Overview

This project implements a client API service with MongoDB integration, deployed using Docker and Kubernetes with SSL/TLS support via cert-manager.

Prerequisites

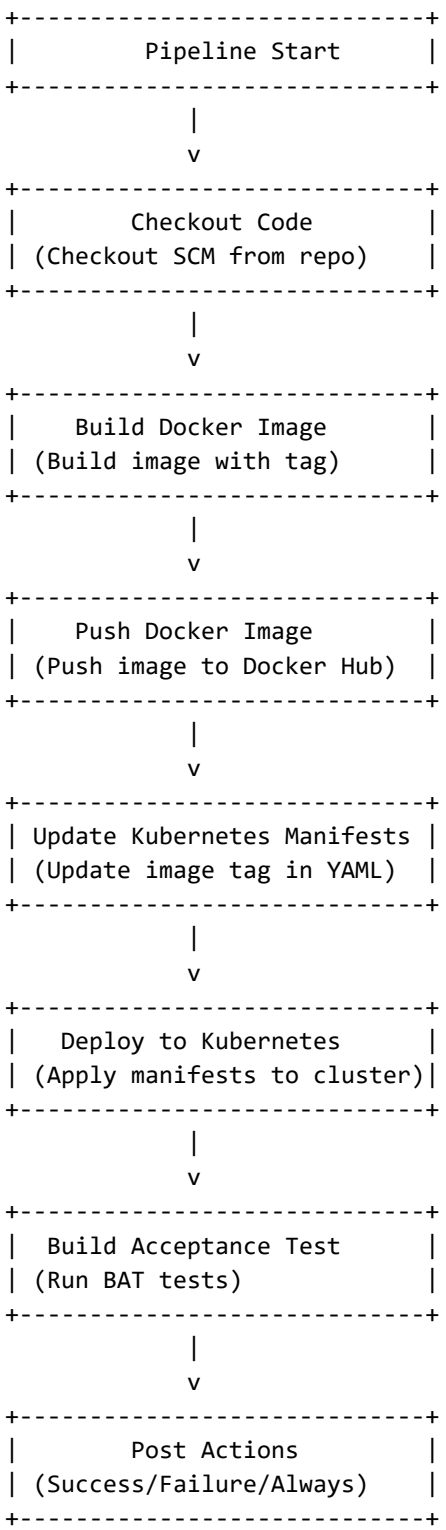
- Docker
- Kubernetes cluster
- kubectl CLI
- Python 3.10+
- Jenkins (for CI/CD)

Project Structure

```
.
├── client_api/
│   ├── Dockerfile
│   ├── main.py
│   ├── requirements.txt
│   └── debug_mongo.sh
├── manifests/
│   ├── cert-manager.yaml
│   ├── certificate.yaml
│   ├── client_api.yaml
│   ├── cluster-issuer.yaml
│   ├── ingress.yaml
│   └── mongodb.yaml
├── Jenkinsfile
└── Documentation.pdf
```

CI/CD Pipeline (Jenkinsfile)

The project uses Jenkins for continuous integration and deployment. The pipeline:



API Endpoints (Client API Code)

Root Endpoint

- **URL**: /
- **Method**: GET
- **Response**:

```
{
    "message": "Welcome to Client APIs"
}
```

Health Check

- **URL**: /health
- **Method**: GET
- **Response Success (200)**:

Deployment

Prerequisites

1. Kubernetes cluster is running
2. kubectl is configured with correct context
3. Docker registry credentials are configured

Kubernetes Deployed Components Overview

Component	File Name	Purpose	Configuration Details	Dependencies
Cert-Manager	cert-manager.yaml	Certificate Manager		None
ClusterIssuer	cluster-issuer.yaml	Certificate Issuer Config	- Type: Let's Encrypt Staging - Email: dinesh.pundkar@gmail.com - Challenge: HTTP01	Cert-Manager
Certificate	certificate.yaml	SSL/TLS Certificate	- Domains: deltacapita.com, *.deltacapita.com - Secret: dsp-capita-cert-tls	ClusterIssuer
Ingress	ingress.yaml	Traffic Routing	- Host: clients.api.deltacapita.com - TLS Enabled	Nginx Ingress, Certificate

MongoDB	mongodb.yaml	Database	- Port: 27017 - Single Standalone instance mode - No persistent storage configured	None
dsp-api	client-api.yaml	API	- Client API pods & Service	Client API

API Deployment Steps

1. Install Nginx Ingress Controller:

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/kind/deploy.yaml
```

2. Install cert-manager:

```
kubectl apply -f manifests/cert-manager.yaml
```

3. Configure SSL/TLS:

```
kubectl apply -f manifests/cluster-issuer.yaml
kubectl apply -f manifests/certificate.yaml
```

4. Deploy MongoDB and API:

```
kubectl apply -f manifests/mongodb.yaml
kubectl apply -f manifests/client_api.yaml
```

5. Configure Ingress:

```
kubectl apply -f manifests/ingress.yaml
```

Verifying Deployment

```
# Check pods status
kubectl get pods

# Check services
kubectl get svc

# Check ingress
kubectl get ingress

# Check certificates
kubectl get certificates
```

Future Improvements

Infrastructure & Scalability

- Implement MongoDB replica set for high availability instead of standalone mode
- Configure Persistent Volumes (PV/PVC) for MongoDB data persistence
- Set up Horizontal Pod Autoscaling (HPA) based on CPU/memory metrics
- Implement pod disruption budgets for zero-downtime updates

Monitoring & Observability

- Deploy Prometheus/Grafana stack for metrics collection and visualization
- Set up ELK stack (Elasticsearch, Logstash, Kibana) for centralized logging
- Create custom dashboards for business-specific KPIs
- Implement automated alerting

Security Enhancements

- Set up Vault for secrets management
- Enable MongoDB authentication and encryption at rest
- Regular security scanning using tools like Trivy

Development & CI/CD

- Set up GitOps workflow using ArgoCD or Flux
- Implement canary deployments for safer releases
- Implement feature flags for controlled rollouts

Backup & Recovery

- Implement automated MongoDB backup solution
- Implement backup for entire K8s cluster using Velero
- Implement automated backup testing
- Set up disaster recovery procedures

Compliance & Governance

- Implement audit logging for all system changes
- Add data retention and archival policies