04-Lab2 管道整合实验

情景

在实验中，你扮演了一个开发人员，他想部署一个新的CI/CD管道来管理和监控应用程序的开发周期。

- 该应用程序通过不同的阶段进展，并在生产阶段达到高潮。
- 这个演示需要在开始之前做一些准备。

1. 为演示做准备

在这一节中，你要为演示设置先决条件。注意以下几点。

- 说明很简短，并不意味着要在演示观众面前进行。
- 在本节的末尾，提供了一个简单的脚本来创建演示。

1.1. 从你的学生虚拟机登录OpenShift

1. 使用学生虚拟机配置电子邮件中的 SSH 命令和密码登录到学生虚拟机中。
2. 登录后，设置你的**OCP_USERNAME**变量并登录Red Hat® OpenShift®容器平台。

```
$ export OCP_USERNAME=youropentlc-account.com
$ oc login -u $OCP_USERNAME https://api.shared-
na4.na4.openshift.opentlc.com:6443
Username: username-domain.com
Password: ********** [ This is your OPENTLC Account password ]
```

结果输出

```
Login successful.
```

1.2. 创建演示项目

1. 为你的演示创建几个项目。你将在Dev中构建你的应用程序，并通过Test和Prod来推广它。

   提示：你必须选择一个唯一的标识符（GUID），以便你的项目名称和路由不会与OpenShift集群中其他用户的项目名称和路由冲突。使用学生虚拟机的供应电子邮件中的GUID。

```
$ export GUID=r5d4
$ oc new-project pipeline-${GUID}-dev  --description="Cat of the Day
Development Environment" --display-name="Cat Of The Day - Dev"
$ oc new-project pipeline-${GUID}-test --description="Cat of the Day Testing
Environment"     --display-name="Cat Of The Day - Test"
$ oc new-project pipeline-${GUID}-prod --description="Cat of the Day
Production Environment"  --display-name="Cat Of The Day - Prod"
```

2. 显示您创建的项目：

```
$ oc get projects
```

结果输出

```
NAME                 DISPLAY NAME            STATUS
pipeline-r5d4-dev    Cat Of The Day - Dev    Active
pipeline-r5d4-prod   Cat Of The Day - Prod   Active
pipeline-r5d4-test   Cat Of The Day - Test   Active
```

3. 切换到 `pipeline-${GUID}-dev` 项目，这是大部分工作要做的地方:

```
$ oc project pipeline-${GUID}-dev
```

1.3. 部署 CI/CD 环境

1. 部署Jenkins来控制你的构建和部署管道:

```
$ oc new-app jenkins-persistent -p ENABLE_OAUTH=false -e
JENKINS_PASSWORD=openshiftpipelines -n pipeline-${GUID}-dev
```

结果输出

```
--> Deploying template "openshift/jenkins-persistent" to project pipeline-
r5d4-dev

     Jenkins
     ---------
     Jenkins service, with persistent storage.

     NOTE: You must have persistent volumes available in your cluster to use
this template.

     A Jenkins service has been created in your project.  Log into Jenkins
with your OpenShift account.  The tutorial at
https://github.com/openshift/origin/blob/master/examples/jenkins/README.md
contains more information about using this template.

     * With parameters:
        * Jenkins Service Name=jenkins
        * Jenkins JNLP Service Name=jenkins-jnlp
        * Enable OAuth in Jenkins=false
        * Memory Limit=1Gi
        * Volume Capacity=1Gi
        * Jenkins ImageStream Namespace=openshift
        * Disable memory intensive administrative monitors=false
        * Jenkins ImageStreamTag=jenkins:2
        * Fatal Error Log File=false
        * Allows use of Jenkins Update Center repository with invalid SSL
certificate=false

--> Creating resources ...
    route.route.openshift.io "jenkins" created
    persistentvolumeclaim "jenkins" created
    deploymentconfig.apps.openshift.io "jenkins" created
    serviceaccount "jenkins" created
    rolebinding.authorization.openshift.io "jenkins_edit" created
    service "jenkins-jnlp" created
    service "jenkins" created
--> Success
```

```
    Access your application via route 'jenkins-pipeline-r5d4-
dev.apps.shared-na4.na4.openshift.opentlc.com'
    Run 'oc status' to view your app.
```

2. 记下Jenkins的`admin`密码。
   - 该密码由`JENKINS_PASSWORD`参数定义。
3. 上面的命令在Dev项目中创建了一个Jenkins部署和服务账户。由于你希望能够使用Jenkins在Test和Prod项目中运行镜像，你需要给Jenkins pservice账户授权来管理这些项目的资源。为此，使用`oc policy`使Jenkins服务账户能够管理`pipeline-${GUID}-test`和`pipeline-${GUID}-prod`项目中的资源。

```
$ oc policy add-role-to-user edit system:serviceaccount:pipeline-${GUID}-
dev:jenkins -n pipeline-${GUID}-test
$ oc policy add-role-to-user edit system:serviceaccount:pipeline-${GUID}-
dev:jenkins -n pipeline-${GUID}-prod
```

