PARTIE 2 : Déploiement Kubernetes

Durée : 60 minutes

Objectifs

Dans cette deuxième partie, vous allez déployer l'architecture microservices complète sur Kubernetes en créant tous les manifests nécessaires.

Compétences évaluées : - Création de Deployments avec probes et resources - Configuration des Services (ClusterIP) - Mise en place d'un Ingress pour le routing - Déploiement d'un StatefulSet PostgreSQL avec volumes persistants - Gestion des ConfigMaps et Secrets

---

Travail à Réaliser

Tâche 2.1 : Namespace et Structure (2 points)

Créez la structure de dossiers et le namespace pour isoler l'application.

Fichier : k8s/namespaces/cloudshop-prod.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: cloudshop-prod
  labels:
    name: cloudshop-prod
    environment: production
```

Structure des dossiers :

```
k8s/
 namespaces/
   cloudshop-prod.yaml
 configs/
   configmaps/
   secrets/
 deployments/
 services/
```

```
statefulsets/
ingress/
```

Tâche 2.2 : ConfigMaps et Secrets (5 points)

ConfigMap pour les URLs des services

Fichier : k8s/configs/configmaps/app-config.yaml

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
  namespace: cloudshop-prod
data:
  # URLs internes des microservices
  AUTH_SERVICE_URL: "http://auth-service:8081"
  PRODUCTS_SERVICE_URL: "http://products-service:8082"
  ORDERS_SERVICE_URL: "http://orders-service:8083"

  # Configuration applicative
  LOG_LEVEL: "info"
  NODE_ENV: "production"
```

Secret pour les credentials de base de données

Fichier : k8s/configs/secrets/db-credentials.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: db-credentials
  namespace: cloudshop-prod
type: Opaque
data:
  # Base64 encoded values
  # Utilisez : echo -n 'valeur' | base64
  POSTGRES_USER: YWRtaW4=          # admin
  POSTGRES_PASSWORD: cGFzc3dvcmQxMjM=   # password123
  POSTGRES_DB: Y2xvdWRzaG9w          # cloudshop
```

Commande pour encoder :

```
echo -n 'admin' | base64
echo -n 'password123' | base64
echo -n 'cloudshop' | base64
```

Secret pour JWT

Fichier : k8s/configs/secrets/jwt-secret.yaml

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: jwt-secret
  namespace: cloudshop-prod
type: Opaque
data:
  JWT_SECRET: c3VwZXItc2VjcmV0LWp3dC1rZXktMjU2LWJpdHM=  # super-secret-jwt-key-256-bits
```

Tâche 2.3 : StatefulSet PostgreSQL (6 points)

Déployez PostgreSQL en mode StatefulSet avec volume persistant.

Fichier : k8s/statefulsets/postgres.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: postgres
  namespace: cloudshop-prod
  labels:
    app: postgres
spec:
  ports:
  - port: 5432
    name: postgres
  clusterIP: None
  selector:
    app: postgres
---
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: postgres
  namespace: cloudshop-prod
spec:
  serviceName: "postgres"
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
        version: "15"
    spec:
      containers:
      - name: postgres
        image: postgres:15-alpine
        ports:
        - containerPort: 5432
          name: postgres
        env:
        - name: POSTGRES_USER
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: POSTGRES_USER
        - name: POSTGRES_PASSWORD
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: POSTGRES_PASSWORD
        - name: POSTGRES_DB
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: POSTGRES_DB
        - name: PGDATA
          value: /var/lib/postgresql/data/pgdata
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
        volumeMounts:
        - name: postgres-storage
          mountPath: /var/lib/postgresql/data
        livenessProbe:
          exec:
            command:
            - pg_isready
            - -U
            - admin
          initialDelaySeconds: 30
          periodSeconds: 10
        readinessProbe:
          exec:
            command:
            - pg_isready
            - -U
            - admin
          initialDelaySeconds: 5
          periodSeconds: 5
  volumeClaimTemplates:
  - metadata:
      name: postgres-storage
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 1Gi
```

Points clés : - serviceName obligatoire pour StatefulSet - volumeClaimTemplates pour créer un PVC par pod - Probes avec pg_isready - PGDATA configuré pour éviter les conflits

---

Tâche 2.4 : Deployments des Microservices (8 points)

Créez les Deployments pour les 5 microservices.

Deployment Auth Service

Fichier : k8s/deployments/auth-service.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: auth-service
  namespace: cloudshop-prod
  labels:
    app: auth-service
    version: v1.0.0
spec:
  replicas: 2
  selector:
    matchLabels:
      app: auth-service
  template:
    metadata:
      labels:
        app: auth-service
        version: v1.0.0
    spec:
      containers:
      - name: auth-service
        image: cloudshop/auth-service:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 8081
          name: http
        env:
        - name: PORT
          value: "8081"
        - name: DATABASE_URL
          value: "postgresql://$(POSTGRES_USER):$(POSTGRES_PASSWORD)@postgres:5432/$
(POSTGRES_DB)"
        - name: POSTGRES_USER
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: POSTGRES_USER
        - name: POSTGRES_PASSWORD
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: POSTGRES_PASSWORD
        - name: POSTGRES_DB
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: POSTGRES_DB
        - name: JWT_SECRET
          valueFrom:
            secretKeyRef:
              name: jwt-secret
              key: JWT_SECRET
        - name: LOG_LEVEL
          valueFrom:
            configMapKeyRef:
              name: app-config
              key: LOG_LEVEL
        resources:
          requests:
            memory: "128Mi"
            cpu: "100m"
          limits:
            memory: "256Mi"
            cpu: "250m"
        livenessProbe:
          httpGet:
            path: /health
            port: 8081
          initialDelaySeconds: 15
          periodSeconds: 10
          timeoutSeconds: 3
          failureThreshold: 3
        readinessProbe:
          httpGet:
            path: /health
            port: 8081
          initialDelaySeconds: 5
          periodSeconds: 5
          timeoutSeconds: 2
          failureThreshold: 3
        securityContext:
          runAsNonRoot: true
          runAsUser: 1001
          allowPrivilegeEscalation: false
```

