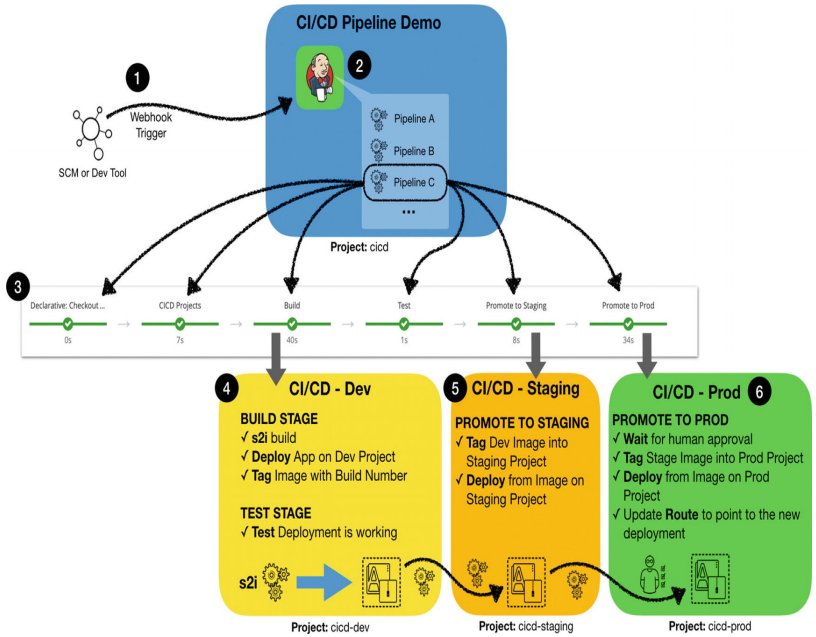


Figure 8-12 Multiproject Pipeline

In this case, "Pipeline C" (#3 on Figure 8-12) has multiple stages across three Projects. A reference Jenkinsfile implementing this type of Pipeline is shown in Listing 8-5.

```
pipeline {
    agent any
    options {
        // set a timeout of 20 minutes for this
pipeline
        timeout(time: 20, unit: 'MINUTES')
    } //options

    environment {
        APP_NAME    = "podcicd"
        GIT_REPO    =
"https://github.com/williamcaban/podcicd.git"
        GIT_BRANCH  = "master"
        CONTEXT_DIR = "myapp"

        CICD_PRJ    = "cicd"
        CICD_DEV    = "${CICD_PRJ}"+"-dev"
        CICD_PROD   = "${CICD_PRJ}"+"-prod"
```

```
        CICD_STAGE  = "${CICD_PRJ}"+"-staging"
        SVC_PORT    = 8080
    }
    stages {
            stage('CICD Projects'){
                steps {
                    echo "Making sure CI/CD projects
exist"
                    script {
                        openshift.withCluster() {
                            echo "Current Pipeline
environment"
                            sh 'env | sort'
                            echo "Making sure required
CI/CD projects exist"
                            try {
                                openshift.selector("pr
ojects",CICD_DEV).exists()
                                echo "Good! Project
${CICD_DEV} exist"
                            } catch (e) {
                                error "Missing
${CICD_DEV} Project or RBAC policy to work with
Project"
                            }
                            try {
                                openshift.selector("pr
ojects",CICD_STAGE).exists()
                                echo "Good! Project
${CICD_STAGE} exist"
                            } catch (e) {
                                error "Missing
${CICD_STAGE} Project or RBAC policy to work with
Project"
                            }
                            try {
                                openshift.selector("pr
ojects",CICD_PROD).exists()
```

```
                                    echo "Good! Project
${CICD_PROD} exist"
                                } catch (e) {
                                    error "Missing
${CICD_PROD} Project or RBAC policy to work with
Project"
                                }

                            } // cluster
                        } // script
                    } //steps
                } // stage - projects

                stage('Build') {
                    steps {
                        echo "Sample Build stage using
project ${CICD_DEV}"
                        script {
                            openshift.withCluster() {
                                openshift.withProject("${C
ICD_DEV}")
                                {

                                    if
(openshift.selector("bc",APP_NAME).exists()) {
                                        echo "Using
existing BuildConfig. Running new Build"
                                        def bc =
openshift.startBuild(APP_NAME)
                                        openshift.set("env
dc/${APP_NAME} BUILD_NUMBER=${BUILD_NUMBER}")
                                        // output build
logs to the Jenkins conosole
                                        echo "Logs from
build"
                                        def result =
bc.logs('-f')
                                        // actions that
took place
```

```
                                    echo "The logs
operation require ${result.actions.size()} 'oc'
interactions"
                                    // see exactly
what oc command was executed.
                                    echo "Logs
executed: ${result.actions[0].cmd}"
                                } else {
                                    echo "No proevious
BuildConfig. Creating new BuildConfig."
                                    def myNewApp =
openshift.newApp (
                                        "${GIT_REPO}#$
{GIT_BRANCH}",
                                        "--
name=${APP_NAME}",
                                        "--context-
dir=${CONTEXT_DIR}",
                                        "-e
BUILD_NUMBER=${BUILD_NUMBER}",
                                        "-e
BUILD_ENV=${openshift.project()}"
                                        )
                                    echo "new-app
myNewApp ${myNewApp.count()} objects named:
${myNewApp.names()}"
                                    myNewApp.describe(
)
                                    // selects the
build config
                                    def bc =
myNewApp.narrow('bc')
                                    // output build
logs to the Jenkins conosole
                                    echo "Logs from
build"
                                    def result =
bc.logs('-f')
```

```