4. 当一个应用程序被部署在一个项目中时，项目中的服务账户会进行工作。由于样本应用程序的镜像将由开发项目拥有，它默认情况下不能被测试和开发项目访问。Test和Prod项目中的服务账户需要能够提取Dev项目中的镜像。使用`oc policy`启用从`pipeline-${GUID}-dev`项目到`pipeline-${GUID}-test`和`pipeline-${GUID}-prod`项目的镜像拉取。

```
$ oc policy add-role-to-group system:image-puller
system:serviceaccounts:pipeline-${GUID}-test -n pipeline-${GUID}-dev
$ oc policy add-role-to-group system:image-puller
system:serviceaccounts:pipeline-${GUID}-prod -n pipeline-${GUID}-dev
```

1.4. 部署示例应用程序

1. 在`pipeline-${GUID}-dev`项目中部署 "今日之猫"（`cotd`）应用程序。

```
$ oc new-app php~https://github.com/StefanoPicozzi/cotd2 --as-deployment-
config -n pipeline-${GUID}-dev
```

   - 注意，在 "php "和 "http "之间有一个波浪号（~）而不是连字符。这意味着**php**镜像将被用于运行GitHub repo **cotd2**中的代码。
2. 观察构建日志直到构建完成：

```
$ oc logs -f build/cotd2-1 -n pipeline-${GUID}-dev
```

   结果输出

```
[...] much text omitted [...]
Successfully pushed image-registry.openshift-image-
registry.svc:5000/pipeline-r5d4-
dev/cotd2@sha256:809c3fcb41ea49ae27fe9224aeab73d599cb22bdc9b62ec0cc70c0e50d5
a25b8
Push successful
```

3. 验证构建完成并标记镜像。首先，你给**cotd2**的`latest`构建贴上`testready`标签。然后用`prodready`标记同一镜像。

```
$ oc tag cotd2:latest cotd2:testready -n pipeline-${GUID}-dev
$ oc tag cotd2:testready cotd2:prodready -n pipeline-${GUID}-dev
```

4. 检查镜像流以验证是否创建了标记。镜像流是OpenShift存储的特殊镜像元数据:

```
$ oc describe imagestream cotd2 -n pipeline-${GUID}-dev
```

结果输出

```
Name:            cotd2
Namespace:       pipeline-r5d4-dev
Created:         12 minutes ago
Labels:          app=cotd2
                 app.kubernetes.io/component=cotd2
                 app.kubernetes.io/instance=cotd2
Annotations:     openshift.io/generated-by=OpenShiftNewApp
Image Repository:default-route-openshift-image-registry.apps.shared-
na4.na4.openshift.opentlc.com/pipeline-r5d4-dev/cotd2
Image Lookup:    local=false
Unique Images:i  1
Tags:            3

latest
  no spec tag

  * image-registry.openshift-image-registry.svc:5000/pipeline-r5d4-
dev/cotd2@sha256:809c3fcb41ea49ae27fe9224aeab73d599cb22bdc9b62ec0cc70c0e50d5
a25b8
      11 minutes ago

prodready
  tagged from
cotd2@sha256:809c3fcb41ea49ae27fe9224aeab73d599cb22bdc9b62ec0cc70c0e50d5a25b
8

  * image-registry.openshift-image-registry.svc:5000/pipeline-r5d4-
dev/cotd2@sha256:809c3fcb41ea49ae27fe9224aeab73d599cb22bdc9b62ec0cc70c0e50d5
a25b8
      About a minute ago

testready
  tagged from
cotd2@sha256:809c3fcb41ea49ae27fe9224aeab73d599cb22bdc9b62ec0cc70c0e50d5a25b
8

  * image-registry.openshift-image-registry.svc:5000/pipeline-r5d4-
dev/cotd2@sha256:809c3fcb41ea49ae27fe9224aeab73d599cb22bdc9b62ec0cc70c0e50d5
a25b8
      About a minute ago
```

5. 想象一下，你已经在开发项目中成功地进行了冒烟测试，现在你将把它推广到测试项目中。想象一下，镜像在测试项目中被测试，并被推广到Prod。在 `pipeline-${GUID}-test` 和 `pipeline-${GUID}-prod` 项目中部署 `cotd2` 应用程序。

```
$ oc new-app pipeline-${GUID}-dev/cotd2:testready --name=cotd2 --as-
deployment-config -n pipeline-${GUID}-test
$ oc new-app pipeline-${GUID}-dev/cotd2:prodready --name=cotd2 --as-
deployment-config -n pipeline-${GUID}-prod
```

6. 为这三个应用程序创建路由：

```
$ oc expose service cotd2 -n pipeline-${GUID}-dev
$ oc expose service cotd2 -n pipeline-${GUID}-test
$ oc expose service cotd2 -n pipeline-${GUID}-prod
```

