|  |
| --- |
| **OC Pizza**  **PizzaFlow**  Dossier d'exploitation  Version 1.0 |
| **Auteur**  Paul-Emmanuel DOS SANTOS FACAO  *Analyste programmeur* |

Table des matières

1 - Versions 7

2 - Introduction 8

2.1 - Objet du document 8

2.2 - Références 8

2.3 - Diagramme de déploiement 9

3 - Pré-requis 10

3.1 - Création d'une paire de clé 10

3.2 - Système 12

3.2.1 - Serveur de Base de données User 12

3.2.1.1 - Description 12

3.2.1.2 - Caractéristiques techniques 12

3.2.2 - Serveur de Base de données Stock 13

3.2.2.1 - Description 13

3.2.2.2 - Caractéristiques techniques 14

3.2.3 - Serveur de Base de données Gestion 15

3.2.3.1 - Description 15

3.2.3.2 - Caractéristiques techniques 15

3.2.4 - Serveur de configuration 16

3.2.4.1 - Description 16

3.2.4.2 - Caractéristiques techniques 17

3.2.5 - user-api 18

3.2.5.1 - Description 18

3.2.5.2 - Caractéristiques techniques 18

3.2.6 - stock-api 19

3.2.6.1 - Description 19

3.2.6.2 - Caractéristiques techniques 20

3.2.7 - web-api 21

3.2.7.1 - Description 21

3.2.7.2 - Caractéristiques techniques 21

3.2.8 - production-api 22

3.2.8.1 - Description 22

3.2.8.2 - Caractéristiques techniques 23

3.2.9 - gestion-api 24

3.2.9.1 - Description 24

3.2.9.2 - Caractéristiques techniques 24

3.2.10 - Serveur de Fichiers 24

3.2.10.1 - Description 24

3.3 - Bases de données 25

3.4 - Les contrôleurs 26

3.5 - Web-services 26

3.6 - Serveur de configuration 26

3.7 - Les fichiers statiques 27

4 - Procédure de déploiement 28

4.1 - Configuration des outils 28

4.1.1 - Configuration de pgAdmin 28

4.1.2 - Configuration de PuTTY 31

4.2 - Installation de la base User 35

4.2.1 - Création du schéma de la base User 35

4.2.2 - Insérer les données initiales de la base User 37

4.3 - Installation de la base Stock 37

4.3.1 - Création du schéma de la base Stock 37

4.3.2 - Insérer les données initiales de la base Stock 37

4.4 - Installation de la base Gestion 37

4.4.1 - Création du schéma de la base Gestion 37

4.4.2 - Insérer les données initiales de la base Gestion 38

4.5 - Déploiement de config-server 38

4.5.1 - Récupération de l'archive 38

4.5.2 - Variables d'environnement 38

4.5.3 - Lancer le microservice 39

4.5.4 - Vérifications 39

4.6 - Déploiement de user-api 40

4.6.1 - Récupération de l'archive 40

4.6.2 - Variables d'environnement 40

4.6.3 - Configuration 41

4.6.4 - Lancer le microservice 42

4.6.5 - Vérifications 42

4.7 - Déploiement de stock-api 43

4.7.1 - Récupération de l'archive 43

4.7.2 - Variables d'environnement 43

4.7.3 - Configuration 44

4.7.4 - Lancer le microservice 45

4.7.5 - Vérifications 45

4.8 - Déploiement des fichiers statiques 46

4.8.1 - Récupération de l'archive 46

4.8.2 - Vérifications 47

4.9 - Déploiement de web-api 48

4.9.1 - Récupération de l'archive 48

4.9.2 - Variables d'environnement 48

4.9.3 - Configuration 49

4.9.4 - Lancer le microservice 50

4.9.5 - Vérifications 50

4.10 - Déploiement de production-api 51

4.10.1 - Récupéreration de l'archive 51

4.10.2 - Variables d'environnement 51

4.10.3 - Configuration 52

4.10.4 - Lancer le microservice 53

4.10.5 - Vérifications 53

4.11 - Déploiement de gestion-api 54

4.11.1 - Récupéreration de l'archive 54

4.11.2 - Variables d'environnement 54

4.11.3 - Configuration 55

4.11.4 - Lancer le microservice 56

4.11.5 - Vérifications 56

5 - Procédure de démarrage / arrêt 58

5.1 - Base de données User 58

5.1.1 - Préalable 58

5.1.2 - Démarrage 59

5.1.3 - Arrêt 60

5.2 - Base de données Stock 60

5.3 - Base de données Gestion 60

5.4 - Config-server 60

5.4.1 - Préalable 60

5.4.2 - Démarrage 61

5.4.3 - Arrêt 61

5.5 - user-api 61

5.5.1 - Préalable 61

5.5.2 - Démarrage 61

5.5.3 - Arrêt 61

5.6 - stock-api 62

5.6.1 - Préalable 62

5.6.2 - Démarrage 62

5.6.3 - Arrêt 62

5.7 - web-api 62

5.7.1 - Préalable 62

5.7.2 - Démarrage 63

5.7.3 - Arrêt 63

5.8 - production-api 63

5.8.1 - Préalable 63

5.8.2 - Démarrage 63

5.8.3 - Arrêt 64

5.9 - gestion-api 64

5.9.1 - Préalable 64

5.9.2 - Démarrage 64

5.9.3 - Arrêt 64

6 - Procédure de mise à jour 65

6.1 - Base de données User 65

6.1.1 - Préalable 65

6.1.2 - Mise à jour 65

6.1.3 - Finalisation 65

6.2 - Base de données Stock 65

6.2.1 - Préalable 65

6.2.2 - Mise à jour 66

6.2.3 - Finalisation 66

6.3 - Base de données Gestion 66

6.3.1 - Préalable 66

6.3.2 - Mise à jour 66

6.3.3 - Finalisation 67

6.4 - Microservice user-api 67

6.4.1 - Préalable 67

6.4.2 - Mise à jour 67

6.4.3 - Finalisation 68

6.5 - Microservice stock-api 68

6.5.1 - Préalable 68

6.5.2 - Mise à jour 68

6.5.3 - Finalisation 69

6.6 - Microservice web-api 69

6.6.1 - Préalable 69

6.6.2 - Mise à jour 69

6.6.3 - Finalisation 70

6.7 - Microservice production-api 70

6.7.1 - Préalable 70

6.7.2 - Mise à jour 70

6.7.3 - Finalisation 71

6.8 - Microservice gestion-api 71

6.8.1 - Préalable 71

6.8.2 - Mise à jour 71

6.8.3 - Finalisation 72

6.9 - Serveur de configuration 72

6.9.1 - Préalable 72

6.9.2 - Mise à jour 72

6.9.3 - Finalisation 73

7 - Supervision/Monitoring 74

7.1 - Supervision hardware des instances des microservices 74

7.2 - Supervision hardware des instances des bases de données 76

7.3 - CloudWatch Metrics 78

7.4 - Analyse des logs 79

7.5 - Les sondes 80

8 - Procédure de sauvegarde et restauration 81

8.1 - Base de données 81

8.1.1 - Sauvegarde d'une base de données 81

8.1.2 - Restauration d'une base de données 83

8.2 - Microservices 84

8.2.1 - Sauvegarde 84

8.2.1.1 - Sauvegarde sur AWS. 84

8.2.1.2 - Backup. 86

8.2.2 - Restauration ancienne configuration 86

8.2.2.1 - Restauration des binaires 87

8.2.2.2 - Restauration d'une image d'instance sur AWS 87

9 - Glossaire 91

# Versions

|  |  |  |  |
| --- | --- | --- | --- |
| Auteur | Date | Description | Version |
| PEDSF | 09/05/2020 | Création du document | 1.0 |
| PEDSF | 16/05/2020 | Utilisation d'un serveur Nexus pour archiver les releases | 2.0 |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Objet du document

Le présent document constitue le dossier d’exploitation de l'application **PizzaFlow** à l’attention des mainteneurs et de l’équipe technique du client.

Les éléments du présent dossier découlent :

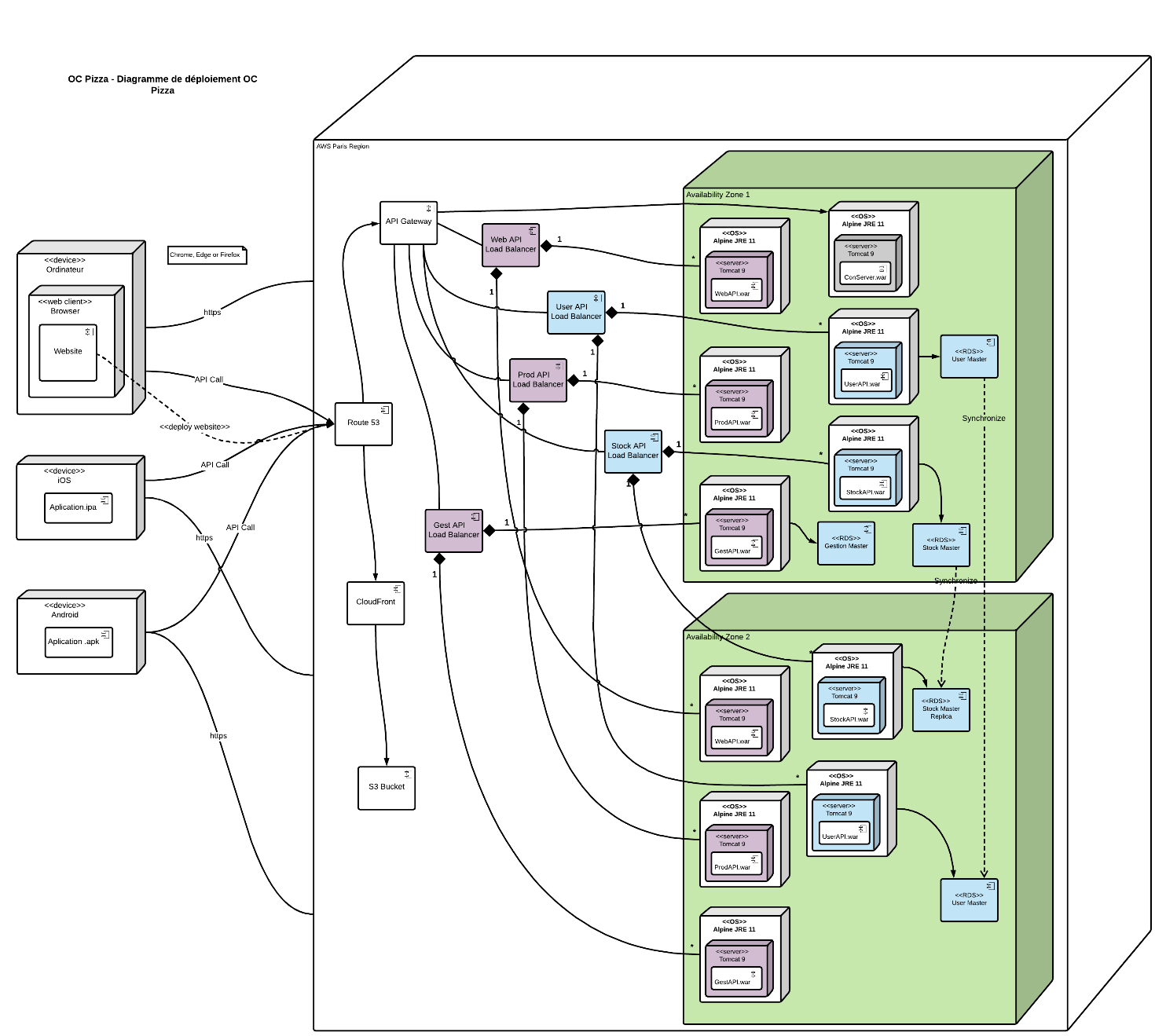
1. Des besoins exprimés par le client **OC Pizza** lors du premier contact,
2. De l'analyse des besoins de **OC** **Pizza**,
3. De la rédaction du dossier de conception fonctionnelle.
4. De la rédaction du dossier de conception technique.

## Références

Pour de plus amples informations, se référer également aux éléments suivants :

* **DCT - PDOCPizza\_01\_fonctionnelle** : Dossier de conception fonctionnelle de l'application.
* **DCT - PDOCPizza\_02\_technique** : Dossier de conception technique de l'application.

