Setup & Deployment Guide (macOS)

Overview

This document provides a step-by-step guide for setting up, deploying, and verifying the ServiceExample application on a local Kubernetes cluster using Minikube.

The setup replicates a production-style deployment incorporating key DevOps and GitOps principles, including Persistent Storage (OpenEBS), Service Mesh (implicit via components), Observability (Prometheus & Grafana), and GitOps Automation (Argo CD).

All commands were executed and tested on macOS (Apple Silicon/Intel).

Why this is production-ready :

Clean containerization

Multi-stage Dockerfile → small final image, deterministic builds.
`.NET 9` with explicit port and environment-variable driven config.

Operational health

/healthz for probes, /metrics via prometheus-net.AspNetCore.
Resource requests/limits in Helm. Non-root, no privilege escalation.

Helm done right

Parameterized values (image repo/tag, connection strings).
Value.schema.json enforces correct types at install time.
Optional ServiceMonitor for Prometheus scraping.

GitOps automation

Argo CD watches this repo, automated sync, prune, and self-heal enabled.
Helm chart is the single source of truth, updating image tag triggers rollout.

Observability

kube-prometheus-stack compatible, app exposes Prometheus metrics.
Grafana-ready (dashboards in guide).

Stateful backing

Designed to run with OpenEBS Local PV (documented) for Mongo/Redis PVCs.

Supply-chain security

Cosign-signed Docker image and Helm chart, public key included for verification.
Trivy vulnerability scanning in CI.

Publication

Image on Docker Hub, Helm chart published as OCI and indexed in Artifact Hub with repo ownership metadata.

Local Environment Setup

System

OS: macOS (Apple Silicon / Intel)

Kubernetes: via Minikube

Cluster Driver: Docker

Shell: zsh / bash

Tools Installed

Homebrew, kubectl, Minikube, Helm, Argo CD CLI, Docker, Cosign (Sigstore).

Start a Local Kubernetes Cluster

minikube start --driver=docker --cpus=4 --memory=8192

Verify :

kubectl get nodes

Deploy Persistent Storage (OpenEBS)

Since the assignment required a storage solution, I used OpenEBS Local PV, which provides lightweight persistent volumes for stateful applications.

Installing :

helm repo add openebs https://openebs.github.io/charts
helm repo update
helm install openebs openebs/openebs -n openebs —create-namespace

Verify :

kubectl get pods -n openebs

kubectl get sc

Deploy Supporting Services (MongoDB, Redis, NATS)

The application relies on **MongoDB**, **Redis**, and **NATS**. These were installed using Helm charts from Bitnami.

MongoDB:

```
helm upgrade --install mongodb bitnami/mongodb -n demo \
  --create-namespace \
  --set architecture=standalone \
  --set auth.enabled=false \
  --set persistence.enabled=true \
  --set persistence.size=2Gi \
  --set persistence.storageClass=openebs-hostpath \
  --set fullnameOverride=mongodb
```

Redis:

```
helm upgrade --install redis bitnami/redis -n demo \
  --set architecture=standalone \
  --set auth.enabled=false \
  --set master.persistence.enabled=true \
  --set master.persistence.size=1Gi \
  --set master.persistence.storageClass=openebs-hostpath \
  --set fullnameOverride=redis
```

NATS:

```
helm repo add nats https://nats-io.github.io/k8s/helm/charts/
helm repo update
helm install nats nats/nats -n demo
```

Verify all pods:

```
kubectl get pods -n demo
```

Deploy the Application via Helm

The .NET API was containerized and packaged as a Helm chart located in charts/serviceexample/.

Local install for testing:

```
helm install svc ./charts/serviceexample -n demo \
  --set image.repository=sathyafire/serviceexample \
  --set image.tag=latest \
  --set env.mongo="mongodb://mongodb:27017" \
  --set env.redis="redis-master:6379" \
  --set env.nats="nats://nats:4222"
```

Verify:

```
kubectl get pods -n demo
kubectl get svc -n demo
```

Verify Application Locally

Port-forward the service:

```
kubectl -n demo port-forward svc/svc-serviceexample 9080:9080
```

Access in browser:

Swagger: http://localhost:9080/swagger/index.html

Prometheus Metrics: http://localhost:9080/metrics

Set Up Argo CD (GitOps)

Argo CD was used to automate deployment from GitHub.

Install Argo CD:

```
kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

Access Argo CD UI:

```
kubectl port-forward svc/argocd-server -n argocd 8080:443
```

Open https://localhost:8080

To Get password:

```
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath="{.data.password}" | base64 -d; echo
```

Login as:

Username: admin

Password: (output from above)

Create GitOps Application

Using the manifest in gitops/serviceexample-app.yml :

```
kubectl apply -f gitops/serviceexample-app.yaml -n argocd
```

Argo CD will:

Watch the GitHub repo

Auto-sync on new commits

Deploy Helm chart to namespace demo

Verify it using kubectl command for argocd

Deploy Observability Stack (Prometheus + Grafana)

Install kube-prometheus-stack:

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
helm install kps prometheus-community/kube-prometheus-stack -n monitoring --create-namespace
```

Access Grafana:

```
kubectl -n monitoring port-forward svc/kps-grafana 3000:80
```

Grafana: http://localhost:3000
Username: admin
Password:

```
kubectl -n monitoring get secret kps-grafana -o jsonpath="{.data.admin-password}" | base64 -d; echo
```

Add Dashboard for Application Metrics

Once logged into Grafana:

Go to Connections → Data Sources → Add Prometheus
URL: http://kps-kube-prometheus-stack-prometheus.monitoring.svc:9090

Import a default .NET Core dashboard (ID: `10915`).

Verify metrics under http_requests_total, etc.

Security and Signing

The Docker image and Helm chart were digitally signed using Cosign (Done Manually).

Generate Keys:

cosign generate-key-pair

Sign Docker image:
cosign sign --key cosign.key registry-1.docker.io/sathyafire/serviceexample:0.1.1

Sign Helm chart:

cosign sign-blob --key cosign.key serviceexample-0.1.1.tgz

Public key uploaded to repository for verification:

cosign.pub

Also you need to login to docker registry hub using username and PAT access token before signing.

Cleanup

helm uninstall kps -n monitoring

helm uninstall openebs -n openebs

kubectl delete ns demo argocd monitoring openebs

minikube delete