7. 在你的演示中为所有的部署配置禁用 `automatic` 部署。

```
$ oc get dc cotd2 -o yaml -n pipeline-${GUID}-dev  | sed 's/automatic:
true/automatic: false/g' | oc replace -f -
$ oc get dc cotd2 -o yaml -n pipeline-${GUID}-test | sed 's/automatic:
true/automatic: false/g' | oc replace -f -
$ oc get dc cotd2 -o yaml -n pipeline-${GUID}-prod | sed 's/automatic:
true/automatic: false/g' | oc replace -f -
```

提示：当你设置 `automatic: false` 时，该参数被删除。用 `oc get dc cotd2 -o yaml` 再次检查。

现在，您知道可以在所有必要的名称空间中成功部署应用程序。现在，您可以添加一个构建管道来自动运行此过程。

1.5. 创建初始BuildConfig管道

构建配置为新的容器镜像定义了一个构建过程。有三种可能的构建方式：使用Dockerfile的容器镜像构建；使用特别准备的基础镜像，接受可以使其运行的源代码的Source-to-Image构建；以及可以运行任意容器镜像作为基础并接受构建参数的自定义构建。构建在集群上运行，完成后被推送到 "输出 "部分中指定的容器镜像注册表。构建可以通过webhook触发，当基础镜像改变时，或者当用户手动请求创建一个新的构建时，都可以触发。

在本节中，你将使用下面列出的两种方法之一创建初始BuildConfig管道。

BuildConfig管道样本

```
kind: "BuildConfig"
apiVersion: build.openshift.io/v1
metadata:
  name: "pipeline-demo"
spec:
  triggers:
  - github:
      secret: 5Mlic4Le
    type: GitHub
  - generic:
      secret: FiArdDBH
    type: Generic
  strategy:
    type: "JenkinsPipeline"
    jenkinsPipelineStrategy:
      jenkinsfile: |
                    pipeline {
```

```
                    agent any
                    stages{
                      stage("Build") {
                        steps{
                          script{
                            openshift.withCluster() {
                              openshift.withProject() {
                                echo '*** Build Starting ***'
                                openshift.selector('bc', 'cotd2').startBuild("-
--wait").logs('-f')
                                echo '*** Build Complete ***'
                              }
                            }
                          }
                        }
                      }
                      stage("Deploy and Verify in Development Env"){
                        steps{
                          script{
                            openshift.withCluster() {
                              openshift.withProject() {
                                echo '*** Deployment Starting ***'
                                openshift.selector('dc',
'cotd2').rollout().latest()
                                echo '*** Deployment Complete ***'
                              }
                            }
                          }
                        }
                      }
                    }
```

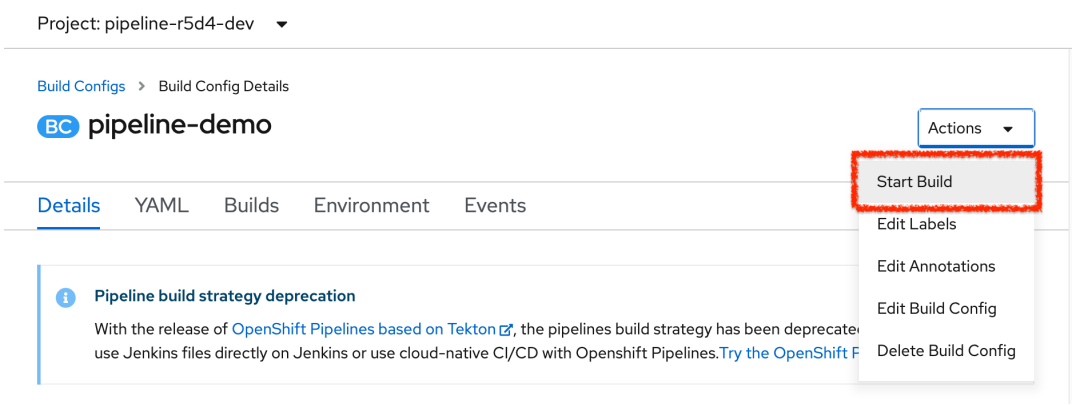
1. 在继续之前，验证Jenkins已经部署，并确保你可以用`admin`/`openshiftpipelines`的凭证连接到Jenkins的网络界面。
2. 找到路由：

```
$ oc get routes -n pipeline-${GUID}-dev
```

3. 使用以下任一方法来创建初始BuildConfig管道。
   - **方法1**。在任何带有OpenShift客户端的主机上使用命令行。
     1. 创建一个包含上述BuildConfig管道样本内容的文件。
     2. 使用`oc create -f FILENAME.yaml`命令创建BuildConfig管道。
   - **方法2**。使用 OpenShift Container Platform Web 控制台：
     1. 登录到OPENTLC共享的OpenShift容器平台Web控制台。
     2. 选择你的`pipeline-${GUID}-dev`项目。
     3. 在左边的菜单中，从下拉列表中选择**开发者**视图。
     4. 选择**添加→导入YAML**，在文本框中粘贴BuildConfig管道文本，然后点击**创建**。