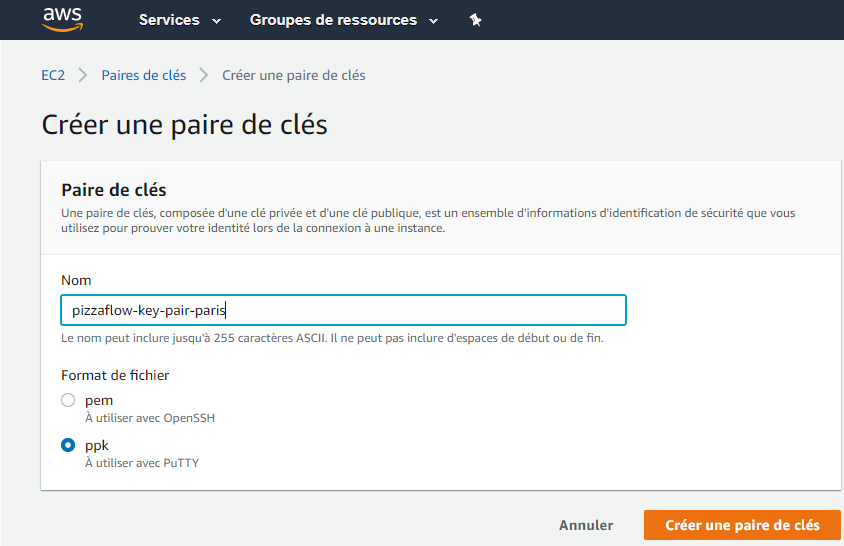
## Diagramme de déploiement



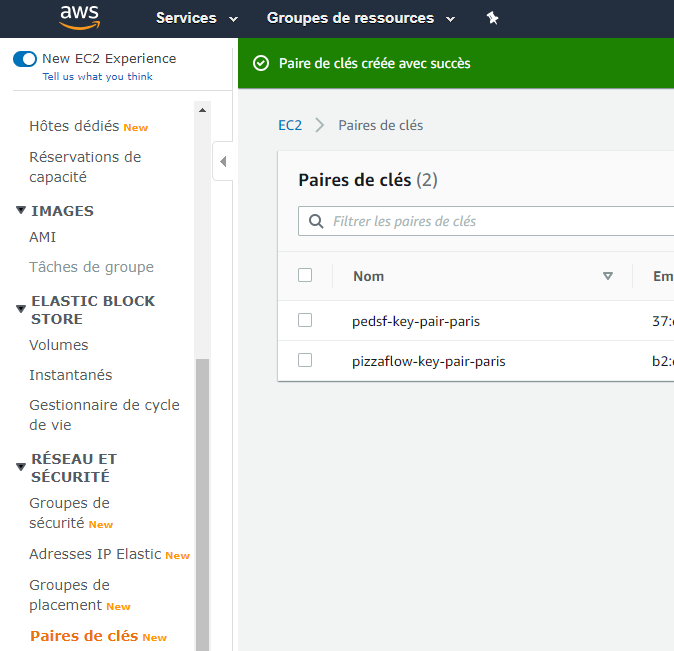
# Pré-requis

## Création d'une paire de clé

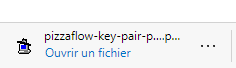
Après création d'un compte sur AWS, on doit générer une paire de clés qui servira d'authentification pour accéder aux **VM**. Lors de la création des instances RDS et EC2, on spécifiera l'utilisation de cette clé. Dans la console de gestion des instances sur **AWS** et on sélectionne dans le menu de gauche "RESEAU ET SECURITE" le sous-menu "Paires de Clés" et on sélectionne créer une paire de clés. Le menu de création de paire de clé apparait.



Rentrer le nom de la paire de clé et sélectionnez un format de fichier "ppk" pour l'utiliser avec **PuTTY**. Après validation sur la touche "Créer une paire de clés", la paire de clés générée apparait dans l'écran des paires de clés.



Le fichier PPK contenant la paire de clé générée est téléchargé via le navigateur. Copier le fichier sur votre ordinateur dans un endroit sécurisé.



## Système

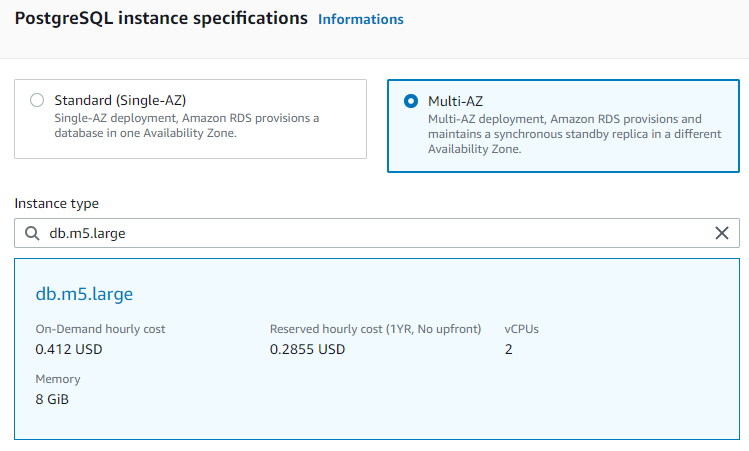
### Serveur de Base de données User

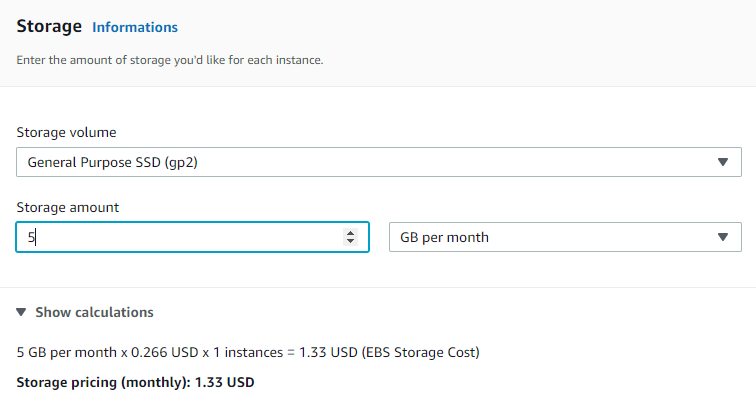
#### Description

On utilise une instance **Amazon** **RDS** (**Relational Database Service**) for **PostgreSQL** avec une réservation d'une année pour réduire le coût et permettre de modifier en fonction de l'utilisation du site web le type de l'instance. On sélectionne l'option **Multi-AZ** pour avoir une seconde instance en standby synchronisée avec la première par sécurité au cas où un problème advienne sur la base de données master.

La base de données stockant uniquement les données des utilisateurs, elle n'a pas besoin d'une grande quantité de stockage. En outre, elle reçoie la majorité des requêtes sont lors de la connexion et sont seulement en lecture donc on n'a pas besoin de puissance de calcul.

#### Caractéristiques techniques





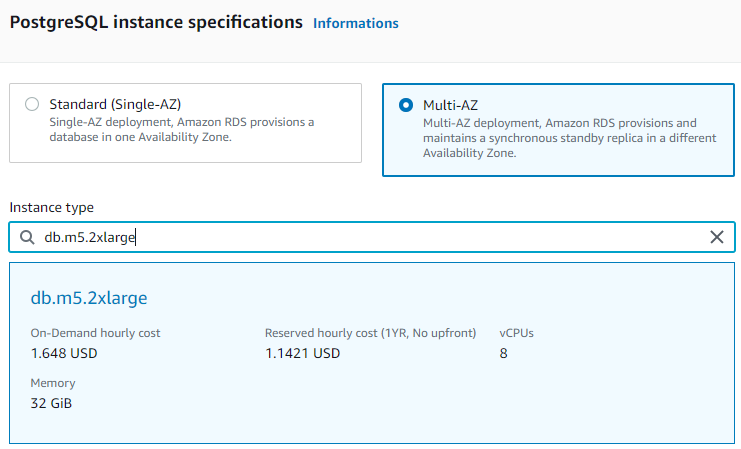
### Serveur de Base de données Stock

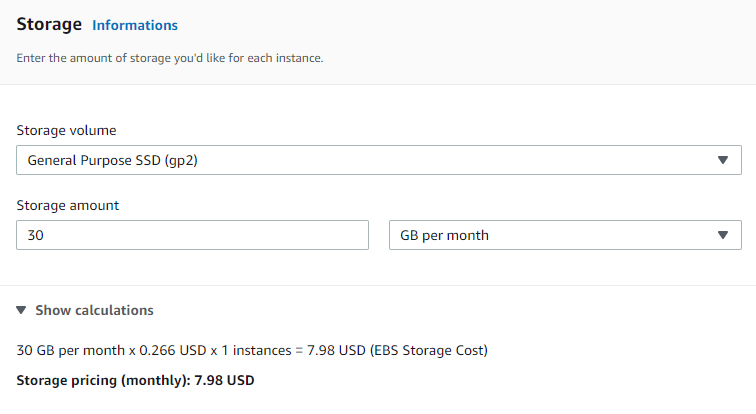
#### Description

On utilise une instance **Amazon** **RDS** for **PostgreSQL** avec une réservation d'une année pour réduire le coût et permettre de modifier en fonction de l'utilisation du site web le type de l'instance. On sélectionne l'option **Multi-AZ** pour avoir une seconde instance en standby synchronisée avec la première par sécurité au cas où un problème advienne sur la base de données master.

La base de données stocke les informations des stocks, des paniers et des commandes. Elle a besoin de puissance de calcul et d'une réserve de stockage.

#### Caractéristiques techniques





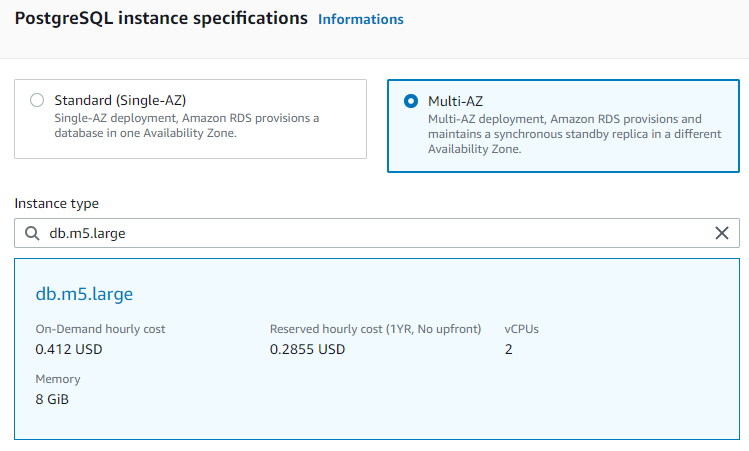
### Serveur de Base de données Gestion

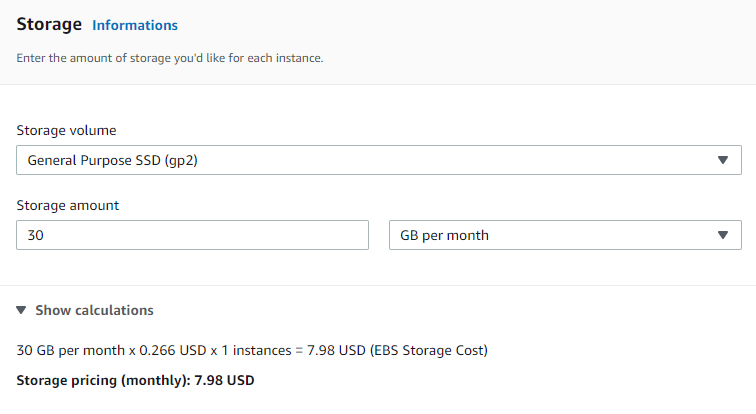
#### Description

On utilise une instance **Amazon** **RDS** for **PostgreSQL** avec une réservation d'une année pour réduire le coût et permettre de modifier en fonction de l'utilisation du site web le type de l'instance. On sélectionne l'option **Multi-AZ** pour avoir une seconde instance en standby synchronisée avec la première par sécurité au cas où un problème advienne sur la base de données master.

La base de données stocke les données des ventes pour la gestion. Elle est utilisée par les Managers et la Direction donc elle a simplement besoin d'un espace de stockage conséquent sans puissance de calcul.

