Kubernetes Capstone Project

Problem Statement:

Design and implement an end-to-end cloud infrastructure automation and deployment solution using Terraform, Ansible, Docker, and Kubernetes.

Terraform Task:

Problem Statement:

Launch a Ubuntu EC2 instance (t2.micro) to be used as your terraform workstation. From that WS, using Terraform, launch an EC2 instance (instance type: t2.micro, OS: Ubuntu) to be used as an ansible workstation for the ansible task. Please make sure that you create a key (using ssh-keygen) and use it while launching the EC2 so that we can SSH into the ansible WS once it is created.

Ansible Tasks:

Problem Statement:

Once you have created a new instance using Terraform (as part of Terraform task), ssh into that instance and install Ansible in it. After that, you have to install a httpd web server in the managed node. You do not have separate managed nodes. So use your ansible workstation itself as the managed node by adding the below line in your host inventory file: localhost ansible_connection = local

Docker & Kubernetes Task:

Problem Statement:

1. Build a docker image to use the python api and push it to the DockerHub. Create a pod and nodeport service with that Docker image.
2. Two Tier Architecture: Deploy Nextcloud with PostgreSQL on Kubernetes

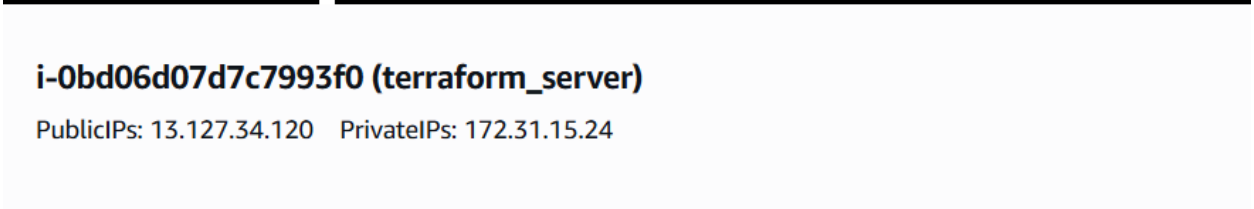
Task 1: Terraform task

Launch a terraform server:

Step 1: Create an instance

- Manually Launch a `t2.micro` instance with OS version as `Ubuntu 22.04 LTS` in ap-south-1 region.
- Enable `SSH`, `HTTP`, `HTTPS`
- Create a keypair
- Configure Storage: `10 GiB`
- Once Launched, Connect to the Instance using `MobaXterm` or `Putty` or `EC2 Instance Connect` with username "ubuntu"
- Set the hostname to terraform server by running the commands:
  sudo hostnamectl set-hostname terraform
  bash

```
ubuntu@ip-172-31-15-24:~$ sudo hostnamectl set-hostname terraform
ubuntu@ip-172-31-15-24:~$ bash
ubuntu@terraform:~$
```



Step 2: Install terraform on the instance

- Run the commands:
  sudo apt update -y
  sudo apt install -y wget unzip
- Install terraform.zip file from hashicorp using the commands:
  TERRAFORM_VERSION="1.8.2"

  wget https://releases.hashicorp.com/terraform/${TERRAFORM_VERSION}/terraform_${TERRAFORM_VERSION}_linux_amd64.zip
- Unzip the file
  unzip terraform_${TERRAFORM_VERSION}_linux_amd64.zip
  rm terraform_1.8.2_linux_amd64.zip

```
ubuntu@terraform:~$ ls
main.tf  outputs.tf  terraform_1.8.2_linux_amd64.zip  variables.tf
ubuntu@terraform:~$ unzip terraform_${TERRAFORM_VERSION}_linux_amd64.zip
Archive:  terraform_1.8.2_linux_amd64.zip
  inflating: LICENSE.txt
  inflating: terraform
ubuntu@terraform:~$ rm terraform_1.8.2_linux_amd64.zip
ubuntu@terraform:~$ ls
LICENSE.txt  main.tf  outputs.tf  terraform  variables.tf
ubuntu@terraform:~$
```

i-0bd06d07d7c7993f0 (terraform_server)

PublicIPs: 13.127.34.120 PrivateIPs: 172.31.15.24

- Move the file to local binary
  sudo mv terraform /usr/local/bin/
- Verify the installation
  terraform -v

Launch an instance managed by terraform to be used as ansible server

Step 1: Generate a keypair using ssh keygen

Run this command on the Terraform server:

ssh-keygen -t rsa -b 4096 -f ~/.ssh/terraform-key -N ""

