OC PIZZA

Projet 8

Dossier d'exploitation
Version 1.0

Auteur
Charlotte Vanhuyse
Développeur d'application junior

TABLE DES MATIÈRES

Versions **5**

Introduction **6**

Objet du document 6

Références 6

Pré-requis **7**

Caractèristique du serveur dédié 7

Serveur dédié 7

Caractéristiques techniques 7

Bases de données 8

Caractéristiques techniques 8

Microservices 8

Caractéristiques techniques 8

Installation sur le serveur dédié 9

Accéder au serveur 9

Installer Git 9

Installer Docker 9

Installer Docker-Compose 10

Procédure de déploiement **11**

Diagramme de déploiement 11

diagramme de déploiement 11

Importer le code source 11

Déployer l'application 12

Le fichier docker-compose.yml 12

La base de données 17

Les fichiers Dockerfile des microservices 18

Config-server 18

Eureka-server 18

Authentification 19

Gestion-commande 19

Gestion-administration 19

Api-gateway 20
UI-client 20
UI-collaborateur 21
Lancement 21
Vérification 21

Procédure de démarrage / arrêt 23
Démarrage des conteneurs 23
démarrer tous les conteneurs 23
démarrer un seul conteneur 23
redémarrer un conteneur 23
Arrêt des conteneurs 23
arrêter tous les conteneurs 23
arrêter un seul conteneur 23

Procédure de mise à jour 24
Mise à jour des conteneurs 24
Choisir une version du code 24
Mettre à jour tous les conteneurs 24
Mettre à jour un seul conteneur 24
Mise à jour du système d'exploitation 25

Supervision/Monitoring 26
Supervision via Docker 26
les statistiques 26
les logs 26
Supervision via AWS 26
Surveillance des métriques 26
Surveillance des logs 27

Procédure de sauvegarde et restauration 28
Bases de données 28
Sauvegarde 28
sauvegarde automatique 28
sauvegarde manuelle 28
Restauration 28
à partir d'un volume 28

à partir d'un fichier 28
Supervision via AWS 29
choisir la version du code 29
restaurer le conteneur 29
Glossaire **31**

1 - Versions

Auteur	Date	Description	Version
Charlotte Vanhuyse	16/11/2020	Création du document	1.0

2 - Introduction

2.1 -Objet du document

Le présent document constitue le dossier d'exploitation de l'application OC Pizza.

Il a pour but de préciser la démarche de déploiement et de maintenance de l'application.

2.2 -Références

Pour de plus amples informations, se référer :

1. **Projet 8 - Dossier de conception fonctionnelle** : Dossier de conception fonctionnelle de l'application
2. **Projet 8 - Dossier de conception technique** : Dossier de conception technique de l'application
3. **PV - projet OC Pizza** : Procès verbal de l'application

3 - Pré-requis

3.1 -Caractèristique du serveur dédié

3.1.1 - Serveur dédié

L'application sera déployée sur un **serveur dédié virtuel AWS**.

On estime les besoins minimums en ressources de notre application OC Pizza (OCPizza/Apace/MySQL) a :

- 2CPU,
- 4G de RAM
- 4G d'espace disque

Commencez par créer un compte AWS et une paire de clés pour pouvoir accéder aux instances.

3.1.1.1 - Caractéristiques techniques

On choisira une **instance Amazon EC2 de type T3**.
Les instances T3 sont des instances à capacité extensible qui fournissent un niveau de départ en matière de capacité de CPU, avec la possibilité de dépasser le seuil de base. Elles sont adaptées aux sites web qui sont confrontés à des courbes de trafic non linéaires.

Créez une instance EC2 de type **T3.Large** avec Amazon Linux 2 AMI qui fournit un noyau Linux 4.14 pour des performances optimales sur Amazon EC2.
L'instance T3.large fournit 2CPU et 8GiB de RAM. On pourra redimensionner cette instance par la suite si les besoins en ressources de l'application augmentent.

Pour l'espace disque on ajoutera un volume **EBS de type SSD** à usage général comme volume de stockage de l'instance. On commencera par 8GB mais ces volumes sont modifiables sans avoir à arrêter notre instance (type de volume, taille du volume et capacité IOPS).

