OpenclassRooms

La place du marché

Dossier d'exploitation

Version 1.0

Auteur
Yann
Etudiant développeur

TABLE DES MATIERES

1 - VERSIONS

Auteur	Date	Description	Version
Yan K	05/02/2019	Création du document	1.0

2 - INTRODUCTION

2.1 - Objet du document

Le présent document constitue le dossier d'exploitation de l'application LPDM (La Place Du Marché).

Objectif du document...

2.2 - Références

Pour de plus amples informations, se référer :

1. Dossier de conception technique de l'application
2. Dossier de conception fonctionnelle de l'application
3. Code source de l'application

3 - PRE-REQUIS

3.1 - Système

3.1.1 - Serveur de Base de données

L'application dans sa globalité repose sur une architecture micro-service et nécessite 7 serveurs de base de données intimement liés à chaque micro-service.

3.1.1.1 - Caractéristiques techniques

Les différents serveurs de bases de données reposent sur des systèmes de gestion de base de données relationnelles PostgreSQL comme suit:

1. L'ensemble des systèmes de gestion utilisent la version 11 de PostgreSQL
2. Chaque serveur de base de données est exécuté de façon autonome dans un conteneur Docker ayant pour base système une distribution Linux Alpine.
3. Chaque conteneur Docker exécutant un serveur de base de données se voit associé à un volume de données qui lui est propre, afin de faire persister le contenu des données si un conteneur est stoppé dans son exécution.
4. Afin que les communications puissent correctement s'effectuer, chaque conteneur exécutant un serveur de base de données expose un port différent.

3.1.1.2 - Bases de données liées à un micro-service

Le liste ci-dessous énumère les différentes bases de données nécessaires au bon fonctionnement de l'application, associées au micro-service auquel elles sont rattachées :

- DB Auth → micro-service à l'authentification
- DB Order → micro-service dédié aux commandes
- DB Product → micro-service aux produits
- DB Stock → micro-service dédié aux stocks de produits
- DB Location → micro-service dédié à la gestion des adresses
- DB Store → micro-service dédié aux magasins
- DB Storage → micro-service dédié à l'enregistrement de fichiers

3.1.1.3 - Sécurité

Pour que chaque micro-service puisse être autorisé à communiquer avec la base de données auquel il est rattaché, des identifiants de connexion sont stockés dans des fichiers de propriétés. Afin de ne pas exposer sur le réseau les identifiants de connexion aux base de données, ces derniers sont chiffrés via la bibliothèque Java Jasypt avec une clé de chiffrement. Cette clé doit être passé en tant que variable d'environnement lors de l'exécution du conteneur d'un micro-service pour que le data-source puisse être correctement lu par l'application et transmis à l'API JDBC.

3.1.2 - Serveur Web

Chaque micro-service composant l'application s'exécute sur un serveur d'application Apache Tomcat 9.0.13 et expose un port de communication différent qui lui est propre.

Chaque micro-service communique via une API Rest et donc seules des données de contenu de type MIME text/plain transitent au sein de l'application, à l'exception de deux micro-services :

- le micro-service des commandes produit, lors de la génération de factures, un MIME application/octet-

stream afin de proposer le téléchargement d'un fichier PDF.

- le micro-service dédié au stockage de fichiers, traite également, en entrée, des données de type MIME application/octet-stream afin de récupérer des fichiers et de les persister.

3.1.2.1 - Caractéristiques techniques

L'application est hébergée sur un serveur dédié, fournit par la société OVH via la marque Kimsufi, exécutant une distribution Linux Ubuntu 18.04 LTS 64bits.

La configuration hardware du serveur propose un processeur Intel i5-750 4 coeurs / 4 threads, avec 16Go DDR3 de mémoire vive cadencés à 1333Mhz, 2 To d'espace de stockage, et une vitesse de transfert de données atteignant 100Mbps.

Le serveur DNS est configuré pour proposer un accès à l'application via le nom de domaine lpdm.kybox.fr.

L'ensemble des communications sont effectuées via le protocole TLS en HTTPS avec un certificat X.509 valide.

3.1.3 - Serveur de Fichiers

Le micro-service « storage », qui permet le transfert et le stockage de fichiers, est indirectement lié à un serveur HTTP Apache 2 sur lequel il enregistre les différents fichiers et stock en base de données leur adresse web.

Le serveur HTTP Apache 2 permet également la redirection de sous-domaines vers des ports pointant sur les micro-services. Ces redirections ont été mises en place à des fins d'opérations de tests et de vérifications lors du développement de l'application, et sont impérativement à désactiver lors de la mise en production de l'application.

