

Advanced Container CI/CD Workshop

Labs

Lab 1 - Explore Environment

Explore OpenShift Environment

<https://console.ocpws.kee.vizuri.com:8443>

Username: student-x

Password: workshop1!

You have 4 projects created for you.

- Customer Development
 - customerdb
- Customer Test
 - customerdb
- Customer Prod
 - customerdb
- CICD
 - anchore
 - anchoredb
 - jenkins
 - nexus
 - sonardb
 - sonar
 - sonarqube

Explore Jenkins

<http://jenkins-student-x-cicd.apps.ocpws.kee.vizuri.com/>

Username: student-x

Password: workshop1!

Explore Nexus

<http://nexus-student-x-cicd.apps.ocpws.kee.vizuri.com>

Explore Quay Registry

<https://registry.kee.vizuri.com/repository/>

Username: student-x

Password: workshop1!

Explore SonarQube

<http://sonarqube-student-x-cicd.apps.ocpws.kee.vizuri.com>

Username: admin

Password: admin

Explore Gogs (git repositories)

<http://gogs.apps.ocpws.kee.vizuri.com/>

Username: student-x

Password: workshop1!

Repositories:

- customer-service - SpringBoot REST Web Service utilized as demo project to orchestrate through our CI/CD process.

Lab 2 - Configure Jenkins Plugins

Install Jenkins Plugins

In Jenkins, navigate to Manage Jenkins.

Choose Manage Plugins.

Choose the Available tab.

Install the following Plugins:

- Anchore Container Image Scanner
- Sonar Quality Gates Plugin
- SonarQube Scanner for Jenkins
- xUnit plugin
- Gogs

Choose Install with Restart.

Configure Kubernetes Cloud

The Kubernetes Cloud plugin allows for the running of Kubernetes/OpenShift PODs as Jenkins JNLP Slaves

The OpenShift Jenkins Template configures a Kubernetes Cloud configuration for the current OpenShift environment. It provides two out-of-the-box Kubernetes Pod Templates to be Jenkins Slaves; maven and nodejs.

Configure Podman Kubernetes Pod Template

We will be using podman to build our container images so we need to include a new Kubernetes Pod Template that includes the podman binary.

The Dockerfile for the podman image can be found here:

<https://github.com/Vizuri/openshift-cicd-podman-jenkins-slave>

This container extends the OpenShift Maven image and just adds the podman binary.

In Jenkins, Navigate to the Manage Jenkins -> Configure System. Scroll down to the Cloud->Kubernetes section.

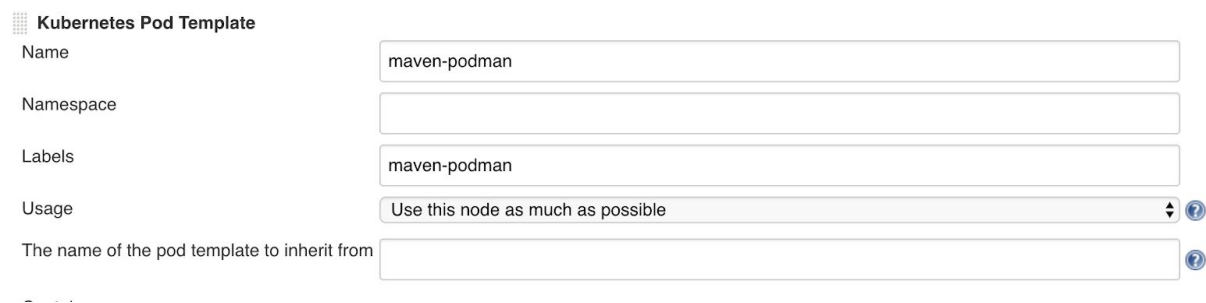
Notice the provided configuration.

Let's add our Podman Kubernetes Pod Template.

Click on the Add Pod Template button and Choose Kubernetes Pod Template.

Enter the following values:

- Name: maven-podman
- Labels maven-podman



The screenshot shows the 'Kubernetes Pod Template' configuration form in Jenkins. The form has the following fields and values:

Field	Value
Name	maven-podman
Namespace	
Labels	maven-podman
Usage	Use this node as much as possible
The name of the pod template to inherit from	

Click on the Add Container Button and Choose Container Template.