3.1.2 - Bases de données

Il y a trois bases de données différentes, chacune n'est accessible que par un seul microservice :

- dbauth,
- dbcommande,
- dbadministration.

3.1.2.1 - Caractéristiques techniques

Les bases de données utilisent un serveur MySQL.
Les bases de données sont déployées dans un conteneur Docker. Le conteneur Docker sera construit à partir du fichier docker-compose.yml et utilisera l'image de bitnami/mysql

3.1.3 - Microservices

L'application est divisée en microservices :

- 2 pour le Frontend
  - ui-client
  - ui-collaborateur
- 6 pour le Backend
  - config-server
  - eureka-server
  - api-gateway
  - authentification
  - gestion des commandes
  - gestion administrative.

3.1.3.1 - Caractéristiques techniques

Chaque microservice est développé avec SpringBoot qui intègre un serveur Tomcat. Un microservice est déployé dans son propre conteneur Docker.

3.2 -Installation sur le serveur dédié

L'installation de Docker et Docker-Compose sur le serveur dédié est nécessaire au déploiement des conteneurs sur le serveur. Git sera utilisé pour importer le code source de l'application ainsi que les mises à jour.

3.2.1 - Accéder au serveur

Connectez-vous à votre instance EC2 à l'aide de votre paire de clés créées précédemment. Accédez au terminal de votre serveur et faites une mise à jour de Amazon Linux 2 AMI.

```
sudo yum update
```

3.2.2 - Installer Git

Accédez au terminal de votre serveur et installez Git.

```
sudo yum install -y git
```

3.2.3 - Installer Docker

Accédez au terminal de votre serveur et installez Docker.

```
sudo yum install -y docker
```

Paramétrez les autorisations

```
sudo usermod -a -G docker ec2-user
```

Démarrez Docker.

```
sudo service docker start
```

Configurez le démarrage automatique de Docker.

```
sudo chkconfig docker on
```

Redémarrez votre instance pour vérifier que votre installation est réussie.

```
sudo reboot
```

3.2.4 - Installer Docker-Compose

Accédez au terminal de votre serveur et installez docker-compose.

```
sudo curl -L
```

```
https://github.com/docker/compose/releases/download/1.21.0/docker-compose-`uname -s`-`uname -m` | sudo tee /usr/local/bin/docker-compose > /dev/null
```

Paramétrez les autorisations.

```
sudo chmod +x /usr/local/bin/docker-compose
```

Vérifiez la version de docker-compose installée.

```
docker-compose --version
```