3.2 - Bases de données

Les bases de données et schémas SQL suivants doivent être accessibles et à jour.

- Base de données liée au micro-service des commandes :

```
create table coupon
(
  id serial not null constraint coupon_pkey primary key,
  active boolean default false not null,
  amount double precision not null,
  code varchar not null,
  description varchar(255)
);

create table delivery
(
  id serial not null constraint delivery_pkey primary key,
  method varchar(30) not null,
  amount double precision not null
);

create table payment
(
  id serial not null constraint payment_pkey primary key,
  label varchar(20) not null constraint uk_ijunsgo2jk9wmfavo2hyibdud unique
);

create table "order"
(
  id serial not null constraint order_pkey primary key,
  customer_id integer not null,
  order_date timestamp,
  status_id integer,
  store_id integer,
  total double precision,
  payment_id integer constraint fkal550jx92fbea8sry5q4siyn1 references payment,
  coupon integer constraint fkiv08ux3yuf76s37722or3c6j7 references coupon,
  delivery integer default 1 constraint fkt5bqfi3ksom90m3utcjsndqcs references delivery,
  tax_amount double precision
);

create table ordered_product
(
  id integer not null constraint ordered_product_pkey primary key,
  price double precision,
  product_id integer,
  quantity integer,
  order_id integer not null constraint fk684l0g7wdvypylqnc39jfjkld references "order",
  tax double precision default 5.5 not null
);

create table invoice
(
  id serial not null constraint invoice_pkey primary key,
  order_id integer not null constraint invoice_order_fk references "order",
  reference varchar(50) not null
);
```

- Base de données liée au micro-service des magasins :

```
create table store
(
  id serial not null constraint store_pkey primary key,
  address_id integer not null,
  name varchar(50) not null
);
```

- Base de données liée au micro-service du stockage de fichiers :

```
create table storage
(
  id serial not null constraint storage_pkey primary key,
  file_type varchar(255) not null,
  owner integer not null,
  url varchar(255) not null
);
```

- Base de données liée au micro-service d'authentification :

```
create table app_role
(
  id serial not null constraint app_role_pkey primary key,
  role_name varchar(255)
);

create table app_user
(
  id serial not null constraint app_user_pkey primary key,
  active boolean,
  address_id integer,
  birthday date,
  email varchar(255) not null,
  first_name varchar(255),
  name varchar(255),
  password varchar(255) not null,
  registration_date timestamp,
  tel varchar(255)
);

create table app_role_app_user
(
  app_role_id integer not null constraint fki0rl707b9g0190knculbwhshs references
app_role,
```

- Base de données liée au stock de produits :

```
create table stock
(
  id serial  not null constraint stock_pk primary key,
  quantity integer not null,
  expire_date date not null,
  packaging varchar not null,
  unit_by_package integer not null,
  product_id integer not null,
  description varchar not null
);
```

- Base de données liée au micro-service des produits :

```
create table category
(
  id serial not null constraint category_pk primary key,
  name varchar not null
);

create table product
(
  id serial not null constraint product_pk primary key,
  name varchar not null,
  label varchar not null,
  price double precision not null,
  tax double precision not null,
  producer_id integer not null,
  picture varchar not null,
  category_id integer not null constraint category_product_fk references category,
  deactivate  boolean not null
);
```

- Base de données liée au micro-service de gestion d'adresses

```
CREATE SEQUENCE public.region_id_seq;
CREATE TABLE public.region (
   id INTEGER NOT NULL DEFAULT nextval('public.region_id_seq'),
   code varchar(3) NOT NULL,
   name varchar(255) NOT NULL,
   slug varchar(255) NOT NULL,
 CONSTRAINT region_pk PRIMARY KEY (id)
);
CREATE SEQUENCE public.department_id_seq;
CREATE TABLE public.department (
  id INTEGER NOT NULL DEFAULT nextval('public.department_id_seq'),
  region_code varchar(3) NOT NULL,
  code varchar(3) NOT NULL,
  name varchar(255) NOT NULL,
  slug varchar(255) NOT NULL,
  CONSTRAINT department_pk PRIMARY KEY (id)

);
CREATE SEQUENCE public.city_id_seq;
CREATE TABLE public.city (
  id INTEGER NOT NULL DEFAULT nextval('public.city_id_seq'),
  department_code varchar(3) NOT NULL,
  insee_code varchar(5) NULL,
  zip_code varchar(5) NULL,
  name varchar(255) NOT NULL,
  slug varchar(255) NOT NULL,
  gps_lat  DOUBLE PRECISION NOT NULL,
  gps_lng  DOUBLE PRECISION NOT NULL,
  CONSTRAINT city_pk PRIMARY KEY (id)

);
CREATE SEQUENCE public.address_id_seq;
CREATE TABLE public.address (
                id INTEGER NOT NULL DEFAULT nextval('public.address_id_seq'),
                street_name VARCHAR NOT NULL,
                street_number VARCHAR NOT NULL,
                complement VARCHAR,
                city_id INTEGER NOT NULL,
                CONSTRAINT address_pk PRIMARY KEY (id)
);
```

