Multi-Tier Java Application Deployment to Kubernetes

Practitioner Assessment

Contents

Title: Multi-Tier Java Application Deployment to Kubernetes

Difficulty Level

Practitioner

Duration

120 minutes

What you will learn

By the end of this, you will be able to:

- How to containerize a multi-tier Java based web application using Docker and Docker-Compose.
- Using Maven to manage dependencies and build artifacts.
- Setting up Jenkins pipelines for automation.
- Deploying containerized applications on Kubernetes.
- Configuring Kubernetes resources like Deployments, Services, and Ingress.

What you will be provided

- A Linux Virtual Machine with the necessary software, including Visual Studio Code, Kubectl, Docker, Maven, Minikube, Jenkins, and Java libraries, is available in the lab.
- The project folder, containing the required files, is located at Desktop > Project.

What you need to know

- Familiarity with Docker, including building and running containers.
- Basic knowledge of Jenkins and pipelines for continuous integration.
- Understanding Pods, Deployments, Services, and ConfigMaps.
- Some experience with Java, Maven, kubernetes and Git.

Skill Tags

- Docker
- Jenkins
- Git
- Creating a Pipeline Job
- Maven
- Docker Hub
- Kubernetes

What you will do

You are tasked with deploying a multi-tier Java application for a growing startup company. The application needs to be containerized to ensure consistent deployment, scalability, and resilience. As a DevOps

engineer, you will be responsible for setting up the environment, containerizing the application, automating the build and deployment processes, and ensuring high availability and fault tolerance in Kubernetes. You will integrate key tools like Jenkins, Maven, and Git to streamline the development and deployment process.

Note:

The user must log in to their GitHub and Docker Hub accounts using their credentials.

Activities

1. Setting Up Docker

1. Install Maven using the commands below:
   sudo apt update
   sudo apt install maven

2. You can find the details of the jenkins credentials and MySQL database local passwords in Readme.txt file on Desktop.

3. Add Jenkins to the Docker group and restart Jenkins

Note: Use the `docker compose` command instead of the legacy `docker-compose`, as it is integrated into Docker CLI (v20.10+), eliminating the need for a separate binary. It provides better performance, consistency, and is actively maintained, unlike the legacy command.

2. Building the Docker Image and docker-compose.yaml

1. Navigate to the Project folder on the Desktop. Open the Dockerfile using Visual Studio Code in the VM lab.
2. Build the project using **maven:3.9.9-eclipse-temurin-17**.
3. Run the project using **openjdk:17-slim**
4. Expose the application port on ***port 8081***.
5. Build the image with name “todo-application-image" with tag latest.
6. In the docker-compose.yaml , the service names of the components should be todo-application and mysql-db.
7. The local host port and container port should be 8082:8081 for the todo-application service.
8. Use “root” as the username and “Root@123” as password for mysql server.
9. Create a custom bridge network named todo-network.
10. Use database name as “tododb”.
11. The local host port and container port should be 3307:3306 for the mysql-db service.
12. Use volumes to persist data. The volume name should be “mysql-data”.
13. Create a repository with the name “todo-application” in Docker Hub using this link.

3. Setting Up GitHub Repository and Docker Hub

1. Create a public repository in your personal GitHub account using the provided link.
2. Ensure the repository is publicly accessible. If it is private, generate a Personal Access Token (PAT) to access it. Update the changes in the Jenkinsfile with the GitHub credentials. The project should be in the master branch.
3. Once all updates with your Docker Hub and GitHub details are complete, commit the application or project to the GitHub repository created earlier using the Linux terminal.

4. Configuring Jenkins Pipeline

1. Go to Manage Jenkins > Credentials > Global > Add Credentials.
2. Add your Docker Hub credentials and save them with the ID “docker-hub-credentials.”
3. If the GitHub repository created is private, you will need to add the credentials for GitHub.
4. Create a new Jenkins pipeline named "todo-application-pipeline".
5. Clone the repository from GitHub and build the project with Maven by skipping tests.
6. Build and push the docker image to docker hub. Use docker hub credentials.
7. Deploying the application using Docker Compose and verify the services
8. Clean the workspace using “rm -rf *”
9. Once the Jenkinsfile is configured. Commit the repository to the previously created github repository.
10. After a successful build, the Maven app will be visible on port 8082 in Chrome within the VM lab.

5. Deploy application using Kubernetes

1. Start the Minikube.
2. Navigate to the project Folder, you can find todo-application-deployment.yaml file and mysql-deployment.yaml.
3. You can use them as the files are preconfigured as part of the assessment.
4. Create a secret in kubectl with name “my-registry-secret” which holds the details of docker hub credentials.
5. Replace the image details with the details of your docker hub repository.
6. Expose Node port 30080.
7. Apply the files and the application can be available on http://<minikube-ip>:30080 .

Testcases

1. Checking if the user 'jenkins' is part of the 'docker' group. [5 marks]
2. Checking if the Docker image ‘todo-application-image’ is created. [5 marks]
3. Checking if the Docker image ' todo-application-image ' is tagged correctly with latest. [5 marks]

4. Checking if the Docker volume 'mysql-data ' is created after executing the Jenkins pipeline. [5 marks]
5. Checking if the Docker network 'todo-network' is created after executing the Jenkins pipeline. [5 marks]
6. Checking if the application created as part of the Jenkins pipeline execution is active in the Docker container. [10 marks]
7. Checking if the MySQL database container ' mysql-db-1' was created and exists as part of the Jenkins pipeline execution. [5 marks]
8. Checking if the To-Do application container ' todo-application-1' was created and exists as part of the Jenkins pipeline execution. [5 marks]
9. Checking if the Kubernetes secret with the name ' my-registry-secret ' exists. [5 marks]
10. Checking if the application is running on Minikube at port 30080. [10 marks]
11. Checking if the application hosted on Minikube at port 30080 serves the expected content. [15 marks]
12. Checking if the latest build of 'todo-application-pipeline' is successful. [10 marks]
13. Checking if all required application pods are in a Running state. [10 marks]
14. Checking if the database 'tododb' exists in the MySQL container. [5 marks]