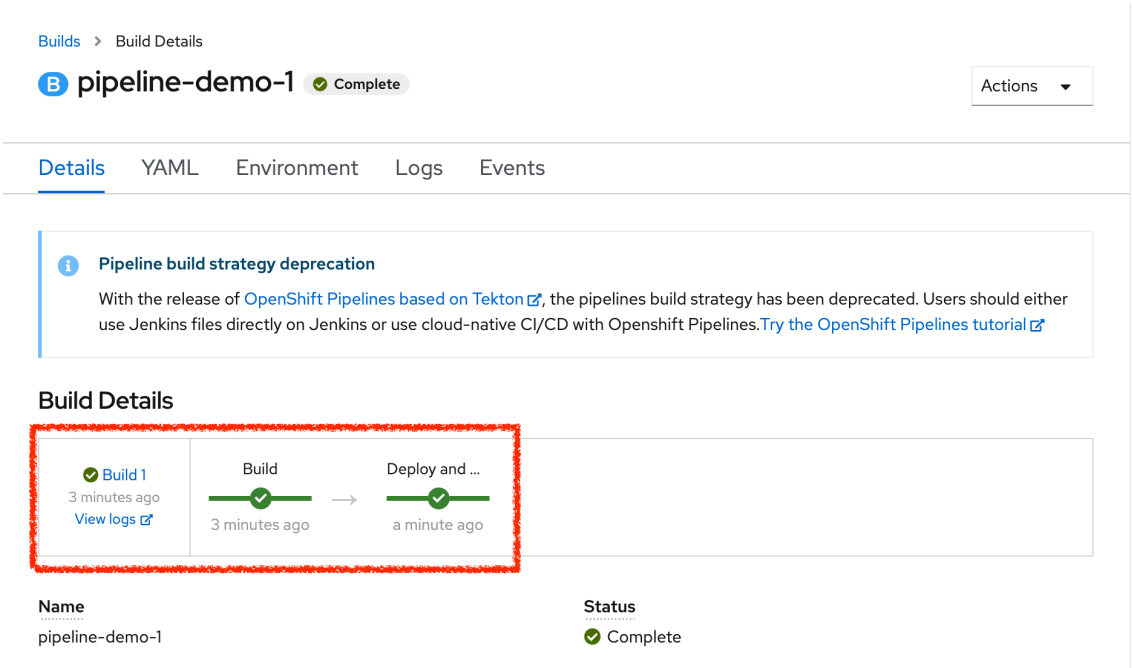
1.6. Test New Pipeline

1. 使用OpenShift容器平台web控制台，确保管道已经创建：
    1. 选择**Builds**，然后选择 `BC pipeline-demo`。
    2. 点击 **Actions** → **Start Build**。



2. 查看管道YAML并注意以下内容。
    - 这个基本管道只有两个阶段。

- 每个阶段都有多个动作，你可以定义。
- 在**Build**阶段，流水线会触发构建和验证。
- 在**部署...**阶段，流水线会触发一个新的部署。
- `openshift.io/build.start-policy` 被设置为**Serial**，以便一次只运行一个构建。



2. 演示OpenShift管道集成

1. **行动**：通过点击左侧的**Build**来编辑BuildConfig。
2. **行动**：点击**BC pipel-demo**，点击**YAML**标签，编辑BuildConfig的strategy片段。YAML界面打开。
3. **行动**：通过复制和粘贴以下CI/CD BuildConfig管道的内容，增加一些步骤，以取代你现有的管道。

   注意：确保将 `GUID` 的值改为你的唯一标识符。
4. 修改 `BuildConfig` 中的 `jenkinsfile` 部分，用下面的管道定义替换当前管道。