4 - Procédure de déploiement

4.1 -Diagramme de déploiement

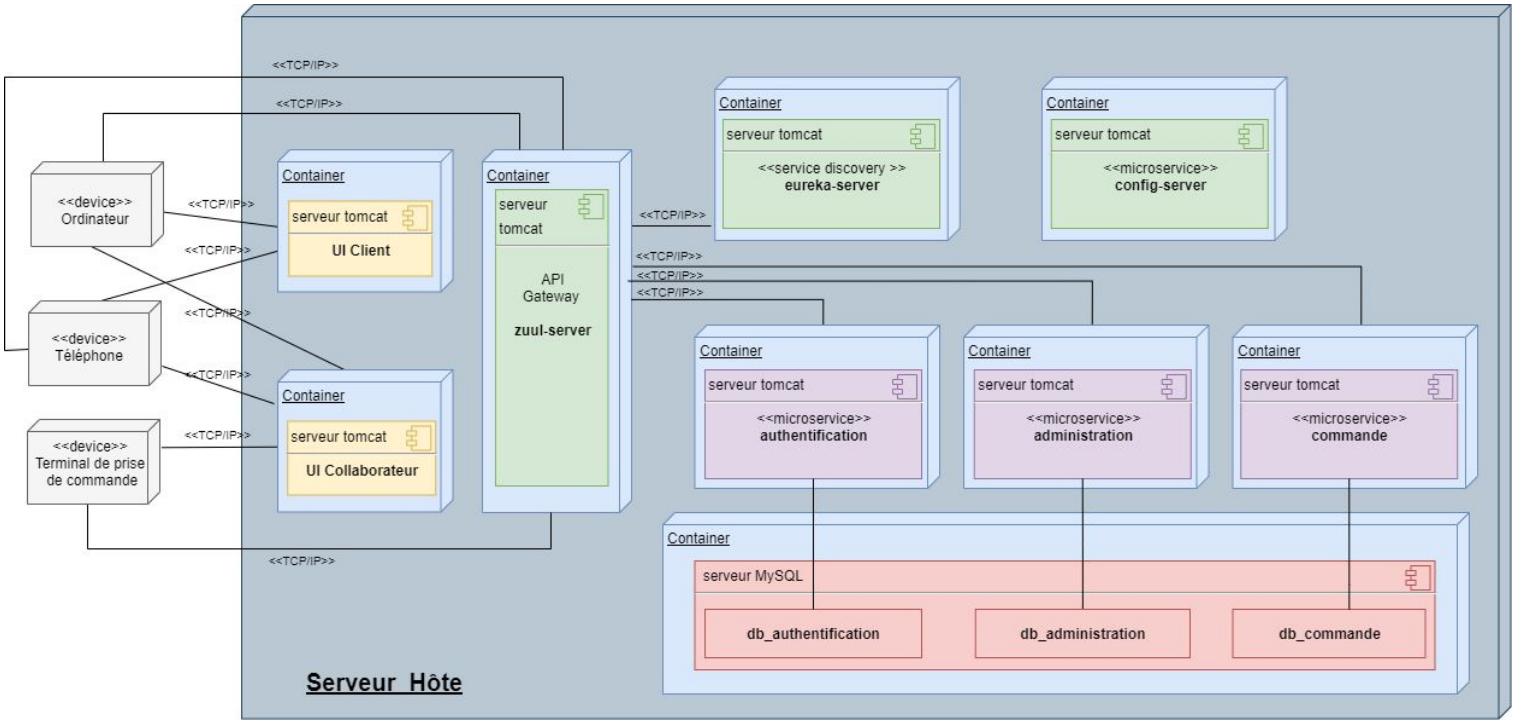

diagramme de déploiement

4.2 -Importer le code source

Le code source de l'application se trouve dans un repository **Github.**
Accédez au terminal du serveur et cloner le repository :

```
git clone https://github.com/vancharlotte/projet8-ocpizza.git
```