Enter the following values:

- Name: jnlp
- Docker Image: registry.kee.vizuri.com/vizuri/podman-slave:v1.0
- Working directory: /tmp
- Command to run: <EMPTY>
- Arguments to pass to the command: \${computer.jnlpMac} \${computer.name}

Containers

Container Template

Name

jnlp

Docker image

registry.kee.vizuri.com/vizuri/podman-slave:v1.0

Always pull image

☐

Working directory

/tmp

Command to run

Arguments to pass to the command

\${computer.jnlpmac} \${computer.name}

Allocate pseudo-TTY

☐

EnvVars

Add Environment Variable

List of environment variables to set in agent pod

Advanced...

Delete Container

Add Container

List of container in the agent pod

Add Environment Variable

Click the Advanced Button to see more options.

Check the Run in privileged mode button.

\${computer.jnlpmac} \${computer.name}

Allocate pseudo-TTY

☐

EnvVars

Add Environment Variable

List of environment variables to set in agent pod

Run in privileged mode

☒

Click the Add Volume Button and Choose Empty Dir Volume.

Enter the following values:

- Mount path: /var/lib/containers

Volumes

List of environment variables to set in all container of the pod

Empty Dir Volume

In Memory ☐

Mount path

Delete Volume

Add Volume

List of volumes to mount in agent pod

Save you changes.

Configure Anchore Plugin

In Jenkins, Navigate to the Manage Jenkins -> Configure System. Scroll down to the Anchore Plugin Mode section.

Enter the following values:

- Engine URL: <http://anchore-student-x-cicd.apps.ocpws.kee.vizuri.com>
- Engine Username: admin
- Engine Password: foobar

Anchore Plugin Mode

Engine Mode

Engine URL

Engine Username

Engine Password

Verify SSL

☐

☐ Local Anchore Scanner Mode - DEPRECATED

Enable DEBUG logging

☐

Click Save

Configure SonarQube Plugins

In Jenkins, Navigate to the Manage Jenkins -> Configure System.

Scroll down to the SonarQube servers section.

Click the Add SonarQube button.

Enter the following values:

- Name: sonar
- Server URL: <http://sonarqube-student-x-cicd.apps.ocpws.kee.vizuri.com>

SonarQube servers

Environment variables ☐ Enable injection of SonarQube server configuration as build environment variables
If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

SonarQube installations

Name

Server URL
Default is http://localhost:9000

Server authentication token
SonarQube authentication token. Mandatory when anonymous access is disabled.

[Advanced...](#)

[Delete SonarQube](#)

Scroll down to the Quality Gates - SonarQube section.

Click the Add Sonar Instance Button.

Enter the following values:

- Name: sonar
- SonarQube Server URL: <http://sonarqube-student-x-cicd.apps.ocpws.kee.vizuri.com>
- SonarQube account login: admin
- SonarQube account password: admin

Quality Gates - SonarQube

Name ⓘ
Make sure the name is unique value

SonarQube Server URL ⓘ
Default value is 'http://localhost:9000'

SonarQube account token ⓘ
Use token instead of user and password

SonarQube account login ⓘ
Default value is 'admin'

SonarQube account password ⓘ
Default value is 'admin'

Maximum waiting time (milliseconds)
Default value is '300000' or 5 minutes

Time to wait next check (milliseconds)
Default value is '10000' or 10 seconds

[Add Sonar instance](#)

[Delete](#)

Click the Save Button.

Configure SonarQube Jenkins WebHook.

Login into your SonarQube Server.

<http://sonarqube-student-5-cicd.apps.ocpws.kee.vizuri.com>

Click the login button and enter

Username: admin

Password: admin

And press the login button.

Click “skip this tutorial” on the pop-up.

Click on Administration and then Choose WebHooks.