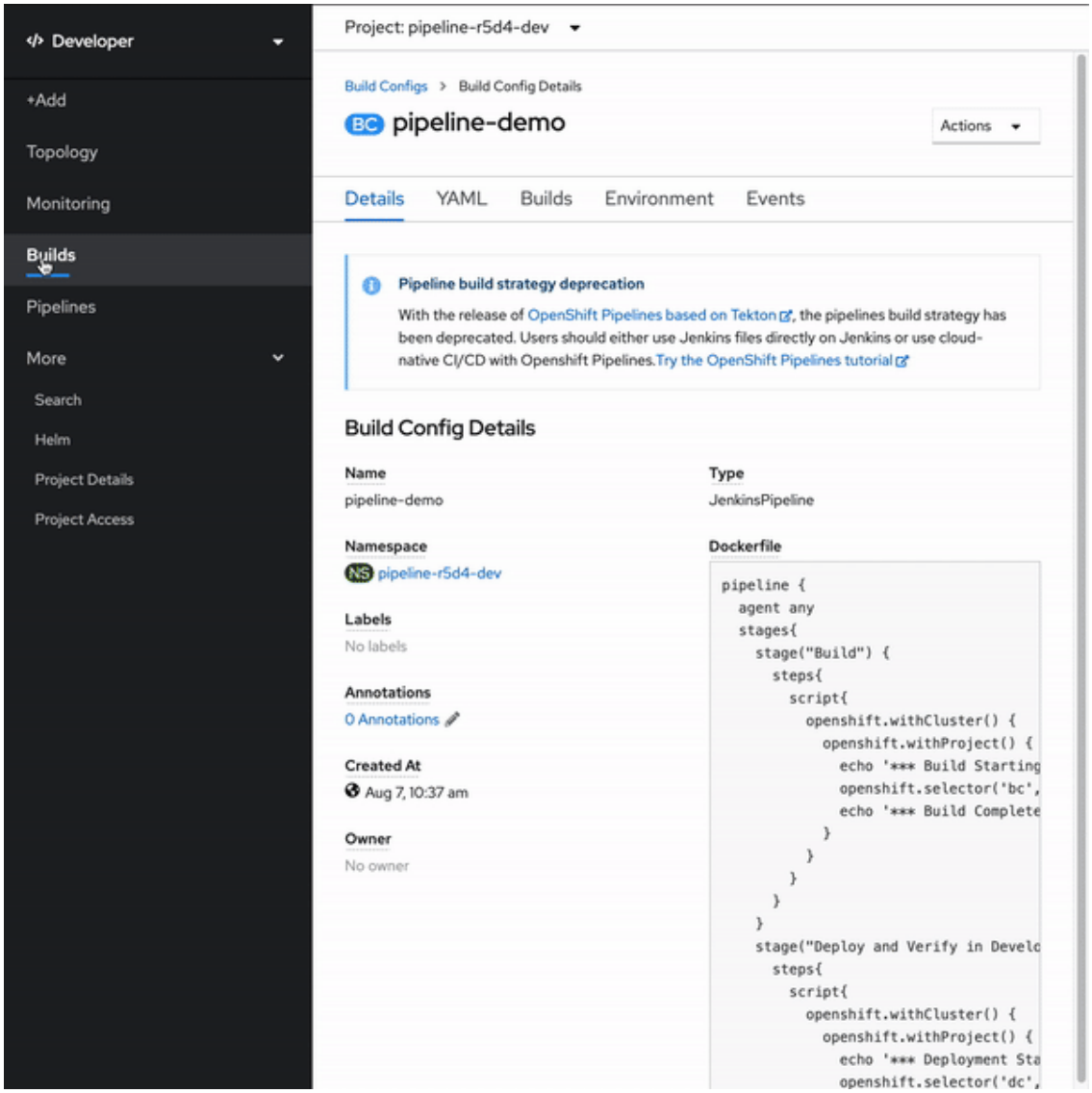
CI/CD BuildConfig管道样本

```
  strategy:
    type: JenkinsPipeline
    jenkinsPipelineStrategy:
      jenkinsfile: |
        // Begin replacing Jenkinsfile here
        // Set project names
        def GUID       = "GUID"
        def devProj    = "pipeline-$GUID-dev"
        def testProj   = "pipeline-$GUID-test"
        def prodProj   = "pipeline-$GUID-prod"
        def svc_name   = "cotd2"
        pipeline {
          agent any
          stages{
            stage("Build"){
              steps{
                echo '*** Build Starting ***'
                script{
                  openshift.withCluster() {
                    openshift.withProject("${devProj}") {
                      openshift.selector('bc', 'cotd2').startBuild("--
wait").logs('-f')
                    }
                  }
                echo '*** Build Complete ***'
                }
              }
            }
            stage ("Deploy and Verify in Development Env") {
              steps{
                echo '*** Deployment Starting ***'
                script{
                  openshift.withCluster() {
                    openshift.withProject("${devProj}") {
                      // Deploy the cotd application in the devProject
                      openshift.selector('dc', 'cotd2').rollout().latest();
                      // Wait for application to be deployed
                      def dc = openshift.selector("dc", "cotd2").object()
                      while (dc.spec.replicas != dc.status.availableReplicas) {
                        sleep 1
                      }
                    }
                  }
                }
                echo '*** Deployment Complete ***'
                echo '*** Service Verification Starting ***'
                script{
                  openshift.withCluster() {
                    openshift.withProject("${devProj}") {
                      // def svc = openshift.selector("svc", "cotd2")
                      // openshiftVerifyService apiURL:
'https://openshift.default.svc.cluster.local', authToken: '', namespace:
'pipeline-${GUID}-dev', svcName: 'cotd2', verbose: 'false'
                      def connected = openshift.verifyService("${svc_name}")
                      if (connected) {
```