4.3 -Déployer l'application

4.3.1 - Le fichier docker-compose.yml

Le fichier docker-compose.yml est présent à la racine du projet. Dans celui-ci nous avons décrit l'ensemble des ressources et services nécessaires au déploiement de l'application.

```
version: '3'

services:
  db:
    image: bitnami/mysql:8.0.22-debian-10-r6
    container_name: db
    restart: always
    environment:
          MYSQL_ROOT_PASSWORD: password
          MYSQL_DATABASE: db_auth
          MYSQL_USER: admin
          MYSQL_PASSWORD: admin123!
     ports:
          - "6033:3306"
     volumes:
          - ./databases:/docker-entrypoint-initdb.d
          - dbdata:/bitnami/mysql/data
     networks:
          - ocpizza-network

  config-server:
    image: config-server
    container_name: config-server
    build:
        context: ./config-server
        dockerfile: dockerfile
     ports:
          - "9101:9101"
     expose:
          - "9101"
     environment:
```

```
      SPRING_CONFIG_SERVER_URI: https://github.com/vancharlotte/config-server-ocpizza.git
    networks:
      - ocpizza-network

eureka-server:
  depends_on:
    - config-server
  image: eureka-server
  container_name: eureka-server
  build:
    context: ./eureka-server
    dockerfile: dockerfile
  ports:
    - "8761:8761"
  expose:
    - "8761"
  environment:
    SPRING_CONFIG_URI: http://config-server:9101
    EUREKA_SERVER_URL : http://eureka-server:8761/eureka
  networks:
    - ocpizza-network

authentification:
  depends_on:
    - config-server
    - eureka-server
    - db
  image: authentification
  container_name: authentification
  restart: always
  build:
    context: ./authentification
    dockerfile: dockerfile
  ports:
    - "9001:9001"
  expose:
    - "9001"
  environment:
    SPRING_CONFIG_URI: http://config-server:9101
    EUREKA_SERVER_URL : http://eureka-server:8761/eureka
    SPRING_DATASOURCE_URI : jdbc:mysql://db:3306/dbauth?createDatabaseIfNotExist=true
```

```
    networks:
        - ocpizza-network

  gestion-commande:
    depends_on:
         - config-server
         - eureka-server
         - db
    image: gestion-commande
    container_name: gestion-commande
    restart: always
    build:
        context: ./gestion-commande
        dockerfile: dockerfile
    ports:
        - "9002:9002"
    expose:
        - "9002"
    environment:
        SPRING_CONFIG_URI: http://config-server:9101
        EUREKA_SERVER_URL : http://eureka-server:8761/eureka
        SPRING_DATASOURCE_URI:dbc:mysql://db:3306/dbcommande?createDatabaseIfNotExist=true
    networks:
        - ocpizza-network

  gestion-administration:
    depends_on:
         - config-server
         - eureka-server
         - db
    image: gestion-administration
    container_name: gestion-administration
    restart: always
    build:
        context: ./gestion-administration
        dockerfile: dockerfile
    ports:
        - "9003:9003"
    expose:
        - "9003"
    environment:
```

```
      SPRING_CONFIG_URI: http://config-server:9101
      EUREKA_SERVER_URL : http://eureka-server:8761/eureka
SPRING_DATASOURCE_URI:dbc:mysql://db:3306/dbadministration?createDatabaseIfNotExist=true
    networks:
      - ocpizza-network

  api-gateway:
    depends_on:
       - config-server
       - eureka-server
       - authentification
       - gestion-commande
       - gestion-administration
    image: api-gateway
    container_name: api-gateway
    restart: always
    build:
      context: ./api-gateway
      dockerfile: Dockerfile
    ports:
      - "9004:9004"
    expose:
      - "9004"
    environment:
      SPRING_CONFIG_URI: http://config-server:9101
      EUREKA_SERVER_URL : http://eureka-server:8761/eureka
    networks:
      - ocpizza-network

  ui-client:
    depends_on:
       - config-server
       - eureka-server
    image: ui-client
    container_name: ui-client
    restart: always
    build:
      context: ./ui-client
      dockerfile: Dockerfile
    ports:
      - "80:8080"
```

```
    expose:
      - "8080"
    environment:
      SPRING_CONFIG_URI: http://config-server:9101
      EUREKA_SERVER_URL : http://eureka-server:8761/eureka
    networks:
      - ocpizza-network

  ui-collaborateur:
    depends_on:
       - config-server
       - eureka-server
    image: ui-collaborateur
    container_name: ui-collaborateur
    restart: always
    build:
      context: ./ui-collaborateur
      dockerfile: Dockerfile
    ports:
      - "80:8081"
    expose:
      - "8081"
    environment:
      SPRING_CONFIG_URI: http://config-server:9101
      EUREKA_SERVER_URL : http://eureka-server:8761/eureka
    networks:
      - ocpizza-network

volumes:
  dbdata:

network:
  ocpizza-network:
```

4.3.2 - La base de données

```
services:
 db:
   image: bitnami/mysql:8.0.22-debian-10-r6
   container_name: db
   restart: always
   environment:
         MYSQL_ROOT_PASSWORD: password
        MYSQL_DATABASE: db_auth
        MYSQL_USER: admin
        MYSQL_PASSWORD: admin123!
    ports:
        - "6033:3306"
    volumes:
        - ./databases:/docker-entrypoint-initdb.d
        - dbdata:/bitnami/mysql/data
    networks:
        - ocpizza-network
```

On utilise l'image Docker de "binami/mysql:8.0.22-debin-10-r6" pour la construction du conteneur "db".

Les scripts de la base de données sont présents dans le fichier "databases" qui se trouve à la racine du projet. Il contient les requêtes nécessaires à la création des trois bases de données.

Les scripts sont récupérés à la construction du conteneur "db" et les base de données sont créées. Vous pouvez accéder à la base de données via MySQL Workbench.

La persistance des données se fait dans le volume "ocpizza_dbdata" créé en même temps que le conteneur "db". Un dossier est créé en local sur votre serveur.

Avant de démarrer le conteneur "db", vous pouvez modifier les variables d'environnement directement dans le fichier "docker-compose.yml".