This will create:

- ~/.ssh/terraform-key (private key)
- ~/.ssh/terraform-key.pub (public key)

```
ubuntu@terraform:~$ ssh-keygen -t rsa -b 4096 -f ~/.ssh/terraform-key -N ""
Generating public/private rsa key pair.
Your identification has been saved in /home/ubuntu/.ssh/terraform-key
Your public key has been saved in /home/ubuntu/.ssh/terraform-key.pub
The key fingerprint is:
SHA256:Zd1IYO+KJ8jx6hkO9nHzQxi/oAobwxDJywmURpn3x9k ubuntu@terraform
The key's randomart image is:
+---[RSA 4096]----+
|.o+       o..    |
|o* .     . + o   |
|=.. . . o o + .  |
|o.o  . +.E .     |
|.+    o S+  .    |
| o   . +o.o.     |
|  = o =.*oo.     |
|   * +.* =o      |
|  . .o*   ..     |
+----[SHA256]-----+
ubuntu@terraform:~$
```

Step 2: Create a variables.tf file

Run the command

vi variables.tf

Add the script to the file and save it

```
variable "region" {
  default = "ap-south-1"
}

variable "instance_type" {
  default = "t2.micro"
}

variable "key_name" {
  default = "terraform-key"
}

variable "public_key_path" {
  default = "~/.ssh/terraform-key.pub"
}

variable "ami_id" {
  default = "ami-0f918f7e67a3323f0"
}
```

Step 3: Create a main.tf file

Run the command

vi main.tf

Add the below script

```
provider "aws" {
  region = var.region
}

# Read public key
resource "tls_private_key" "generated" {
  algorithm = "RSA"
  rsa_bits  = 4096
}

resource "aws_key_pair" "deployer_key" {
  key_name   = var.key_name
```

```
  public_key = file(var.public_key_path)
}

resource "aws_security_group" "allow_ssh" {
  name        = "allow_ssh"
  description = "Allow SSH inbound traffic"

  ingress {
    description = "SSH"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
 ingress {
    description = "Allow HTTP"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_instance" "ec2_instance" {
  ami           = var.ami_id
  instance_type = var.instance_type
  key_name      = aws_key_pair.deployer_key.key_name
  security_groups = [aws_security_group.allow_ssh.name]

  tags = {
    Name = "Ansible-EC2"
  }
}
```

Step 4: Create an Output file

Run the command

vi outputs.tf

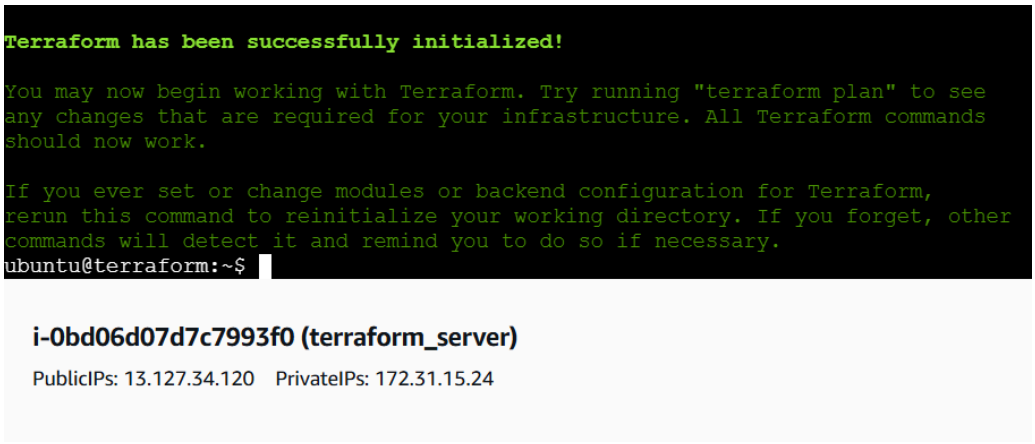
Add the lines to the file

```
output "instance_public_ip" {
  value = aws_instance.ec2_instance.public_ip
}

output "ssh_command" {
  value = "ssh -i ~/.ssh/terraform-key ubuntu@${aws_instance.ec2_instance.public_ip}"
}
```

Step 5: Run the terraform commands

Run the following commands one after the other

Initialize: terraform init



Validate: terraform validate

Plan: terraform plan

```
Plan: 4 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + instance_public_ip = (known after apply)
  + ssh_command        = (known after apply)


Note: You didn't use the -out option to save this plan, so Te
ubuntu@terraform:~$
```

i-0bd06d07d7c7993f0 (terraform_server)