Enter the following values:

- Name: Jenkins
- URL: <http://jenkins-student-x-cicd.apps.ocpws.kee.vizuri.com/sonarqube-webhook/>

Name	URL
Jenkins	http://jenkins-student-5-cicd.apps.ocpws.kee.vizuri.com/sonarqub

Click Save

Lab 3 - Build Java Pipeline

In this lab you will create a Jenkins Pipeline Job that checks out a SpringBoot microservice code and builds a Jar archive.

Create Jenkinsfile for Build

Log into the Gogs git Repository.

<http://gogs.apps.ocp-nonprod-01.kee.vizuri.com>

Username: student-x

Password: workshop1!

Click on the customer-service repository.

Create a new file in the root of the customer-service repository called Jenkinsfile with the following contents.

```
#!/usr/bin/groovy

def app_name = "customer";
def nexusUrl = "http://nexus-student-x-cicd.apps.ocpws.kee.vizuri.com";
def release_number;

node ("maven-podman") {

    stage('Checkout') {
        echo "In checkout"
        checkout scm

        if(BRANCH_NAME ==~ /(release.*/)) {
            def tokens = BRANCH_NAME.tokenize( '/' )
            branch_name = tokens[0]
            branch_release_number = tokens[1]
            release_number = branch_release_number
        }
        else {
            sh (
                script: "mvn -B help:evaluate
-Dexpression=project.version | grep -e '^[^\\[]' > release.txt",
                returnStdout: true,
                returnStatus: false
            )
            release_number = readFile('release.txt').trim()
            echo "release_number: ${release_number}"
        }
    }
}
```

```

    }

    stage('Build') {
        echo "In Build"
        sh "mvn -s configuration/settings.xml -Dnexus.url=${nexusUrl}
-DskipTests=true -Dbuild.number=${release_number} clean install"
    }

    stage ('Unit Test') {
        sh "mvn -s configuration/settings.xml -Dnexus.url=${nexusUrl}
-Dbuild.number=${release_number} test"
        junit "target/surefire-reports/*.xml"

        step([$class: 'XUnitBuilder',
            thresholds: [
                [$class: 'FailedThreshold', unstableThreshold: '1']
            ],
            tools: [
                [$class: "JUnitType", pattern: "target/surefire-reports/*.xml"]
            ]
        ])
    }
}

```

Configure Jenkins Job to Build Code

Log into Jenkins.

<http://jenkins-student-x-cicd.apps.ocpws.kee.vizuri.com/>

Username: student-x

Password: workshop1!

Click on New Item.

Enter the following values:

Enter an Item Name: customer-service

Choose: Multibranch pipeline

Enter an item name

customer-service

» Required field



Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.



Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.



Multi-configuration project

Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.



Bitbucket Team/Project

Scans a Bitbucket Cloud Team (or Bitbucket Server Project) for all repositories matching some defined markers.



Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.



GitHub Organization

Scans a GitHub organization (or user account) for all repositories matching some defined markers.



Multibranch Pipeline

Creates a set of Pipeline projects according to detected branches in one SCM repository.

OK

Click OK to Create the customer-service project.

Under Branch Sources, click Add source->Git.

Enter the following values:

Project repository:

<http://gogs.apps.ocp-nonprod-01.kee.vizuri.com/student-x/customer-service.git>

Git
X

Project Repository
?

Credentials

- none -
Add

?

Behaviors

Discover branches
X
?

Add

Property strategy

All branches get the same properties

Add property

Click Save.

This will trigger a build of you develop branch. Navigate to customer-service->develop to see the status of the build.

The Job should execute three stages; Checkout, Build and Unit Test.

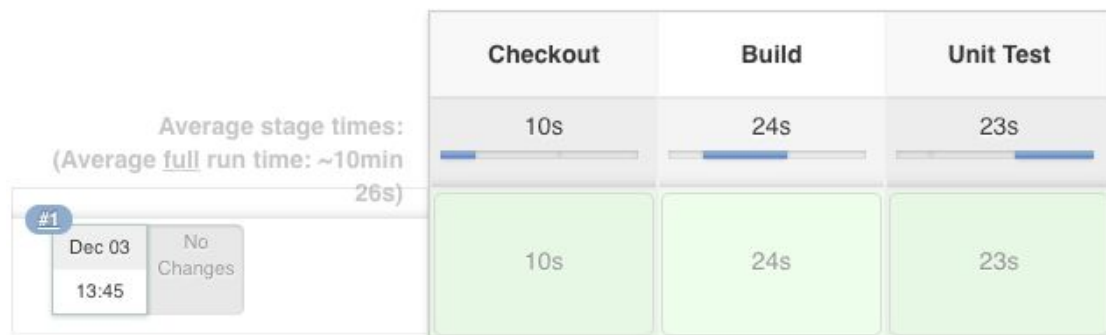
Pipeline develop

Full project name: customer-service/develop



[Recent Changes](#)