4 - PROCEDURE DE DEPLOIEMENT

4.1 - Déploiement des conteneurs

4.1.1 - Bases de données

Les conteneurs Docker de base de données doivent être déployés AVANT les conteneurs applicatifs.

Chaque conteneur de base de données est associé à un fichier docker-compose décrivant les différents paramètres d'exécution, à savoir :

- Le nom de la base de données
- Le port exposé
- Les variables d'environnement indiquant le nom de la base de données, l'identifiant de connexion ainsi que le mot de passe
- Le nom du conteneur
- Les chemins vers les volumes associés (fichiers SQL et volume de persistance)
- Le nom du réseau
- Le mode de redémarrage du conteneur

Exemple de fichier docker-compose au format yml :

```
version: '3.5'
services:
  db-product:
    image: postgres:11-alpine
    ports:
      - '28185:5432'
    container_name: LPDM-ProductDB
    environment:
      POSTGRES_DB: db_product
      POSTGRES_USER: usr_product
      POSTGRES_PASSWORD: pwd_product
    volumes:
      - './sql:/docker-entrypoint-initdb.d'
      - data:/var/lib/postgresql/data
    restart: always
    network_mode: bridge
volumes:
  data:
```

4.1.2 - Micro-services

Les conteneurs Docker applicatifs des différents micro-services composant l'ensemble de l'application doivent être lancés dans un ordre particulier.

1. Le premier conteneur à exécuter est celui du CloudConfig, qui permet de distribuer les différents fichiers de propriétés aux différents micro-services.
2. Le second conteneur à exécuter doit être celui d'Eureka, permettant ensuite aux autres micro-services de pouvoir s'enregistrer dans l'annuaire.
3. Logiquement, le troisième micro-service à exécuter, doit être l'API Gateway Zuul. C'est par ce micro-service que toutes les communications internes à l'application doivent transiter.
4. Une fois que les trois micro-services cités ci-dessus sont exécutés, l'ordre de lancement des autres applications n'a aucune incidence.

Chaque exécution d'un conteneur Docker applicatif doit renseigner les informations suivantes :

1. Nom du conteneur
2. Le port exposé
3. Un lien privilégié avec le conteneur de base de données associé
4. Le mode de redémarrage du conteneur
5. L'indication de ne pas utiliser de mémoire swap pour de se prémunir des ralentissements et de crashs de type OOM qui s'en suivent
6. Une variable d'environnement comprenant la clé de chiffrement afin de lire les propriétés sensibles
7. L'image du conteneur identifié dans le Docker Hub

Exemple d'exécution d'un conteneur en ligne de commande :

```
docker run -d
--name LPDM-OrderMS
-p 28083:28083
--link LPDM-OrderDB
--restart always
--memory-swappiness=0
-e 'JAVA_TOOL_OPTIONS=-Djasypt.encryptor.password=$KEY'
vyjorg/lpdm-order:latest
```

Note :

Avec cette configuration, le conteneur n'est soumis à aucune limitation de mémoire.

Après une étude sur la monté en charge de chaque conteneur, il serait pertinent de limiter la consommation de mémoire en l'indiquant au lancement du conteneur.

4.1.3 - Intégration continue

Les dépôts associés à chaque micro-service, contenant les sources du code, sont disponibles sur l'organisation GitHub à l'adresse internet https://github.com/vyjorg

C'est, reposant sur cette organisation de dépôts, qu'intervient le serveur d'automatisations Java, open source, Jenkins, sur lequel sont liés chaque dépôt dont les commits initialisent automatiquement des builds suivant une procédure détaillée dans un fichier Jenkinsfile à la racine des dépôts.