#### Caractéristiques techniques



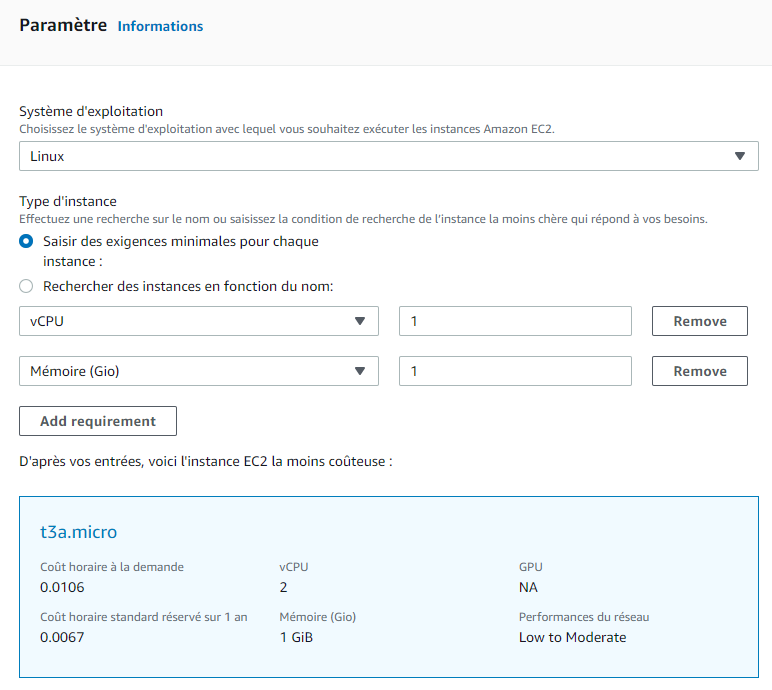


### Serveur de configuration

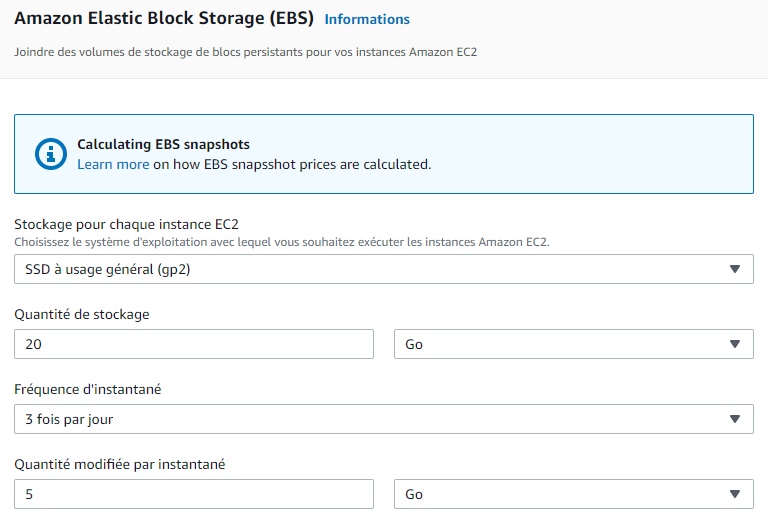
#### Description

Il sert juste à distribuer les configurations et ne nécessite pas de puissance de calcul. On prendra la plus petite des instances **Amazon** **EC2** (**Elastic Cloud Compute**).

#### Caractéristiques techniques



Le stockage sera partagé par toutes les instances Amazon EC2 pour mutualiser les ressources. On utilise 20Go de stockage avec 3 pics de 5Go modifié par jour vers les heures d'affluence et pour les transferts de données journaliers pour la maintenance en heure creuse.

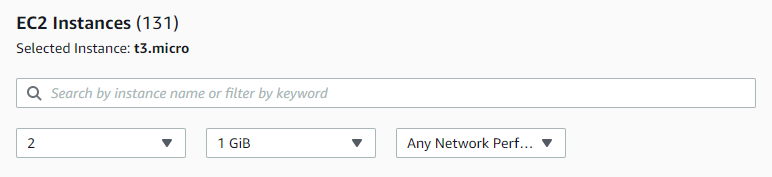


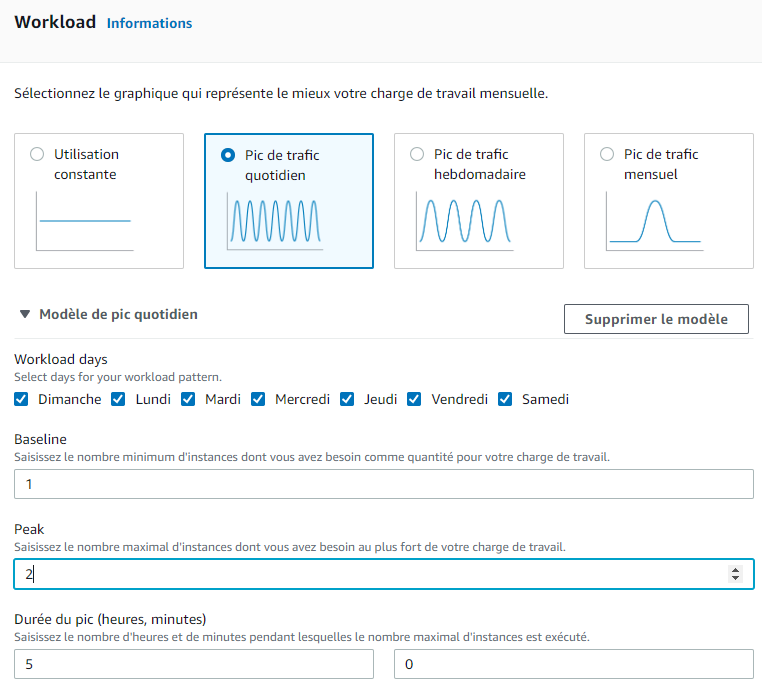
### user-api

#### Description

On prendra une petite instance avec 2 cœurs **Amazon** **EC2** scalable à 2 pendant 5h tous les jours.

#### Caractéristiques techniques



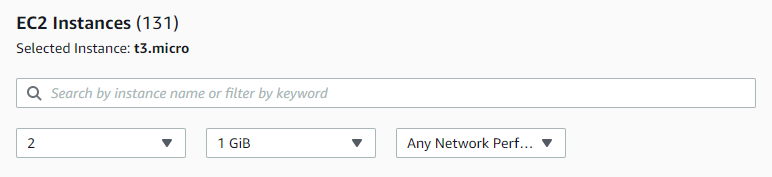


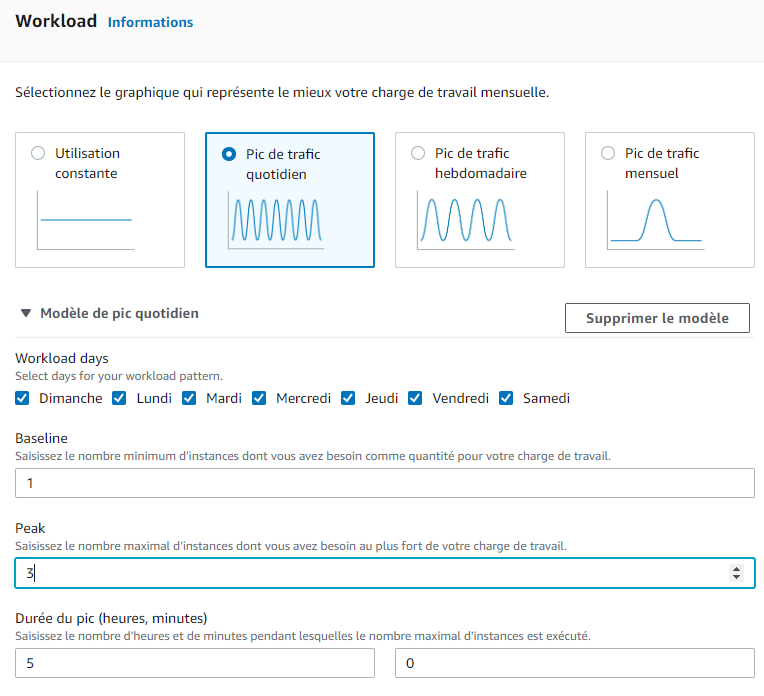
### stock-api

#### Description

On prendra une instance avec 2 cœurs **Amazon** **EC2** scalable à 3 pendant 5h tous les jours.

#### Caractéristiques techniques



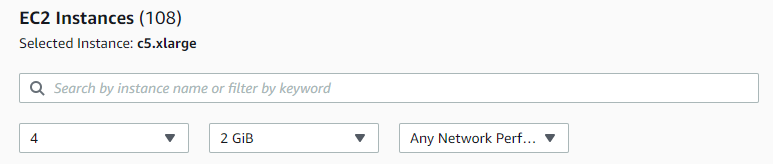


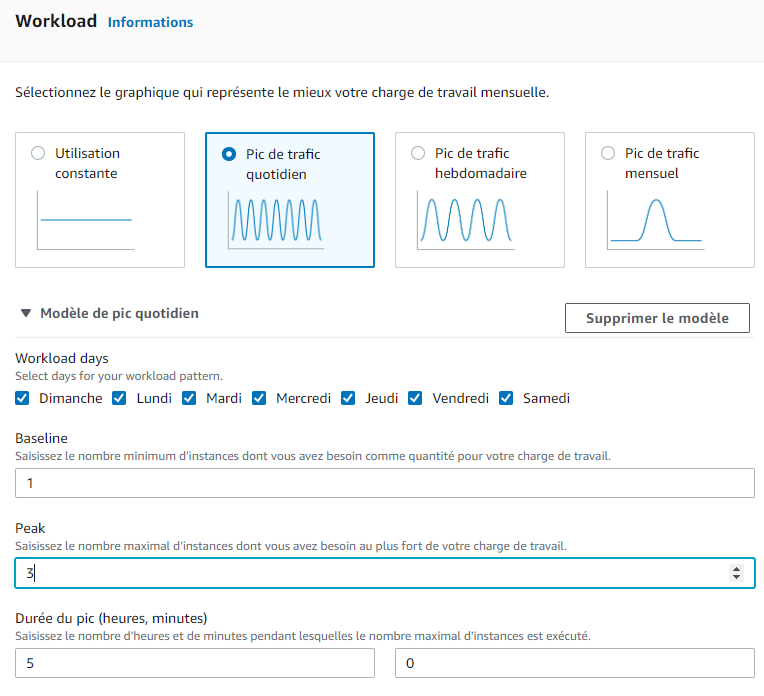
### web-api

#### Description

On prendra une instance avec 4 cœurs **Amazon** **EC2** scalable à 3 pendant 5h tous les jours. On prend plus de puissance de calcul pour les microservices frontend.

#### Caractéristiques techniques



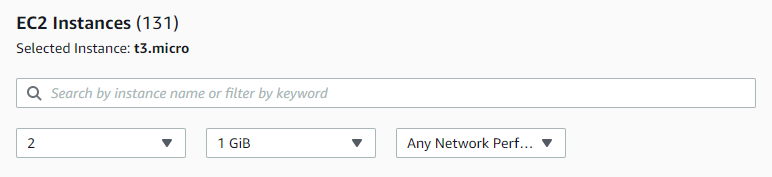


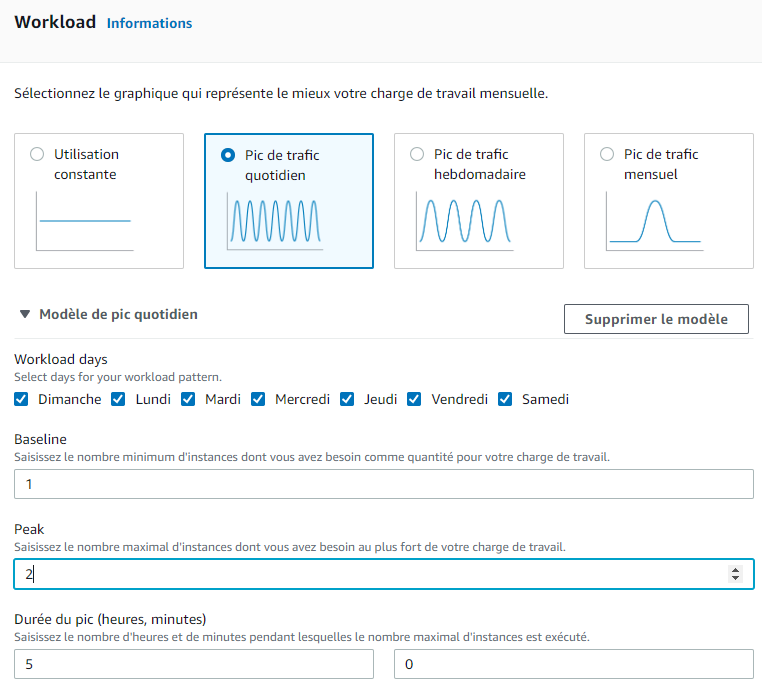
### production-api

#### Description

On prendra une instance avec 2 cœurs **Amazon** **EC2** scalable à 2 pendant 5h tous les jours. On prend plus de puissance de calcul pour les microservices frontend.

#### Caractéristiques techniques



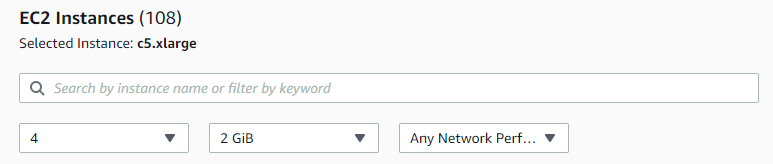


### gestion-api

#### Description

On prendra une instance avec 4 cœurs **Amazon** **EC2** sans load-balancing. On prend plus de puissance de calcul pour les microservices frontend de gestion.