Par défaut, MYSQL_ROOT_HOST = root

```
        MYSQL_ROOT_PASSWORD: password
        MYSQL_DATABASE: db_auth
        MYSQL_USER: admin
        MYSQL_PASSWORD: admin123!
```

4.3.3 - Les fichiers Dockerfile des microservices

Pour chaque microservice, un fichier “Dockerfile” va décrire l’image Docker nécessaire pour le projet.

La première étape consiste à lancer un “maven clean install” pour construire le jar du microservice.

La deuxième étape va lancer le jar du microservice.

4.3.3.1 - Config-server

```
# Build stage
FROM maven:3.5-jdk-8-alpine AS build
WORKDIR /config-server
COPY pom.xml /
COPY src /src
RUN mvn -f /pom.xml clean install

# Run stage
FROM openjdk:8-jdk-alpine
COPY --from=build /target/config-server.jar /config-server.jar
EXPOSE 9101
ENTRYPOINT ["java", "-jar", "/config-server.jar"]
```

4.3.3.2 - Eureka-server

```
# Build stage
FROM maven:3.5-jdk-8-alpine AS build
WORKDIR /eureka-server
COPY pom.xml /
COPY src /src
RUN mvn -f /pom.xml clean install

# Run stage
FROM openjdk:8-jdk-alpine
COPY --from=build /target/eureka-server.jar /eureka-server.jar
EXPOSE 8761
ENTRYPOINT ["java", "-jar", "/eureka-server.jar"]
```

4.3.3.3 - Authentification

```
# Build stage
FROM maven:3.5-jdk-8-alpine AS build
WORKDIR /authentification
COPY pom.xml /
COPY src /src
RUN mvn -f /pom.xml clean install

# Run stage
FROM openjdk:8-jdk-alpine
COPY --from=build /target/authentification.jar /authentification.jar
EXPOSE 9001
ENTRYPOINT ["java", "-jar", "/authentification.jar"]
```

4.3.3.4 - Gestion-commande

```
# Build stage
FROM maven:3.5-jdk-8-alpine AS build
WORKDIR /gestion-commande
COPY pom.xml /
COPY src /src
RUN mvn -f /pom.xml clean install

# Run stage
FROM openjdk:8-jdk-alpine
COPY --from=build /target/gestion-commande.jar /gestion-commande.jar
EXPOSE 9002
ENTRYPOINT ["java", "-jar", "/gestion-commande.jar"]
```

4.3.3.5 - Gestion-administration

```
# Build stage
FROM maven:3.5-jdk-8-alpine AS build
WORKDIR /gestion-administrative
COPY pom.xml /
COPY src /src
```

```
RUN mvn -f /pom.xml clean install

# Run stage
FROM openjdk:8-jdk-alpine
COPY --from=build /target/gestion-administrative.jar /gestion-administrative.jar
EXPOSE 9003
ENTRYPOINT ["java", "-jar", "/gestion-administrative.jar"]
```

4.3.3.6 - Api-gateway

```
# Build stage
FROM maven:3.5-jdk-8-alpine AS build
WORKDIR /api-gateway
COPY pom.xml /
COPY src /src
RUN mvn -f /pom.xml clean install

# Run stage
FROM openjdk:8-jdk-alpine
COPY --from=build /target/api-gateway.jar /api-gateway.jar
EXPOSE 9004
ENTRYPOINT ["java", "-jar", "/api-gateway.jar"]
```

4.3.3.7 - UI-client

```
# Build stage
FROM maven:3.5-jdk-8-alpine AS build
WORKDIR /ui-client
COPY pom.xml /
COPY src /src
RUN mvn -f /pom.xml clean install

# Run stage
FROM openjdk:8-jdk-alpine
COPY --from=build /target/ui-client.jar /ui-client.jar
EXPOSE 8000
ENTRYPOINT ["java", "-jar", "/ui-client.jar"]
```

4.3.3.8 - UI-collaborateur

```
# Build stage
FROM maven:3.5-jdk-8-alpine AS build
WORKDIR /ui-collaborateur
COPY pom.xml /
COPY src /src
RUN mvn -f /pom.xml clean install

# Run stage
FROM openjdk:8-jdk-alpine
COPY --from=build /target/ui-collaborateur.jar /ui-collaborateur.jar
EXPOSE 8001
ENTRYPOINT ["java", "-jar", "/ui-collaborateur.jar"]
```

