**Graphical user interface, text

Description automatically generated**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Deploying a Docker Container with Jenkins Pipelines**

**Introduction**

Docker is a great tool for containerization. However, there are some problems that need to be solved when transitioning from a Jenkins Pipeline deployment that uses code artifacts to one that uses Docker. This learning activity will guide you through the process of deploying an app as a Docker container using a Jenkins Pipeline.

After completing this exercise, you will have the knowledge necessary to Dockerize applications while still maintaining automated deployment with Jenkins Pipelines.

**Solution**

Log in to the Jenkins instance using the credentials provided in a new browser tab:

<PUBLIC\_IP\_ADDRESS>:8080

Log in to the Jenkins server using the credentials provided to retrieve the temporary admin password for the Jenkins instance:

ssh cloud\_user@<PUBLIC\_IP\_ADDRESS>

**Configure Jenkins to run the new dockerized train-schedule pipeline.**

1. Access the Jenkins instance and create the first admin user.
2. Using the cloud\_user password, create a global Jenkins credential for the production server.
   * Kind: Username with password
   * Scope: Global
   * Username: deploy
   * Password: <CLOUD\_USER\_PASSWORD>
   * ID: webserver\_login
   * Description: Webserver Login
3. Create a global Jenkins credential for the Docker image registry (Docker Hub).
   * Kind: Username with password
   * Username: <DOCKER\_HUB\_USERNAME>
   * Password: <DOCKER\_HUB\_PASSWORD>
   * ID: docker\_hub\_login
   * Description: Docker Hub Login

**Note:** You will need a Docker hub account in order to use Docker Hub as an image registry.

1. Configure a global property in Jenkins to store the production server IP by navigating to **Manage Jenkins** > **Configure System** and adding a environment variable.
   * Name: prod\_ip
   * Value: <PRODUCTION\_SERVER\_PUBLIC\_IP\_ADDRESS>
2. Make a personal fork of the GitHub repo at:

https://github.com/linuxacademy/cicd-pipeline-train-schedule-dockerdeploy

1. Generate a new GitHub API key to allow Jenkins to access the forked repo by navigating to **Profile** > **Settings** > **Developer Settings** > **Personal Access Tokens** > **Generate New Token**.
   * Token Description: Jenkins
   * Permissions: admin-repo\_hook
2. Copy the GitHub token.
3. In Jenkins, create a Multibranch Pipeline project named *train-schedule*.
4. Under Credentials, add the GitHub account used to fork the repo to Jenkins.
   * Kind: Username with password
   * Scope: Global
   * Username: <GITHUB\_USERNAME>
   * Password: <GITHUB\_API\_KEY>
   * ID: github\_key
   * Description: Github Key
5. Select the *Github Key* and the forked repository.
   * Owner: <GITHUB\_USERNAME>
   * Repository: cicd-pipeline-train-schedule-dockerdeploy
6. Click **save**.

**Successfully deploy the train-schedule app to production as a Docker container using the Jenkins Pipeline.**

1. Modify the Jenkinsfile in GitHub to build and push the Docker image to Docker Hub, and commit the changes.
2. pipeline {
3. agent any
4. stages {
5. stage('Build') {
6. steps {
7. echo 'Running build automation'
8. sh './gradlew build --no-daemon'
9. archiveArtifacts artifacts: 'dist/trainSchedule.zip'
10. }
11. }
12. stage('Build Docker Image') {
13. when {
14. branch 'master'
15. }
16. steps {
17. script {
18. app = docker.build("<DOCKER\_HUB\_USERNAME>/train-schedule")
19. app.inside {
20. sh 'echo $(curl localhost:8080)'
21. }
22. }
23. }
24. }
25. stage('Push Docker Image') {
26. when {
27. branch 'master'
28. }
29. steps {
30. script {
31. docker.withRegistry('https://registry.hub.docker.com', 'docker\_hub\_login') {
32. app.push("${env.BUILD\_NUMBER}")
33. app.push("latest")
34. }
35. }
36. }
37. }
38. }

}

1. In Jenkins, click **Build Now**.

**Note:** The initial build may take several minutes to complete.

1. In Docker Hub, under Repositories, select the *train-schedule* app.
2. Click the **Tags** tab to verify that the build was pushed successfully.
3. In GitHub, modify the Jenkinsfile to include a stage that pushes the build to the production server, and commit the changes.
4. stage ('DeployToProduction') {
5. when {
6. branch 'master'
7. }
8. steps {
9. input 'Deploy to Production'
10. milestone(1)
11. withCredentials ([usernamePassword(credentialsId: 'webserver\_login', usernameVariable: 'USERNAME', passwordVariable: 'USERPASS')]) {
12. script {
13. sh "sshpass -p '$USERPASS' -v ssh -o StrictHostKeyChecking=no $USERNAME@${env.prod\_ip} \"docker pull <DOCKER\_HUB\_USERNAME>/train-schedule:${env.BUILD\_NUMBER}\""
14. try {
15. sh "sshpass -p '$USERPASS' -v ssh -o StrictHostKeyChecking=no $USERNAME@${env.prod\_ip} \"docker stop train-schedule\""
16. sh "sshpass -p '$USERPASS' -v ssh -o StrictHostKeyChecking=no $USERNAME@${env.prod\_ip} \"docker rm train-schedule\""
17. } catch (err) {
18. echo: 'caught error: $err'
19. }
20. sh "sshpass -p '$USERPASS' -v ssh -o StrictHostKeyChecking=no $USERNAME@${env.prod\_ip} \"docker run --restart always --name train-schedule -p 8080:8080 -d <DOCKER\_HUB\_USERNAME>/train-schedule:${env.BUILD\_NUMBER}\""
21. }
22. }
23. }

}

1. In Jenkins, click **Build Now**.
2. Once the build is complete, using a web browser, verify that the application has been deployed successfully.

<PRODUCTION\_SERVER\_PUBLIC\_IP\_ADDRESS>:8080

**Conclusion**

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