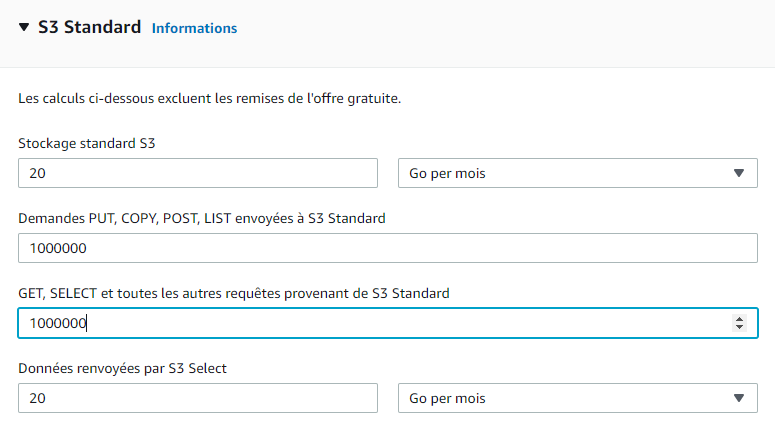
#### Caractéristiques techniques



### Serveur de Fichiers

#### Description

On utilise une instance **Amazon S3** (**Simple Storage Service**) pour stocker les fichiers statiques des microservices frontend. On table sur 2 millions de transactions par mois et 20Go de données transférées.



## Bases de données

Les bases de données et schémas suivants doivent être accessibles et à jour dans le repository **Nexus** privé.

|  |  |  |  |
| --- | --- | --- | --- |
| Description | GroupId | ArtifactId | Version |
| **user schéma** | com.itcd.delivery | pizzaflow-user-init | 1.0 |
| **user datas** | com.itcd.delivery | pizzaflow-user-datas | 1.0 |
| **stock schéma** | com.itcd.delivery | pizzaflow-stock-init | 1.0 |
| **stock datas** | com.itcd.delivery | pizzaflow-stock-datas | 1.0 |
| **gestion schéma** | com.itcd.delivery | pizzaflow-gestion-init | 1.0 |
| **gestion datas** | com.itcd.delivery | pizzaflow-gestion-datas | 1.0 |

## Les contrôleurs

Les contrôleurs suivants doivent être accessibles et à jour dans le repository **Nexus**.

|  |  |  |  |
| --- | --- | --- | --- |
| Description | GroupId | ArtifactId | Version |
| **user-api** | com.itcd.delivery | pizzaflow-user-api | 1.0 |
| **stock-api** | com.itcd.delivery | pizzaflow-stock-api | 1.0 |
| **gestion-api** | com.itcd.delivery | pizzaflow-gestion-api | 1.0 |

## Web-services

Les web services suivants doivent être accessibles et à jour dans le repository **Nexus**.

|  |  |  |  |
| --- | --- | --- | --- |
| Description | GroupId | ArtifactId | Version |
| **web-api** | com.itcd.delivery | pizzaflow-web-api | 1.0 |
| **production-api** | com.itcd.delivery | pizzaflow-production-api | 1.0 |
| **gestion-api** | com.itcd.delivery | pizzaflow-gestion-api | 1.0 |

## Serveur de configuration

Le serveur config-server et les fichiers de configurations doivent être accessibles et à jour dans le repository **Nexus**.

|  |  |  |  |
| --- | --- | --- | --- |
| Description | GroupId | ArtifactId | Version |
| **fichier JAR** | com.itcd.delivery | config-server | 1.0 |
| **fichiers properties** | com.itcd.delivery | pizzaflow-prod-properties | 1.0 |

## Les fichiers statiques

Les fichiers statiques pour les applications web doivent être accessibles et à jour dans le repository **Nexus**.

|  |  |  |  |
| --- | --- | --- | --- |
| Description | GroupId | ArtifactId | Version |
| **images** | com.itcd.delivery | pizzaflow-prod-images | 1.0 |
| **templates** | com.itcd.delivery | pizzaflow-prod-templates | 1.0 |
| **locales** | com.itcd.delivery | pizzaflow-prod-locales | 1.0 |
| **css** | com.itcd.delivery | pizzaflow-prod-css | 1.0 |
| **js** | com.itcd.delivery | pizzaflow-prod-js | 1.0 |

# Procédure de déploiement

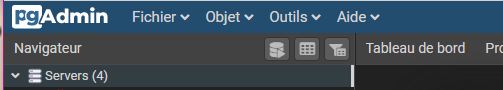
## Configuration des outils

### Configuration de pgAdmin

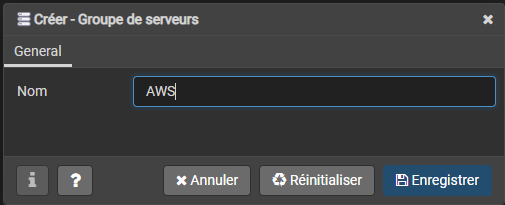
Pour accéder aux bases de données on doit installer **pgAdmin 4** (**PostgreSQL Admin**). Le lien pour installer le logiciel sur son ordinateur est :

[**https://www.pgadmin.org/download/**](https://www.pgadmin.org/download/)

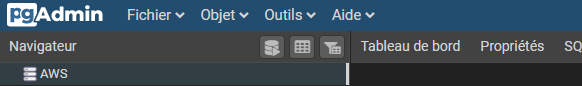
Une fois le logiciel installé, il faut configurer la connexion.



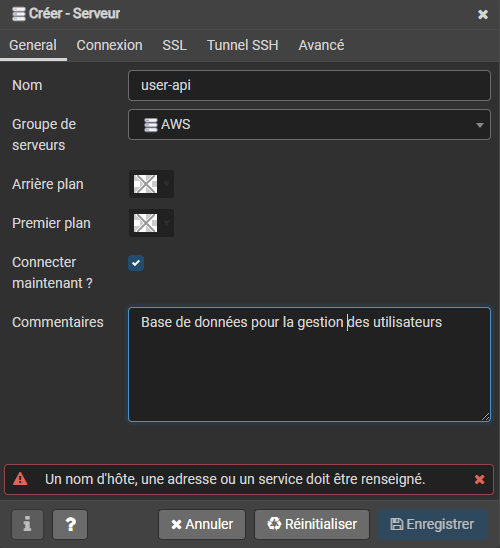
Il faut créer un groupe de serveur pour classer les connexions aux bases de données dans le menu **"Objet/Créer/Groupe de serveur".**



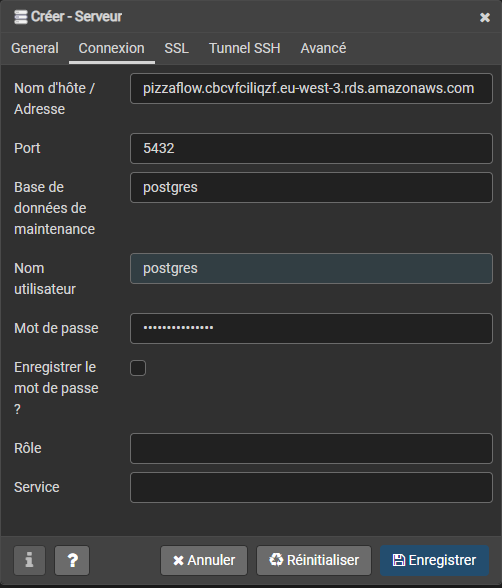
Sélectionner le groupe de serveur créé.



Cliquer sur le menu "**Objet/Créer/Serveur"** et indiquer le **"Nom"** de la connexion et une description dans la partie "Commentaires".



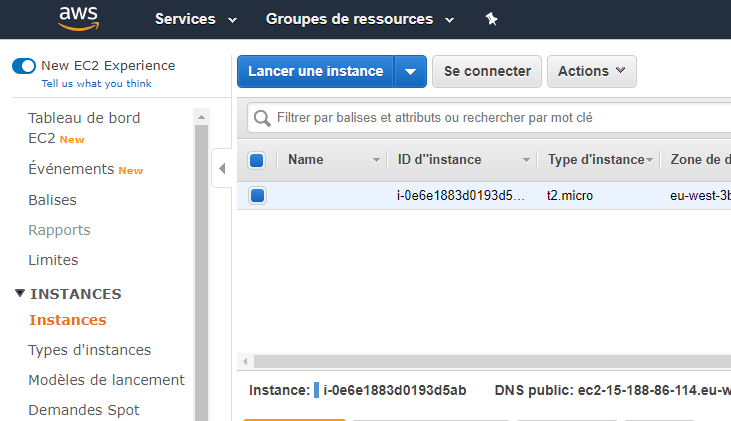
Sélectionner l'onglet **"Connexion"** pour renseigner les informations de connexion à la base de données. Entrer le **"Nom d'hôte/Adresse"** de la base de données, le **"Nom utilisateur"** et son **"Mot de passe"** en fonction des indications données lors de la création de la base de données et enregistrer.



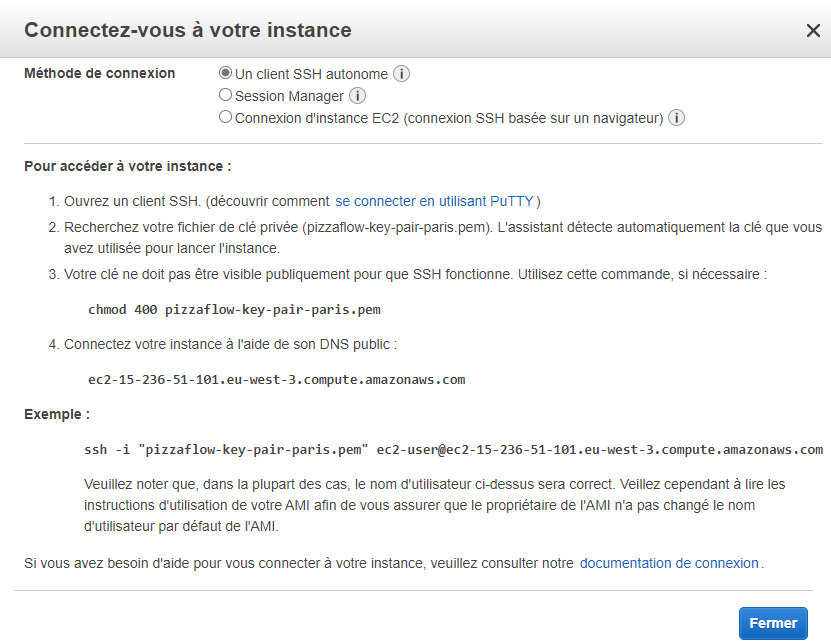
Il faut créer une connexion pour chaque base de données (User, Stock et Gestion).

### Configuration de PuTTY

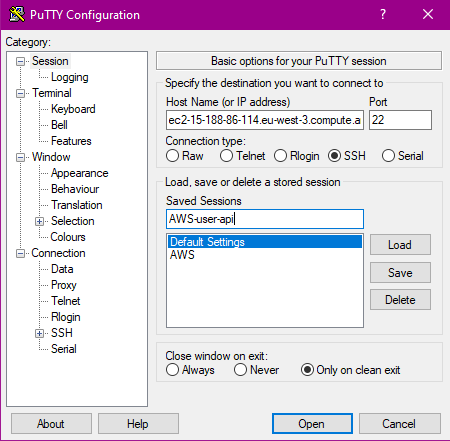
On doit créer une configuration pour chaque connexion. Pour avoir les informations nécessaires on se connecte sur la console de gestion des instances sur **AWS** et on sélectionne dans le menu de gauche **"INSTANCES"** le sous-menu **"Instances"**.



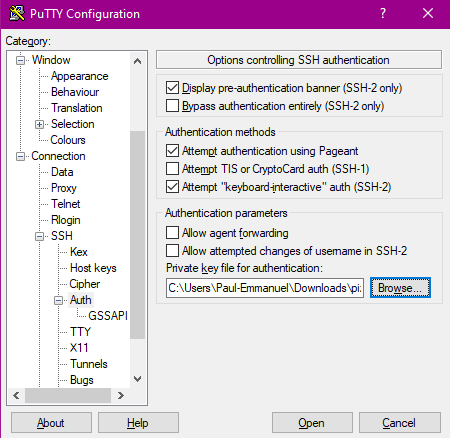
On sélectionne l'instance voulue puis on clique sur **"Se connecter"**.



* Copier le DNS public de l'instance et lancer PuTTY.
* Coller le nom du DNS dans la partie **"Host name"**.
* Renseigner le **"Port"** 22.
* Sélectionner **"SSH"** dans le type de connexion.
* Renseigner le nom de sauvegarde de la connexion en spécifiant le nom du microservice dans **"Saved Session"**.



* Dans le volet **"Catégory"**, entrer dans le sous-menu **"Connexion/SSH/Auth"**.
* Choisir **"Parcourir"** et sélectionner le fichier de paire de clé **PPK**.