Deployment Products API

Fichier : k8s/deployments/products-api.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: products-api
  namespace: cloudshop-prod
  labels:
    app: products-api
    version: v1.0.0
spec:
  replicas: 2
  selector:
    matchLabels:
      app: products-api
  template:
    metadata:
      labels:
        app: products-api
        version: v1.0.0
    spec:
      containers:
      - name: products-api
        image: cloudshop/products-api:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 8082
          name: http
        env:
        - name: PORT
          value: "8082"
        - name: DATABASE_URL
          value: "postgresql://$(POSTGRES_USER):$(POSTGRES_PASSWORD)@postgres:5432/$
(POSTGRES_DB)"
        - name: POSTGRES_USER
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: POSTGRES_USER
        - name: POSTGRES_PASSWORD
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: POSTGRES_PASSWORD
        - name: POSTGRES_DB
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: POSTGRES_DB
        resources:
          requests:
            memory: "128Mi"
            cpu: "100m"
          limits:
            memory: "256Mi"
            cpu: "250m"
        livenessProbe:
          httpGet:
            path: /health
            port: 8082
          initialDelaySeconds: 15
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /health
            port: 8082
          initialDelaySeconds: 5
          periodSeconds: 5
        securityContext:
          runAsNonRoot: true
          runAsUser: 1001
          allowPrivilegeEscalation: false
```

Deployment Orders API

Fichier : k8s/deployments/orders-api.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: orders-api
  namespace: cloudshop-prod
  labels:
    app: orders-api
    version: v1.0.0
```

```yaml
spec:
  replicas: 2
  selector:
    matchLabels:
      app: orders-api
  template:
    metadata:
      labels:
        app: orders-api
        version: v1.0.0
    spec:
      containers:
      - name: orders-api
        image: cloudshop/orders-api:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 8083
          name: http
        env:
        - name: PORT
          value: "8083"
        - name: DATABASE_URL
          value: "postgresql://$(POSTGRES_USER):$(POSTGRES_PASSWORD)@postgres:5432/$
(POSTGRES_DB)"
        - name: POSTGRES_USER
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: POSTGRES_USER
        - name: POSTGRES_PASSWORD
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: POSTGRES_PASSWORD
        - name: POSTGRES_DB
          valueFrom:
            secretKeyRef:
              name: db-credentials
              key: POSTGRES_DB
        resources:
          requests:
            memory: "64Mi"
            cpu: "50m"
          limits:
            memory: "128Mi"
            cpu: "150m"
        livenessProbe:
          httpGet:
            path: /health
            port: 8083
          initialDelaySeconds: 10
          periodSeconds: 10
        readinessProbe:
          httpGet:
            path: /health
            port: 8083
          initialDelaySeconds: 5
          periodSeconds: 5
        securityContext:
          runAsNonRoot: true
          runAsUser: 1001
          allowPrivilegeEscalation: false
```