4.3.4 - Lancement

Avant de lancer le déploiement de l'application, vous pouvez modifier la valeur associées aux variables d'environnement directement dans le fichier docker-compose.yml

```
cd ocpizza

nano docker-compose.yml
```

Pour lancer le déploiement de l'application sur le serveur, placez-vous à la racine du projet et entrez la commande :

```
docker-compose up -d
```

4.3.5 - Vérification

Pour vérifier que tous les conteneurs ont été correctement lancés, placez-vous à la racine du projet et entrez :

```
docker-compose ps
```

L'output attendu est :

```
        Name                   Command               State           Ports
-------------------------------------------------------------------------------------
db                    /opt/bitnami/scripts/mysql ...   Up      0.0.0.0:6033->3306/tcp
config-server          java -jar /config-server.jar    Up      0.0.0.0:9101->9101/tcp
eureka-server          java -jar /eureka-server.ja     Up      0.0.0.0:8761->8761/tcp
authentification       java -jar / authentification.jar Up     0.0.0.0:9001->9001/tcp
gestion-commande      java -jar / gestion-commande.jar Up      0.0.0.0:9002->9002/tcp
gestion-administrative java -jar / gestion-administrative.jar Up 0.0.0.0:9003->9003/tcp
api-gateway           java -jar api-gateway.jar        Up      0.0.0.0:9004->9004/tcp
ui-client              java -jar /ui-client            Up      0.0.0.0:8000->8000/tcp
ui-collaborateur       java -jar /ui-collaborateur.j   Up      0.0.0.0:8000->8001/tcp
```

5 - Procédure de démarrage / arrêt

5.1 -Démarrage des conteneurs

5.1.1 - démarrer tous les conteneurs

Pour lancer le déploiement des conteneurs sur le serveur, placez-vous à la racine du projet et entrez la commande :

```
docker-compose up -d
```

5.1.2 - démarrer un seul conteneur

Pour démarrer un seul conteneur, placez-vous à la racine du projet et entrez la commande :

```
docker-compose run -d nomduservice
```

Attention l'image Docker doit avoir été créée auparavant.

5.1.3 - redémarrer un conteneur

Pour redémarrer un conteneur, placez-vous à la racine du projet et entrez la commande :

```
docker-compose restart -d nomduservice
```

5.2 -Arrêt des conteneurs

5.2.1 - arrêter tous les conteneurs

Pour arrêter un conteneur, placez-vous à la racine du projet et entrez la commande :

```
docker-compose down -d nomduservice
```

5.2.2 - arrêter un seul conteneur

Pour arrêter un conteneur, placez-vous à la racine du projet et entrez la commande :

```
docker-compose down -d nomduservice
```

6 - Procédure de mise à jour

6.1 -Mise à jour des conteneurs

6.1.1 - Choisir une version du code

Se placer à la racine du projet sur le serveur dédié et importer la dernière version de l'application :

Mettez à jour votre répertoire locale avec les données du repository distant :

```
git fetch --all –tags
```

Listez les différents tags disponibles :

```
git tag
```

Basculez sur la version souhaitée (où <tag> est le nom du tag souhaité et <branch> le nom de la branche créée)

```
git checkout tags/<tag> -b <branch>
```

6.1.2 - Mettre à jour tous les conteneurs

Pour mettre à jour tous les conteneurs, placez-vous à la racine du et entrez :

```
docker-compose up --build -d
```

6.1.3 - Mettre à jour un seul conteneur

Pour mettre à jour un conteneur, placez-vous à la racine du et entrez :

```
docker-compose up --build --force-recreate -d nomduservice
```

6.2 -Mise à jour du système d'exploitation

Sur le serveur dédié est installé Amazon Linux 2 spécialement optimisé pour Amazon EC2. Amazon a créé sa propre distribution compacte et parfaitement intégrée à l'univers de services d'AWS pour animer les VM Linux exécutées sous AWS.