* Revenir sur le sous-menu **"Session"** et cliquer sur **"Save"** pour enregistrer les modifications.
* Cliquer sur **"Open"** pour ouvrir une session sur la machine distante.

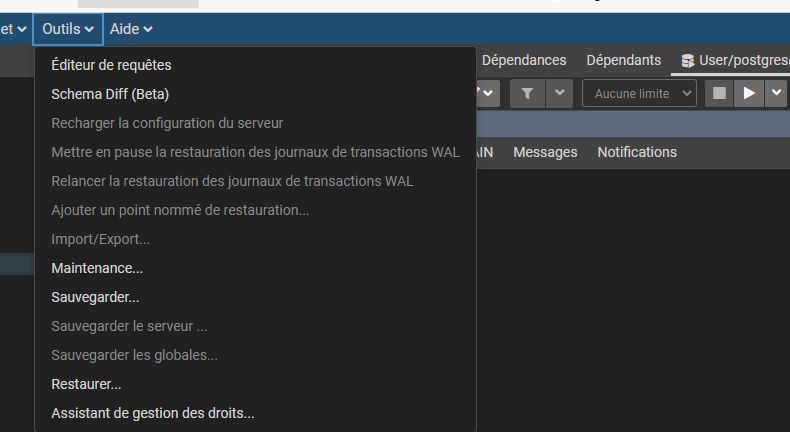
## Installation de la base User

### Création du schéma de la base User

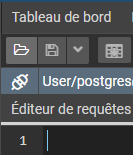
Télécharger le schéma de la base de données **User** dans le repository **Nexus** à l'adresse suivante :

[**http://itcd.com/nexus/pizzaflow**](http://itcd.com/nexus/pizzaflow)**/com.itcd.delivery/1.0/pizzaflow-user-schema-1.0**

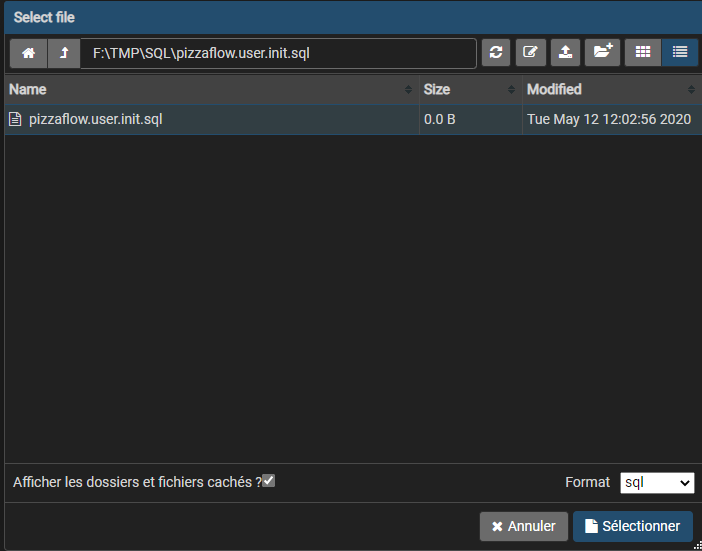
Dans **pgAdmin** il faut sélectionner la base de données **User** et le menu **Outils/Editeur de requêtes**. La fenêtre de l'éditeur s'ouvre dans la partie gauche de l'écran.



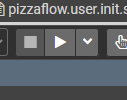
Cliquer sur l'icône **"Ouvir fichier"** en haut à gauche de la fenêtre de requêtes.



Sélectionner le fichier **pizzaflow-user-schema-1.0.sql** et valider.



Dans la fenêtre des requêtes cliquer sur l'icône exécuter.



### Insérer les données initiales de la base User

Télécharger les données initiales de la base **User** dans le repository **Nexus** à l'adresse suivante :

[**http://itcd.com/nexus/pizzaflow**](http://itcd.com/nexus/pizzaflow)**/com.itcd.delivery/1.0/pizzaflow-user-datas-1.0**

Faire la même opération avec le fichier **pizzaflow-user-datas-1.0.sql** pour insérer les données dans la base.

## Installation de la base Stock

### Création du schéma de la base Stock

Récupérer le schéma de la base de données **Stock** sur le repository **Nexus** et effectuer les mêmes opérations que la base **User**.

[**http://itcd.com/nexus/pizzaflow/com.itcd.delivery/1.0/pizzaflow-stock-shema-1.0**](http://itcd.com/nexus/pizzaflow/com.itcd.delivery/1.0/pizzaflow-stock-shema-1.0)

### Insérer les données initiales de la base Stock

Récupérer les données initiales de la base de données **Stock** sur le repository **Nexus** et effectuer les mêmes opérations que la base **User**.

[**http://itcd.com/nexus/pizzaflow**](http://itcd.com/nexus/pizzaflow)**/com.itcd.delivery/1.0/pizzaflow-stock-datas-1.0**

## Installation de la base Gestion

### Création du schéma de la base Gestion

Récupérer le schéma de la base de données **Gestion** sur le repository **Nexus** et effectuer les mêmes opérations que la base **User**.

[**http://itcd.com/nexus/pizzaflow**](http://itcd.com/nexus/pizzaflow)**/com.itcd.delivery/1.0/pizzaflow-gestion-schema-1.0**

### Insérer les données initiales de la base Gestion

Récupérer les données initiales de la base de données **Gestion** sur le repository **Nexus** et effectuer les mêmes opérations que la base **User**.

[**http://itcd.com/nexus/pizzaflow**](http://itcd.com/nexus/pizzaflow)**/com.itcd.delivery/1.0/pizzaflow-gestion-datas-1.0**

## Déploiement de config-server

### Récupération de l'archive

Se connecter à la **VM** du microservice avec **PuTTY** et se placer dans le répertoire **/srv** de l'instance.

|  |
| --- |
| $cd /srv |

Récupérer le **ZIP** de **config-server** qui est dans un repository **Nexus** et décompresser-le avec les commandes :

|  |
| --- |
| /srv$ wget -O config-server.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a=config-server \  &v=1.0 \  &p=zip  /srt$unzip config-server.zip -d config-server  /srt$cd config-server  /srt/config-server$ |

### Variables d'environnement

On doit spécifier les variables d'environnement pour les microservices afin qu'ils puissent récupérer leur configuration. On définit une variable d'environnement avec la commande :

|  |
| --- |
| $export NOM\_VARIABLE=valeur |

Ou peut renseigner ces valeurs dans le script **/srt/config-server/start.sh**.

|  |  |  |
| --- | --- | --- |
| Nom | Obligatoire | Description |
| **JASYPT\_SECRET** | oui | Clé pour de cryptage des données sensibles des fichiers de propriétés. |
| **PROPERTIES\_PATH** | oui | Chemin du repository contenant les fichiers de propriétés |

### Lancer le microservice

Un script de commande fourni permet de lancer le microservice en tâche de fond après avoir défini les variables d'environnement et déployer les fichiers statiques. Se placer dans le répertoire du microservice, rendre exécutable les scripts et exécuter le script de démarrage.

|  |
| --- |
| $ cd /srv/config-server  /srv/config-server$sudo chmod g+x u+x \*.sh  /srv/config-server$sudo ./start.sh |

### Vérifications

Exécuter la commande suivante pour vérifier que le EDGE microservice **config-server** fonctionne en tâche de fond.

|  |
| --- |
| /srv/config-server$ ps -l | grep config-server  4 S 0 1 0 0 80 0 - 967 - pts/0 00:00:00 config-server |

Une ligne est affichée donc le microservice fonctionne et l'identifiant du process est indiqué en 4ème colonne. Se connecter sur **logit.io** pour avoir une confirmation du fonctionnement en regardant les logs.

## Déploiement de user-api

### Récupération de l'archive

Se connecter à la **VM** du microservice avec **PuTTY** et se placer dans le répertoire **/srv** de l'instance.

|  |
| --- |
| $cd /srv |

Récupérer le **ZIP** de **user-api** qui est dans un repository **Nexus** et décompresser-le avec les commandes :

|  |
| --- |
| /srv$ wget -O user-api.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a=user-api \  &v=1.0 \  &p=zip  /srt$unzip user-api.zip -d user-api  /srt$cd user-api  /srt/user-api$ |

### Variables d'environnement

On doit spécifier les variables d'environnement pour les microservices **user-api** afin qu'ils puissent récupérer leur configuration. On définit une variable d'environnement avec la commande :

|  |
| --- |
| $export NOM\_VARIABLE=valeur |

Ou peut renseigner ces valeurs dans le script **/srv/user-api/start.sh**.

|  |  |  |
| --- | --- | --- |
| Nom | Obligatoire | Description |
| **JASYPT\_SECRET** | oui | Clé pour de cryptage des données sensibles des fichiers de propriétés. |
| **CONFIG\_IP** | oui | Addresse IP du serveur de configuration |
| **USER\_DB\_USERNAME** | oui | Identifiant de connexion à la base de données User |
| **USER\_DB\_PASSWORD** | oui | Mot de passe correspondant |

### Configuration

Le fichier de configuration **user-api-prod.properties** comme les fichiers de configuration de tous les microservices sont dans le repository **git** suivant :

[**https://github.com/pizzaflow/properties.git**](https://github.com/pizzaflow/properties.git)

Lors du démarrage, le microservice le demande au serveur de configuration qui va le chercher et le transmet.

|  |
| --- |
| ## port par défaut pour la première instance de user-api  ## utiliser la gamme des ports 4XXX pour les duplications d'instances  server.port=4000  ## Propriétés générales de l'application PizzaFlow  pizzaflow.nomPropriété=Valeur  ## Propriétés particulières du microservice user-api  pizzaflow.user.nomPropriété=Valeur  ## configuration du pool de connexion par défaut  spring.datasource.hikari.connectionTimeout=20000  spring.datasource.hikari.maximumPoolSize=5  ## configuration de PostgreSQL remplacer localhost par la vrai adresse IP  spring.datasource.url=jdbc:postgresql://localhost:5432/user  spring.datasource.driverClassName=org.postgresql.Driver  ## identifiants temporaire de connexion à changer et crypter avec Jasypt pour la production  spring.datasource.username=${USER\_DB\_USERNAME}  spring.datasource.password=${USER\_DB\_PASSWORD}  # DANGER!! mettre à create pour refaire le schéma  spring.jpa.hibernate.ddl-auto=update  ## Properties pour les logs  logging.level.org.springframework.web=ERROR  logging.level.com.pizzaflow=DEBUG  ## le pattern pour la console  logging.pattern.console= "%d{yyyy-MM-dd HH:mm:ss} - %msg%n"  ## le pattern pour le nom du fichier  logging.pattern.file= "%d{yyyy-MM-dd HH:mm:ss}[%thread]%-5level %logger{36} - %msg%n"  ## le nom du fichier de log  logging.file=/var/log/pizzaflow/user-api.log |

### Lancer le microservice

Un script de commande fourni permet de lancer le microservice **user-api** en tâche de fond après avoir défini les variables d'environnement et déployer les fichiers statiques. Se placer dans le répertoire du microservice, rendre exécutable les scripts et exécuter le script de démarrage.

|  |
| --- |
| $ cd /srv/user-api  /srv/user-api$sudo chmod g+x u+x \*.sh  /srv/user-api$sudo ./start.sh |

### Vérifications

Exécuter la commande suivante pour vérifier que le microservice **user-api** fonctionne en tâche de fond.

|  |
| --- |
| /srv/user-api$ ps -l | grep user-api  4 S 0 1 0 0 80 0 - 967 - pts/0 00:00:00 user-api |

Une ligne est affichée donc le microservice fonctionne et l'identifiant du process est indiqué en 4ème colonne. Se connecter sur **logit.io** pour avoir une confirmation du fonctionnement en regardant les logs.