PublicIPs: 13.127.34.120 PrivateIPs: 172.31.15.24

Apply changes: terraform apply

```
tls_private_key.generated: Creating...
aws_key_pair.deployer_key: Creating...
aws_security_group.allow_ssh: Creating...
aws_key_pair.deployer_key: Creation complete after 0s [id=terraform-key]
aws_security_group.allow_ssh: Creation complete after 2s [id=sg-014c4e8077d47c1bd]
aws_instance.ec2_instance: Creating...
tls_private_key.generated: Creation complete after 4s [id=3dcec4494e6d2d0acd70f4b43383480af1f4d999]
aws_instance.ec2_instance: Still creating... [00m10s elapsed]
aws_instance.ec2_instance: Creation complete after 12s [id=i-0e226ad65ebf6033d]

Apply complete! Resources: 4 added, 0 changed, 0 destroyed.

Outputs:

instance_public_ip = "35.154.119.39"
ssh_command = "ssh -i ~/.ssh/terraform-key ubuntu@35.154.119.39"
ubuntu@terraform:~$
```

i-0bd06d07d7c7993f0 (terraform_server)

PublicIPs: 13.127.34.120 PrivateIPs: 172.31.15.24

less than a minute ago

Find Instance by attribute or tag (case-sensitive) | All states

Name	Instance ID	Instance state	Instance type	Status check	Alarm status
Ansible-EC2	i-0e226ad65ebf6033d	Running	t2.micro	2/2 checks passec	View alarms +
terraform_server	i-0bd06d07d7c7993f0	Running	t2.micro	2/2 checks passec	View alarms +

i-0e226ad65ebf6033d (Ansible-EC2)

Details | Status and alarms | Monitoring | Security | Networking | Storage | Tags

▼ Instance summary Info

Instance ID	Public IPv4 address	Private IPv4 addresses
i-0e226ad65ebf6033d	35.154.119.39 | open address	172.31.6.217

Step 6: SSH into the instance:

After applying, SSH into the instance using the command

ssh -i ~/.ssh/terraform-key ubuntu@35.154.119.39

```
ubuntu@ip-172-31-6-217:~$ sudo hostnamectl set-hostname ansible
ubuntu@ip-172-31-6-217:~$ bash
ubuntu@ansible:~$
```

i-0bd06d07d7c7993f0 (terraform_server)

PublicIPs: 13.127.34.120 PrivateIPs: 172.31.15.24

Task 2: Ansible task

Step 1: SSH into Ansible server
Run the command to ssh

ssh -i ~/.ssh/terraform-key ubuntu@35.154.119.39

Step 2: Install Ansible on the server by running the commands

sudo apt update
sudo apt install ansible -y

```
ubuntu@ansible:~$ ansible --version
ansible [core 2.16.3]
  config file = None
  configured module search path = ['/home/ubuntu/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/ubuntu/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.12.3 (main, Feb  4 2025, 14:48:35) [GCC 13.3.0] (/usr/bin/python3)
  jinja version = 3.1.2
  libyaml = True
ubuntu@ansible:~$
```

Step 3: Setup inventory ini file:

- Create /etc/ansible directory
  sudo mkdir -p /etc/ansible
- Create an ini file
  sudo nano /etc/ansible/hosts
- Add the line
  localhost ansible_connection=local

Step 4: Test Ansible Connection to Localhost

ansible localhost -m ping

```
ubuntu@ansible:~$ sudo mkdir -p /etc/ansible
ubuntu@ansible:~$ sudo nano /etc/ansible/hosts
ubuntu@ansible:~$ ansible localhost -m ping
localhost | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
ubuntu@ansible:~$
```

Step 5: Create a playbook `install_httpd.yml`:

Run the command

sudo vi install_httpd.yml

Add the script below

```
- name: Install HTTPD on localhost
  hosts: localhost
  tasks:
    - name: Install Apache
      apt:
        name: apache2
        state: present
        update_cache: yes
```