Deployment API Gateway

Fichier : k8s/deployments/api-gateway.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-gateway
  namespace: cloudshop-prod
  labels:
    app: api-gateway
    version: v1.0.0
spec:
  replicas: 3
  selector:
    matchLabels:
      app: api-gateway
  template:
    metadata:
      labels:
        app: api-gateway
        version: v1.0.0
    spec:
      containers:
```

```yaml
- name: api-gateway
  image: cloudshop/api-gateway:latest
  imagePullPolicy: Always
  ports:
  - containerPort: 8080
    name: http
  env:
  - name: PORT
    value: "8080"
  - name: AUTH_SERVICE_URL
    valueFrom:
      configMapKeyRef:
        name: app-config
        key: AUTH_SERVICE_URL
  - name: PRODUCTS_SERVICE_URL
    valueFrom:
      configMapKeyRef:
        name: app-config
        key: PRODUCTS_SERVICE_URL
  - name: ORDERS_SERVICE_URL
    valueFrom:
      configMapKeyRef:
        name: app-config
        key: ORDERS_SERVICE_URL
  - name: LOG_LEVEL
    valueFrom:
      configMapKeyRef:
        name: app-config
        key: LOG_LEVEL
  resources:
    requests:
      memory: "128Mi"
      cpu: "100m"
    limits:
      memory: "256Mi"
      cpu: "250m"
  livenessProbe:
    httpGet:
      path: /health
      port: 8080
    initialDelaySeconds: 15
    periodSeconds: 10
  readinessProbe:
    httpGet:
      path: /health
      port: 8080
    initialDelaySeconds: 5
    periodSeconds: 5
  securityContext:
    runAsNonRoot: true
    runAsUser: 1001
    allowPrivilegeEscalation: false
```

Deployment Frontend

Fichier : k8s/deployments/frontend.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  namespace: cloudshop-prod
  labels:
    app: frontend
    version: v1.0.0
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
        version: v1.0.0
    spec:
      containers:
      - name: frontend
        image: cloudshop/frontend:latest
        imagePullPolicy: Always
        ports:
        - containerPort: 80
          name: http
        env:
        - name: API_URL
          value: "http://api-gateway:8080"
        resources:
```

```
        requests:
          memory: "64Mi"
          cpu: "50m"
        limits:
          memory: "128Mi"
          cpu: "150m"
      livenessProbe:
        httpGet:
          path: /
          port: 80
        initialDelaySeconds: 10
        periodSeconds: 10
      readinessProbe:
        httpGet:
          path: /
          port: 80
        initialDelaySeconds: 5
        periodSeconds: 5
      securityContext:
        runAsNonRoot: true
        runAsUser: 101
        allowPrivilegeEscalation: false
```

Tâche 2.5 : Services (4 points)

Créez les Services pour exposer les Deployments.

Fichier : k8s/services/services.yaml

```
---
apiVersion: v1
kind: Service
metadata:
  name: auth-service
  namespace: cloudshop-prod
  labels:
    app: auth-service
spec:
  type: ClusterIP
  ports:
  - port: 8081
    targetPort: 8081
```

```yaml
    protocol: TCP
    name: http
  selector:
    app: auth-service
---
apiVersion: v1
kind: Service
metadata:
  name: products-service
  namespace: cloudshop-prod
  labels:
    app: products-api
spec:
  type: ClusterIP
  ports:
  - port: 8082
    targetPort: 8082
    protocol: TCP
    name: http
  selector:
    app: products-api
---
apiVersion: v1
kind: Service
metadata:
  name: orders-service
  namespace: cloudshop-prod
  labels:
    app: orders-api
spec:
  type: ClusterIP
  ports:
  - port: 8083
    targetPort: 8083
    protocol: TCP
    name: http
  selector:
    app: orders-api
---
apiVersion: v1
kind: Service
metadata:
  name: api-gateway
  namespace: cloudshop-prod
  labels:
    app: api-gateway
spec:
  type: ClusterIP
  ports:
  - port: 8080
    targetPort: 8080
    protocol: TCP
    name: http
  selector:
    app: api-gateway
---
apiVersion: v1
kind: Service
metadata:
  name: frontend
  namespace: cloudshop-prod
  labels:
    app: frontend
spec:
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: http
  selector:
    app: frontend
```

Tâche 2.6 : Ingress (4 points)

Créez l'Ingress pour exposer le frontend et l'API Gateway.