Exemple d'un fichier Jenkinsfile utilisant le mode pipeline, situé à la racine d'un dépôt GitHub :

```
pipeline {
  agent any
    tools { maven 'Apache Maven 3.5.2' }
    environment { KEY = "" }
    stages{
      stage('Checkout') {
       steps { git 'https://github.com/vyjorg/LPDM-Order' }
      }
      stage('Load Key') {
        steps {
          script {
            configFileProvider([configFile(fileId:
            '2bd4e734-a03f-4fce-9015-aca988614b4e',
            targetLocation: 'lpdm.key')]) {
              lpdm_keys = readJSON file: 'lpdm.key'
              KEY = lpdm_keys.lpdm
            }
          }
        }
      }
      stage('Tests') {
        steps { sh 'mvn clean test' }
        post {
          always { junit 'target/surefire-reports/**/*.xml' }
          failure { error 'The tests failed'}
        }
      }
      stage('Push to DockerHub') {
        steps { sh 'mvn clean package' }
      }
      stage('Deploy') {
        steps {
          sh "docker stop LPDM-OrderMS
              || true && docker rm LPDM-OrderMS || true"
          sh "docker pull vyjorg/lpdm-order:latest"
          sh "docker run -d
              --name LPDM-OrderMS
              -p 28083:28083
              --link LPDM-OrderDB
              --restart always
              --memory-swappiness=0
              -e 'JAVA_TOOL_OPTIONS=-Djasypt.encryptor.password=$KEY'
              vyjorg/lpdm-order:latest"
        }
      }
   }
}
```

4.1.4 - Variables d'environnement

Chaque micro-service lié à une base de données reçoit, dans un fichier de propriétés, le data-source JDBC ainsi que les identifiants de connexion à cette dernière, par l'intermédiaire du micro-service CloudConfig.

Dans un soucis de sécurité, ces informations sont chiffrées dans les fichiers de propriétés. Afin de procéder à la lecture correcte des informations de connexion, une clé de chiffrement doit être injectée en tant que variable d'environnement aussi bien pour les tests que pour la mise en production des différents conteneur Docker.

Le tableau ci-dessous énumère les micro-services nécessitant une variable d'environnement.

Micro-service	Obligatoire	Description
Order	oui	Data-source & API Paypal
Product	oui	Data-source
Stock	oui	Data-source
Location	oui	Data-source
Store	oui	Data-source
Storage	oui	Data-source
User	oui	Clé de chiffrement JWT

Définissez les variables d'environnement nécessaires comme suit :

```
-Djasypt.encryptor.password={clé de chiffrement}
```

5 - PROCEDURE DE DEMARRAGE / ARRET

5.1 - Bases de données

Étant donné que chaque base de données est exécutée dans un conteneur Docker par l'intermédiaire d'un fichier docker-compose, la procédure de démarrage se réalise à la ligne de commande :

```
dossier-du-fichier-docker-compose/docker-compose up
```

La procédure d'arrêt d'un conteneur de base de donnée se réalise comme suit :

```
docker stop NOM_DU_CONTENEUR
```

Il est à noter que chaque conteneur de base de données est lié à un volume de données qui lui est propre. De ce fait, l'arrêt d'un conteneur n'a aucune incidence sur les données persistées, et seront rechargées dans la base de données lorsque le conteneur sera de nouveau lancé.

5.2 - Micro-services

Les micro-services, à l'instar des base de données, ne bénéficient pas de fichiers docker-compose.

La procédure de démarrage se réalise suivant le schéma suivant :

```
docker run -d –name {nom_du_contenteur} -p {ports_externe:interne}
--link {nom_du_conteneur_DB} --restart {mode_de_redémarrage}
--memory-swapiness=0 -e
'JAVA_TOOL_OPTIONS={variable_environnement}'
```

Exemple pour le micro-service des commandes :

```
docker run -d --name LPDM-OrderMS -p 28083:28083
--link LPDM-OrderDB --restart always --memory-swappiness=0
-e 'JAVA_TOOL_OPTIONS=-Djasypt.encryptor.password=$KEY'
vyjorg/lpdm-order:latest
```

La procédure d'arrêt reste identique à celle d'un conteneur pour une base de données :

```
docker stop NOM_DU_CONTENEUR
```

6 - PROCEDURE DE MISE A JOUR

6.1 - Bases de données

La mise à jour d'une base de données peut s'effectuer de 2 façons, soit :

- Par le micro-service associé, mis à jour, à l'aide de la propriété ddl-auto = update
- Par une plate-forme d'administration telle que PgAdmin

6.2 - Micro-services

La mise à jour d'un micro-service peut se réaliser de 2 manières différentes suivant le degré de modification.

Si seules les propriétés sont impactées par la mise à jour, dans se cas, ces dernières seront transmises automatiquement au micro-service concerné en quelques secondes, sans qu'aucune autre tâche ne soit nécessaire.

En revanche, la procédure de mise à jour du code source d'un micro-service doit s'achever (après compilation) par le lancement de la création de l'image Docker associée, et son transfert sur le site DockerHub, afin que l'image puisse être ensuite récupérer lors de la procédure d'exécution du conteneur.