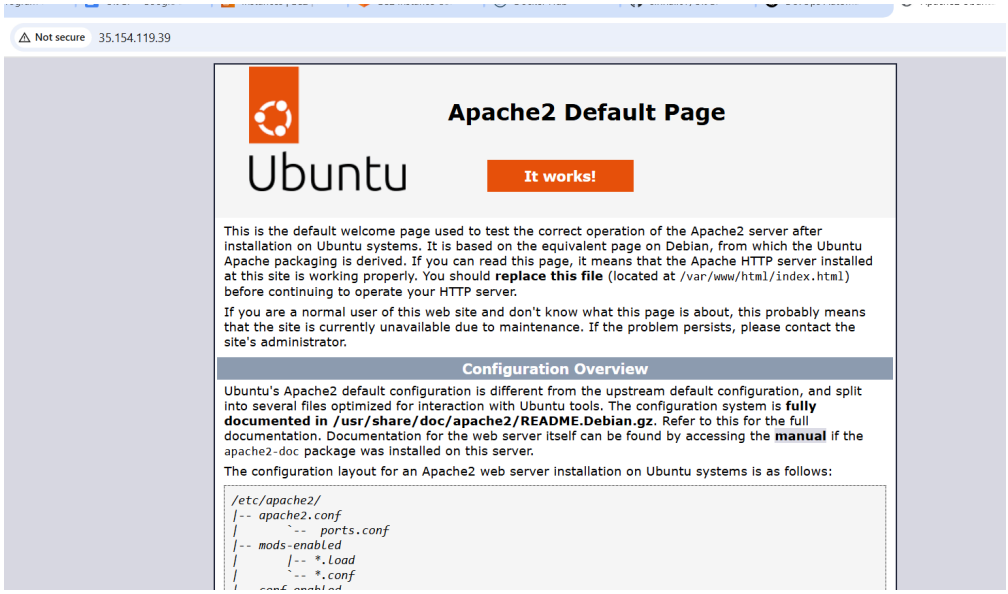
Step 6: Run the playbook:

ansible-playbook install_httpd.yml

Step 7: Check if apache is installed

http://<publicipoftheinstance>

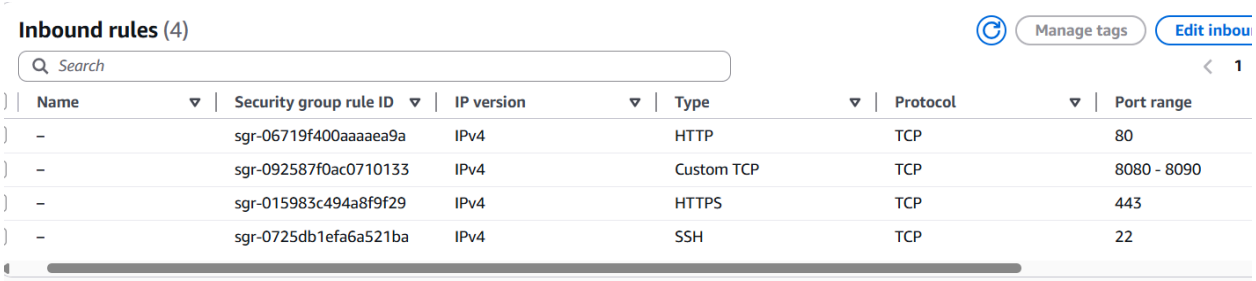
http://35.154.119.39/



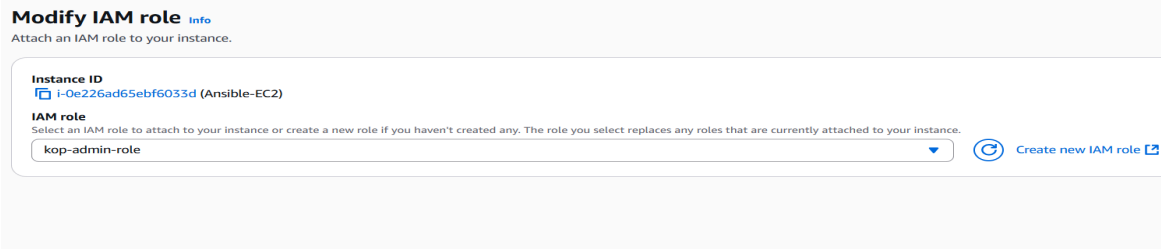
Task 3: Docker & Kubernetes Task

Build a docker image to use the python api and push it to the DockerHub. Create a pod and nodeport service with that Docker image.

Step 1: Edit the security group of the ansible server such that the port range is set to allow traffic from 8080-8090.



Step 2: Modify the IAM role to add Administrator full access permission to the instance.



