DevOps exercise

DevOps Assignment

Overview

- Deploy a Kubernetes microservice app using **Helm And Docker**
- Set up **monitoring** and **tracing** (Prometheus, Grafana, OpenTelemetry, Jaeger)
- Configure **autoscaling** with **Karpenter**
- (Bonus) Automate infrastructure with **Pulumi** or your IaC tool of choice
- (Bonus) Integrate **CI/CD** using **Jenkins**
- (Bonus) Add **Istio** service mesh integration

Core Requirements

1. Microservice Deployment with Helm

Deploy a simple service on Kubernetes:

- Build a FastAPI service with this route:

```
from fastapi import FastAPI, Query
import requests

app = FastAPI()

BASE_URL = "https://api.coingecko.com/api/v3/simple/price?ids=bitcoin&vs_currencies=usd"
@app.get("/price")
def get_crypto_price(crypto: str = Query("bitcoin", description="Cryptocurrency name")):
    params = {
        "ids": crypto,
        "vs_currencies": "usd"
    }
    response = requests.get(BASE_URL, params=params)
    response.raise_for_status()
    data = response.json()

    if crypto not in data:
        return {"error": f"Price not found for '{crypto}'"}

    return {"crypto": crypto, "price_usd": data[crypto]["usd"]}

```

Create ILS and USD as currencies separate by env vars and print both.

- Create Dockerfile
- Make sure the FastAPI app is deployed via a **Helm chart**:
  - `values.yaml` should define the environment variables like `BITCOIN_API_URL`
    - Use templates after

      ```
      helm create
      ```

  - Liveness and readiness probes should be configured
    - explain difference
  - The service should be deployed in a namespace like `test-bitcoin-price`
    - Create the namespace using helm
  - Kubernetes service should be named `test-api-service`

References:

-

Helm | Docs Home

Everything you need to know about how the documentation is organized.

helm.sh

-
-

Configure Liveness, Readiness and Startup Probes

This page shows how to configure liveness, readiness and startup probes for containers. For more information about probes, see Liveness, Readiness and Startup Probes The kubelet uses liveness probes to know when to restart a container. For...

Kubernetes

-

2. Observability with Prometheus and Grafana

Deploy monitoring for your app and the cluster:

- Install **kube-prometheus-stack** via Helm
- Monitor the FastAPI app
- Add a custom metric to the app (use Prometheus client library)
- Create at least two Grafana dashboards(or use builtin):
  - One for Kubernetes components (nodes, pods, CPU/memory)
  - One for the Bitcoin price app metrics

References:

-

helm-charts/charts/kube-prometheus-stack at main · prometheus-community/helm-charts

Prometheus community Helm charts. Contribute to prometheus-community/helm-charts development by creating an account on GitHub.

GitHub

3. Distributed Tracing with OpenTelemetry and Jaeger(Bonus Observability)

- Deploy **OpenTelemetry Collector** and **Jaeger** using Helm
- Configure your FastAPI app to emit traces
- Verify the traces appear in Jaeger

References:

-

Collector

Vendor-agnostic way to receive, process and export telemetry data.

OpenTelemetry

-
-

jaeger 3.4.1 · jaegertracing/jaegertracing

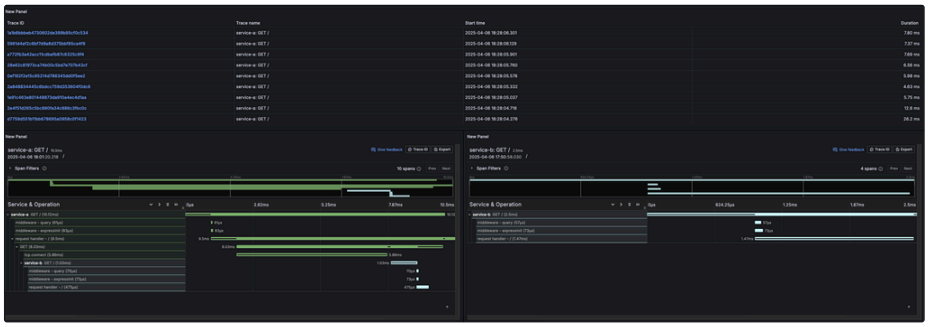
A Jaeger Helm chart for Kubernetes

artifacthub.io

ArtifactHUB

Find, install and publish Kubernetes packages

Example of traces in Grafana:



4. Autoscaling with Karpenter

- **Karpenter** using Helm
- Create a **Provisioner** that uses both Spot and On-Demand instances(Mock don't deploy)

References:

-

Documentation

Just-in-time Nodes for Any Kubernetes Cluster

karpenter.sh

Karpenter

Tasks

CI/CD with Jenkins(or preferred tool)

if using any tool that is not Jenkins please provide tool and explanation usage

- Write a `Jenkinsfile` that:
  - Checks out your repo
  - Lints Kubernetes YAMLs
  - Builds and pushes a Docker image to ECR
  - Deploys via Helm upgrade/install
  - (Optional) Sends a notification or reloads Prometheus config

References:

- Pipeline Syntax
  Jenkins – an open source automation server which enables developers around the world to reliably build, test, and deploy their software
  Pipeline Syntax
-

Pulumi IaC

- Use **Pulumi** (Python) to provision:
  - EKS cluster
  - RDS instance and RDS Aurora cluster
- Create two classes:
  - `EksCluster`
  - `RdsInfrastructure`

Note: If you don't use Pulumi, you can use another IaC tool Pulumi is a bonus.

References:

- Programming Model
  This content has moved. Redirecting to Pulumi Concepts....
  pulumi
-

Bonus #2: Istio Integration

- Install **Istio**
- Inject sidecars to your backend app
- Configure:
  - Ingress Gateway
  - Telemetry with Prometheus and Jaeger

References:

- Documentation
  Learn how to deploy, use, and operate Istio.
  Istio

Deliverables

At the end of the assignment, please submit:

- **Git Repository**
  - Helm charts
  - Kubernetes YAMLs
  - FastAPI service code
  - Grafana dashboard JSON exports
  - Jaeger screenshots
  - Prometheus screenshots (targets and metrics)
  - Pulumi (or other IaC) code if done
  - Istio configs if done
- **README.md** in the root of the repository with:

  - How to deploy the system
  - How to run the app locally (Docker Compose/Minikube if possible)
  - Required environment variables
  - How to connect to Grafana, Prometheus, and Jaeger
  - Any known limitations
- **Screenshots and Logs**
  - Grafana dashboards
  - Jaeger traces
  - Prometheus targets
  - Any relevant service logs

`Please Provide a README.md in the git.`

Good luck!