

Lab Program 4

DD	MM	YY	YY
24	04	20	25

4) Container Orchestration With Kubernetes

Tools: Kubernetes

Program: * set up Kubernetes cluster (use minikube or cloud provider)

* Deploy a sample application using a Deployment and service

* scale the application using kubectl scale

Sol: Start Minikube using DockerStep 1:- Open powershell as administrator and Run
➤ minikube start --driver=dockerStep 2:- Check the status

➤ minikube status

Step 3:- check cluster

➤ kubectl get nodes

⇒ Create a Simple Pod.yaml file

apiVersion: v1

kind: pod

metadata:

name: my-nginx

spec:

containers:

- name: nginx

image: nginx:latest

ports:

- containerPort: 80

D.S.C.E.

Run the following commands in the terminal of Visual studio:

➤ `kubectl apply -f pod.yaml`

➤ `kubectl get pods`

➤ `kubectl get pods -o wide`

➤ `minikube ssh`

➤ `curl 10.244.0.3`

⇒ Create a Kubernetes Deployment and Service for a simple python web application (like Flask) running in Minikube

Step 1 :- Create app.py file

```
from flask import Flask
```

```
app = Flask(__name__)
```

```
@app.route('/')
def hello():
```

```
    return "Hello from App 1 !! Kubernetes,  
    also know as K8S"
```

```
if __name__ == '__main__':
```

```
    app.run(host='0.0.0.0', port=5000)
```

Step 2 :- Create requirements.txt

```
flask==3.0.0
```

Step 3 :- Create Dockerfile

FROM python:3.12-slim

WORKDIR /app

COPY requirements.txt

RUN pip install --no-cache-dir -r requirements.txt

COPY app.py

EXPOSE 5000

CMD ["python", "app.py"]

Step 4 :- run The below commands in visual studio

> docker build -t lds22cs141/app1:k8s:latest

> docker push lds22cs141/app1:k8s:latest

Step 5 :- Create deployment.yaml file

apiVersion: apps/v1

kind: Deployment

metadata:

name: hw-deployment

spec:

replicas: 2

selector:

matchLabels:

app: hello-world

template:

metadata:

labels:

app: hello-world

spec:

- name: hw-container

image: lds22cs141/app1-k8s:latest

ports:

- containerport: 5000

Step 6:- create service.yaml file

apiVersion: v1

kind: Service

metadata:

name: hello-world

spec:

type: NodePort

selector:

app: hello-world

ports:

- port: 5000

targetPort: 5000

nodeport: 30005

Step 7:- Run the below commands in Visual Studio

> kubectl apply -f deployment.yaml

> kubectl apply -f service.yaml

> kubectl get pods

> kubectl get svc

> kubectl scale deployment/hw-deployment
--replicas=3

> kubectl get deployment

> kubectl get pods

> kubectl port-forward svc/hello-world
5000:5000
D.S.G.E.

Step 8:- <https://localhost:5000/>

⇒ Simple python application in kubernetes using Configmap and Secret.

Step 1:- Create app1.py file

```
from flask import Flask
```

```
import os
```

```
app = Flask(__name__)
```

```
@app.route('/')
def index:
```

```
    app.env = os.getenv("APP_ENV", "not set")
```

```
    db_password = os.getenv("DB_PASSWORD",
                             "not set")
```

```
    return f"APP_ENV : {app.env} & >
```

```
            DB_PASSWORD : {db_password}"
```

```
if __name__ == '__main__':
```

```
    app.run(host='0.0.0.0', port=5000)
```

Step 2:- Create Dockerfile

```
FROM python:3.9-slim
```

```
WORKDIR /app
```

```
COPY app1.py .
```

```
RUN pip install flask
```

```
CMD ["python", "app1.py"]
```

Step 4:- run the below commands in visual studio:

```
> docker build -t id522cs141/python-app:latest
```

```
> docker push id522cs141/python-app:latest
```

