Multi-Tier User Management Web Application with Kubernetes and CI/CD

Proficient Assessment

Contents

Title: Multi-Tier User Management Web Application with Kubernetes and CI/CD

Difficulty Level

Proficient

Duration

120 minutes

What you will learn

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

- Learn how to containerize a Java application along with its MySQL database to ensure consistent deployment.
- Set up a Jenkins pipeline to automate the build, test, and deployment processes, ensuring faster and more reliable delivery.
- Deploy and manage the application using Kubernetes, ensuring scalability, high availability, and fault tolerance.

What you will be provided

- A Linux virtual machine with the necessary software, including Visual Studio Code, Kubectl, Docker, MySQL, 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 understanding of Continuous Integration and Continuous Deployment (CI/CD) workflows using Jenkins.
- Knowledge of MySQL database operations, including schema creation, queries, and application connectivity.
- Familiarity with Kubernetes components such as Pods, Deployments, Services, and Secrets.
- Some experience with Java, Maven, Linux, and Git.

Skill Tags

- Docker
- Jenkins
- Git
- MySQL
- Java
- Maven

- Docker Hub
- Kubernetes

What you will do

You're tasked with containerizing a Java-based user management application, which includes a backend and MySQL database, for a client. The user is responsible for setting up Docker to containerize the application, creating a Jenkins pipeline to automate the build, test, and deployment process, and pushing the Docker images to Docker Hub. The final step is deploying the application on Kubernetes for high availability and scalability. This project will ensure the application is easily maintainable and deployable in any environment.

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-jdk-slim**
4. Expose the application port on ***port 8081***.
5. Build the image with the name "usermanagement-application-image" with tag latest.
6. In the docker-compose.yaml , the service names and container names of the components should be **usermanagement-application** and **mysql-db**.
7. The local host port and container port should be 8082:8081 for the usermanagement-application service.
8. Use "root" as the username and "Root@123" as password for mysql server.

9. Create a custom bridge network named **user-network**.
10. Use database name as “**userdb**”.
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 “**usermanagement**” 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 "**usermanagementpipeline**".
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 user management application will be visible on port 8082 in Chrome within the VM lab.

5. Configuring Kubernetes file for application and database

1. Start the Minikube.
2. Create a kubernetes configuration file for usermanagement application in which the deployment and service of application will be there.
3. The node port should be **30080.**
4. Configure 2 replicas for the application.
5. Use docker hub image that has been created as part of the project.
6. Create a kubectl secret with the name “**my-registry-secret**” to store docker hub credentials that are going to be used in Kubernetes files.
7. Version of MySQL is “**mysql:8.0**”.
8. The container port will be **3306** for MySQL db.
9. Use the concept of persistent volumes and persistent volume claim for the deployment configuration of MySQL database in Kubernetes.

10. 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 ‘usermanagement-application-image’ is created. [5 marks]
3. Checking if the Docker image ' usermanagement-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 'user-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 application container ' usermanagement-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 'usermanagementpipeline' is successful. [10 marks]
13. Checking if all required application pods are in a Running state. [10 marks]
14. Checking if the database 'userdb' exists in the MySQL container. [5 marks]