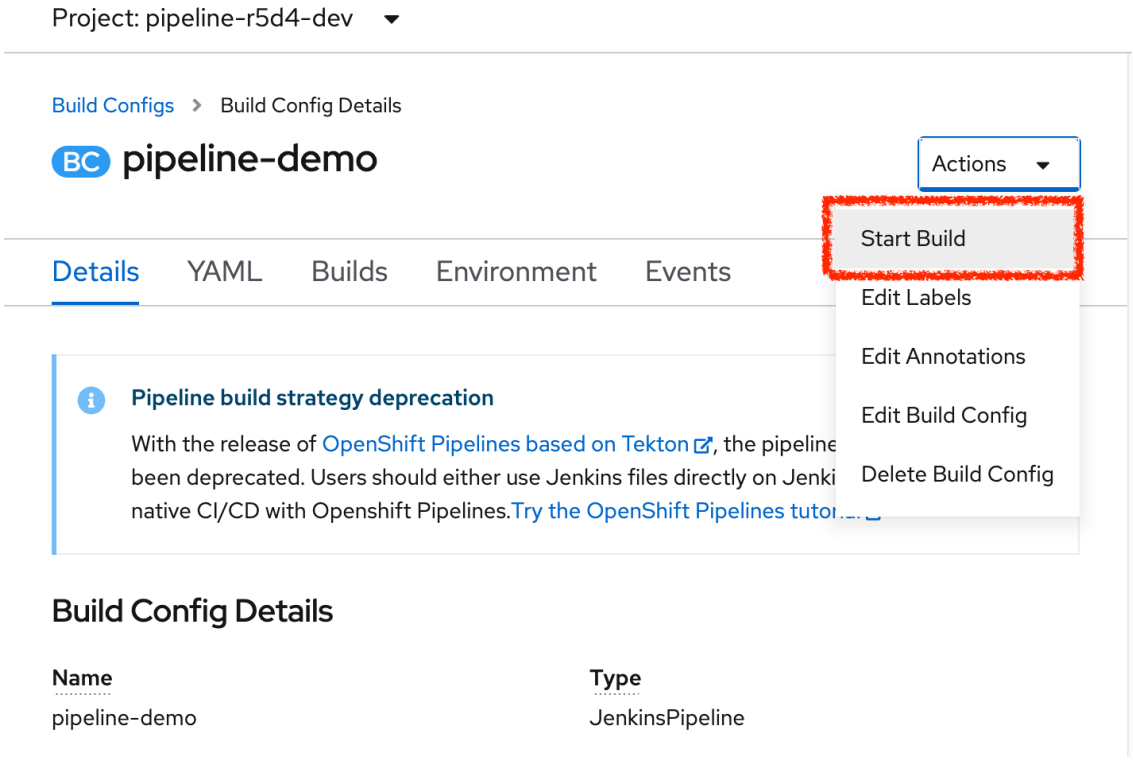
```
                        echo "Able to connect to ${svc_name}"
                      } else {
                        echo "Unable to connect to ${svc_name}"
                      }

                      // openshiftTag(srcStream: 'cotd2', srcTag: 'latest',
destStream: 'cotd2', destTag: 'testready')
                      openshift.tag("${devProj}/cotd2:latest",
"${devProj}/cotd2:testready")
                    }
                  }
                }
                echo '*** Service Verification Complete ***'
              }
            }
            stage ('Deploy and Test in Testing Env') {
              steps{
                echo "*** Deploy testready build in pipeline-${GUID}-test project
***"
                script {
                  openshift.withCluster() {
                    openshift.withProject("${testProj}") {
                      // openshiftDeploy apiURL:
'https://openshift.default.svc.cluster.local', authToken: '', depCfg: 'cotd2',
namespace: 'pipeline-${GUID}-test', verbose: 'false', waitTime: ''
                      // openshiftVerifyDeployment apiURL:
'https://openshift.default.svc.cluster.local', authToken: '', depCfg: 'cotd2',
namespace: 'pipeline-${GUID}-test', replicaCount: '1', verbose: 'false',
verifyReplicaCount: 'false', waitTime: '10'
                      // Deploy the cotd application in the testProject
                      openshift.selector('dc', 'cotd2').rollout().latest();
                      // Wait for application to be deployed
                      def dc = openshift.selector("dc", "cotd2").object()
                      while (dc.spec.replicas != dc.status.availableReplicas) {
                        sleep 1
                      }
                      // curl the testProject route to get cats
                      // sh 'curl http://cotd2-pipeline-${GUID}-test.apps.shared-
na4.na4.openshift.opentlc.com/ | grep cats -q'
                      def route = openshift.selector("route", "cotd2").object()
                      def the_route = "${route.spec.host}"
                      echo "route: ${the_route}"
                      sh "curl -s http://${the_route}/item.php | grep -q cats"
                    }
                  }
                }
              }
            }
            stage ('Promote and Verify in Production Env') {
              steps{
                echo '*** Waiting for Input ***'
                script{
                  openshift.withCluster() {
                    openshift.withProject("${prodProj}") {
                      input message: 'Should we deploy to Production?', ok:
"Promote"
                      // openshiftTag(srcStream: 'cotd2', srcTag: 'testready',
destStream: 'cotd2', destTag: 'prodready')
```

```
                    openshift.tag("${devProj}/cotd2:testready",
"${devProj}/cotd2:prodready")
                    // openshiftDeploy apiURL:
'https://openshift.default.svc.cluster.local', authToken: '', depCfg: 'cotd2',
namespace: 'pipeline-${GUID}-prod', verbose: 'false', waitTime: ''
                    echo '*** Deploying to Production ***'
                    def dc = openshift.selector("dc", "cotd2").object()
                    while (dc.spec.replicas != dc.status.availableReplicas) {
                      sleep 1
                    }
                    // openshiftVerifyDeployment apiURL:
'https://openshift.default.svc.cluster.local', authToken: '', depCfg: 'cotd2',
namespace: 'pipeline-${GUID}-prod', replicaCount: '1', verbose: 'false',
verifyReplicaCount: 'false', waitTime: '10'
                    sleep 10
                    // test route
                    def route = openshift.selector("route", "cotd2").object()
                    def the_route = "${route.spec.host}"
                    echo "route: ${the_route}"
                    sh "curl -s http://${the_route}/item.php | grep -q cats"
                  }
                }
              }
            }
          }
        }
```

