PARTIE 1 : Conteneurisation Docker

Durée : 45 minutes

Objectifs

Dans cette première partie, vous allez conteneuriser les 5 microservices de la plateforme CloudShop en appliquant les **best practices** apprises durant les Jours 1 et 2.

Compétences évaluées : - Multi-stage builds pour optimiser la taille des images - Sécurité (utilisateur non-root, scan de vulnérabilités) - Images optimisées (< 200MB par service) - Docker Compose pour l'orchestration locale

---

Travail à Réaliser

Tâche 1.1 : Dockerfiles Multi-Stage (15 points)

Créez un Dockerfile optimisé pour chaque microservice :

1.1.1 - Frontend (React + Vite)

Fichier : src/frontend/Dockerfile

Contraintes : - **Stage 1 (build)** : Node.js 20-alpine, installer dépendances, build Vite - **Stage 2 (runtime)** : Nginx alpine, copier uniquement /dist - Taille cible : < 50MB - Port : 80

Indices :

```
# Stage 1: Build
FROM node:20-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY . .
RUN npm run build

# Stage 2: Production
FROM nginx:alpine
COPY --from=builder /app/dist /usr/share/nginx/html
```

```
COPY nginx.conf /etc/nginx/nginx.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Configuration Nginx requise (nginx.conf) :

```
server {
  listen 80;
  location / {
    root /usr/share/nginx/html;
    try_files $uri $uri/ /index.html;
  }
}
```

1.1.2 - API Gateway (Node.js Express)

Fichier : src/api-gateway/Dockerfile

Contraintes : - Base image : node:20-alpine - Utilisateur non-root (créer node user) - Taille cible : < 150MB - Port : 8080 - Healthcheck HTTP /health

Template :

```
FROM node:20-alpine

# Créer un utilisateur non-root
RUN addgroup -g 1001 appgroup && \
  adduser -D -u 1001 -G appgroup appuser

WORKDIR /app

# Copier seulement package.json d'abord (cache layer)
COPY --chown=appuser:appgroup package*.json ./
RUN npm ci --only=production && npm cache clean --force

# Copier le code
COPY --chown=appuser:appgroup . .

# Changer d'utilisateur
USER appuser

EXPOSE 8080
```

```
# Healthcheck
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
  CMD node -e "require('http').get('http://localhost:8080/health', (r) =>
{process.exit(r.statusCode === 200 ? 0 : 1)})"

CMD ["node", "server.js"]
```

1.1.3 - Auth Service (Node.js)

Fichier : src/auth-service/Dockerfile

Contraintes : - Identique à API Gateway - Port : 8081 - Healthcheck : /health

---

1.1.4 - Products API (Python FastAPI)

Fichier : src/products-api/Dockerfile

Contraintes : - **Stage 1** : python:3.11-slim avec build dependencies - **Stage 2** : python:3.11-slim runtime only - Utilisateur non-root - Taille cible : < 180MB - Port : 8082

Template :

```
# Stage 1: Build
FROM python:3.11-slim AS builder

WORKDIR /app

# Installer build dependencies
RUN apt-get update && apt-get install -y --no-install-recommends \
    gcc \
    && rm -rf /var/lib/apt/lists/*

# Copier requirements et installer
COPY requirements.txt .
RUN pip install --no-cache-dir --user -r requirements.txt

# Stage 2: Runtime
FROM python:3.11-slim
```

```
WORKDIR /app

# Créer utilisateur non-root
RUN useradd -m -u 1001 appuser

# Copier les packages Python depuis builder
COPY --from=builder /root/.local /home/appuser/.local
ENV PATH=/home/appuser/.local/bin:$PATH

# Copier le code
COPY --chown=appuser:appuser . .

USER appuser

EXPOSE 8082

HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
  CMD python -c "import urllib.request;
urllib.request.urlopen('http://localhost:8082/health').read()"

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8082"]
```

1.1.5 - Orders API (Go)

Fichier : src/orders-api/Dockerfile

Contraintes : - **Stage 1** : golang:1.21-alpine pour build - **Stage 2** : alpine:latest (distroless optionnel si bonus) - Binary statique (CGO_ENABLED=0) - Taille cible : < 20MB - Port : 8083

Template :

```
# Stage 1: Build
FROM golang:1.21-alpine AS builder

WORKDIR /app

# Copier go.mod et go.sum
COPY go.mod go.sum ./
RUN go mod download
```

```
# Copier le code et build
COPY . .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o orders-api .

# Stage 2: Runtime
FROM alpine:latest

RUN apk --no-cache add ca-certificates

WORKDIR /root/

# Créer utilisateur non-root
RUN addgroup -g 1001 appgroup && \
    adduser -D -u 1001 -G appgroup appuser

# Copier le binary
COPY --from=builder --chown=appuser:appgroup /app/orders-api .

USER appuser

EXPOSE 8083

HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
  CMD wget --no-verbose --tries=1 --spider http://localhost:8083/health || exit 1

CMD ["./orders-api"]
```

Tâche 1.2 : .dockerignore (2 points)

Créez un fichier .dockerignore pour chaque service afin d'exclure les fichiers inutiles.

Exemple générique :