                                    // actions that
took place
                                    echo "The logs
operation require ${result.actions.size()} 'oc'
interactions"
                                    // see exactly
what oc command was executed.
                                    echo "Logs
executed: ${result.actions[0].cmd}"
                                } //else

                                echo "Tag Container
image with 'build number' as version"
                                openshift.tag("${APP_N
AME}:latest", "${APP_NAME}:v${BUILD_NUMBER}")

                                echo "Validating Route
for Service exist, if Not create Route"
                                if
(!openshift.selector("route",APP_NAME).exists()) {
                                    openshift.selector
("svc",APP_NAME).expose()
                                }

                            } // project
                        } // cluster
                    } // script
                } // steps
            } //stage-build
            stage('Test') {

                steps {
                    echo "Testing if 'Service'
resource is operational and responding"
                    script {
                        openshift.withCluster() {
                                openshift.withProject(
) {
                                    echo sh (script:
"curl -I
```

```
${APP_NAME}.${CICD_DEV}.svc:${SVC_PORT}/healthz",
returnStdout: true)
                                } // withProject
                            } // withCluster
                        } // script
                    } // steps
                } //stage

                stage('Promote to Staging') {
                    steps {
                        echo "Setup for Staging"
                        script {
                            openshift.withCluster() {
                                openshift.withProject("${C
ICD_STAGE}") {
                                    echo "Tag new image
for staging"
                                    openshift.tag("${CICD_
DEV}/${APP_NAME}:v${BUILD_NUMBER}",
"${CICD_STAGE}/${APP_NAME}:v${BUILD_NUMBER}")
                                    //openshift.tag("${CIC
D_STAGE}/${APP_NAME}:v${BUILD_NUMBER}",
"${CICD_STAGE}/${APP_NAME}:latest")
                                    echo "Deploying to
project: ${openshift.project()}"
                                    def myStagingApp =
openshift.newApp(
                                        "${APP_NAME}:v${BU
ILD_NUMBER}",
                                        "--
name=${APP_NAME}-v${BUILD_NUMBER}",
                                        "-e
BUILD_NUMBER=${BUILD_NUMBER}",
                                        "-e
BUILD_ENV=${openshift.project()}"
                                    )
                                    myStagingApp.narrow("s
vc").expose()
                                }
                            }
```

```
                    } // script
                } //steps
            } //stage
            stage('Promote to Prod'){
                steps {
                    echo "Promote to production?
Waiting for human input"
                    timeout(time:10, unit:'MINUTES'){
                        input message: "Promote to
Production?", ok: "Promote"
                    }
                    script {
                        openshift.withCluster() {
                            openshift.withProject("${C
ICD_PROD}") {
                                echo "Tag Staging
Image for Production"
                                openshift.tag("${CICD_
STAGE}/${APP_NAME}:v${BUILD_NUMBER}",
"${CICD_PROD}/${APP_NAME}:v${BUILD_NUMBER}")

                                echo "Deploying to
project: ${openshift.project()}"
                                def myProdApp =
openshift.newApp(
                                    "${APP_NAME}:v${BU
ILD_NUMBER}",
                                    "--
name=${APP_NAME}-v${BUILD_NUMBER}",
                                    "-e
BUILD_NUMBER=${BUILD_NUMBER}",
                                    "-e
BUILD_ENV=${openshift.project()}"
                                )

                                if
(openshift.selector("route",APP_NAME).exists()){
                                    echo "Sending the
traffic the the latest version"
```

```
                                        openshift.set("rou
te-backends",APP_NAME,"${APP_NAME}-
v${BUILD_NUMBER}=100%")
                                    } else {
                                        echo "Creating new
Route"
                                        myProdApp.narrow("
svc").expose("--name=${APP_NAME}")
                                    }

                                } // project
                            }
                        } // script
                    } // steps
                } //stage

    } // stages
} // pipeline
```

