

EXPERIMENT NO.: 8. Integrate Kubernetes and Docker

AIM: To integrate Kubernetes and Docker

DESCRIPTION: Kubernetes and Docker are both popular technologies for managing containers, but they are used for different purposes. Kubernetes is an orchestration platform that provides a higher-level abstractions for managing containers, while Docker is a containerization technology that provides a lower-level runtime for containers.

To integrate Kubernetes and Docker, you need to use Docker to build and package your application as a container image, and then use Kubernetes to manage and orchestrate the containers.

Step1: Build a Docker image.

Use Docker to build a Docker image of your application. You can use a Dockerfile to specify the base image, copy the application into the container, and specify the command to run the application.

Step2: Push the Docker image to a registry

Push the Docker image to a container registry, such as Docker Hub or Google Container Registry, so that it can be easily accessed by Kubernetes. Deploy the Docker image to a Kubernetes cluster.

Step3: Use Kubernetes to deploy the Docker image to a cluster.

This involves creating a **deployment .yaml** that specifies the number of replicas and the image to be used, and creating a **service.yaml** that exposes the deployment to the network.

Step4: Monitor and manage the containers

Use Kubernetes to monitor and manage the containers. This includes scaling the number of replicas, updating the image, and rolling out updates to the containers.

EXPERIMENT NO.: 9.

Automate the process of running containerized application developed in exercise 7 using Kubernetes

AIM: Automate the process of running containerized application developed in exercise 7 using Kubernetes

DESCRIPTION:

To automate the process of running the containerized application , follow these steps

- Create a Kubernetes cluster: Create a Kubernetes cluster using a cloud provider, such as Google Cloud or Amazon Web Services, or using a local installation of Minikube.

Step1: Create a docker image

- Create a Dockerfile for your application

- Build the Docker image

```
C:\Users\kuppa\dockersample>docker build -t myapp .
[+] Building 10.3s (8/8) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 312B
=> [internal] load metadata for docker.io/library/nginx:alpine
=> [auth] library/nginx:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build context
=> => transferring context: 32B
=> [1/2] FROM docker.io/library/nginx:alpine@sha256:a45ee5d042aaa9e81e013f97ae40c3dda26f98f22b6251acdf28e579560d55
=> CACHED [2/2] COPY index.html /usr/share/nginx/html/
=> exporting to image
=> => exporting layers
=> => writing image sha256:b1f8c33c58455da995ad5d9feb743ea860e004f31ff3583656b843bb578dc2f7
=> => naming to docker.io/library/myapp

View build details: docker-desktop://dashboard/build/default/default/lgtj6nkg04wbzwuf3u3o3evbq

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
```

- Push the Docker image to a registry: Push the Docker image of your application to a container registry, such as Docker Hub or Google Container Registry.

```
C:\Users\kuppa\dockersample>docker tag myapp swetha328/myapp
C:\Users\kuppa\dockersample>docker push swetha328/myapp
Using default tag: latest
The push refers to repository [docker.io/swetha328/myapp]
7d9abf1752da: Layer already exists
a51b172d7184: Layer already exists
b7486fe26981: Layer already exists
320c8baef084: Layer already exists
d2cef4a1b224: Layer already exists
4275164ce225: Layer already exists
5e92270dbfe6: Layer already exists
p5d2e1fcf1ad: Layer already exists
af9a70194aa4: Layer already exists
latest: digest: sha256:8d1d85e26fd1e3f3e95382b60cd4b559e6ad005fde12dab9eeaec459bfaf9b8f size: 2197
```

Step2: Create a deployment

- Make sure Minikube is running. If it's not, start it with the command:

minikube start

- Create a deployment: Create a deployment in Kubernetes that specifies the number of replicas and the Docker image to use. Here's an example of a deployment YAML file:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: html-app-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - name: htmlcontainer
          image: swetha328/myapp:latest
          ports:
            - containerPort: 80
```

- Create a service: Create a service in Kubernetes that exposes the deployment to the network. Here's an example of a service YAML file

```
apiVersion: v1
kind: Service
metadata:
  name: my-html-app-service
  namespace: default # Use your desired namespace
spec:
  type: NodePort
  selector:
    app: myapp # Ensure this matches the label in your Deployment
  ports:
    - protocol: TCP
      port: 80 # Port that the service will expose
      targetPort: 80 # Port on the container to which traffic will be forwarded
      nodePort: 30070 # The port on each node on which this service is exposed
```

- Apply the deployment and service to the cluster: Apply the deployment and service to the cluster using the kubectl command line tool.

```
C:\Users\kuppa\dockersample>kubectl apply -f deployment.yaml
deployment.apps/html-app-deployment created

C:\Users\kuppa\dockersample>kubectl apply -f services.yaml
service/my-html-app-service created
```

Step3: Port Forwarding

- Verify the deployment: Verify the deployment by checking the status of the pods and the service.

```
C:\Users\kuppa\dockersample>kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
html-app-deployment-fb7875db6-5blrs 1/1     Running   0           60s
html-app-deployment-fb7875db6-bw4jb 1/1     Running   0           60s
html-app-deployment-fb7875db6-n88qq 1/1     Running   0           61s

C:\Users\kuppa\dockersample>kubectl get services
NAME            TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
kubernetes      ClusterIP   10.96.0.1    <none>        443/TCP          97s
my-html-app-service NodePort    10.96.34.89  <none>        80:30070/TCP     59s
```

- port forward the service

```
C:\Users\kuppa\dockersample>kubectl port-forward service/my-html-app-service 8085:80
Forwarding from 127.0.0.1:8085 -> 80
Forwarding from [::1]:8085 -> 80
```

Step 4: Access the Application

Once you have set up port forwarding, you can access your application at <http://localhost:8085>.

The screenshot shows a web browser window with the address bar displaying 'localhost:8085'. The page content is a 'Registration Form' with the following fields:

- First Name:
- Last Name:
- Email:
- Password:
- Re-type Password:
- Contact:
- Gender: (dropdown menu)

A green 'Submit' button is located at the bottom of the form.

EXPERIMENT NO.: 10. Install and Explore Selenium for automated testing

AIM: Install and Explore Selenium for automated testing

DESCRIPTION: To install and explore Selenium for automated testing, you can follow these steps: