Documentation for FastAPI Application Deployment on AWS & Kubernetes

Overview

The objective is to develop a FastAPI Python application that counts the number of requests it responds to. The application should store the count in a database and expose an endpoint (/count) that returns the count in a JSON format.

The task includes:

Dockerizing the app.

Preparing deployment manifests for AWS and Kubernetes.

Demonstrating DevOps best practices such as security, and cost-efficiency.

This document provides the architecture, deployment details, and how the system will be deployed in both local (Minikube) and AWS environments.

1. FastAPI Python Application

1.1 Application Overview

The FastAPI Python application provides an HTTP endpoint /count that increments and returns the number of requests received. The count is stored in a database SQLite. This is how the system works:

POST /count: Increments the count each time it is called.

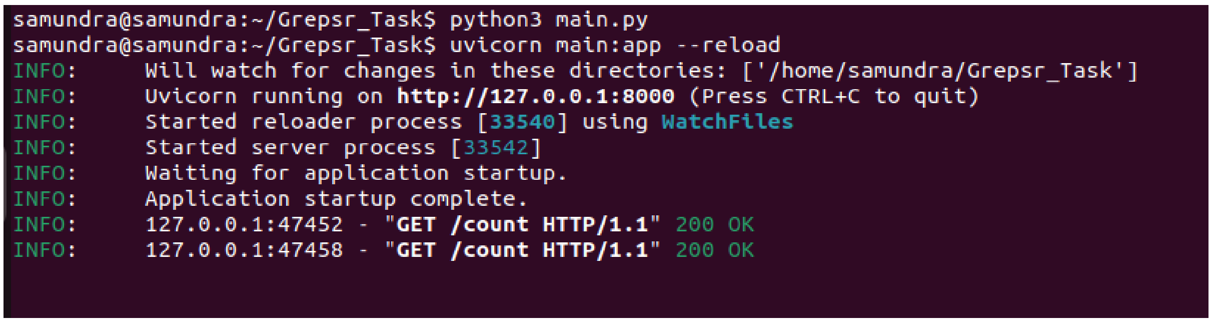
GET /count: Returns the current count in the format { "count": <int> }.

1.2 Application Code

```
from fastapi import FastAPI
from sqlalchemy import create_engine, Column, Integer
from sqlalchemy.orm import sessionmaker, declarative_base

app = FastAPI()

# Database setup
DATABASE_URL = "sqlite:///./count.db"
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()

class Counter(Base):
    __tablename__ = "counters"
    id = Column(Integer, primary_key=True, index=True)
    count = Column(Integer, default=0)
```

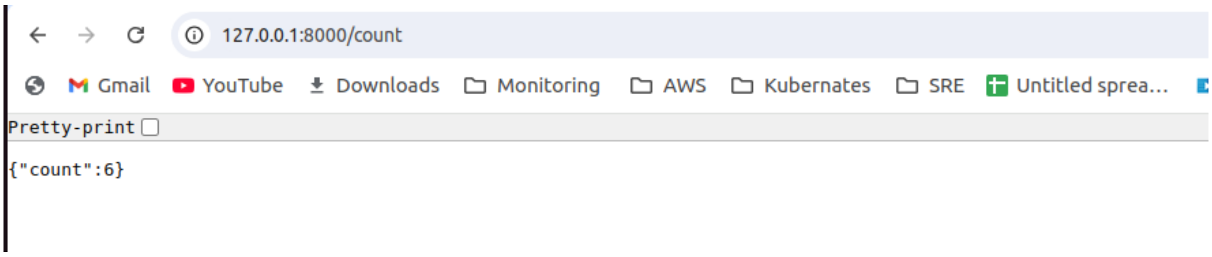
```
Base.metadata.create_all(bind=engine)

# FastAPI route
@app.get("/count")
def read_count():
    db = SessionLocal()
    counter = db.query(Counter).first()
    if not counter:
        counter = Counter(count=0)
        db.add(counter)
        db.commit()
    counter.count += 1
    db.commit()
    return {"count": counter.count}
```

1.3 The application is deployed as shown.



```
samundra@samundra:~/Grepsr_Task$ python3 main.py
samundra@samundra:~/Grepsr_Task$ uvicorn main:app --reload
INFO:     Will watch for changes in these directories: ['/home/samundra/Grepsr_Task']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [33540] using WatchFiles
INFO:     Started server process [33542]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:47452 - "GET /count HTTP/1.1" 200 OK
INFO:     127.0.0.1:47458 - "GET /count HTTP/1.1" 200 OK
```

1.4 The application is accessible as displayed.



```
{"count":6}
```

1.5 Dockerizing the Application

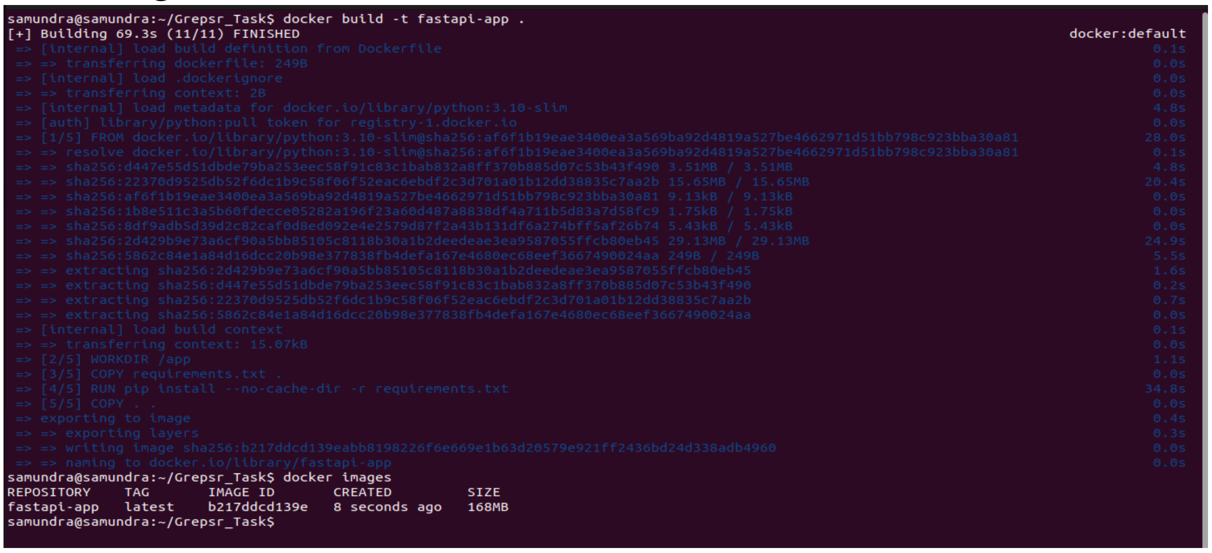
Create a Dockerfile to containerize the FastAPI app:

```
FROM python:3.10-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
EXPOSE 8000
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000", "--reload"]
```

1.7 we also have the requirements.txt as these inclused:

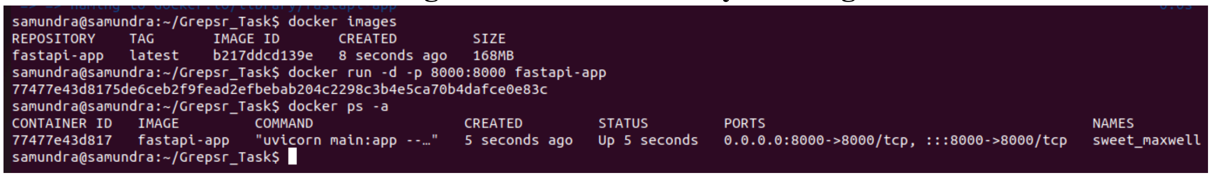
```
fastapi
uvicorn
sqlite3
```

1.8 Building the docker file as a docker image. We can further verify the status as the docker images command as shown below.



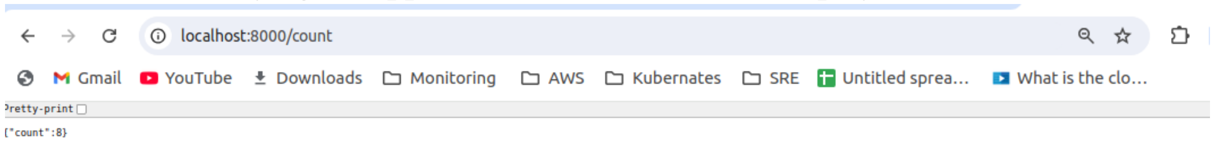
```
samundra@samundra:~/Grepsr_Task$ docker build -t fastapi-app .
[+] Building 69.3s (11/11) FINISHED                                                                     docker:default
 => [internal] load build definition from Dockerfile                                                              0.1s
 => => transferring dockerfile: 249B                                                                              0.0s
 => [internal] load .dockerignore                                                                                 0.0s
 => => transferring context: 2B                                                                                   0.0s
 => [internal] load metadata for docker.io/library/python:3.10-slim                                               4.8s
 => [auth] library/python:pull token for registry-1.docker.io                                                     0.0s
 => [1/5] FROM docker.io/library/python:3.10-slim@sha256:af6f1b19eae3400ea3a569ba92d4819a527be4662971d51bb798c923bba30a81   28.0s
 => => resolve docker.io/library/python:3.10-slim@sha256:af6f1b19eae3400ea3a569ba92d4819a527be4662971d51bb798c923bba30a81    0.1s
 => => sha256:d447e55d51dbde79ba253eec58f91c83c1bab832a8ff370b885d07c53b43f490 3.51MB / 3.51MB                    4.8s
 => => sha256:22370d9525db52f6dc1b9c58f06f52eac6ebdf2c3d701a01b12dd38835c7aa2b 15.65MB / 15.65MB                 20.4s
 => => sha256:af6f1b19eae3400ea3a569ba92d4819a527be4662971d51bb798c923bba30a81 9.13kB / 9.13kB                    0.0s
 => => sha256:1b8e511c3a5b60fdecce05282a196f23a60d487a8838df4a711b5d83a7d58fc9 1.75kB / 1.75kB                    0.0s
 => => sha256:8df9adb5d39d2c82caf0d8ed092e4e2579d87f2a43b131df6a274bff5af26b74 5.43kB / 5.43kB                    0.0s
 => => sha256:2d429b9e73a6cf90a5bb85105c8118b30a1b2deedeae3ea9587055ffcb80eb45 29.13MB / 29.13MB                 24.9s
 => => sha256:5862c84e1a84d16dcc20b98e377838fb4defa167e4680ec68eef3667490024aa 249B / 249B                        5.5s
 => => extracting sha256:2d429b9e73a6cf90a5bb85105c8118b30a1b2deedeae3ea9587055ffcb80eb45                         1.6s
 => => extracting sha256:d447e55d51dbde79ba253eec58f91c83c1bab832a8ff370b885d07c53b43f490                         0.2s
 => => extracting sha256:22370d9525db52f6dc1b9c58f06f52eac6ebdf2c3d701a01b12dd38835c7aa2b                         0.7s
 => => extracting sha256:5862c84e1a84d16dcc20b98e377838fb4defa167e4680ec68eef3667490024aa                         0.0s
 => [internal] load build context                                                                                 0.1s
 => => transferring context: 15.07kB                                                                              0.0s
 => [2/5] WORKDIR /app                                                                                            1.1s
 => [3/5] COPY requirements.txt .                                                                                 0.0s
 => [4/5] RUN pip install --no-cache-dir -r requirements.txt                                                     34.8s
 => [5/5] COPY . .                                                                                                0.0s
 => exporting to image                                                                                            0.4s
 => => exporting layers                                                                                           0.3s
 => => writing image sha256:b217ddcd139eabb8198226f6e669e1b63d20579e921ff2436bd24d338adb4960                      0.0s
 => => naming to docker.io/library/fastapi-app                                                                    0.0s
samundra@samundra:~/Grepsr_Task$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
fastapi-app   latest    b217ddcd139e   8 seconds ago   168MB
samundra@samundra:~/Grepsr_Task$
```

1.9 we then run the docker image and it is successfully running.



```
samundra@samundra:~/Grepsr_Task$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED         SIZE
fastapi-app   latest    b217ddcd139e   8 seconds ago   168MB
samundra@samundra:~/Grepsr_Task$ docker run -d -p 8000:8000 fastapi-app
77477e43d8175de6ceb2f9fead2efbebab204c2298c3b4e5ca70b4dafce0e83c
samundra@samundra:~/Grepsr_Task$ docker ps -a
CONTAINER ID   IMAGE         COMMAND                  CREATED         STATUS         PORTS                                       NAMES
77477e43d817   fastapi-app   "uvicorn main:app --…"   5 seconds ago   Up 5 seconds   0.0.0.0:8000->8000/tcp, :::8000->8000/tcp   sweet_maxwell
samundra@samundra:~/Grepsr_Task$ 
```

1.10 Further verifying the application. It is accessible as displayed.



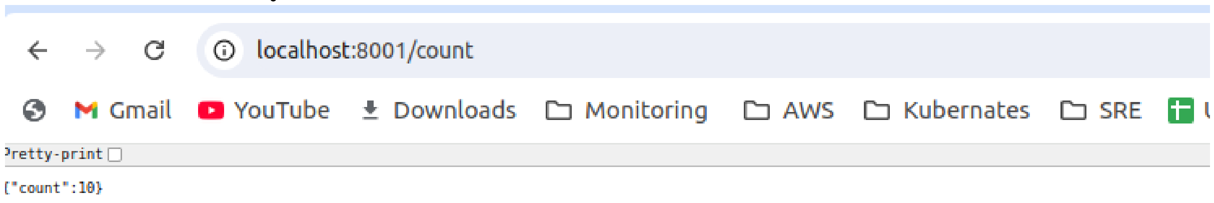
```
{"count":8}
```

1.11 Local Testing with Docker Compose

Create a docker-compose.yaml to run the app locally for testing:

```yaml
version: '3.9'

services:
  fastapi-app:
    build: .
    container_name: fastapi-app
    ports:
      - "8001:8000"
    volumes:
      - ./count.db:/app/count.db
    restart: always
```

1.12 It is successfully accessible.



```
{"count":10}
```

2. AWS Services and Architecture Diagram

To deploy the FastAPI application, we will require the following AWS services:

2.1 Amazon Route 53:

Provides domain name resolution to the application.

2.2 Elastic Load Balancer (ELB):

Distributes user requests across multiple nodes in the EKS cluster.

2.3 Amazon Elastic Kubernetes Service (EC2):

Hosts specific application components or custom workloads, ensuring flexibility for varied use cases.

2.4 Amazon Relational Database Service (RDS):

Stores the request count in a managed database.

2.5 Amazon S3:

Used for storing static assets or application logs.

2.6 AWS IAM:

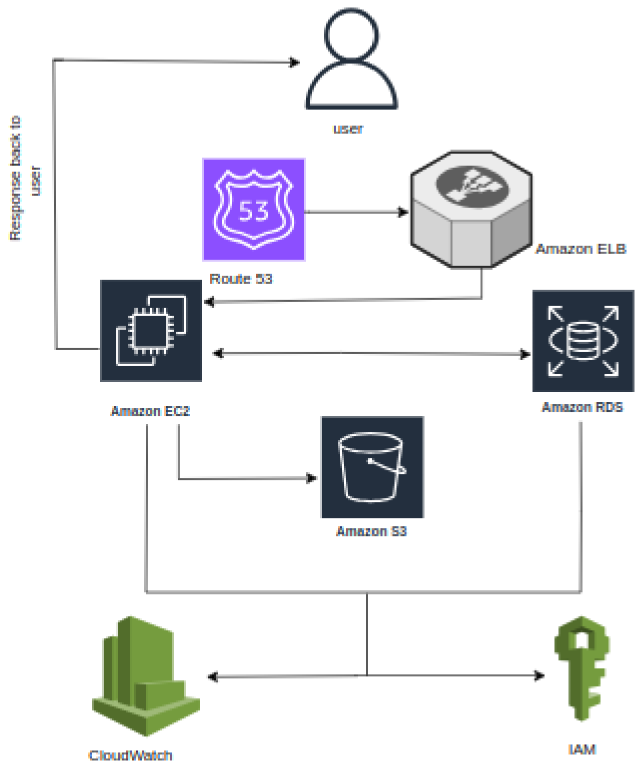
Provides role-based access control for EKS, RDS, and other AWS services.

2.7 Amazon CloudWatch:

Monitors application logs and metrics.

3. Architecture Explanation

3.1 Architecture Workflow:



The diagram represents an optimized AWS architecture using managed services. Amazon Route 53 handles DNS routing, Amazon EC2 supports custom workloads, and Amazon RDS offers fully managed database services, reducing operational overhead. Amazon S3 stores static assets cost-effectively. CloudWatch provides monitoring, and IAM enforces secure access controls.

Cost Optimization

Managed services eliminate overprovisioning and unnecessary operational costs with pay-as-you-go pricing. Services like S3 and CloudFront minimize storage and transfer expenses through caching and scalability.

Security

IAM ensures strict access controls and built-in features like encryption and automated updates enhance data and application security. This architecture reduces costs, improves performance, and strengthens reliability.

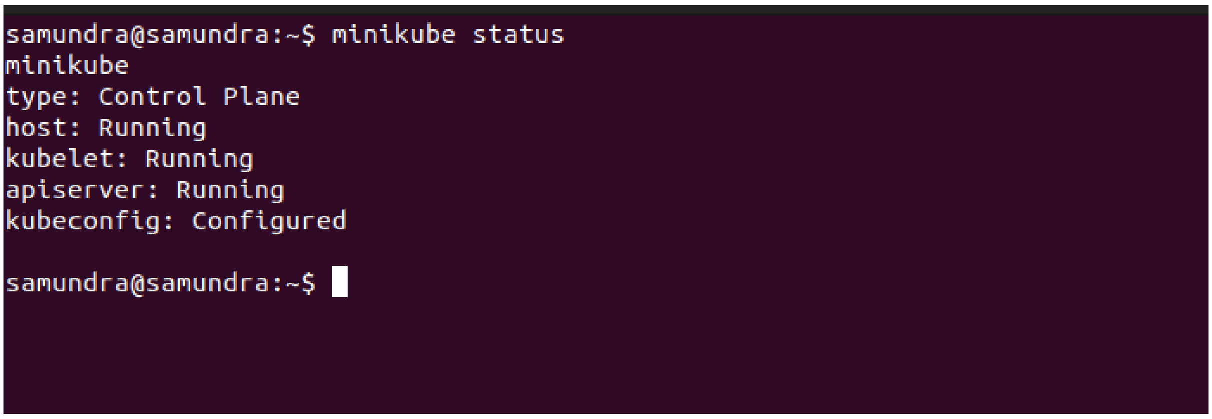
Monitoring

CloudWatch enables real-time performance tracking and operational insights.

4. Kubernetes Deployment

4.1 Kubernetes Deployment Manifests

Firstly, we have installed the minikube to run the single Kubernetes cluster within the local machine.



```
samundra@samundra:~$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

samundra@samundra:~$ 
```

To deploy the FastAPI application on Kubernetes, we will create the following Kubernetes resources:

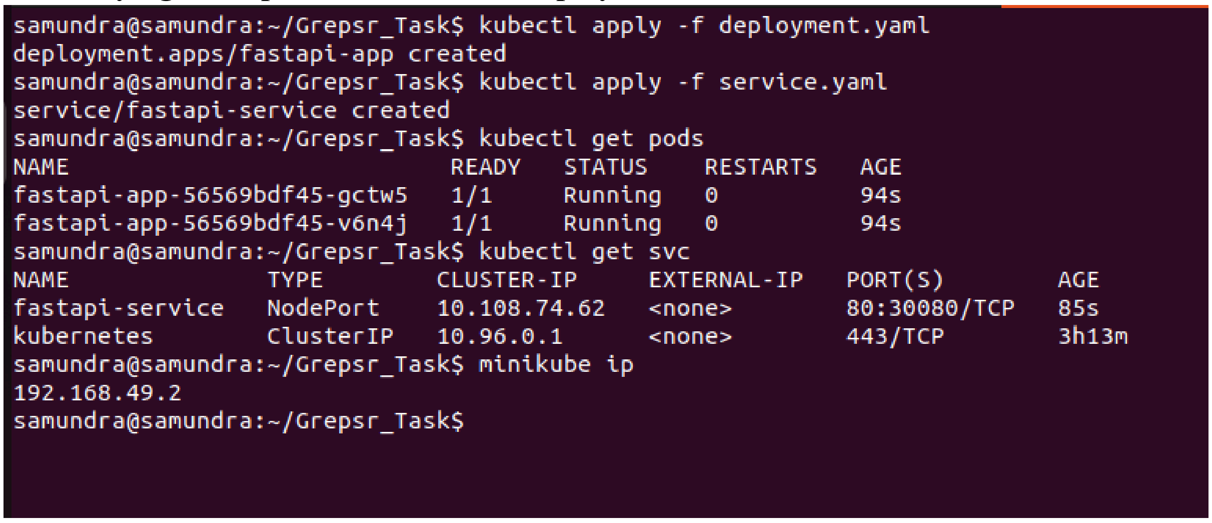
Deployment YAML: Defines the application pods.

Service YAML: Exposes the FastAPI application internally within the cluster.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: fastapi-app
spec:
  replicas: 2
  selector:
    matchLabels:
      app: fastapi-app
  template:
    metadata:
      labels:
        app: fastapi-app
    spec:
      containers:
      - name: fastapi
        image: samundra77/fastapi-app:latest
        ports:
        - containerPort: 8000
```

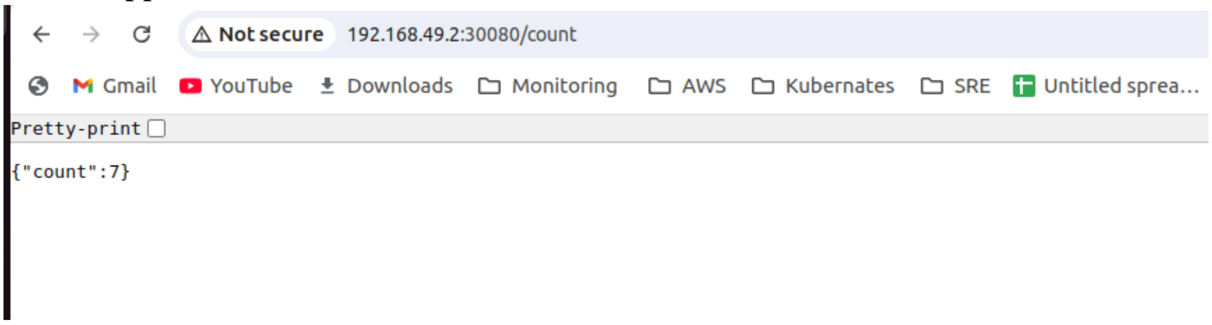
```yaml
apiVersion: v1
kind: Service
metadata:
  name: fastapi-service
spec:
  selector:
    app: fastapi-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8000
      nodePort: 30080
  type: NodePort
```

4.2 Verifying if the pods and service is deployed or not.



```
samundra@samundra:~/Grepsr_Task$ kubectl apply -f deployment.yaml
deployment.apps/fastapi-app created
samundra@samundra:~/Grepsr_Task$ kubectl apply -f service.yaml
service/fastapi-service created
samundra@samundra:~/Grepsr_Task$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
fastapi-app-56569bdf45-gctw5   1/1     Running   0          94s
fastapi-app-56569bdf45-v6n4j   1/1     Running   0          94s
samundra@samundra:~/Grepsr_Task$ kubectl get svc
NAME              TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
fastapi-service   NodePort    10.108.74.62   <none>        80:30080/TCP   85s
kubernetes        ClusterIP   10.96.0.1      <none>        443/TCP        3h13m
samundra@samundra:~/Grepsr_Task$ minikube ip
192.168.49.2
samundra@samundra:~/Grepsr_Task$
```

4.3 The application is accessible.



```
{"count":7}
```

5. Helm chart

Helm chart includes

values.yaml: Contains default configuration values for the chart (image, replica count) that can be customized during deployment.

Chart.yaml: Metadata about the Helm chart (name, version, dependencies).

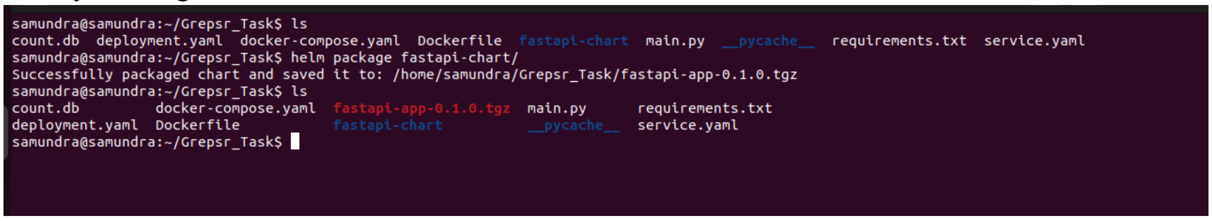
templates/: Kubernetes resource templates (like Deployments, Services) that use Go templating to generate configurations.

charts/: Stores subcharts (dependencies) that can be included in the parent chart.

In this part, I have used a helm chart. I have created a directory and followed the standards pattern for creating the helm chart. Below is the created helm file for deployment of the application.