Stage View



Lab 4: Add SonarQube Code Analysis

SonarQube analyzes the source code for common issue and test coverage.

Edit the Jenkinsfile for the customer-service project and add the following two stages to the build pipeline.

```
stage('Unit Test') {
    sh "mvn -s configuration/settings.xml -Dnexus.url=${nexusUrl}
-Dbuild.number=${release_number} test"
    junit "target/surefire-reports/*.xml"

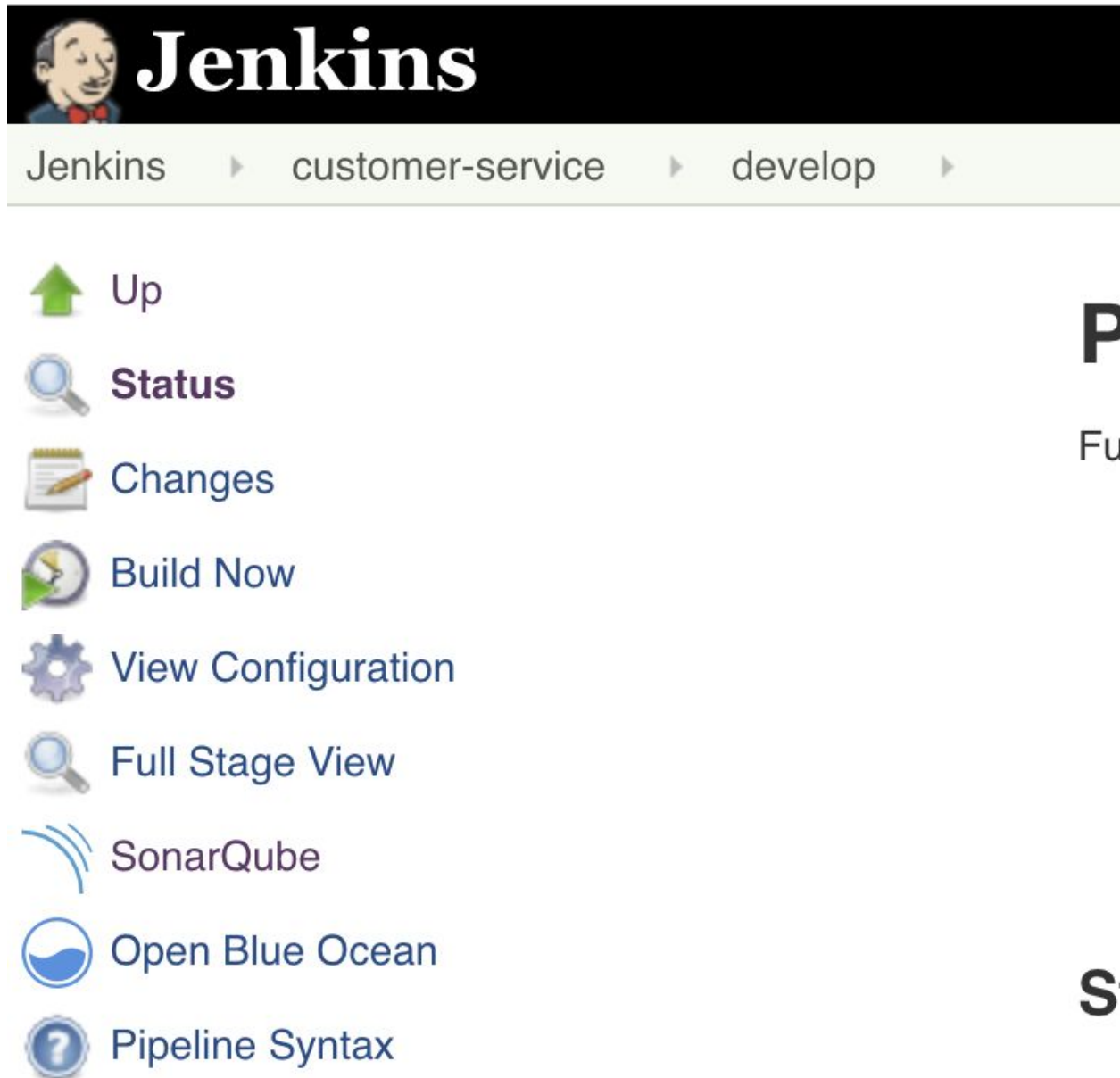
    step([$class: 'XUnitBuilder',
        thresholds: [
            [$class: 'FailedThreshold', unstableThreshold: '1']
        ],
        tools: [
            [$class: "JUnitType", pattern: "target/surefire-reports/*.xml"]
        ])
}

stage('SonarQube Analysis') {
    withSonarQubeEnv('sonar') { sh "mvn -s configuration/settings.xml
-Dnexus.url=${nexusUrl} -Dbuild.number=${release_number} sonar:sonar" }
}

stage("Quality Gate"){
    timeout(time: 1, unit: 'HOURS') {
        def qg = waitForQualityGate()
        if (qg.status != 'OK') {
            error "Pipeline aborted due to quality gate failure: ${qg.status}"
        }
    }
}
```

