# TD Final - Déploiement d'une Plateforme E-Commerce Cloud Native

## Master 2 Full Stack - Docker & Kubernetes

### Contexte & Mission

Vous êtes **SRE/DevOps Engineer** chez **CloudShop**, une startup e-commerce qui vient de lever des fonds. L'équipe technique vous confie la mission de **moderniser l'infrastructure** en migrant vers une architecture **Cloud Native**.

**Contraintes business** : - Déploiements multiples par jour (CI/CD automatisé) - Sécurité renforcée (images signées, policies strictes) - Observabilité complète (métriques, logs, traces) - Haute disponibilité (99.9% SLO) - Tests de résilience obligatoires (Chaos Engineering)

### Architecture Microservices

Votre plateforme est composée de **5 microservices** :

1. **Frontend** (React + Vite)
    - Interface utilisateur SPA
    - Port: 3000
    - Langage: JavaScript/TypeScript
2. **API Gateway** (Node.js Express)
    - Point d'entrée unique
    - Routing vers les microservices
    - Rate limiting, CORS
    - Port: 8080
3. **Auth Service** (Node.js)
    - Authentification JWT
    - Gestion des users
    - Port: 8081
4. **Products API** (Python FastAPI)
    - CRUD produits
    - Elasticsearch pour recherche
    - Port: 8082

5. **Orders API** (Go)
    - Gestion des commandes
    - Connexion PostgreSQL
    - Port: 8083

### Travail à Réaliser

Le TD est découpé en **5 parties** correspondant aux 5 jours de formation.

---

## PARTIE 1 : Conteneurisation Docker

**Fichier détaillé** : PARTIE1_DOCKER.md

### Objectifs

- Créer des Dockerfiles multi-stage optimisés
- Configurer Docker Compose pour le développement local
- Appliquer les best practices de sécurité

### Livrables attendus

5 Dockerfiles (un par microservice)
1 docker-compose.yml fonctionnel
1 .dockerignore par service
Images < 200MB
Utilisateur non-root

### Critères d'évaluation

- Multi-stage builds : 5 pts
- Taille des images : 5 pts
- Sécurité (non-root, scan) : 5 pts
- Docker Compose complet : 5 pts

---

## PARTIE 2 : Déploiement Kubernetes

**Fichier détaillé** : PARTIE2_KUBERNETES.md

### Objectifs

- Créer les manifests Kubernetes (Deployments, Services, Ingress)

- Déployer PostgreSQL en StatefulSet
- Configurer les ConfigMaps et Secrets
- Exposer l'application via Ingress

### Livrables attendus

Namespace cloudshop-prod
5 Deployments avec requests/limits
5 Services (ClusterIP)
1 Ingress (shop.local, api.local)
1 StatefulSet PostgreSQL avec PVC
ConfigMaps & Secrets

---

## PARTIE 3 : GitOps avec ArgoCD

**Fichier détaillé** : PARTIE3_GITOPS.md

### Objectifs

- Installer ArgoCD
- Créer des Applications ArgoCD
- Configurer la synchronisation automatique
- Tester un rollback

### Livrables attendus

ArgoCD installé et accessible
1 Application ArgoCD par environnement (dev, staging, prod)
Auto-sync activé
Healt checks configurés

### Critères d'évaluation (15 points)

- Installation ArgoCD : 4 pts
- Applications configurées : 6 pts
- Sync policies : 3 pts
- Rollback testé : 2 pts

---

# PARTIE 4 : Observabilité (Prometheus, Grafana)

**Fichier détaillé** : PARTIE4_OBSERVABILITE.md

## Objectifs

- Déployer la stack Prometheus + Grafana
- Créer des ServiceMonitors
- Configurer des dashboards Grafana
- Définir des SLI/SLO et alertes

## Livrables attendus

Prometheus + Grafana déployés
ServiceMonitors pour chaque microservice
2 dashboards Grafana (Overview, SLOs)
Alerting rules (HighErrorRate, LatencyP95)

## Critères d'évaluation (15 points)

- Stack monitoring : 4 pts
- ServiceMonitors : 4 pts
- Dashboards Grafana : 4 pts
- Alertes configurées : 3 pts

---

# PARTIE 5 : Sécurité, SRE & Chaos

**Fichier détaillé** : PARTIE5_SECURITE_SRE.md

## Objectifs

- Implémenter Policy as Code avec Kyverno
- Créer un pipeline CI/CD avec signature d'images (Cosign)
- Calculer les Error Budgets
- Exécuter un Chaos Experiment avec Litmus

## Livrables attendus

3 ClusterPolicies Kyverno (deny-latest, require-resources, check-signature)
Pipeline GitHub Actions avec Cosign
Calcul Error Budget (SLO 99.9%)
1 Chaos Experiment (pod-delete)

---

# Démarrage

## Prérequis

**Outils nécessaires** :

```
docker --version        # >= 24.0
docker-compose --version  # >= 2.20
kubectl version --client  # >= 1.28
helm version           # >= 3.12
kind version           # >= 0.20 (ou k3d, minikube)
```

**Accès** : - Compte GitHub (pour GitOps et CI/CD) - Accès à un cluster Kubernetes (Kind, K3d ou minikube)

## Étape 1 : Setup du Cluster Local

```
# Créer un cluster Kind avec Ingress
cat <<EOF | kind create cluster --config=-
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
name: cloudshop
nodes:
- role: control-plane
  kubeadmConfigPatches:
  - |
    kind: InitConfiguration
    nodeRegistration:
      kubeletExtraArgs:
        node-labels: "ingress-ready=true"
  extraPortMappings:
  - containerPort: 80
    hostPort: 80
  - containerPort: 443
    hostPort: 443
EOF

# Vérifier
kubectl cluster-info
kubectl get nodes
```