```
samundra@samundra:~/Grepsr_Task/fastapi-chart$ ls
charts  Chart.yaml  templates  values.yaml
samundra@samundra:~/Grepsr_Task/fastapi-chart$ cat values.yaml
replicaCount: 2

image:
  repository: samundra77/fastapi-app
  tag: latest
  pullPolicy: Always

service:
  type: NodePort
  port: 80
  targetPort: 8000

ingress:
  enabled: false
samundra@samundra:~/Grepsr_Task/fastapi-chart$ cat Chart.yaml
apiVersion: v2
name: fastapi-app
description: A Helm chart for deploying FastAPI application
version: 0.1.0
appVersion: "1.0"

samundra@samundra:~/Grepsr_Task/fastapi-chart$
```

5.1 Helm Package
The helm package is an important factor as it is a standardized version of our chart and helps in easy sharing and installation.

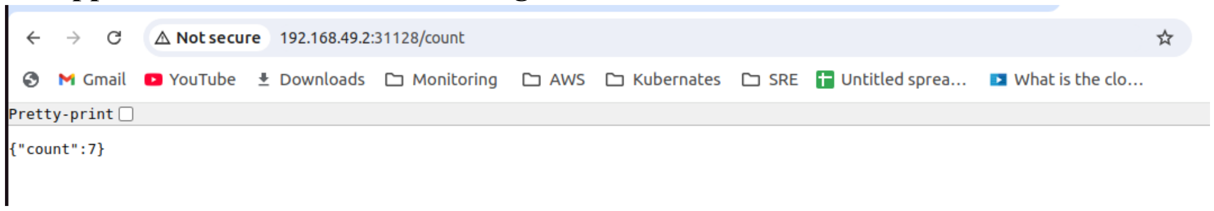


```
samundra@samundra:~/Grepsr_Task$ ls
count.db  deployment.yaml  docker-compose.yaml  Dockerfile  fastapi-chart  main.py  __pycache__  requirements.txt  service.yaml
samundra@samundra:~/Grepsr_Task$ helm package fastapi-chart/
Successfully packaged chart and saved it to: /home/samundra/Grepsr_Task/fastapi-app-0.1.0.tgz
samundra@samundra:~/Grepsr_Task$ ls
count.db         docker-compose.yaml  fastapi-app-0.1.0.tgz  main.py      requirements.txt
deployment.yaml  Dockerfile           fastapi-chart          __pycache__  service.yaml
samundra@samundra:~/Grepsr_Task$
```

5.2 Application is updated from Helm chart
The helm chart is installed in my local machine and is working fine as displayed.

```
samundra@samundra:~/Grepsr_Task$ helm upgrade --install
fastapi-app ./fastapi-chart
Release "fastapi-app" does not exist. Installing it now.
NAME: fastapi-app
LAST DEPLOYED: Tue Nov 19 16:09:59 2024
NAMESPACE: default
STATUS: deployed
REVISION: 1
NOTES:
1. Get the application URL by running these commands:
  export NODE_PORT=$(kubectl get --namespace default -o
jsonpath="{.spec.ports[0].nodePort}" services fastapi-app)
  export NODE_IP=$(kubectl get nodes --namespace default -o
jsonpath="{.items[0].status.addresses[0].address}")
  echo http://$NODE_IP:$NODE_PORT
samundra@samundra:~/Grepsr_Task$ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
fastapi-app-7f8f956f7b-6ztgs   1/1     Running   0          15s
fastapi-app-7f8f956f7b-fphcz   1/1     Running   0          15s
samundra@samundra:~/Grepsr_Task$ kubectl get svc
NAME          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
fastapi-app   NodePort    10.96.233.126   <none>
80:31128/TCP   19s
kubernetes    ClusterIP   10.96.0.1       <none>        443/TCP
27h
samundra@samundra:~/Grepsr_Task$ minikube ip
192.168.49.2
```

5.3 Application is accessible from the given above details



```
{"count":7}
```

Currently, the helm chart is installed within the local machine. To make it accessible from the public network we can use Artifact Hub and Helm Hub to package and make them publicly available.

Challenges:

Deploying the FastAPI application posed challenges with database migration (from SQLite to AWS RDS), containerization, networking (exposing the app through Kubernetes and ELB), scaling (managing replicas in Kubernetes), and managing cloud costs and security with IAM roles.

Solutions:

The solution involved transitioning to AWS RDS for the database, building a Docker image for containerization, using Kubernetes for deployment and scaling, managing the process with Helm charts, and monitoring with AWS CloudWatch while securing access with IAM roles.

Conclusions:

The deployment was successful, achieving scalability and security with Kubernetes, AWS, and Helm. Best practices like CI/CD, and secure access were implemented, and future enhancements include integrating observability tools and automating deployments with GitHub Actions or Jenkins.