## Déploiement de stock-api

### Récupération de l'archive

Se connecter à la **VM** du microservice avec **PuTTY** et se placer dans le répertoire **/srv** de l'instance.

|  |
| --- |
| $cd /srv |

Récupérer le **ZIP** de **stock-api** qui est dans un repository **Nexus** et décompresser-le avec les commandes :

|  |
| --- |
| /srv$ wget -O stock-api.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a= stock-api \  &v=1.0 \  &p=zip  /srt$unzip stock-api.zip -d stock-api  /srt$cd stock-api  /srt/stock-api$ |

### Variables d'environnement

On doit spécifier les variables d'environnement pour les microservices afin qu'ils puissent récupérer leur configuration. On définit une variable d'environnement avec la commande :

|  |
| --- |
| $export NOM\_VARIABLE=valeur |

Ou peut renseigner ces valeurs dans le script **/srv/stock-api/start.sh**.

|  |  |  |
| --- | --- | --- |
| Nom | Obligatoire | Description |
| **JASYPT\_SECRET** | oui | Clé pour de cryptage des données sensibles des fichiers de propriétés. |
| **CONFIG\_IP** | oui | Addresse IP du serveur de configuration |
| **STOCK\_DB\_USERNAME** | oui | Identifiant de connexion à la base de données Stock |
| **STOCK\_DB\_PASSWORD** | oui | Mot de passe correspondant |

### Configuration

Le fichier de configuration **stock-api-prod.properties** comme les fichiers de configuration de tous les microservices sont dans le repository **git** suivant :

[**https://github.com/pizzaflow/properties.git**](https://github.com/pizzaflow/properties.git)

Lors du démarrage, le microservice le demande au serveur de configuration qui va le chercher et le transmet.

|  |
| --- |
| ## port par défaut pour la première instance de stock-api  ## utiliser la gamme des ports 5XXX pour les duplications d'instances  server.port=5000  ## Propriétés générales de l'application PizzaFlow  pizzaflow.nomPropriété=Valeur  ## Propriétés particulières du microservice stock-api  pizzaflow.stock.nomPropriété=Valeur  ## configuration du pool de connexion par défaut  spring.datasource.hikari.connectionTimeout=20000  spring.datasource.hikari.maximumPoolSize=5  ## configuration de PostgreSQL remplacer localhost par la vrai adresse IP  spring.datasource.url=jdbc:postgresql://localhost:5432/stock  spring.datasource.driverClassName=org.postgresql.Driver  ## identifiants temporaire de connexion à changer et crypter avec Jasypt pour la production  spring.datasource.username=${STOCK\_DB\_USERNAME}  spring.datasource.password=${STOCK\_DB\_PASSWORD}  # DANGER!! mettre à create pour refaire le schéma  spring.jpa.hibernate.ddl-auto=update  ## Properties pour les logs  logging.level.org.springframework.web=ERROR  logging.level.com.pizzaflow=DEBUG  ## le pattern pour la console  logging.pattern.console= "%d{yyyy-MM-dd HH:mm:ss} - %msg%n"  ## le pattern pour le nom du fichier  logging.pattern.file= "%d{yyyy-MM-dd HH:mm:ss}[%thread]%-5level %logger{36} - %msg%n"  ## le nom du fichier de log  logging.file=/var/log/pizzaflow/stock-api.log |

### Lancer le microservice

Un script de commande fourni permet de lancer le microservice en tâche de fond après avoir défini les variables d'environnement et déployer les fichiers statiques. Se placer dans le répertoire du microservice, rendre exécutable les scripts et exécuter le script de démarrage.

|  |
| --- |
| $ cd /srv/stock-api  /srv/stock-api$sudo chmod g+x u+x \*.sh  /srv/stock-api$sudo ./start.sh |

### Vérifications

Exécuter la commande suivante pour vérifier que le microservice fonctionne en tâche de fond.

|  |
| --- |
| /srv/stock-api$ ps -l | grep stock-api  4 S 0 1 0 0 80 0 - 967 - pts/0 00:00:00 stock-api |

Une ligne est affichée donc le microservice fonctionne et l'identifiant du process est indiqué en 4ème colonne. Se connecter sur **logit.io** pour avoir une confirmation du fonctionnement en regardant les logs.

## Déploiement des fichiers statiques

### Récupération de l'archive

Se connecter au bucket **S3** avec **PuTTY** et créer un répertoire **/resources** et se placer dedans.

|  |
| --- |
| $mkdir /resources  $cd /resources |

Récupérer le **ZIP** des fichiers properties et décompresser-le qui est dans un repository **Nexus** avec la commande :

|  |
| --- |
| /resources$ wget -O images.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a= pizzaflow-prod-images \  &v=1.0 \  &p=zip  /resources$unzip images.zip -d images  /resources$ wget -O templates.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a= pizzaflow-prod-templates \  &v=1.0 \  &p=zip  /resources$unzip templates.zip -d templates  /resources$ wget -O locales.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a= pizzaflow-prod-locales \  &v=1.0 \  &p=zip  /resources$unzip locales.zip -d locales  /resources$ wget -O css.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a= pizzaflow-prod-css \  &v=1.0 \  &p=zip  /resources$unzip css.zip -d css  /resources$ wget -O js.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a= pizzaflow-prod-js \  &v=1.0 \  &p=zip  /resources$unzip js.zip -d js |

### Vérifications

Exécuter la commande suivante pour vérifier que les répertoires ont bien été créés.

|  |
| --- |
| /resources$ ls -l  total 92  -rw-r--r-- 1 root root 125 May 16 08:21 css.zip  drwxr-xr-x 1 root root 1204 May 16 08:21 css  -rw-r--r-- 1 root root 29492 May 16 08:21 images.zip  drwxr-xr-x 1 root root 4096 May 16 08:21 images  -rw-r--r-- 1 root root 7621 May 16 08:21 js.zip  drwxr-xr-x 1 root root 4096 May 16 08:21 js  -rw-r--r-- 1 root root 1245 May 16 08:21 locales.zip  drwxr-xr-x 1 root root 4096 May 16 08:21 locales  -rw-r--r-- 1 root root 7621 May 16 08:21 template.zip  drwxr-xr-x 1 root root 4096 May 16 08:21 template |

On voit les archives récupérées et un répertoire pour chaque archive.

## Déploiement de web-api

### Récupération de l'archive

Se connecter à la **VM** du microservice avec **PuTTY** et se placer dans le répertoire **/srv** de l'instance.

|  |
| --- |
| $cd /srv |

Récupérer le **ZIP** de **web-api** qui est dans un repository **Nexus** et décompresser-le avec les commandes :

|  |
| --- |
| /srv$ wget -O web-api.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a=web-api \  &v=1.0 \  &p=zip  /srt$unzip web-api.zip -d web-api  /srt$cd web-api  /srt/web-api$ |

### Variables d'environnement

On doit spécifier les variables d'environnement pour les microservices **web-api** afin qu'ils puissent récupérer leur configuration. On définit une variable d'environnement avec la commande :

|  |
| --- |
| $export NOM\_VARIABLE=valeur |

Ou peut renseigner ces valeurs dans le script **/srv/web-api/start.sh**.

|  |  |  |
| --- | --- | --- |
| Nom | Obligatoire | Description |
| **JASYPT\_SECRET** | oui | Clé pour de cryptage des données sensibles des fichiers de propriétés. |
| **CONFIG\_IP** | oui | Addresse IP du serveur de configuration |
| **USER\_API\_IP** | oui | Addresse IP de **user-api** |
| **STOCK\_API\_IP** | oui | Addresse IP de **stock-api** |
| **RESOURCES\_PATH** | oui | Chemin vers le bucket des ressources |

### Configuration

Le fichier de configuration **web-api-prod.properties** comme les fichiers de configuration de tous les microservices sont dans le repository **git** suivant :

[**https://github.com/pizzaflow/properties.git**](https://github.com/pizzaflow/properties.git)

Lors du démarrage, le microservice le demande au serveur de configuration qui va le chercher et le transmet.

|  |
| --- |
| ## port par défaut pour la première instance de web-api  ## utiliser la gamme des ports 7XXX pour les duplications d'instances  server.port=7000  ## Propriétés générales de l'application PizzaFlow  pizzaflow.nomPropriété=Valeur  ## Propriétés particulières du microservice web-api  pizzaflow.web.nomPropriété=Valeur  ## Properties pour les logs  logging.level.org.springframework.web=ERROR  logging.level.com.pizzaflow=DEBUG  ## le pattern pour la console  logging.pattern.console= "%d{yyyy-MM-dd HH:mm:ss} - %msg%n"  ## le pattern pour le nom du fichier  logging.pattern.file= "%d{yyyy-MM-dd HH:mm:ss}[%thread]%-5level %logger{36} - %msg%n"  ## le nom du fichier de log  logging.file=/var/log/pizzaflow/web-api.log |

### Lancer le microservice

Un script de commande fourni permet de lancer le microservice **web-api** en tâche de fond après avoir défini les variables d'environnement et déployer les fichiers statiques. Se placer dans le répertoire du microservice, rendre exécutable les scripts et exécuter le script de démarrage.

|  |
| --- |
| $ cd /srv/web-api  /srv/web-api$sudo chmod g+x u+x \*.sh  /srv/web-api$sudo ./start.sh |

### Vérifications

Exécuter la commande suivante pour vérifier que le microservice **web-api** fonctionne en tâche de fond.

|  |
| --- |
| /srv/web-api$ ps -l | grep web-api  4 S 0 1 0 0 80 0 - 967 - pts/0 00:00:00 web-api |

Une ligne est affichée donc le microservice fonctionne et l'identifiant du process est indiqué en 4ème colonne. Se connecter sur **logit.io** pour avoir une confirmation du fonctionnement en regardant les logs.

## Déploiement de production-api

### Récupéreration de l'archive

Se connecter à la **VM** du microservice avec **PuTTY** et se placer dans le répertoire **/srv** de l'instance.

|  |
| --- |
| $cd /srv |

Récupérer le **ZIP** de **production-api** qui est dans un repository **Nexus** et décompresser-le avec les commandes :

|  |
| --- |
| /srv$ wget -O production-api.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a= production-api \  &v=1.0 \  &p=zip  /srt$unzip production-api.zip -d production-api  /srt$cd production-api  /srt/ production-api$ |

### Variables d'environnement

On doit spécifier les variables d'environnement pour les microservices afin qu'ils puissent récupérer leur configuration. On définit une variable d'environnement avec la commande :

|  |
| --- |
| $export NOM\_VARIABLE=valeur |

Ou peut renseigner ces valeurs dans le script **/srv/production-api/start.sh**.

|  |  |  |
| --- | --- | --- |
| Nom | Obligatoire | Description |
| **JASYPT\_SECRET** | oui | Clé pour de cryptage des données sensibles des fichiers de propriétés. |
| **CONFIG\_IP** | oui | Addresse IP du serveur de configuration |
| **USER\_API\_IP** | oui | Addresse IP de **user-api** |
| **STOCK\_API\_IP** | oui | Addresse IP de **stock-api** |
| **WEB\_API\_IP** | oui | Addresse IP de **web-api** |
| **RESOURCES\_PATH** | oui | Chemin vers le bucket des ressources |

### Configuration

Le fichier de configuration **production-api-prod.properties** comme les fichiers de configuration de tous les microservices sont dans le repository **git** suivant :

[**https://github.com/pizzaflow/properties.git**](https://github.com/pizzaflow/properties.git)

Lors du démarrage, le microservice le demande au serveur de configuration qui va le chercher et le transmet.

|  |
| --- |
| ## port par défaut pour la première instance de production-api  ## utiliser la gamme des ports 8XXX pour les duplications d'instances  server.port=8000  ## Propriétés générales de l'application PizzaFlow  pizzaflow.nomPropriété=Valeur  ## Propriétés particulières du microservice production-api  pizzaflow.production.nomPropriété=Valeur  ## Properties pour les logs  logging.level.org.springframework.web=ERROR  logging.level.com.pizzaflow=DEBUG  ## le pattern pour la console  logging.pattern.console= "%d{yyyy-MM-dd HH:mm:ss} - %msg%n"  ## le pattern pour le nom du fichier  logging.pattern.file= "%d{yyyy-MM-dd HH:mm:ss}[%thread]%-5level %logger{36} - %msg%n"  ## le nom du fichier de log  logging.file=/var/log/pizzaflow/production-api.log |

### Lancer le microservice

Un script de commande fourni permet de lancer le microservice en tâche de fond après avoir défini les variables d'environnement et déployer les fichiers statiques. Se placer dans le répertoire du microservice, rendre exécutable les scripts et exécuter le script de démarrage.

|  |
| --- |
| $ cd /srv/production-api  /srv/ production-api$sudo chmod g+x u+x \*.sh  /srv/ production-api$sudo ./start.sh |

### Vérifications

Exécuter la commande suivante pour vérifier que le microservice fonctionne en tâche de fond.

|  |
| --- |
| /srv/ production-api$ ps -l | grep production-api  4 S 0 1 0 0 80 0 - 967 - pts/0 00:00:00 production-api |

Une ligne est affichée donc le microservice fonctionne et l'identifiant du process est indiqué en 4ème colonne. Se connecter sur **logit.io** pour avoir une confirmation du fonctionnement en regardant les logs.