Step 3: Create a kops cluster

Create a file named kops.sh by running the command

vi kops.sh

Add the below script to the file and save it.

```
#!/bin/bash

echo "Let's get started with Kubernetes cluster creation using KOPS!"
echo "Enter your name:"
read username
lower_username=$(echo -e $username | sed 's/ //g' | tr '[:upper:]' '[:lower:]')
date_now=$(date "+%F-%H-%m")
clname=$(echo $lower_username-$date_now.k8s.local)
echo "Your Kubernetes cluster name will be $clname"

TOKEN=$(curl -X PUT "http://169.254.169.254/latest/api/token" -H
"X-aws-ec2-metadata-token-ttl-seconds: 21600")

az=$(curl -H "X-aws-ec2-metadata-token: $TOKEN"
http://169.254.169.254/latest/meta-data/placement/availability-zone)
region=$(curl -H "X-aws-ec2-metadata-token: $TOKEN"
http://169.254.169.254/latest/meta-data/placement/region)

sudo sed -i "/$nrconf{restart}/d" /etc/needrestart/needrestart.conf
echo "\$nrconf{restart} = 'a';" | sudo tee -a /etc/needrestart/needrestart.conf
export DEBIAN_FRONTEND=noninteractive
export NEEDRESTART_MODE=a
```

```
sudo apt update -y
sudo apt install nano curl python3-pip -y
sudo snap install aws-cli --classic

# Install kubectl
curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
chmod +x ./kubectl
sudo mv ./kubectl /usr/local/bin/kubectl

# Install kops
curl -LO "https://github.com/kubernetes/kops/releases/download/$(curl -s
https://api.github.com/repos/kubernetes/kops/releases/latest | grep tag_name | cut -d '"' -f
4)/kops-linux-amd64"
chmod +x kops-linux-amd64
sudo mv kops-linux-amd64 /usr/local/bin/kops

# Generate SSH key
ssh-keygen -t rsa -N "" -f $HOME/.ssh/id_rsa

# Create S3 bucket for kops state store
aws s3 mb s3://$clname --region $region

# Set KOPS_STATE_STORE environment variable
export KOPS_STATE_STORE=s3://$clname

# Create Kubernetes cluster
kops create cluster --node-count=2 --master-size="t2.medium" --node-size="t2.medium"
--master-volume-size=20 --node-volume-size=20 --zones $az --name $clname --ssh-public-key
~/.ssh/id_rsa.pub --yes
kops update cluster $clname --yes

# Export KOPS_STATE_STORE to bashrc
```

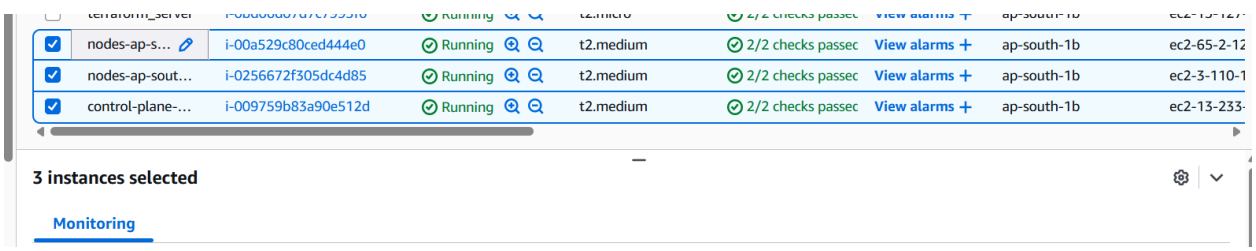
```
echo "export KOPS_STATE_STORE=s3://$clname" >> /home/ubuntu/.bashrc
source /home/ubuntu/.bashrc

# Export kubectl configuration
kops export kubecfg --admin

# Validate cluster
for (( x=0 ; x < 30 ; x++ )); do
  echo "Validating Cluster"
  if kops validate cluster > status.txt 2>/dev/null && grep -q "is ready" status.txt; then
    echo "Your Cluster is now ready!"
    break
  else
    sleep 20
    echo "x: $x"
  fi
done
```

Run the command . kops.sh to execute the file and create a kops cluster.