Actuellement, cette tâche est automatisé à l'aide de Jenkins lors d'un commit sur la branche « master » d'un dépôt GitHub.

Jenkins récupérera le code source mis à jour, lancera les tests, créera l'image Docker qui sera transférée sur le site DockerHub et déploiera automatiquement l'application.

7 - SUPERVISION/MONITORING

7.1 - Supervision de l'application

La supervision de l'ensemble des micro-services se réalise à l'aide du serveur Jenkins sur lequel sont répertoriés les différents dépôt GitHub associés.

La place du marché

CLASSROOM

Folder name: Vyjorg
Projet 12 - DA Java / JEE - Openclassrooms

Repositories (11)

S	M	Name ↓	Description
		LPDM-Auth	La Place du marché - MS-Authenticazion
		LPDM-Config	Microservice Config
		LPDM-Eureka	microservice lié à Eureka
		LPDM-Location	Microservice Location
		LPDM-Order	Microservice Order
		LPDM-Product	Microservice Product
		LPDM-Stock	Microservice Stock
		LPDM-Storage	Microservice storage
		LPDM-Store	Microservice Store
		LPDM-User	
		LPDM-Zuul-Server	Zuul Server for LPDM

Il est également possible d'avoir une vue d'ensemble des différents conteneurs en cours d'exécution à l'aide de la commande en ligne suivante :

```
docker stats
```

CONTAINER ID	NAME	CPU %	MEM USAGE / LIMIT	MEM %	NET I/O	BLOCK I/O	PIDS
07f54723aae7	LPDM-ZuulMS	0.13%	709.6MiB / 15.66GiB	4.43%	20MB / 23.6MB	24.6kB / 0B	46
670fa719fb2f	LPDM-OrderMS	0.18%	623.8MiB / 15.66GiB	3.89%	12.5MB / 14.1MB	114MB / 0B	44
ec19fb1d0e45	LPDM-StoreMS	0.08%	532.4MiB / 15.66GiB	3.32%	12.1MB / 13.8MB	4.6MB / 0B	42
6f97870db9f3	LPDM-UserMS	1.74%	388.6MiB / 15.66GiB	2.42%	21.8MB / 20.5MB	26.5MB / 0B	45
a24397f4094e	LPDM-ProductMS	0.08%	437.1MiB / 15.66GiB	2.73%	62.8MB / 65MB	31.3MB / 0B	44
3bee8abbb118	LPDM-AuthMS	0.09%	464.5MiB / 15.66GiB	2.90%	68.3MB / 57.1MB	30.7MB / 0B	44
6c503b41923b	LPDM-StockMS	0.09%	959.2MiB / 15.66GiB	5.98%	30.5MB / 28.5MB	19.5MB / 0B	44
282986c32cab	LPDM-LocationMS	0.15%	652.7MiB / 15.66GiB	4.07%	47.4MB / 48.5MB	15.1MB / 0B	44
adac9656dda5	LPDM-StorageMS	0.11%	570.7MiB / 15.66GiB	3.56%	14.5MB / 16.5MB	35.9MB / 0B	42
88ea3145a126	LPDM-ConfigMS	0.09%	314.4MiB / 15.66GiB	1.96%	2.21MB / 1.48MB	70.8MB / 3.61MB	36
20a4e2dc47e9	LPDM-ProductDB	0.00%	18.27MiB / 15.66GiB	0.11%	2.38MB / 5.92MB	9.59MB / 216MB	17

8 - PROCEDURE DE SAUVEGARDE ET RESTAURATION

Les données persistées sont stockées dans des volumes Docker.

La restauration des données, suite à un arrêt de conteneur, est donc automatique.

Si un besoin de sauvegarde des données persistées en base de données est nécessaire, il suffit de lancer la commande suivante dans un terminal connecté au serveur :

```
docker exec -t NOM_DU_CONTENEUR pg_dumpall -c -U postgres > NOM_DU_FICHIER.sql
```

Si un besoin de restauration des données persistées en base de données est nécessaire, après création du fichier SQL (comme montré ci-dessus) il suffira de lancer la commande suivante pour mettre à jour la base de données :

```
cat NOM_DU_FICHIER.sql | docker exec -i NOM DU CONTENEUR psql -U postgres
```

Les différents micro-services étant indépendants des données persistées, aucune procédure de sauvegarde et de restauration des données n'est nécessaire.

Si un conteneur s'arrête pour n'importe quelle raison, ce dernier sera relancé de façon automatique grâce au paramètre « --restart always » comme mentionné dans la partie 5.2 du présent document.

9 - GLOSSAIRE