5. **行动**：通过点击页面底部的**保存**来保存更改。



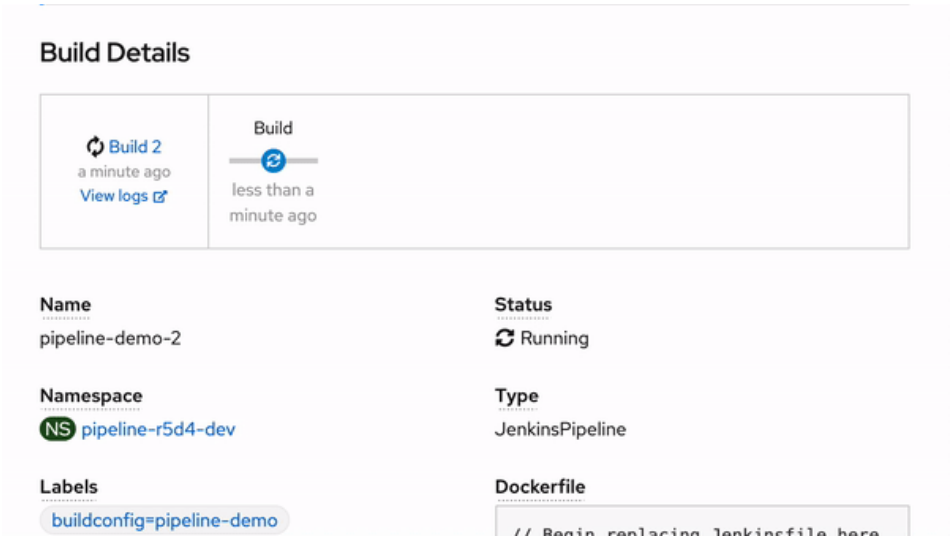
6. **行动**：在Jenkins控制台输出页面的底部，当最终出现 `Should we deploy to Production?` 的小确认对话框时，点击**Promote**来完成管道。



   - **解释**：该管道的以下方面。

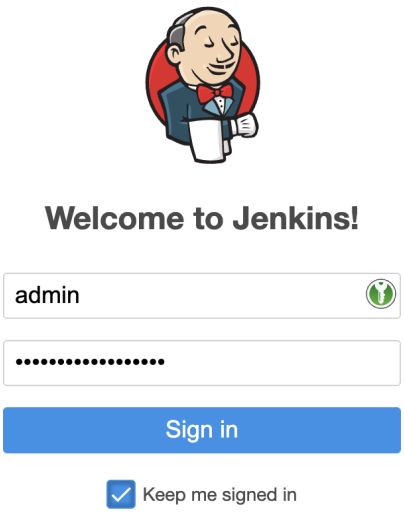
- 这个CI/CD流水线有四个阶段。
- 每个阶段都有多个动作，你可以定义。
- 串行运行策略被设置为每次只运行一个构建。

- **解释**：管道中的不同阶段。
  - **构建**：OpenShift构建并验证一个成功的构建。
    - **在开发环境中部署和验证**：OpenShift在开发项目中部署最新的应用，验证容器和服务的部署是否正确，并将镜像标记为`testready`。
    - **在测试环境中进行部署和测试**：OpenShift在测试项目中部署标记为`testready`的镜像，运行不同的集成测试--在这种情况下，只是一个cURL命令--如果所有测试都通过，则将镜像标记为`prodready`。
    - **在生产环境中进行推广和验证**。OpenShift会等待人工批准，然后在生产项目中部署并验证容器和服务。