Once the cluster is ready run the following commands.

```
kops get cluster
kubectl get nodes
```

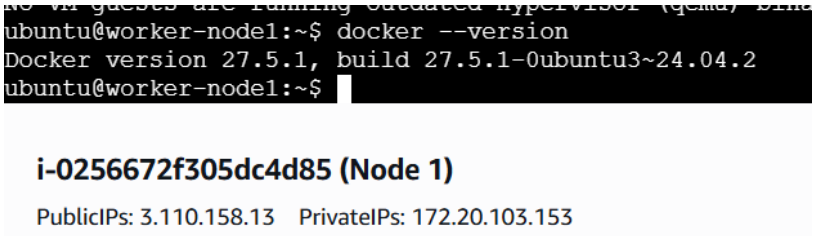
```
Validating Cluster
Your Cluster is now ready!
ubuntu@ansible:~$ kops get cluster
NAME                                    CLOUD   ZONES
sameera-2025-06-24-09-06.k8s.local      aws     ap-south-1b
ubuntu@ansible:~$ kubectl get nodes
NAME                  STATUS   ROLES           AGE     VERSION
i-009759b83a90e512d   Ready    control-plane   6m4s    v1.32.4
i-00a529c80ced444e0   Ready    node            3m36s   v1.32.4
i-0256672f305dc4d85   Ready    node            3m17s   v1.32.4
ubuntu@ansible:~$
```



Step 4: Install docker on the worker node

SSH into the worker node and install docker on it.

- Update the package index
  sudo apt update
- Install docker.io
  sudo apt install -y docker.io
- Enable and start the Docker service
  sudo systemctl enable docker
  sudo systemctl start docker
- Verify Docker installation
  docker --version



Step 5: Create the Python API Project

- Create a directory named python-api

  mkdir python-api && cd python-api

- Create a simple Flask API.
  vi app.py
- Add the script to the file

```
        from flask import Flask
app = Flask(__name__)

@app.route("/")
def hello():
    return "Hello from Kubernetes Python API!"

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000)
```

Step 6: Create a Dockerfile

- Run the command

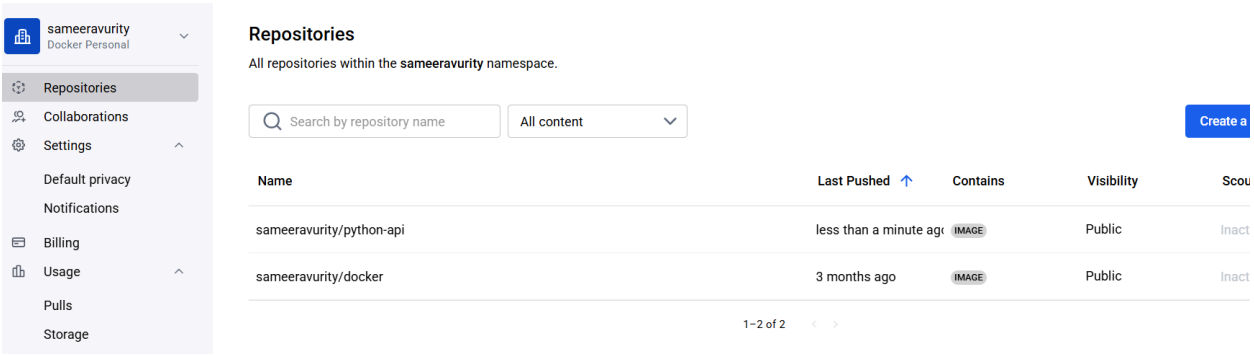
vi Dockerfile

- Add the script in the file

```
FROM python:3.9-slim

WORKDIR /app
COPY app.py .

RUN pip install flask

EXPOSE 5000
CMD ["python", "app.py"]
```

Step 7: Build & Push Docker Image to DockerHub

- Run the command
  docker login
- Build the Docker image
  docker build -t sameeravurity/python-api .
- Push it to DockerHub
  docker push sameeravurity/python-api



Step 8: Create Kubernetes YAML Files

- Create a deployment file

  vi python-api-deployment.yaml
- Add the script and save the file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: python-api
```

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: python-api
  template:
    metadata:
      labels:
        app: python-api
    spec:
      containers:
        - name: python-api
          image: sameeravurity/python-api
          ports:
            - containerPort: 5000
```

- Create a Service YAML file.

  vi python-api-service.yaml
- Add the below script and save the file.

```
apiVersion: v1
kind: Service
metadata:
  name: python-api-service
spec:
  selector:
    app: python-api
  type: NodePort
  ports:
    - port: 80
      targetPort: 5000
      nodePort: 30001
```

Step 9: Deploy to KOPS Kubernetes Cluster

Apply the deployment and service by running the commands

```
kubectl apply -f python-api-deployment.yaml
kubectl apply -f python-api-service.yaml
```

Verify the creation by running the commands

```
kubectl get pods
kubectl get svc
```