## Étape 2 : Installer les Prérequis

```
# Ingress Controller (Traefik)
helm repo add traefik https://traefik.github.io/charts
helm install traefik traefik/traefik \
  --namespace traefik --create-namespace

# Cert-Manager (optionnel pour HTTPS)
kubectl apply -f
https://github.com/cert-manager/cert-manager/releases/download/v1.13.0/cert-
manager.yaml
```

## Étape 3 : Explorer le Code

```
cd tp-etudiant/src

# Structure fournie :
src/
 frontend/         # React (package.json fourni, Dockerfile à créer)
 api-gateway/      # Node.js (server.js fourni, Dockerfile à créer)
 auth-service/     # Node.js (app.js fourni, Dockerfile à créer)
 products-api/     # Python (main.py fourni, Dockerfile à créer)
 orders-api/       # Go (main.go fourni, Dockerfile à créer)
```

## Étape 4 : Commencer le TD

```
# Ouvrir l'énoncé de la Partie 1
cat PARTIE1_DOCKER.md

# Lancer le projet en mode dev
docker-compose up -d
```

---

# Ressources & Documentation

## Documentation Officielle

- Docker Best Practices
- Kubernetes Documentation
- ArgoCD Documentation
- Prometheus Operator
- Kyverno Policies

## Cours de la Formation

- **Jour 1-2** : Docker, Images, Compose, Sécurité
- **Jour 3** : Kubernetes Orchestration
- **Jour 4** : GitOps, Prometheus, Grafana, Velero
- **Jour 5** : Backstage, Kyverno, SRE, Litmus, Tekton

## Exemples de Commandes Utiles

```
# Docker
docker build -t myapp:v1.0 .
docker images | grep myapp
docker run -p 8080:8080 myapp:v1.0
docker-compose up -d
docker-compose logs -f api-gateway

# Kubernetes
kubectl create namespace cloudshop-prod
kubectl apply -f k8s/
kubectl get pods -n cloudshop-prod
kubectl logs -f deployment/frontend -n cloudshop-prod
kubectl describe pod <pod-name> -n cloudshop-prod
kubectl port-forward svc/api-gateway 8080:8080 -n cloudshop-prod

# ArgoCD
argocd app create frontend --repo https://github.com/user/cloudshop.git \
  --path k8s/frontend --dest-server https://kubernetes.default.svc \
  --dest-namespace cloudshop-prod
argocd app sync frontend
argocd app get frontend

# Prometheus
kubectl port-forward svc/prometheus-operated 9090:9090 -n monitoring

# Kyverno
kubectl apply -f policies/deny-latest.yaml
kubectl get clusterpolicy
kubectl describe clusterpolicy disallow-latest-tag
```

---

## Checklist de Validation

Avant de rendre votre travail, vérifiez que :

### Docker & Compose

- [ ] Les 5 Dockerfiles sont créés et optimisés (multi-stage)
- [ ] `docker-compose up` démarre tous les services sans erreur
- [ ] Frontend accessible sur http://localhost:3000
- [ ] API Gateway accessible sur http://localhost:8080
- [ ] Toutes les images < 200MB
- [ ] Scan Trivy ne remonte aucune vulnérabilité CRITICAL

### Kubernetes

- [ ] `kubectl get pods -n cloudshop-prod` affiche 5 pods RUNNING
- [ ] PostgreSQL StatefulSet est persistant (survit à un redémarrage)
- [ ] Ingress fonctionne : http://shop.local redirige vers le frontend
- [ ] Tous les Deployments ont des `requests` et `limits` configurés
- [ ] Secrets et ConfigMaps sont utilisés (pas de credentials en dur)

### GitOps

- [ ] ArgoCD UI accessible (kubectl port-forward)
- [ ] Au moins 1 Application ArgoCD synchronisée
- [ ] Auto-sync fonctionne (changement dans Git = déploiement auto)
- [ ] Health check “Healthy” pour toutes les resources

### Observabilité

- [ ] Prometheus scrape les métriques des 5 microservices
- [ ] Grafana affiche un dashboard “CloudShop Overview”
- [ ] Au moins 2 alertes configurées
- [ ] SLO défini et affiché dans Grafana

### Sécurité & SRE

- [ ] Kyverno installé et au moins 2 policies actives
- [ ] Pipeline GitHub Actions fonctionne (build + push + sign)
- [ ] Image signée avec Cosign et vérifiable
- [ ] Chaos Experiment exécuté avec succès (pod-delete)
- [ ] Application récupère après le chaos test (auto-healing)

---

## En Cas de Blocage

1. **Consultez les aides** : Chaque partie contient un fichier AIDE.md avec des indices
2. **Relisez les cours** : Les concepts sont tous dans les cours Jours 1-5
3. **Debuggez méthodiquement** :

```
kubectl describe pod <pod-name>
kubectl logs <pod-name>
kubectl get events --sort-by='.lastTimestamp'
```

---

## Livrables Finaux

À rendre à la fin du TD :

1. **Code source** : Dossier src/ avec tous les Dockerfiles
2. **Manifests K8s** : Dossier k8s/ complet
3. **Configurations** : docker-compose.yml, ArgoCD apps, Prometheus rules
4. **Documentation** : ARCHITECTURE.md expliquant vos choix techniques
5. **Screenshots** :
    - Frontend accessible
    - ArgoCD dashboard
    - Grafana dashboards
    - Kyverno policies actives
    - Chaos Experiment résultats

---

**Bonne chance ! Ce TD est l'aboutissement de votre formation. Montrez tout ce que vous avez appris !**

*N'oubliez pas : L'objectif n'est pas la perfection, mais la démonstration de votre compréhension des concepts Docker/Kubernetes/GitOps/SRE.*