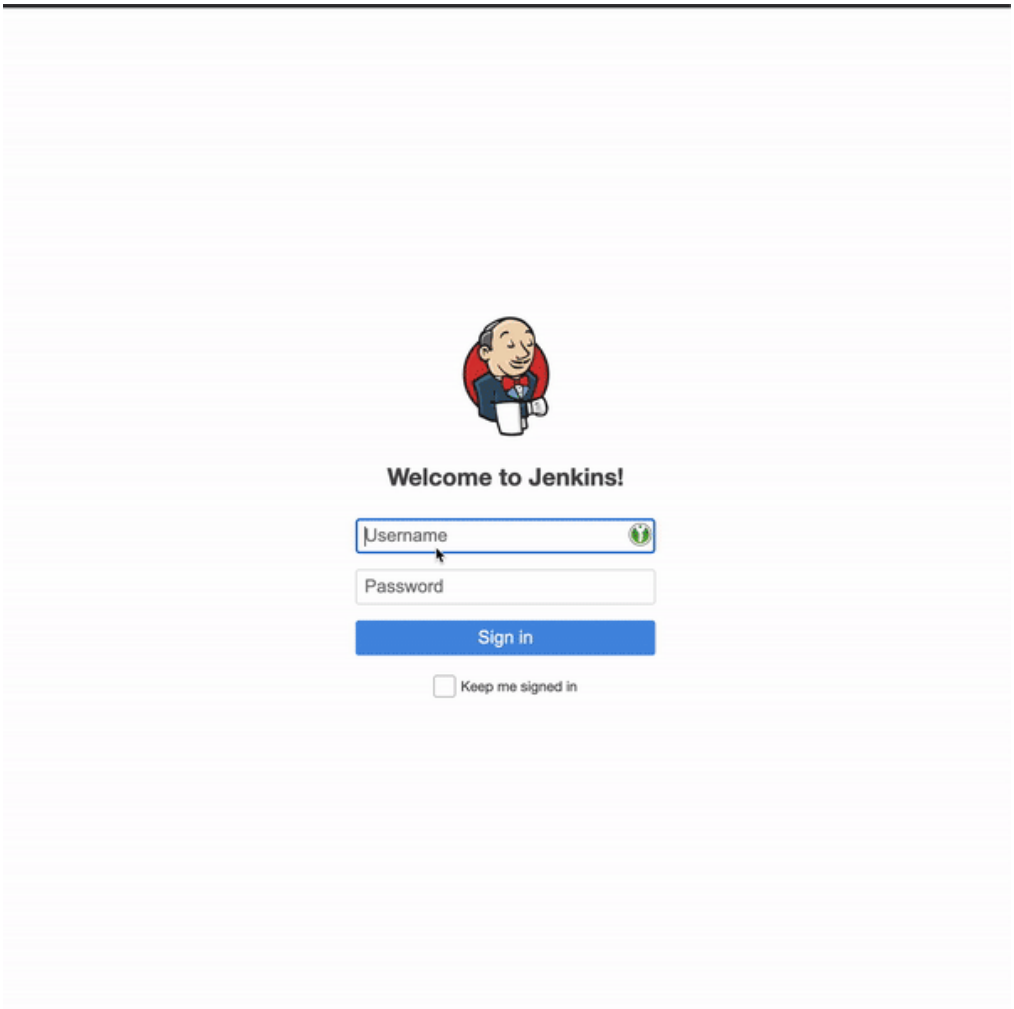
7. **行动**：当你等待管道完成其运行时，点击**查看日志**。

   注意：你必须使用你在创建Jenkins服务器时设置的密码（默认：`openshiftpipelines`），以`admin`身份登录Jenkins。



   - **解释**：OpenShift会引导你到Jenkins控制台日志，在那里你可以从Jenkins界面管理和监控构建。
8. **行动**：在Jenkins控制台输出页面的底部，当最终出现`Should we deploy to Production?`的小型确认对话框时，点击**Promote**来完成管道。

9. **行动**：完成后，点击**返回项目。**
    - **解释**，Jenkins的网络界面也允许你监控、配置和管理你的管道。
        - **Build Now**按钮可以用来启动一个新的管道。
        - 注意，不同的管道及其阶段都有图形化的进度表示。
        - （可选）展示**管道语法**页面，并演示创建一个新管道是多么容易。



10. **行动**：回到OpenShift容器平台Web控制台，向观众提问。

3. 清理环境

如果你不打算进行任何额外的演示，请回到OpenShift容器平台web控制台的主页，删除你的项目。

4. 额外演示

你可以进行以下可选的演示。

- 编辑流水线，删除手动输入步骤。
- 展示如何向上和向下扩展部署。
- 显示一个容器的指标和日志。
- 显示你项目的配额限制。