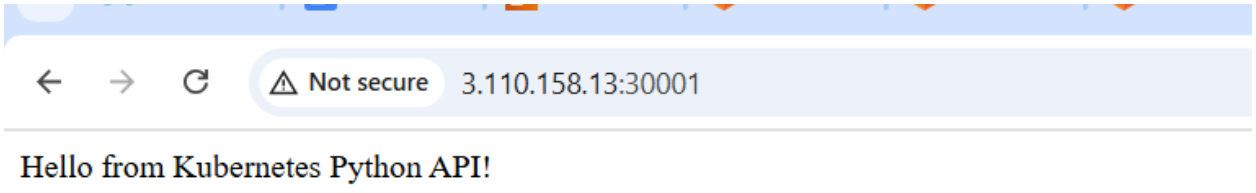
```
kubectl get nodes -o wide
```

```
ubuntu@controlnode:~$ vi python-api-deployment.yaml
ubuntu@controlnode:~$ vi python-api-service.yaml
ubuntu@controlnode:~$ kubectl apply -f python-api-deployment.yaml
deployment.apps/python-api created
ubuntu@controlnode:~$ kubectl get pods
NAME                          READY   STATUS    RESTARTS   AGE
python-api-68dfd477cc-nlgn5   1/1     Running   0          18s
ubuntu@controlnode:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   100.64.0.1   <none>        443/TCP   125m
ubuntu@controlnode:~$ kubectl get nodes -o wide
NAME                  STATUS   ROLES           AGE    VERSION   INTERNAL-IP      EXTERNAL-IP      OS-IMAGE             KERNEL-VERSION   CONTAINER-RUNTIME
i-009759b83a90e512d   Ready    control-plane   125m   v1.32.4   172.20.129.53    13.233.102.147   Ubuntu 24.04.2 LTS   6.8.0-1029-aws   containerd://1.7.25
i-00a529c80ced444e0   Ready    node            122m   v1.32.4   172.20.92.111    65.2.127.33      Ubuntu 24.04.2 LTS   6.8.0-1029-aws   containerd://1.7.25
i-0256672f305dc4d85   Ready    node            122m   v1.32.4   172.20.103.153   3.110.158.13     Ubuntu 24.04.2 LTS   6.8.0-1029-aws   containerd://1.7.27
ubuntu@controlnode:~$
```

i-009759b83a90e512d (control-plane-ap-south-1b.masters.sameera-2025-06-24-09-06.k8s.local)

PublicIPs: 13.233.102.147 PrivateIPs: 172.20.129.53

Open the app in your browser and check with
`http://<worker-node-public-ip>:30001`



Two Tier Architecture: Deploy Nextcloud with PostgreSQL on Kubernetes

```
ubuntu@controlnode:~$ kubectl run pod1 --image nginx --port 80
pod/pod1 created
ubuntu@controlnode:~$ kubectl run pod2 --image nginx --port 80
pod/pod2 created
ubuntu@controlnode:~$ kubectl get pods
NAME                          READY   STATUS              RESTARTS   AGE
pod1                          1/1     Running             0          22s
pod2                          0/1     ContainerCreating   0          6s
python-api-68dfd477cc-nlgn5   1/1     Running             0          17m
ubuntu@controlnode:~$
```

i-009759b83a90e512d (control-plane-ap-south-1b.masters.sameera-2025-06-24-09-06.k8s.local)

PublicIPs: 13.233.102.147 PrivateIPs: 172.20.129.53

Deploying Nextcloud with PostgreSQL in a two-tier architecture on Kubernetes means separating:

- Tier 1: The Application Layer (Nextcloud)
- Tier 2: The Database Layer (PostgreSQL)

Step 1: Create Kubernetes Namespace

```
kubectl create namespace nextcloud
```

Step 2: Create Persistent Volumes (PV/PVC)

- Create a file nextcloud-pvc.yaml

  ```
  vi nextcloud-pvc.yaml
  ```
- Add the script

  ```
  apiVersion: v1
  kind: PersistentVolumeClaim
  metadata:
    name: nextcloud-pvc
    namespace: nextcloud
  spec:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 5Gi
  ```
- Create a file postgres-pvc.yaml

  ```
  vi postgres-pvc.yaml
  ```
- Add the script

  ```
  apiVersion: v1
  kind: PersistentVolumeClaim
  metadata:
    name: postgres-pvc
    namespace: nextcloud
  spec:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 5Gi
  ```

Step 3: PostgreSQL Deployment and Service

- Create a file Postgres-deployment.yaml

  ```
  vi Postgres-deployment.yaml
  ```
- Add the script

