Implementing Fully-Automated Deployment in a CD Pipeline

Introduction

Many companies that practice continuous deployment use a hands-free, fully-automated deployment model. This allows them to ship code to production with various automated tests and sanity checks built into the process, bypassing the need for human intervention. In this lesson, you will implement a basic sanity check and see a simple fully-automated deployment pipeline in action. This will give you a hands-on introduction to the concept of fully-automated deployments.

On the hands-on lab page, locate the *Jenkins Server Public IP* and copy it to your clipboard. Open a new tab in your browser and paste the IP address, followed by the port number `8080`:

```
<JENKINS_SERVER_PUBLIC_IP>:8080
```

Using this same IP address, open a terminal window to connect to the server using SSH:

```
ssh cloud_user@<JENKINS_SERVER_PUBLIC_IP>
```

We need to show the password for the **admin** user to log in to our Jenkins web interface:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Enter the following values in your **Welcome to Jenkins!** screen:

- Username: **admin**
- Password: Copy the string that is output from the command above and paste it here
- Click **Sign in**.

Prepare the Jenkins Environment and Verify Your Configuration with an Initial Deploy

Add GitHub Credentials in Jenkins

> **Note**: You need a GitHub account for this step.

We will use a GitHub API token for the next step. Navigate to the GitHub tab in your browser. Click your profile picture in the top right of the page, click **Settings**, click **Developer settings**, click **Personal access tokens**, and finally click **Generate new token**.

click **Personal access tokens**, and finally click **Generate new token**.

Name this token "jenkins" and be sure to click the checkbox next to **admin:repo_hook**. Click **Generate token** at the bottom of the page. Copy the token to your clipboard.

Back in the Jenkins tab in your browser, click **Manage Jenkins** in the menu on the left of the page and then click **Manage Credentials**, under **Security**. Under **Stores scoped to Jenkins**, click **(global)**, then **Add Credentials** in the menu on the left of the page. Provide the following information:

- *Username*: Provide your GitHub username
- *Password*: Paste the API token from your clipboard.
- *ID*: github_key
- *Description*: GitHub Key

Click **OK**.

Click **Add Credentials** in the menu on the left of the page.

Add Docker Hub Credentials in Jenkins

> **Note**: You need a DockerHub account for this step.

- *Username*: Provide your DockerHub username
- *Password*: Provide your DuckerHub password
- *ID*: docker_hub_login
- *Description*: Docker Hub Login

Click **OK**.

Add the Kubeconfig from the Kubernetes master as a credential in Jenkins

We will need to view the contents of our Kubeconfig for this step. Log in to the Kubernetes master node by navigating to the hands-on lab page, copy the *Kubernetes Master Public IP*, and use the credentials for that instance to log in via SSH:

```
ssh cloud_user@<KUBERNETES_MASTER_PUBLIC_IP>
```

Next, display the contents of our Kubeconfig:

```
cat ~/.kube/config
```

Copy the output of this file to your clipboard. We will need to paste this into Jenkins, so navigate back to the Jenkins tab in your browser.

Click **Add Credentials** in the menu on the left of the page.

Add credentials with the following information:

- *Kind*: Kubernetes configuration (kubeconfig)
- *ID*: kubeconfig
- *Description*: Kubeconfig
- *Kubeconfig*: **Enter directly**
    - *Content*: Paste the contents of `~/.kube/config`

Click **OK**.

Configure Environment Variables

On the main page of Jenkins, click **Manage Jenkins**. Under **System Configuration**, click **Configure System**.

In the *Global Properties* section, click the checkbox next to **Environment variables**. Click **Add**.

- **Name**: KUBE_MASTER_IP
- **Value**: <KUBERNETES_MASTER_IP>

Click **Apply**.

In the *GitHub* section, click **Add GitHub Server** and then click **GitHub Server**.

- **Name**: GitHub
- **Credentials**: Click **Add** and then click **Jenkins**
    - **Kind**: Secret text
    - **Secret**: Paste the GitHub API token from the earlier step
    - **ID**: github_secret
    - **Description**: GitHub Secret
    - Click **Add**
    - Click the dropdown next to *Credentials* and select the **GitHub Secret** we just added
    - Make sure to click the **Manage hooks** checkbox
    - Click **Save**.

Fork the GitHub Repository

Open the following link in a new tab in your browser:

- **https://github.com/ACloudGuru-Resources/cicd-pipeline-train-schedule-autodeploy**

Click **Fork** in the top-right of the page.

Click **Jenkinsfile** to open the file, then click the **Edit** icon in the top-right of the window.

- Change the `DOCKER_IMAGE_NAME` at the top of the Jenkinsfile to use your Docker Hub username instead of `willbla`.
- Click **Commit Changes**.