Rebuild Project.

Once complete, notice the SonarQube link on the left navigation. You can click this to see the results of the analysis.



Lab 5: Publish Build Artifact

The next step in the process is to publish the build artifact in our Nexus Repository.

Add the following stage to the Jenkins file.

```
stage('Deploy Build Artifact') {
    sh "mvn -s configuration/settings.xml -DskipTests=true
```

```
-Dbuild.number=${release_number} -Dnexus.url=${nexusUrl} deploy"
}
```

If you navigate to your nexus repository, you will see a snapshot version of the jar file in the maven-snapshots folder.

Lab 6: Container Build

Now we are going to build our container and publish it to our enterprise registry (Quay).

We will be using podman to build our container.

Add the following variables to the top of the Jenkinsfile.

```
def imageBase = "registry.kee.vizuri.com";
def imageNamespace = "student_x";
def registryUsername = "student-x"
def registryPassword = "workspace1!"
```

Add the following steps to the Jenkinsfile.

```
def tag = "${release_number}"

if (BRANCH_NAME ==~ /(develop|release.+)/) {
    stage('Container Build') {
        sh "podman build -t
${imageBase}/${imageNamespace}/${app_name}:${tag} ."
    }

    stage("Container Push") {
        if (BRANCH_NAME ==~ /(develop|release.+)/) {
            sh "podman login -u ${registryUsername} -p
${registryPassword} ${imageBase}"
            sh "podman push
${imageBase}/${imageNamespace}/${app_name}:${tag}"
        }
    }
}
```

```
}
```

If you log into the Quay registry, you will see your image.

<https://registry.kee.vizuri.com/repository>

Username: student-x

Password: workshop1!

Click on the customer repository then browse the tags. Notice the Security Scan tag. The image is queued for scanning. Once complete you will see the results of the scan.

Lab 7: Scan Container Image

In the next step, we will scan the newly created image for issues and known vulnerabilities. We will be using the Jenkins Anchore Image Scanner for this task.

Add the following lines to the Jenkinsfile.

```
    if (BRANCH_NAME ==~ /(develop|release.+)/) {  
        stage('Container Scan') {  
            writeFile file: 'anchore_images', text:  
"${imageBase}/${imageNamespace}/${app_name}:${tag} Dockerfile"  
            anchore engineRetries: '800', name: 'anchore_images'  
        }  
    }
```

Once complete, you will see the Anchore Report Link associated with the build.