Step 5:- create configmap.yaml file

```
apiVersion: v1
```

```
kind: ConfigMap
```

```
metadata:
```

```
  name: my-config
```

```
data:
```

```
  APP-ENV: production.
```

Step 6:- create secret.yaml

```
apiVersion: v1
```

```
kind: Secret
```

```
metadata:
```

```
  name: my-secret
```

```
type: Opaque
```

```
stringData:
```

```
  DB-PASSWORD: mypassword123
```

Step 7:- Create deployment.yaml

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: python-app
```


Spec:

replicas: 1

Selector:

metichLabeler:

app: python-app

template:

metichdata:

labeler:

app: python-app

Spec:

Containers:

- name: app-container

image: l522cst41/python-

app: latest

ports:

- containerport: 5000

env:

- name: APP_ENV

valueFrom:

configMapRef:

name: APP_ENV

key: my-config

- name: DB_PASSWORD

valueFrom:

secretKeyRef:

name: my-secret

key: DB_PASSWORD

Step 8:- Create Service.yaml

apiVersion: v1

kind: Service

metadata:

name: python-service

spec:

type: NodePort

selector:

app: python-app

ports:

- protocol: TCP

port: 80

targetPort: 5000

nodePort: 30006

Step 9:- run the below commands in Visual Studio

> kubectl apply -f conf.g.yaml

> kubectl apply -f secret.yaml

> kubectl apply -f deployment.yaml

> kubectl apply -f service.yaml

> kubectl get pods

> kubectl get svc

> kubectl port-forward svc/python-service
8095:80

Step 10:- open <http://localhost:8095> in browser

5) Basic Infrastructure Deployment

* Tool : Terraform

* Program : Write a Terraform configuration file to provision a single EC2 instance on AWS.

Use Terraform commands (terraform init, terraform plan, terraform apply, terraform destroy) to manage the infrastructure

Sol STEP-1 :- Sign into the AWS console

Login in user with IAM privileges

STEP-2 :- Navigate to IAM and click Users from the left hand - user.

STEP-3 :- Select (or) Create a User

→ go to Create a new user

→ click Add users

→ Enter username (terraform-user)

→ select Programmatic access

→ click Next

* Attach existing policies like

AmazonEC2FullAccess, AmazonS3Full

Access (or) Administrator Access

→ Finish user creation.

STEP-4 :- Generate Access Key

→ Under Security credentials click Create access key

→ Choose The use case (CLI)

→ Click next

→ copy The access key and Secret Access Key.

⇒ Create main.tf file in visual studio

```
provider = "aws" {
```

```
  region = "ap-south-1"
```

```
resource "aws-instance" "ec2_machine"
```

```
  ami = "ami-0af95698c8786b23a"
```

```
  instance_type = "t2.micro"
```

```
  tags = {
```

```
    Name = "Terra"
```

```
  }
```

```
}
```

→ Follow The given below commands in visual studio

> aws configure

> Access Key : AKIA3RRMOP3Z5N

Secret Key : EKb80K1YKU3hoAHcjYKisIQ

> Terraform init

> Terraform plan

> Terraform apply.

b) Terraform configuration to create an AWS S3 bucket
 sol STEP 1:- Make sure to have a text file ready

sample.txt

Hello

STEP-2:- Create main.tf file

provider "aws" {

region = "ap-south-1" }

resource "aws_instance" "ec2-machine"

ami = "ami-0af56123b23a"

instance_type = "t2.micro"

tags = {

Name = "Terra"

}

}

resource "aws_s3_bucket" "demo_bucket" {

bucket = "my-unique-s3-bucket-2025-
demo-xy2k3"

tags = {

Name = "upload-demo"

}

}

resource "aws_s3_bucket_object" "text-file" {

STEP 3:- bucket = aws_s3_bucket.demo_bucket

bucket

key = "sample.txt"

source = "C:\\Users\\Nithill OneDrive\\Desktop
sample.txt"

D.S.C.E.

STEP 3:- Run the following commands in visual studio

> terraform plan

> terraform apply