Listing 8-5 Jenkinsfile—Multiproject Pipeline

For the successful completion of the Pipeline shown in Figure 8-12 and documented in Listing 8-5, the *jenkins Service Account* in the "cicd" Project must have *edit* privileges in the "cicd-dev," "cicd-staging," and "cicd-prod" Projects (see Listing 8-6).

Step 1: Create the CI/CD Project in the OpenShift cluster

```
oc new-project cicd --description="CI/CD Pipeline
Demo"
oc new-project cicd-dev --description="CI/CD - Dev"
oc new-project cicd-prod --description="CI/CD - Prod"
oc new-project cicd-staging --description="CI/CD -
Staging"
```

Step 2: Give jenkins Service Account edit access to the other Projects

```
oc policy add-role-to-user edit
system:serviceaccount:cicd:jenkins -n cicd-dev
oc policy add-role-to-user edit
system:serviceaccount:cicd:jenkins -n cicd-prod
```

```
oc policy add-role-to-user edit
system:serviceaccount:cicd:jenkins -n cicd-staging
Step 3: Deploy the OpenShift Pipeline from a Git
repository containing the Jenkinsfile
oc new-app https://github.com/williamcaban/podcicd.git
-n cicd
```

Listing 8-6 Deploying a Multiproject Pipeline

Deploying the example in the listing once the Jenkins Master is running and the Pipeline BuildConfig is ready, executing a manual trigger or simulating a Webhook trigger should yield results similar to the ones shown in Figure 8-11.

To start a new pipeline build from GUI, go to "Application Console" ➤ Project "cicd" ➤ Builds ➤ Pipelines and click the "Start Pipeline" button. To start a new pipeline build from CLI, execute `oc start-build podcicd -n cicd`. The Pipeline logs and progress are visible at the "Application Console."

OpenShift Client Plugin

The *OpenShift Client Plugin* [5] or the OpenShift Jenkins Pipeline (DSL) Plugin is a Jenkins Plugin that provides comprehensive *Fluent-style* syntax for use in Jenkins Pipelines interacting with OpenShift clusters. The plugin leverages the *OpenShift "oc"* client binary and integrates with Jenkins credentials and cluster.

The OpenShift Client Plugin exposes any option available with "oc" to the Jenkins Pipeline.

> **NOTE** The *OpenShift Client Plugin for Jenkins* supersedes the previous *OpenShift V3 Plugin for Jenkins* which is now deprecated. [6]

Custom Jenkins Images

The *Jenkins Images* can be customized by using the traditional *Docker* layering capabilities with a *Dockerfile* or by using the *OpenShift* native *Source-to-Image* capabilities.