Build #9 (Dec 3, 2018 11:03:45 PM)



Build Artifacts

 anchore_gates.json	6.33 KB view
 anchore_security.json	6.06 KB view



Changes

1. Update 'Jenkinsfile' ([detail](#))



Started by user [student-5](#)



Revision: 2b8ea58654286e43c86b9b8cbe08a0ea83014a54

- develop



[Test Result](#) (no failures)



[Anchore Report \(PASS\)](#)

Lab 8: Deploy Image to OpenShift

Add configuration to Deploy the image to OpenShift.

Configure Jenkins OpenShift Client Plugin.

Get Jenkins Service Account Token from OpenShift to be used by the OpenShift Client Plugin.

Log into the OpenShift Console.

<https://ocpws.kee.vizuri.com:8443>

Navigate to the CICD Project.

Choose Resources -> Secrets

Locate one of the two jenkins-token-XXXXXX secrets.

Click to view the secret

Click on Reveal the Secret to see the values.

Copy the value of the Token to be used below.

In Jenkins, Click on Manage Jenkins -> Configure System.

Scroll down to the OpenShift Client Plugin Section. Press the Add OpenShift Cluster button and choose OpenShift Cluster.

Enter the following values:


- Cluster name: ocp-ws
- API Server URL: <https://ocpws.kee.vizuri.com:8443>
- Disable TLS Verify: Check
- Credentials: Click Add and Select Jenkins to Create new Credentials.
 - Kind: OpenShift Token for OpenShift Client Plugin
 - ID: ocp-ws
 - Token: Past token retrieved above.




The screenshot shows the 'Jenkins Credentials Provider: Jenkins' interface. The 'Add Credentials' section is active. The 'Domain' is set to 'Global credentials (unrestricted)'. The 'Kind' is set to 'OpenShift Token for OpenShift Client Plugin'. The 'Scope' is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Token' field is empty. The 'ID' field is set to 'ocp-ws'. The 'Description' field is empty. There are 'Add' and 'Cancel' buttons at the bottom.





Click Add to create the new credential.

Select the new credential in the credentials drop down.

 **OpenShift Cluster**

Cluster Name 

API Server URL

Credentials ocp-ws   Add  

Disable TLS Verify ☒

Server Certificate Authority

Default Project

Delete

Click Save to Update the Jenkins Plugin.

Update Jenkinsfile

Add the following variables to the top of your Jenkinsfile.

```
def ocp_cluster = "ocp-ws"
def ocpDevProject = "student-x-customer-dev"
def ocpTestProject = "student-x-customer-test"
def ocpProdProject = "student-x-customer-prod"
```

The add the following to the bottom of the Jenkinsfile.

```
if (BRANCH_NAME ==~ /(develop)/) {
    def ocp_project = ocpDevProject;
    stage("Deploy Openshift ${ocp_project}") {
        openshift.withCluster( "${ocp_cluster}" ) {
            openshift.withProject( "${ocp_project}" ) {
                def dc = openshift.selector("dc", "${app_name}")
                if(!dc.exists()) {
                    dc = openshift.newApp("-f
https://raw.githubusercontent.com/Vizuri/openshift-cicd-pipeline/master/templates/springboot-
dc.yaml -p IMAGE_NAME=${imageBase}/${imageNamespace}/${app_name}:${tag} -p
APP_NAME=${app_name} -p DATABASE_HOST=customerdb -p DATABASE_DB=customer
-p DATABASE_USER=customer -p DATABASE_PASSWORD=customer").narrow("dc")
                }
            }
        }
    }
}
```

```

def dcObject = dc.object()
dcObject.spec.template.spec.containers[0].image
= "${imageBase}/${imageNameSpace}/${app_name}:${tag}"
openshift.apply(dcObject)
}

def rm = dc.rollout()
rm.latest()
timeout(5) {
    def latestDeploymentVersion =
openshift.selector('dc', "${app_name}").object().status.latestVersion
    def rc = openshift.selector('rc',
"${app_name}-${latestDeploymentVersion}")
    rc.untilEach(1){
        def rcMap = it.object()
        return
(rcMap.status.replicas.equals(rcMap.status.readyReplicas))
    }
}
}
}
}

```

Return to the customer-service develop job and trigger a build.