Vérifiez si des mises à jour sont disponibles :

```
sudo yum updateinfo
```

Avant de faire une mise à jour, prenez note de la version de Linux utilisée actuellement.

```
uname -mrs
```

Mettez l'OS à jour.

```
sudo yum --security update
```

Si vous ne souhaitez mettre à jour que les mises à jour de sécurité c'est possible avec la ligne de commande :

```
sudo yum --security update
```

Redémarrez le système pour vous assurer que la mise à jour a fonctionné.

```
sudo reboot
```

Vérifier la version de Linux.

```
uname -mrs
```

7 - Supervision/Monitoring

7.1 -Supervision via Docker

7.1.1 - les statistiques

Pour obtenir les statistiques de consommation des conteneurs en cours d'exécution sur votre serveur, entrez la commande :

```
docker stats
```

7.1.2 - les logs

Pour afficher l'historique des logs d'un conteneur, entrez la commande :

```
docker logs -f nomduservice
```

7.2 -Supervision via AWS

7.2.1 - Surveillance des métriques

On utilise **CloudWatch** pour analyser les besoins en ressources de l'application :

On peut observer l'utilisation des ressources sur le serveur en direct en observant les métriques liées à notre instance EC2.

Dans le tableau de bord EC2, sélectionner l'onglet :

```
» Instance
```

Sélectionner l'instance que vous souhaitez analyser et cliquez sur l'onglet :

```
» Surveillance
```

On peut observer les volumes EBS pour analyser l'utilisation des ressources liées à la mémoire (espace disque, bande passante…).

Dans le tableau de bord EC2, sélectionner l'onglet :

» Elastik Bloc Store

Sélectionner le volume liée à l'instance que vous souhaitez analyser et cliquez sur l'onglet :

» Surveillance

Il est possible de créer des alarmes pour chacun des métriques observables. Cela peut vous permettre d'ajuster les ressources de votre instance en fonction de vos besoins.

7.2.2 - Surveillance des logs

On utilise **CloudWatchLogs** pour recueillir et analyser les logs de notre application.

CloudWatchLogs va enregistrer les logs pour chaque conteneur.

Il est possible de les consulter sur votre compte AWS :

Dans le tableau de bord CloudWatch, sélectionner l'onglet :

» Logs (journaux)

Sélectionner le groupe de logs associé asscocié à votre instance.

Il est possible de paramétrer des alertes pour vous prévenir lorsqu'une erreur intervient ou si elle est récurrente par exemple.

8 - Procédure de sauvegarde et restauration

8.1 -Bases de données

8.1.1 - Sauvegarde

8.1.1.1 - sauvegarde automatique

Les données présentes dans les bases de données du conteneur “db” sont sauvegardées automatiquement dans le volume du conteneur.

8.1.1.2 - sauvegarde manuelle

Créez un dump de la base de données.
Le fichier dump est placé dans le volume du conteneur “db” et il est copié dans le dossier “dbdata” en local sur le serveur dédié.

```
docker-compose exec db mysql -u root -p db auth > bitnami/data/dump-XXXXXXXX.sql
```

8.1.2 - Restauration

8.1.2.1 - à partir d’un volume

Lorsque qu’on redémarre le conteneur « db », les données persistées dans le volume «dbdata» sont récupérées.

8.1.2.2 - à partir d’un fichier

Sélectionnez un dump précédent de la base de données présent dans le dossier « bitnami/mysql/data/ » et restaurez les données.

```
docker-compose exec db mysql -u root -p -D db_auth < /bitnami/mysql/data/
dump-XXXXXXXX.sql.
```

8.2 -Supervision via AWS

8.2.1 - choisir la version du code

Listez les différents tags disponibles :

```
git tag
```

Basculez sur la version souhaitée (où <tag> est le nom du tag souhaité et <branch> le nom de la branche créée).

```
git checkout tags/<tag> -b <branch>
```

8.2.2 - restaurer le conteneur

Pour mettre à jour un conteneur, placez-vous à la racine du projet et entrez la commande:

```
docker-compose up --build --force-recreate -d nomduservice
```

9 - GLOSSAIRE

AMI	Amazon Machine Image
AWS	Amazon Web Service
BDD	Base de données
EBS	Elastik Block Store