Fichier : k8s/ingress/ingress.yaml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: cloudshop-ingress
  namespace: cloudshop-prod
  annotations:
    kubernetes.io/ingress.class: "traefik"
    traefik.ingress.kubernetes.io/router.entrypoints: web
spec:
  rules:
  - host: shop.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend
            port:
              number: 80
  - host: api.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: api-gateway
            port:
              number: 8080
```

Configuration /etc/hosts (pour tests locaux) :

```
# Ajouter à /etc/hosts
127.0.0.1 shop.local
127.0.0.1 api.local
```

Validation

Déploiement

```bash
# 1. Créer le namespace
kubectl apply -f k8s/namespaces/

# 2. Créer les Secrets et ConfigMaps
kubectl apply -f k8s/configs/secrets/
kubectl apply -f k8s/configs/configmaps/
```

```
# 3. Déployer PostgreSQL (StatefulSet)
kubectl apply -f k8s/statefulsets/

# 4. Attendre que PostgreSQL soit prêt
kubectl wait --for=condition=ready pod -l app=postgres -n cloudshop-prod --timeout=120s

# 5. Déployer les microservices
kubectl apply -f k8s/deployments/

# 6. Créer les Services
kubectl apply -f k8s/services/

# 7. Créer l'Ingress
kubectl apply -f k8s/ingress/

# 8. Vérifier l'ensemble
kubectl get all -n cloudshop-prod
```

Tests de Validation

```
# 1. Vérifier que tous les pods sont Running
kubectl get pods -n cloudshop-prod

# Attendu :
# postgres-0              1/1     Running
# auth-service-xxx        1/1     Running
# products-api-xxx        1/1     Running
# orders-api-xxx          1/1     Running
# api-gateway-xxx         1/1     Running
# frontend-xxx            1/1     Running

# 2. Vérifier les Services
kubectl get svc -n cloudshop-prod

# 3. Vérifier l'Ingress
kubectl get ingress -n cloudshop-prod
kubectl describe ingress cloudshop-ingress -n cloudshop-prod

# 4. Tester les endpoints
# Via port-forward
kubectl port-forward svc/api-gateway 8080:8080 -n cloudshop-prod
curl http://localhost:8080/health
```

```
# Via Ingress (si configuré)
curl -H "Host: shop.local" http://localhost
curl -H "Host: api.local" http://localhost/health

# 5. Vérifier la persistence PostgreSQL
kubectl exec -it postgres-0 -n cloudshop-prod -- psql -U admin -d cloudshop -c '\l'

# 6. Vérifier les logs
kubectl logs -l app=api-gateway -n cloudshop-prod --tail=50

# 7. Vérifier les ConfigMaps et Secrets
kubectl get configmap -n cloudshop-prod
kubectl get secrets -n cloudshop-prod
kubectl describe configmap app-config -n cloudshop-prod
```

Debugging

```
# Si un pod ne démarre pas
kubectl describe pod <pod-name> -n cloudshop-prod
kubectl logs <pod-name> -n cloudshop-prod
kubectl logs <pod-name> -n cloudshop-prod --previous

# Si un Service ne fonctionne pas
kubectl get endpoints <service-name> -n cloudshop-prod

# Tester la connectivité interne
kubectl run -it --rm debug --image=busybox --restart=Never -n cloudshop-prod -- sh
# Puis dans le pod :
wget -O- http://api-gateway:8080/health
```

Erreur : ImagePullBackOff

```
# Vérifier l'image
kubectl describe pod <pod-name> -n cloudshop-prod

# Solutions :
# 1. Image n'existe pas : reconstruire et pusher
# 2. Credentials manquants : créer imagePullSecrets
# 3. Tag incorrect : vérifier dans le Deployment
```

Erreur : CrashLoopBackOff

```
# Voir les logs
kubectl logs <pod-name> -n cloudshop-prod --previous

# Causes fréquentes :
# - Variables d'environnement manquantes
# - Database non accessible
# - Port déjà utilisé
```

StatefulSet : Pod en Pending

```
# Vérifier les PVC
kubectl get pvc -n cloudshop-prod

# Si PVC Pending :
# - Vérifier le StorageClass
kubectl get storageclass

# - Installer un provisioner local (si Kind)
kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-provisioner/master/deploy/local-path-storage.yaml
```

Ingress non accessible

```
# Vérifier que l'Ingress Controller est installé
kubectl get pods -n traefik

# Vérifier le status de l'Ingress
kubectl describe ingress cloudshop-ingress -n cloudshop-prod

# Tester avec curl et Host header
curl -v -H "Host: shop.local" http://<cluster-ip>
```

Une fois cette partie terminée, passez à PARTIE3_GITOPS.md