Once finished, you can log into OpenShift and navigate to the Customer Development project. You should now have a customer POD running.

Other Resources

DEPLOYMENT CONFIG

CONTAINERS

customer

- Image: student_5/customer
- Ports: 8080/TCP (web) and 1 other

1
pod

Routes - External Traffic
<http://customer-student-5-customer-dev.apps.ocpws.keee.vizuri.com>
Route customer, target port web

NETWORKING

Service - Internal Traffic
customer
8080/TCP (web) → 8080

Lab 9: Full Pipeline From Shared Library

In this lab, you will build a complete CI/CD pipeline. It builds Feature, Develop and Release Branch Jobs and Orchestrates a release from Dev->Test->Prod.

Review Pipeline Library Functions.

Browse openshift-cicd-pipeline github repository.

<https://github.com/Vizuri/openshift-cicd-pipeline>

Navigate to src -> com -> vizuri -> openshift

Review the functions in PipelineSteps.groovy

Next review the Pipeline defined in JavaDeliveryPipeline.groovy.

Update Jenkinsfile

Now back your Jenkinsfile to Jenkinsfile.BAC.

Update the Jenkins File with the following contents.

```
#!/usr/bin/groovy
@Library('github.com/vizuri/openshift-cicd-pipeline@master')

def javaDeliveryPipeline = new com.vizuri.openshift.JavaDeliveryPipeline();

javaDeliveryPipeline {
    ocpAppSuffix = 'apps.ocpws.kee.vizuri.com'
    imageNamespace = 'student_x';
    registryUsername = 'student-x'
    imageBase = 'registry.kee.vizuri.com'
    registryUsername = 'student-x'
    registryPassword = 'workshop1!'
    app_name = 'customer'
    ocp_dev_cluster = 'ocp-ws'
```

```
    ocp_dev_project = 'student-x-customer-dev'  
    ocp_test_cluster = 'ocp-ws'  
    ocp_test_project = 'student-x-customer-test'  
    ocp_prod_cluster = 'ocp-ws'  
    ocp_prod_project = 'student-x-customer-prod'  
}
```

Configure Gogs Jenkins WebHook

Log into Gogs

<http://gogs.apps.ocp-nonprod-01.kee.vizuri.com>

Username: student-x

Password: workshop1!

Select the customer-service project.

Click on the Settings link in the top right.

Choose Webhooks

Click Add Webhook and Select Gogs.

Enter the following values.

Payload URL:

<http://jenkins-student-x-cicd.apps.ocp-ws.kee.vizuri.com/gogs-webhook/?job=customer-service>

Save

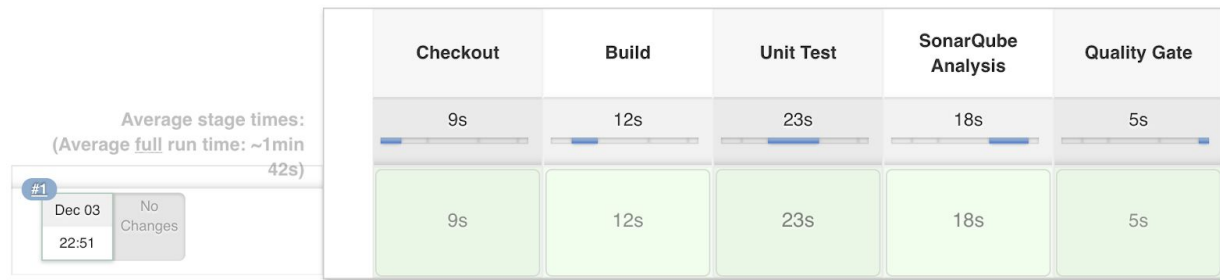
Test Feature Branch

Create a new branch called feature/Feature-1 in the Gogs Repository.

Watch build trigger a new Feature build.

The following steps will be executed.

Stage View



- . .

Test Develop Branch

Create a Pull Request and Merge the Feature Branch into the Develop Branch.

This will trigger the Develop branch build and deploy to development.

Release Develop Branch

Create a release branch called release/1.0

This will trigger a release pipeline.