Experiment 12

Lab Objective:

This experiment aims to demonstrate container orchestration using Kubernetes by guiding you through the deployment, scaling, and management of containerized Applications.

Prerequisites:

● Basic understanding of containerization and Docker.
● Kubernetes cluster (local or cloud-based) set up and running.
● kubectl CLI tool installed and configured to communicate with your Kubernetes cluster.
Outcome:
1. Learn how to deploy and manage containerized applications using Kubernetes.
2. Understand key Kubernetes concepts such as Pods, Deployments, and Services.
3. Gain experience with scaling and updating applications in a Kubernetes environment.

Description:

Task 1: Deploy a Containerized Application
● Create a simple Docker image for a sample application (e.g., a basic web server or API).
● Write a Kubernetes Deployment YAML file to deploy this Docker image to your Kubernetes cluster.
● Apply the Deployment YAML file using the following command:
kubectl apply -f deployment.yaml
● Verify the deployment and check the status of the Pods:
kubectl get deployments
kubectl get pods
Task 2: Expose the Application with a Service
● Write a Kubernetes Service YAML file to expose the deployed application to the outside world.
● Apply the Service YAML file using the following command
kubectl apply -f service.yaml
● Verify the Service and obtain the external IP address or port:
kubectl get services
Task 3: Scale the Application
● Scale the number of replicas for your Deployment using the following command:
kubectl scale deployment <deployment-name> --replicas=3
● Verify that the Pods are scaled and running:
kubectl get pods
Task 4: Update the Application
● Modify the Docker image (e.g., update the application code) and build a new image.
● Update the Deployment to use the new Docker image by editing the Deployment YAML file or using the following command:
kubectl set image deployment/<deployment-name>
<container-name>=<new-image>
● Verify that the new version of the application is deployed and running:
kubectl rollout status deployment/<deployment-name>
Note: Attach screenshots to your document that clearly showcase the steps and outcomes of your work for this lab.
Ask AI

Experiment 12

1. Introduction

Kubernetes has become the industry-standard platform for container orchestration. It allows developers and system administrators to deploy, manage, and scale containerized applications automatically. This experiment aims to provide hands-on experience with Kubernetes and help students understand how it manages container lifecycles, load balancing, scaling, and updates.

In this lab, we worked with a Kubernetes cluster, deployed a containerized web application, exposed it via a Service, scaled the application, and performed a rolling update. The goal was to simulate real-world scenarios encountered in production environments.

2. Lab Objective

This experiment is designed to:
Introduce the basics of Kubernetes for container orchestration.
Deploy and manage containerized applications in a Kubernetes environment.
Understand and use Kubernetes objects like Pods, Deployments, and Services.
Learn how to scale applications and perform rolling updates using Kubernetes commands and YAML configuration files.

3. Prerequisites

Before starting this lab, the following prerequisites were completed:

A Kubernetes cluster was set up using [Minikube/Docker Desktop/kind/cloud provider].

kubectl, the Kubernetes CLI, was installed and configured.

A basic Docker image of a simple web application was created and pushed to a container registry.

Familiarity with writing YAML configuration files was assumed.

4. Expected Outcomes
Upon completion of this lab, the following were achieved:
Successfully deployed a containerized application to Kubernetes.
Exposed the application using a Kubernetes Service.
Scaled the application by adjusting the number of running Pods.
Updated the running application using rolling update strategies.

5. Lab Tasks
Task 1: Deploy a Containerized Application
Step 1: Create Docker Image
We used a simple Python Flask application and created a Docker image using the following Dockerfile:

Dockerfile

Experiment 12

Copy
Edit
FROM python:3.9-slim
WORKDIR /app
COPY app.py .
RUN pip install flask
CMD ["python", "app.py"]
Then we built and tagged the image:

bash
Copy
Edit
docker build -t myapp:1.0 .
Step 2: Write Deployment YAML File
A deployment.yaml file was created:

yaml
Copy
Edit
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
      - name: myapp-container
        image: myapp:1.0
        ports:
        - containerPort: 5000

Step 3: Deploy Application to Kubernetes
bash
Copy
Edit
kubectl apply -f deployment.yaml
kubectl get deployments
kubectl get pods

Task 2: Expose the Application with a Service

Experiment 12

Step 1: Create Service YAML File
yaml
Copy
Edit

```
apiVersion: v1
kind: Service
metadata:
  name: myapp-service
spec:
  type: NodePort
  selector:
    app: myapp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 5000
```

Step 2: Apply and Verify the Service
bash
Copy
Edit

```
kubectl apply -f service.yaml
kubectl get services
```

We accessed the app at http://localhost:<NodePort> to confirm it was working.

Task 3: Scale the Application
To scale the number of pods from 1 to 3:

bash
Copy
Edit

```
kubectl scale deployment myapp-deployment --replicas=3
kubectl get pods
```

This created 2 additional replicas and ensured high availability.

Task 4: Update the Application
Step 1: Modify Docker Image
Updated app.py with new content, rebuilt, and tagged the Docker image as version 2.0:

bash
Copy
Edit

```
docker build -t myapp:2.0 .
```

Step 2: Update Deployment
bash
Copy
Edit

Experiment 12

kubectl set image deployment/myapp-deployment myapp-container=myapp:2.0
kubectl rollout status deployment/myapp-deployment
The rolling update completed successfully, and all pods were updated with the new image.

6. Summary Table of Kubernetes Commands Used
Command Purpose
kubectl apply -f deployment.yaml Deploys application
kubectl get deployments Lists current deployments
kubectl get pods Displays running pods
kubectl apply -f service.yaml Exposes application
kubectl get services Displays available services
kubectl scale deployment ... Scales application
kubectl set image ... Performs rolling update
kubectl rollout status ... Checks update progress

7. Conclusion
This lab provided a comprehensive understanding of Kubernetes and its core capabilities. We successfully deployed a containerized web application, exposed it using a service, scaled it for high availability, and updated it using rolling deployments—all using Kubernetes commands and YAML configuration files.

Through this hands-on activity, we explored key concepts like Pods, Deployments, Services, and rolling updates. This forms a strong foundation for real-world DevOps practices and cloud-native application deployment.

8. References
Kubernetes Official Documentation

kubectl Command Reference

Docker Documentation

YAML Configuration Guide for Kubernetes