## Déploiement de gestion-api

### Récupéreration de l'archive

Se connecter à la **VM** du microservice avec **PuTTY** et se placer dans le répertoire **/srv** de l'instance.

|  |
| --- |
| $cd /srv |

Récupérer le **ZIP** de **gestion-api** qui est dans un repository **Nexus** et décompresser-le avec les commandes :

|  |
| --- |
| /srv$ wget -O gestion-api.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a= gestion-api \  &v=1.0 \  &p=zip  /srt$unzip gestion-api.zip -d gestion-api  /srt$cd gestion-api  /srt/ gestion-api$ |

### Variables d'environnement

On doit spécifier les variables d'environnement pour les microservices afin qu'ils puissent récupérer leur configuration. On définit une variable d'environnement avec la commande :

|  |
| --- |
| $export NOM\_VARIABLE=valeur |

Ou peut renseigner ces valeurs dans le script **/srv/gestion-api/start.sh**.

|  |  |  |
| --- | --- | --- |
| Nom | Obligatoire | Description |
| **JASYPT\_SECRET** | oui | Clé pour de cryptage des données sensibles des fichiers de propriétés. |
| **CONFIG\_IP** | oui | Addresse IP du serveur de configuration |
| **USER\_API\_IP** | oui | Addresse IP de **user-api** |
| **STOCK\_API\_IP** | oui | Addresse IP de **stock-api** |
| **WEB\_API\_IP** | oui | Addresse IP de **web-api** |
| **RESOURCES\_PATH** | oui | Chemin vers le bucket des ressources |
| **GESTION\_DB\_USERNAME** | oui | Identifiant de connexion à la base de données Gestion |
| **GESTION\_DB\_PASSWORD** | oui | Mot de passe correspondant |

### Configuration

Le fichier de configuration **gestion-api-prod.properties** comme les fichiers de configuration de tous les microservices sont dans le repository **git** suivant :

[**https://github.com/pizzaflow/properties.git**](https://github.com/pizzaflow/properties.git)

Lors du démarrage, le microservice le demande au serveur de configuration qui va le chercher et le transmet.

|  |
| --- |
| ## port par défaut pour la première instance de production-api  ## utiliser la gamme des ports 9XXX pour les duplications d'instances  server.port=9000  ## Propriétés générales de l'application PizzaFlow  pizzaflow.nomPropriété=Valeur  ## Propriétés particulières du microservice gestion-api  pizzaflow.gestion.nomPropriété=Valeur  ## configuration du pool de connexion par défaut  spring.datasource.hikari.connectionTimeout=20000  spring.datasource.hikari.maximumPoolSize=5  ## configuration de PostgreSQL remplacer localhost par la vrai adresse IP  spring.datasource.url=jdbc:postgresql://localhost:5432/stock  spring.datasource.driverClassName=org.postgresql.Driver  ## identifiants temporaire de connexion à changer et crypter avec Jasypt pour la production  spring.datasource.username=${GESTION\_DB\_USERNAME}  spring.datasource.password=${GESTION\_DB\_PASSWORD}  # DANGER!! mettre à create pour refaire le schéma  spring.jpa.hibernate.ddl-auto=update  ## Properties pour les logs  logging.level.org.springframework.web=ERROR  logging.level.com.pizzaflow=DEBUG  ## le pattern pour la console  logging.pattern.console= "%d{yyyy-MM-dd HH:mm:ss} - %msg%n"  ## le pattern pour le nom du fichier  logging.pattern.file= "%d{yyyy-MM-dd HH:mm:ss}[%thread]%-5level %logger{36} - %msg%n"  ## le nom du fichier de log  logging.file=/var/log/pizzaflow/gestion-api.log |

### Lancer le microservice

Un script de commande fourni permet de lancer le microservice en tâche de fond après avoir défini les variables d'environnement et déployer les fichiers statiques. Se placer dans le répertoire du microservice, rendre exécutable les scripts et exécuter le script de démarrage.

|  |
| --- |
| $ cd /srv/gestion-api  /srv/ gestion-api$sudo chmod g+x u+x \*.sh  /srv/ gestion-api$sudo ./start.sh |

### Vérifications

Exécuter la commande suivante pour vérifier que le microservice fonctionne en tâche de fond.

|  |
| --- |
| /srv/ gestion-api$ ps -l | grep gestion-api  4 S 0 1 0 0 80 0 - 967 - pts/0 00:00:00 gestion-api |

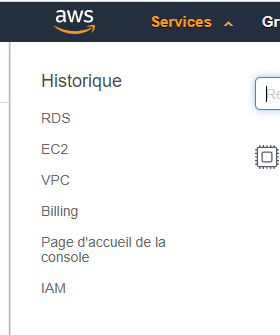
Une ligne est affichée donc le microservice fonctionne et l'identifiant du process est indiqué en 4ème colonne. Se connecter sur **logit.io** pour avoir une confirmation du fonctionnement en regardant les logs.

# Procédure de démarrage / arrêt

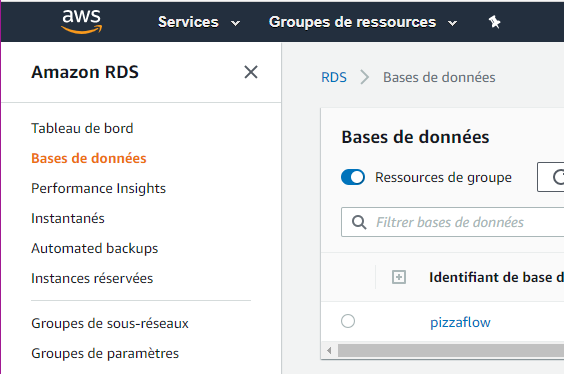
## Base de données User

### Préalable

Se connecter à l'interface de gestion de **AWS** et sélectionner le service **RDS**.



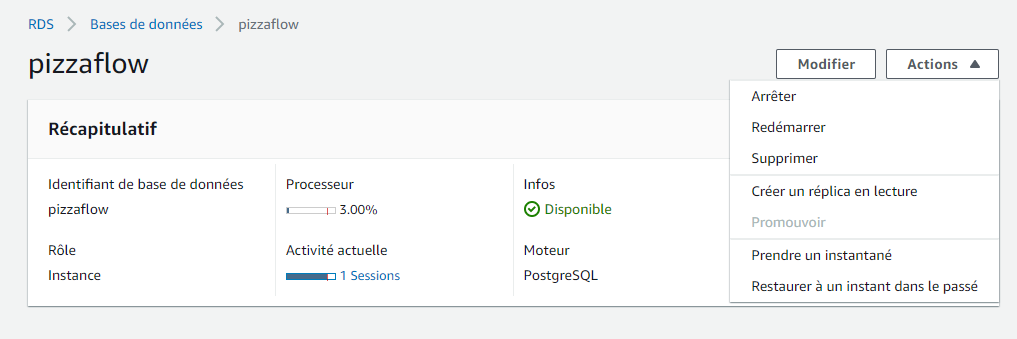
Choisir **"Bases de données"** pour voir les instances des bases de données.



Cliquer sur la base de données **User**.

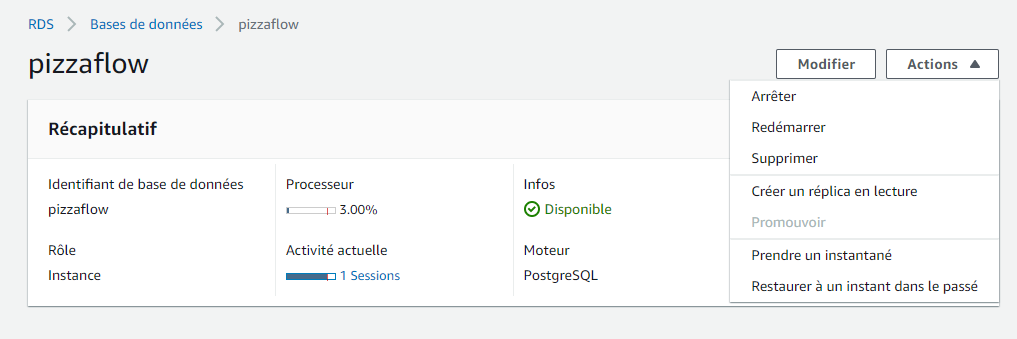
### Démarrage

Dans le menu **"Action"** en haut à droite sélectionner **"Redémarrer"**.



### Arrêt

Dans le menu **"Action"** en haut à droite sélectionner **"Arrêter"**.



## Base de données Stock

Effectuer les mêmes opérations que pour la base de données **User** ci-dessus.

## Base de données Gestion

Effectuer les mêmes opérations que pour la base de données **User** ci-dessus.

## Config-server

### Préalable

Se connecter avec **PuTTY** sur la **VM** de **config-server** et se placer dans le répertoire de l'application.

|  |
| --- |
| $cd /srv/config-server |

### Démarrage

Exécuter le script de démarrage.

|  |
| --- |
| /srv/config-server$sudo ./start.sh |

### Arrêt

Exécuter le script d'arrêt.

|  |
| --- |
| /srv/config-server$sudo ./shutdown.sh |

## user-api

### Préalable

Se connecter avec **PuTTY** sur la **VM** de **user-api** et se placer dans le répertoire de l'application.

|  |
| --- |
| $cd /srv/user-api |

### Démarrage

Exécuter le script de démarrage.

|  |
| --- |
| /srv/user-api$sudo ./start.sh |

### Arrêt

Exécuter le script d'arrêt.

|  |
| --- |
| /srv/user-api$sudo ./shutdown.sh |

## stock-api

### Préalable

Se connecter avec **PuTTY** sur la **VM** de **stock-api** et se placer dans le répertoire de l'application.

|  |
| --- |
| $cd /srv/stock-api |

### Démarrage

Exécuter le script de démarrage.

|  |
| --- |
| /srv/stock-api$sudo ./start.sh |

### Arrêt

Exécuter le script d'arrêt.

|  |
| --- |
| /srv/stock-api$sudo ./shutdown.sh |

## web-api

### Préalable

Se connecter avec **PuTTY** sur la **VM** de **web-api** et se placer dans le répertoire de l'application.

|  |
| --- |
| $cd /srv/web-api |

### Démarrage

Exécuter le script de démarrage.

|  |
| --- |
| /srv/web-api$sudo ./start.sh |

### Arrêt

Exécuter le script d'arrêt.

|  |
| --- |
| /srv/web-api$sudo ./shutdown.sh |

## production-api

### Préalable

Se connecter avec **PuTTY** sur la **VM** de **production-api** et se placer dans le répertoire de l'application.

|  |
| --- |
| $cd /srv/production-api |

### Démarrage

Exécuter le script de démarrage.

|  |
| --- |
| /srv/production-api$sudo ./start.sh |

### Arrêt

Exécuter le script d'arrêt.

|  |
| --- |
| /srv/production-api$sudo ./shutdown.sh |

## gestion-api

### Préalable

Se connecter avec **PuTTY** sur la **VM** de **gestion-api** et se placer dans le répertoire de l'application.

|  |
| --- |
| $cd /srv/gestion-api |

### Démarrage

Exécuter le script de démarrage.

|  |
| --- |
| /srv/gestion-api$sudo ./start.sh |

### Arrêt

Exécuter le script d'arrêt.

|  |
| --- |
| /srv/gestion-api$sudo ./shutdown.sh |

# Procédure de mise à jour

Les mises à jour doivent se faire uniquement en période creuse et de préférence la nuit pour ne pas nuire aux utilisateurs et pour avoir assez de temps pour effectuer les opérations

## Base de données User

### Préalable

Récupérer le script **SQL** de mise à jour de la base de données **User** sur son ordinateur pour pouvoir l'exécuter dans **pgAdmin**.

Arrêter le microservice **user-api** suivant la procédure indiquée plus haut pour qu'ils n'accèdent pas à la base de données.

Faire une sauvegarde de la base de données comme indiqué dans les procédures de sauvegarde et de restauration.

### Mise à jour

Depuis **pgAdmin** se connecter à la base de données **User**. Dans la fenêtre éditeur de requêtes ouvrir le script **SQL** récupéré et l'exécuter comme pour l'ajout des données dans la base lors du déploiement.

### Finalisation

Relancer le microservice **user-api**.

## Base de données Stock

### Préalable

Récupérer le script **SQL** de mise à jour de la base de données **Stock** sur son ordinateur pour pouvoir l'exécuter dans **pgAdmin**.

Arrêter le microservice **stock-api** suivant la procédure indiquée plus haut pour qu'ils n'accèdent pas à la base de données.

Faire une sauvegarde de la base de données comme indiqué dans les procédures de sauvegarde et de restauration.

### Mise à jour

Depuis **pgAdmin** se connecter à la base de données **Stock**. Dans la fenêtre éditeur de requêtes ouvrir le script **SQL** récupéré et l'exécuter comme pour l'ajout des données dans la base lors du déploiement.

### Finalisation

Relancer le microservice **stock-api**.

## Base de données Gestion

### Préalable

Récupérer le script **SQL** de mise à jour de la base de données **Gestion** sur son ordinateur pour pouvoir l'exécuter dans **pgAdmin**.

Arrêter le microservice **gestion-api** suivant la procédure indiquée plus haut pour qu'ils n'accèdent pas à la base de données.

Faire une sauvegarde de la base de données comme indiqué dans les procédures de sauvegarde et de restauration.