```
node_modules
npm-debug.log
.git
.gitignore
README.md
.env
.env.local
```

Dockerfile
docker-compose.yml
*.md
.vscode
.idea
dist
build
*.test.js
coverage

Tâche 1.3 : Docker Compose (3 points)

Créez un docker-compose.yml à la racine du projet pour démarrer **tous les services en local.**

Fichier : docker-compose.yml

Contraintes : - Network cloudshop-net créé - PostgreSQL en volume persistant - Variables d'environnement via .env - Healthchecks configurés - Depends_on avec conditions

Template :

```yaml
version: '3.9'

networks:
  cloudshop-net:
    driver: bridge

volumes:
  postgres-data:

services:
  # Base de données
  postgres:
    image: postgres:15-alpine
    container_name: cloudshop-postgres
    environment:
      POSTGRES_DB: cloudshop
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD:-changeme}
    volumes:
```

```
    - postgres-data:/var/lib/postgresql/data
  networks:
    - cloudshop-net
  healthcheck:
    test: ["CMD-SHELL", "pg_isready -U admin -d cloudshop"]
    interval: 10s
    timeout: 5s
    retries: 5
  ports:
    - "5432:5432"

 # Auth Service
 auth-service:
  build:
    context: ./src/auth-service
    dockerfile: Dockerfile
  container_name: cloudshop-auth
  environment:
    PORT: 8081
    DATABASE_URL: postgresql://admin:${POSTGRES_PASSWORD:-changeme}
@postgres:5432/cloudshop
    JWT_SECRET: ${JWT_SECRET:-secret-key-change-in-prod}
  depends_on:
    postgres:
      condition: service_healthy
  networks:
    - cloudshop-net
  ports:
    - "8081:8081"

 # Products API
 products-api:
  build:
    context: ./src/products-api
    dockerfile: Dockerfile
  container_name: cloudshop-products
  environment:
    PORT: 8082
    DATABASE_URL: postgresql://admin:${POSTGRES_PASSWORD:-changeme}
@postgres:5432/cloudshop
  depends_on:
    postgres:
      condition: service_healthy
```

```
    networks:
      - cloudshop-net
    ports:
      - "8082:8082"

  # Orders API
  orders-api:
    build:
      context: ./src/orders-api
      dockerfile: Dockerfile
    container_name: cloudshop-orders
    environment:
      PORT: 8083
      DATABASE_URL: postgresql://admin:${POSTGRES_PASSWORD:-changeme}
@postgres:5432/cloudshop
    depends_on:
      postgres:
        condition: service_healthy
    networks:
      - cloudshop-net
    ports:
      - "8083:8083"

  # API Gateway
  api-gateway:
    build:
      context: ./src/api-gateway
      dockerfile: Dockerfile
    container_name: cloudshop-gateway
    environment:
      PORT: 8080
      AUTH_SERVICE_URL: http://auth-service:8081
      PRODUCTS_SERVICE_URL: http://products-api:8082
      ORDERS_SERVICE_URL: http://orders-api:8083
    depends_on:
      - auth-service
      - products-api
      - orders-api
    networks:
      - cloudshop-net
    ports:
      - "8080:8080"
```

```yaml
  # Frontend
  frontend:
    build:
      context: ./src/frontend
      dockerfile: Dockerfile
    container_name: cloudshop-frontend
    environment:
      API_URL: http://localhost:8080
    depends_on:
      - api-gateway
    networks:
      - cloudshop-net
    ports:
      - "3000:80"
```

Fichier .env à créer :

```
POSTGRES_PASSWORD=secure_postgres_password_123
JWT_SECRET=super-secret-jwt-key-256-bits
```

Validation

Tests à effectuer :

```bash
# 1. Build toutes les images
docker-compose build

# 2. Vérifier la taille des images
docker images | grep cloudshop

# Attendu :
# cloudshop-frontend      < 50MB
# cloudshop-gateway       < 150MB
# cloudshop-auth          < 150MB
# cloudshop-products      < 180MB
# cloudshop-orders        < 20MB

# 3. Démarrer la stack
docker-compose up -d

# 4. Vérifier que tous les services sont UP
```

```
docker-compose ps

# 5. Tester les endpoints
curl http://localhost:3000          # Frontend (HTML)
curl http://localhost:8080/health   # API Gateway
curl http://localhost:8081/health   # Auth Service
curl http://localhost:8082/health   # Products API
curl http://localhost:8083/health   # Orders API

# 6. Scanner les vulnérabilités (avec Trivy)
trivy image cloudshop-frontend:latest
trivy image cloudshop-gateway:latest
# Aucun CRITICAL ne doit être présent

# 7. Vérifier que les conteneurs tournent en non-root
docker exec cloudshop-gateway whoami  # Doit afficher "appuser" ou similaire

# 8. Logs
docker-compose logs -f api-gateway
```

Barème (20 points)

Aide

Problème : Image trop volumineuse

Solution : - Utiliser des images alpine ou slim - Multi-stage builds (ne copier que le nécessaire) - Supprimer les caches : npm cache clean --force, apt-get clean

Problème : Permission denied

Solution : - Vérifier COPY --chown=user:group - Ajouter USER username avant CMD

Problème : Service ne démarre pas

Solution :

```
docker-compose logs <service-name>
docker exec -it <container> sh
```

Problème : Healthcheck échoue

Solution : - Vérifier que le endpoint /health existe dans le code - Augmenter start-period dans le healthcheck - Tester manuellement : curl http://localhost:8080/health

---

Ressources

- Dockerfile Best Practices
- Multi-Stage Builds
- Docker Compose File Reference
- Trivy Scanner

Une fois cette partie terminée, passez à PARTIE2_KUBERNETES.md !