# API Gateway Deployment Document for docker and kuberneties

**1. Introduction**

This document describes how to deploy the **API Gateway Service** (Spring Cloud Gateway + OAuth2 + Eureka + Rate Limiting) in **Docker** and **Kubernetes** environments.

**2. Prerequisites**

* **Docker** installed (latest version)
* **Kubernetes cluster** (Minikube, Kind, or managed service like EKS/AKS/GKE)
* **kubectl** CLI installed and configured
* **Docker Hub / Private Registry** access for image hosting
* **Keycloak** and **Eureka server** should be accessible to the API Gateway

**3. Docker Deployment**

**3.1 Dockerfile**

# Use official OpenJDK image with Java 21

FROM eclipse-temurin:21-jdk-alpine

# Set working directory

WORKDIR /app

# Copy jar file

COPY target/api-gateway-0.0.1-SNAPSHOT.jar api-gateway.jar

# Expose port

EXPOSE 8080

# Run the application

ENTRYPOINT ["java", "-jar", "api-gateway.jar"]

**3.2 Build & Run Commands**

# Build the jar (Maven)

mvn clean package -DskipTests

# Build docker image

docker build -t myorg/api-gateway:1.0 .

# Run container

docker run -d -p 8080:8080 --name api-gateway myorg/api-gateway:1.0

**4. Kubernetes Deployment**

**4.1 Namespace**

apiVersion: v1

kind: Namespace

metadata:

name: api-gateway-ns

**4.2 ConfigMap (for externalized configs)**

apiVersion: v1

kind: ConfigMap

metadata:

name: api-gateway-config

namespace: api-gateway-ns

data:

SPRING\_PROFILES\_ACTIVE: "prod"

EUREKA\_SERVER\_URL: "http://eureka-server:8761/eureka/"

OAUTH2\_AUTH\_SERVER: "http://keycloak:8080/realms/myrealm/protocol/openid-connect/token"

**4.3 Deployment**

apiVersion: apps/v1

kind: Deployment

metadata:

name: api-gateway

namespace: api-gateway-ns

spec:

replicas: 2

selector:

matchLabels:

app: api-gateway

template:

metadata:

labels:

app: api-gateway

spec:

containers:

- name: api-gateway

image: myorg/api-gateway:1.0

ports:

- containerPort: 8080

envFrom:

- configMapRef:

name: api-gateway-config

resources:

requests:

cpu: "200m"

memory: "512Mi"

limits:

cpu: "500m"

memory: "1Gi"

readinessProbe:

httpGet:

path: /actuator/health

port: 8080

initialDelaySeconds: 10

periodSeconds: 5

**4.4 Service**

apiVersion: v1

kind: Service

metadata:

name: api-gateway-svc

namespace: api-gateway-ns

spec:

type: ClusterIP

selector:

app: api-gateway

ports:

- port: 80

targetPort: 8080

**4.5 Ingress (Optional, for external access)**

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: api-gateway-ingress

namespace: api-gateway-ns

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

spec:

rules:

- host: api-gateway.local

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: api-gateway-svc

port:

number: 80

**5. Deployment Steps**

**Docker**

1. Build the application jar → mvn clean package
2. Build Docker image → docker build -t myorg/api-gateway:1.0 .
3. Run container → docker run -p 8080:8080 myorg/api-gateway:1.0

**Kubernetes**

1. Create namespace → kubectl apply -f namespace.yaml
2. Apply configmap → kubectl apply -f configmap.yaml
3. Deploy → kubectl apply -f deployment.yaml
4. Expose service → kubectl apply -f service.yaml
5. (Optional) Ingress → kubectl apply -f ingress.yaml

**6. Scaling and Monitoring**

* **Scaling:**
* **kubectl scale deployment api-gateway -n api-gateway-ns --replicas=3**
* **Monitoring:**
  + Use Spring Actuator /actuator/health
  + Integrate with Prometheus & Grafana
  + Logs via kubectl logs -f <pod> -n api-gateway-ns

**5. GCP GKE Deployment**

**5.1 Build & Push Image to Google Artifact Registry**

1. Enable Artifact Registry:
2. gcloud services enable artifactregistry.googleapis.com
3. Create a Docker repository in GCP:
4. gcloud artifacts repositories create my-docker-repo \
5. --repository-format=docker \
6. --location=us-central1 \
7. --description="Docker repo for API Gateway"
8. Configure Docker to authenticate with Artifact Registry:
9. gcloud auth configure-docker us-central1-docker.pkg.dev
10. Build and push Docker image:
11. mvn clean package -DskipTests
12. docker build -t us-central1-docker.pkg.dev/<PROJECT\_ID>/my-docker-repo/api-gateway:1.0 .
13. docker push us-central1-docker.pkg.dev/<PROJECT\_ID>/my-docker-repo/api-gateway:1.0

**5.2 Update Kubernetes Deployment for GKE**

apiVersion: apps/v1

kind: Deployment

metadata:

name: api-gateway

namespace: api-gateway-ns

spec:

replicas: 3

selector:

matchLabels:

app: api-gateway

template:

metadata:

labels:

app: api-gateway

spec:

containers:

- name: api-gateway

image: us-central1-docker.pkg.dev/<PROJECT\_ID>/my-docker-repo/api-gateway:1.0

ports:

- containerPort: 8080

envFrom:

- configMapRef:

name: api-gateway-config

readinessProbe:

httpGet:

path: /actuator/health

port: 8080

initialDelaySeconds: 10

periodSeconds: 5

**5.3 Expose Service via LoadBalancer**

apiVersion: v1

kind: Service

metadata:

name: api-gateway-svc

namespace: api-gateway-ns

spec:

type: LoadBalancer # GKE automatically provisions a GCP Load Balancer

selector:

app: api-gateway

ports:

- port: 80

targetPort: 8080

When deployed, GKE will create a **public IP** via a Google Load Balancer.  
Retrieve it using:

kubectl get svc -n api-gateway-ns

**5.4 GKE Ingress (Optional)**

If using **Ingress + ManagedCertificate** for TLS:

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: api-gateway-ingress

namespace: api-gateway-ns

annotations:

kubernetes.io/ingress.global-static-ip-name: "api-gateway-ip"

networking.gke.io/managed-certificates: "api-gateway-cert"

spec:

rules:

- host: api-gateway.mycompany.com

http:

paths:

- path: /

pathType: Prefix

backend:

service:

name: api-gateway-svc

port:

number: 80

**6. Deployment Steps in GCP GKE**

1. **Authenticate with GCP**
2. gcloud auth login
3. gcloud config set project <PROJECT\_ID>
4. **Create GKE cluster**
5. gcloud container clusters create api-gateway-cluster \
6. --num-nodes=3 --region=us-central1
7. **Connect kubectl**
8. gcloud container clusters get-credentials api-gateway-cluster --region=us-central1
9. **Apply manifests**
10. kubectl apply -f namespace.yaml
11. kubectl apply -f configmap.yaml
12. kubectl apply -f deployment-gke.yaml
13. kubectl apply -f service.yaml
14. kubectl apply -f ingress.yaml # optional

**7. Scaling and Monitoring**

* **Scale API Gateway**
* kubectl scale deployment api-gateway -n api-gateway-ns --replicas=5
* **Monitoring**
  + Enable **Cloud Monitoring & Logging** in GCP
  + Collect metrics from Spring Actuator
  + Use Prometheus & Grafana (if required)