Set Up Project

Back in the Jenkins tab in our browser, click **New Item**. Use a *Name* of "train-schedule" and select **Multibranch Pipeline** as the type. Click **OK**.

In the *Branch Sources* section, click **Add source**, and then click **GitHub**.

- **Credentials**: Select the *GitHub Key*
- **Repository HTTPS URL**: Type `https://github.com/<your_GitHub_username>/cicd-pipeline-train-schedule-autodeploy`
- Click **Validate** to ensure your credentials are configured correctly
- In the *Behaviors* section, delete both *Discover pull requests* options by clicking the red **X** in the top right of each of their respective sections.

Click **Save**.

Click **train-schedule** in the top-left of the page and then click on **master**.

The initial build will take some time. Wait a few moments until your build gets to the *DeployToProduction* stage. When it is ready, hover your mouse over the blue box and click **Proceed**.

On the hands-on lab page, copy the **Kubernetes Master Public IP** and navigate to it in a new tab in your browser, using port `8080`.

```
<KUBERNETES_MASTER_PUBLIC_IP>:8080
```

The train-schedule app will load.

Add a Smoke Test with Automated Deployment and Remove the Human Approval Step from the Pipeline, Then Deploy

In the GitHub tab in your browser, click on the **Jenkinsfile** to open it. Click the **Edit** icon in the top-right of the page to edit this file.

Remove the human input step from the deployment and add a smoke test before the production deployment. Your **Jenkinsfile** should look like this:

```
pipeline {
    agent any
    environment {
        //be sure to replace "willbla" with your own Docker Hub username
        DOCKER_IMAGE_NAME = "willbla/train-schedule"
        CANARY_REPLICAS = 0
    }
    stages {
        stage('Build') {
            steps {
                echo 'Running build automation'
                sh './gradlew build --no-daemon'
                archiveArtifacts artifacts: 'dist/trainSchedule.zip'
            }
        }
        stage('Build Docker Image') {
            when {
                branch 'master'
            }
            steps {
                script {
                    app = docker.build(DOCKER_IMAGE_NAME)
                    app.inside {
                        sh 'echo Hello, World!'
                    }
                }
            }
        }
        stage('Push Docker Image') {
            when {
                branch 'master'
            }
            steps {
                script {
                    docker.withRegistry('https://registry.hub.docker.com',
'docker_hub_login') {
                        app.push("${env.BUILD_NUMBER}")
                        app.push("latest")
```

```
                    }
                }
            }
        }
        stage('CanaryDeploy') {
            when {
                branch 'master'
            }
            environment {
                CANARY_REPLICAS = 1
            }
            steps {
                kubernetesDeploy(
                    kubeconfigId: 'kubeconfig',
                    configs: 'train-schedule-kube-canary.yml',
                    enableConfigSubstitution: true
                )
            }
        }
        stage('SmokeTest') {
            when {
                branch 'master'
            }
            steps {
                script {
                    sleep (time: 5)
                    def response = httpRequest (
                        url: "http://$KUBE_MASTER_IP:8081/",
                        timeout: 30
                    )
                    if (response.status != 200) {
                        error("Smoke test against canary deployment
failed.")
                    }
                }
            }
        }
        stage('DeployToProduction') {
            when {
                branch 'master'
```

```
                branch 'master'
            }
            steps {
                milestone(1)
                kubernetesDeploy(
                    kubeconfigId: 'kubeconfig',
                    configs: 'train-schedule-kube.yml',
                    enableConfigSubstitution: true
                )
            }
        }
    }
    post {
        cleanup {
            kubernetesDeploy (
                kubeconfigId: 'kubeconfig',
                configs: 'train-schedule-kube-canary.yml',
                enableConfigSubstitution: true
            )
        }
    }
}
```

Click **Commit Changes** to save your changes to the Jenkins file. The deployment will start automatically and can be viewed in the Jenkins tab of your browser.

Demonstrate the Pipeline in Action

In the GitHub tab in your browser, navigate to the main page of your fork by clicking on **cicd-pipeline-train-schedule-autodeploy** at the top of the page.

Click on **branches** to display the three branches of this repository. Click on **New pull request** for the *new-code* branch.

Change the following fields on this page:

- **base fork**: Set this to your personal fork of the **cicd-pipeline-train-schedule-autodeploy** repo
- **base**: master

The page will update and show the changes from the *new-code* branch to the *master* branch.

Click **Create pull request**. When the page updates, click **Merge pull request**. Finally, click **Confirm merge**.

Back in the Jenkins tab in your browser, a new build should spin up shortly.

Back in the Jenkins tab in your browser, a new build should spin up shortly.

Navigate to the tab in your browser that displays the **train-schedule** application. Refresh this page to see the changes that were made.

Conclusion

Congratulations, you've completed this hands-on lab!