To use the Source-to-Image capabilities, create a Git repository following the structure shown in Figure 8-13.

```
# Git repository structure for Custom Jenkins Image with s2i:
plugins.txt                 <-- List of plugins to install in the format "pluginID:pluginVersion"
plugins/                    <-- binary Jenknins plugins to copy into the Jenkins image
configuration/              <-- The content of this directory will be copied into the /var/lib/jenkins/

# Examples on using the configuration/ folder:
configuration/jobs/         <-- Jenkins job definitions to copy into /var/lib/jenkins/jobs/
configuration/config.xml    <-- Custom Jenkins configuration to copy into /var/lib/jenkins/config.xml
configuration/credentials.xml <-- Credentials configuration to copy into /var/lib/jenkins/credentials.xml
```

Figure 8-13 Git repository structure for custom Jenkins Image with s2i

For the creation of the custom Jenkins Image from the structure defined in a Git repository, create a BuildConfig similar to Listing 8-7.

```
# BuildConfig to customize the Jenkins Image
apiVersion: v1
kind: BuildConfig
metadata:
  name: custom-jenkins-build
spec:
  source:
    git:
      uri: https://github.com/williamcaban/openshift-
custom-jenkins.git
    type: Git
  strategy:
    sourceStrategy:
      from:
        kind: ImageStreamTag
        name: jenkins:latest
        namespace: openshift
    type: Source
  output:
    to:
      kind: ImageStreamTag
      name: custom-jenkins:latest
```

Listing 8-7 BuildConfig for creating custom Jenkins Images

Integrating External CI/CD Pipelines

External Jenkins instances can be integrated with OpenShift in one of the following ways:

- Using the *Jenkins Kubernetes Plugin* [7] which provides the ability for Jenkins agents to be dynamically provisioned [8] on multiple Pods
- Using the *OpenShift Client Plugin* [9] and the *OpenShift Sync Plugin* [10]

The level of integration provided by the *OpenShift Client Plugin* (i.e., embedding pipeline status in the GUI) currently is only available with Jenkins, and it is maintained by Red Hat. Other popular CI/CD tools like GitLab CI, Spinnaker, Bamboo, TeamCity, and so on provide support for OpenShift Container Platform with a vendor-provided plugin for OpenShift or by using their Kubernetes plugin.

Summary

The OpenShift Jenkins Pipelines capabilities enable development teams to continue the adoption of modern development paradigms by providing CI/CD as a first-class service into the platform. When using *Jenkins Pipeline Build Strategy*, or by having a *Jenkinsfile* with the source code, or by using the *OpenShift Jenkins Plugin*, the OpenShift Jenkins Pipelines ease the learning curve for using CI/CD and simplify the management and operation of the Jenkins CI/CD Pipelines.

Beyond knowing how to do the initial administrative tasks or manage value-added features like the CI/CD Pipelines, a cluster administrator should be aware of Day-2 operations and maintenance tasks for maintaining an optimized cluster. Some of these Day-2 tasks are covered in Chapter 9 .

Footnotes

1 Using Jenkins-persistent template: https://docs.openshift.com/container-platform/3.11/install_config/configuring_pipeline_execution.html

2 *OpenShift Pipeline Build Strategy* https://docs.openshift.com/container-platform/3.11/dev_guide/builds/build_strategies.html#pipeline-strategy-options

3 OpenShift *Build* process https://docs.openshift.com/container-platform/3.11/architecture/core_concepts/builds_and_image_streams.ht

4 Additional information on using Webhooks to trigger builds is available from the official documentation: https://docs.openshift.com/container-platform/3.11/dev_guide/builds/triggering_builds.html

5 For the latest documentation and features of the OpenShift Client Plugin, refer to https://github.com/openshift/jenkins-client-plugin

6 For reference to the legacy OpenShift Jenkins Plugin, visit the Git repository: https://github.com/openshift/jenkins-plugin

7 For details about the Jenkins Kubernetes Plugin, refer to https://wiki.jenkins-ci.org/display/JENKINS/Kubernetes+Plugin

8 For configuration details, refer to the OpenShift documentation at https://docs.openshift.com/container-platform/3.11/using_images/other_images/jenkins.html#configuring-the-jenkins-kubernetes-plug-in

9 OpenShift Client Plugin https://docs.openshift.com/container-platform/3.11/using_images/other_images/jenkins.html#client-plug-in

10 OpenShift Sync Plugin https://docs.openshift.com/container-platform/3.11/using_images/other_images/jenkins.html#sync-plug-in