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres
  labels:
    app: postgres
spec:
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
      - name: postgres
        image: postgres:15
        ports:
        - containerPort: 5432
        env:
        - name: POSTGRES_DB
          value: nextcloud
        - name: POSTGRES_USER
          value: nextcloud
        - name: POSTGRES_PASSWORD
          valueFrom:
            secretKeyRef:
              name: postgres-secret
              key: POSTGRES_PASSWORD
        volumeMounts:
        - mountPath: /var/lib/postgresql/data
          name: postgres-storage
      volumes:
      - name: postgres-storage
        persistentVolumeClaim:
          claimName: postgres-pvc
```

- Create a postgres-service.yaml file

```
apiVersion: v1
```

```
kind: Service
metadata:
  name: postgres
  namespace: nextcloud
spec:
  selector:
    app: postgres
  ports:
    - port: 5432
      targetPort: 5432
```

Step 4: Nextcloud Deployment and Service

- Create a nextcloud-deployment.yaml file

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nextcloud
  namespace: nextcloud
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nextcloud
  template:
    metadata:
      labels:
        app: nextcloud
    spec:
      containers:
      - name: nextcloud
        image: nextcloud
        ports:
        - containerPort: 80
        env:
        - name: POSTGRES_HOST
          value: postgres
        - name: POSTGRES_DB
          value: nextcloud
        - name: POSTGRES_USER
```

```
        value: ncuser
      - name: POSTGRES_PASSWORD
        value: ncpass
      volumeMounts:
      - mountPath: /var/www/html
        name: nextcloud-storage
    volumes:
    - name: nextcloud-storage
      persistentVolumeClaim:
        claimName: nextcloud-pvc
```

- Create a nextcloud-service.yaml file

```
apiVersion: v1
kind: Service
metadata:
  name: nextcloud
  namespace: nextcloud
spec:
  selector:
    app: nextcloud
  ports:
    - port: 80
      targetPort: 80
  type: NodePort
```

Step 5: Apply all YAML files

Run the commands

- kubectl apply -f postgres-pvc.yaml
- kubectl apply -f nextcloud-pvc.yaml
- kubectl apply -f postgres-deployment.yaml
- kubectl apply -f postgres-service.yaml
- kubectl apply -f nextcloud-deployment.yaml
- kubectl apply -f nextcloud-service.yaml

Step 6. Access Nextcloud

Run the command to get the service:

```
kubectl get svc -n nextcloud
```

```
ubuntu@controlnode:~$ kubectl get pods -n nextcloud
NAME                         READY   STATUS             RESTARTS      AGE
nextcloud-789db79655-nddwq   1/1     Running            0             15m
postgres-5648967f9b-jrhkq    0/1     CrashLoopBackOff   8 (63s ago)   17m
ubuntu@controlnode:~$ kubectl get pods -n nextcloud -o wide
NAME                         READY   STATUS             RESTARTS      AGE   IP             NODE                  NOMINATED NODE   READINESS GATES
nextcloud-789db79655-nddwq   1/1     Running            0             15m   100.96.4.69    i-0b3c9714c50330b9e   <none>           <none>
postgres-5648967f9b-jrhkq    0/1     CrashLoopBackOff   8 (72s ago)   17m   100.96.4.209   i-0b3c9714c50330b9e   <none>           <none>
ubuntu@controlnode:~$ kubectl get svc -n nextcloud
NAME        TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
nextcloud   NodePort    100.69.86.95    <none>        80:30080/TCP   16m
postgres    ClusterIP   100.70.85.21    <none>        5432/TCP       18m
ubuntu@controlnode:~$
```

i-080d3275edca43d09 (control-plane-ap-south-1b.masters.sameera-2025-06-24-09-06.k8s.local)

PublicIPs: 15.206.164.122 PrivateIPs: 172.20.159.101

Run the command to get the pods:

kubectl get pods -n nextcloud

```
ubuntu@controlnode:~$ kubectl get pods -n nextcloud
NAME                         READY   STATUS    RESTARTS   AGE
nextcloud-789db79655-vwcm2   1/1     Running   0          84m
postgres-694b89d9f9-jqmwz    1/1     Running   0          3m47s
ubuntu@controlnode:~$
```

i-080d3275edca43d09 (control-plane-ap-south-1b.masters.sameera-2025-06-24-09-06.k8s.local)

PublicIPs: 15.206.164.122 PrivateIPs: 172.20.159.101

Access the nextcloud by browsing to

http://<ip of the node><nodeport>