### Mise à jour

Depuis **pgAdmin** se connecter à la base de données **Gestion**. Dans la fenêtre éditeur de requêtes ouvrir le script **SQL** récupéré et l'exécuter comme pour l'ajout des données dans la base lors du déploiement.

### Finalisation

Relancer le microservice **gestion-api**.

## Microservice user-api

### Préalable

Aller dans le répertoire de l'application et arrêter le microservice **user-api** avec les commandes suivantes :

|  |
| --- |
| $cd /srv/user-api  /srv/user-api$sudo ./shutdown.sh |

Effectuer une sauvegarde de l'instance sur **AWS** en suivant la procédure indiquée dans les sauvegardes des microservices.

### Mise à jour

Récupérer le **ZIP** de mise à jour an spécifiant le bon numéro de version (v=?) de **user-api** qui est dans un repository **Nexus** et lancer la mise à jour avec les commandes :

|  |
| --- |
| /srv/user-api$ wget -O user-api.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a= user-api \  &v=2.1 \  &p=zip  /srt/user-api$sudo ./update.sh |

Le script **update.sh** arrête le microservice si c'est nécessaire, effectue une sauvegarde des fichiers de l'application et décompresse l'archive **ZIP**.

### Finalisation

Relancer le microservice **user-api**.

## Microservice stock-api

### Préalable

Aller dans le répertoire de l'application et arrêter le microservice **stock-api** avec les commandes suivantes :

|  |
| --- |
| $cd /srv/stock-api  /srv/stock-api$sudo ./shutdown.sh |

Effectuer une sauvegarde de l'instance sur **AWS** en suivant la procédure indiquée dans les sauvegardes des microservices.

### Mise à jour

Récupérer le **ZIP** de mise à jour an spécifiant le bon numéro de version (v=?) de **stock-api** qui est dans un repository **Nexus** et lancer la mise à jour avec les commandes :

|  |
| --- |
| /srv/stock-api$ wget -O stock-api.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a= stock-api \  &v=2.1 \  &p=zip  /srt/stock-api$sudo ./update.sh |

Le script **update.sh** arrête le microservice si c'est nécessaire, effectue une sauvegarde des fichiers de l'application et décompresse l'archive **ZIP**.

### Finalisation

Relancer le microservice **stock-api**.

## Microservice web-api

### Préalable

Aller dans le répertoire de l'application et arrêter le microservice **web-api** avec les commandes suivantes :

|  |
| --- |
| $cd /srv/web-api  /srv/web-api$sudo ./shutdown.sh |

Effectuer une sauvegarde de l'instance sur **AWS** en suivant la procédure indiquée dans les sauvegardes des microservices.

### Mise à jour

Récupérer le **ZIP** de mise à jour an spécifiant le bon numéro de version (v=?) de **web-api** qui est dans un repository **Nexus** et lancer la mise à jour avec les commandes :

|  |
| --- |
| /srv/web-api$ wget -O web-api.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a= web-api \  &v=2.1 \  &p=zip  /srt/web-api$sudo ./update.sh |

Le script **update.sh** arrête le microservice si c'est nécessaire, effectue une sauvegarde des fichiers de l'application et décompresse l'archive **ZIP**.

### Finalisation

Relancer le microservice **web-api**.

## Microservice production-api

### Préalable

Aller dans le répertoire de l'application et arrêter le microservice **production-api** avec les commandes suivantes :

|  |
| --- |
| $cd /srv/production-api  /srv/production-api$sudo ./shutdown.sh |

Effectuer une sauvegarde du microservice en suivant la procédure indiquée dans les sauvegardes des microservices.

### Mise à jour

Récupérer le **ZIP** de mise à jour an spécifiant le bon numéro de version (v=?) de **production-api** qui est dans un repository **Nexus** et lancer la mise à jour avec les commandes :

|  |
| --- |
| /srv/production-api$ wget -O production-api.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a= production-api \  &v=2.1 \  &p=zip  /srt/production-api$sudo ./update.sh |

Le script **update.sh** arrête le microservice si c'est nécessaire, effectue une sauvegarde des fichiers de l'application et décompresse l'archive **ZIP**.

### Finalisation

Relancer le microservice **production-api**.

## Microservice gestion-api

### Préalable

Aller dans le répertoire de l'application et arrêter le microservice **gestion-api** avec les commandes suivantes :

|  |
| --- |
| $cd /srv/gestion-api  /srv/gestion-api$sudo ./shutdown.sh |

Effectuer une sauvegarde du microservice en suivant la procédure indiquée dans les sauvegardes des microservices.

### Mise à jour

Récupérer le **ZIP** de mise à jour an spécifiant le bon numéro de version (v=?) de **gestion-api** qui est dans un repository **Nexus** et lancer la mise à jour avec les commandes :

|  |
| --- |
| /srv/gestion-api$ wget -O gestion-api.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a= gestion-api \  &v=2.1 \  &p=zip  /srt/gestion-api$sudo ./update.sh |

Le script **update.sh** arrête le microservice si c'est nécessaire, effectue une sauvegarde des fichiers de l'application et décompresse l'archive **ZIP**.

### Finalisation

Relancer le microservice **gestion-api**.

## Serveur de configuration

### Préalable

Aller dans le répertoire de l'application et arrêter le microservice **config-server** avec les commandes suivantes :

|  |
| --- |
| $cd /srv/config-server  /srv/config-server$sudo ./shutdown.sh |

Effectuer une sauvegarde du microservice en suivant la procédure indiquée dans les sauvegardes des microservices.

### Mise à jour

Récupérer le **ZIP** de mise à jour an spécifiant le bon numéro de version (v=?) de **web-api** qui est dans un repository **Nexus** et lancer la mise à jour avec les commandes :

|  |
| --- |
| /srv/config-server$ wget -O config-server.zip \ http://itcd.com/nexus/service/local/artifact/maven/redirect? \  r=pizzaflow \  &g=com.itcd.delivery \  &a= config-server \  &v=2.1 \  &p=zip  /srt/config-server$sudo ./update.sh |

Le script **update.sh** arrête le microservice si c'est nécessaire, effectue une sauvegarde des fichiers de l'application et décompresse l'archive **ZIP**.

### Finalisation

Relancer le microservice **config-server**.

# Supervision/Monitoring

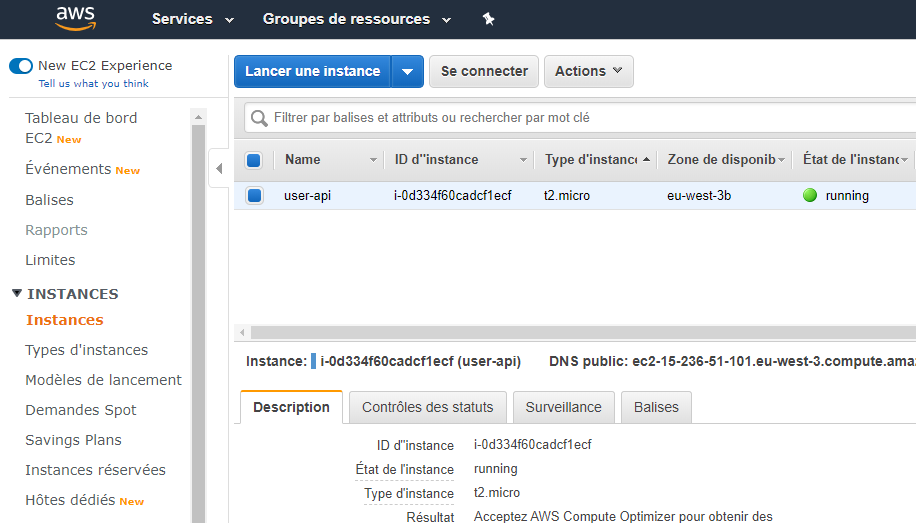
La supervision et le monitoring se font à plusieurs niveaux :

* Au niveau matériel avec les indicateurs des tableaux de bord des instances sur **AWS**.
* Supervision des métriques exposées par **actuator** sur **AWS** **CloudWatch**.
* Analiser les logs en temps réel avec **Kibana**.
* Remontée des sondes sur un compte privé **Twitter**.

## Supervision hardware des instances des microservices

Lancer un navigateur à l'adresse [www.pizzaflow.com](http://www.pizzaflow.com) pour vérifier que la page d'accueil s'affiche correctement.

Se rendre sur l'application de gestion des instances **EC2** sur **AWS** et sélectionner l'écran **"INSTANCES/Instances"** pour voir l'état des instances.

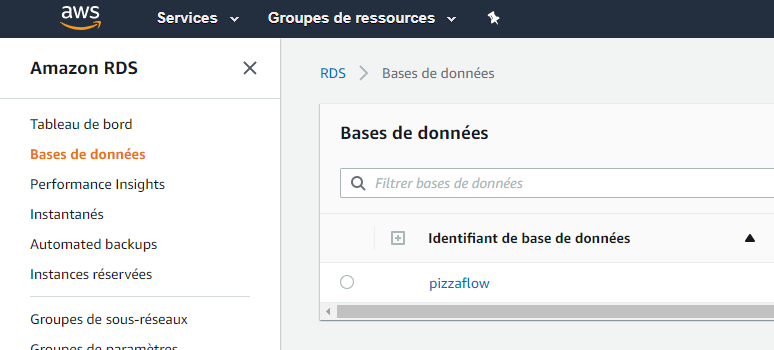


L'état des instances doit être indiqué en vert comme dans l'exemple ci-dessus avec **user-api**. Sélectionner l'instance voulue puis l'onglet "Surveillance" pour voir le tableau de monitoring de l'instance.

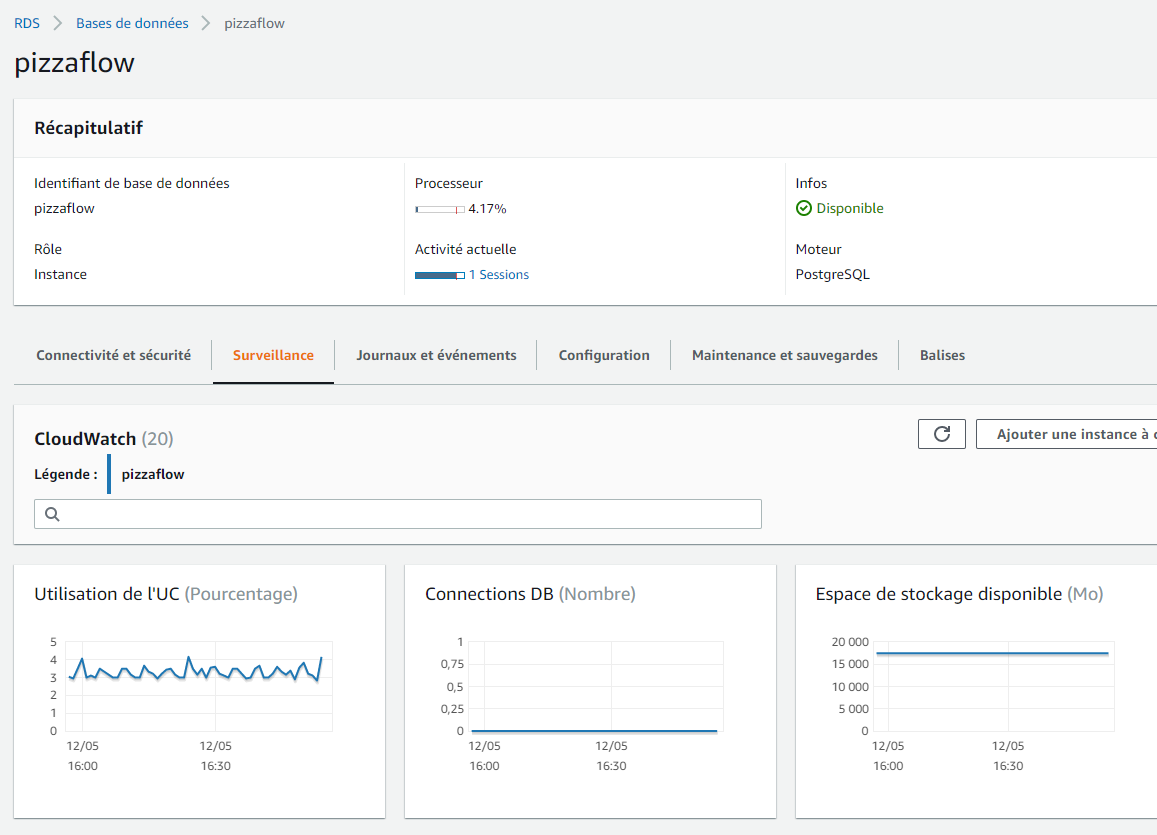


## Supervision hardware des instances des bases de données

Se rendre sur l'application de gestion **Amazon RDS** et sélectionner l'écran **"Base de données"** pour voir l'état des bases.

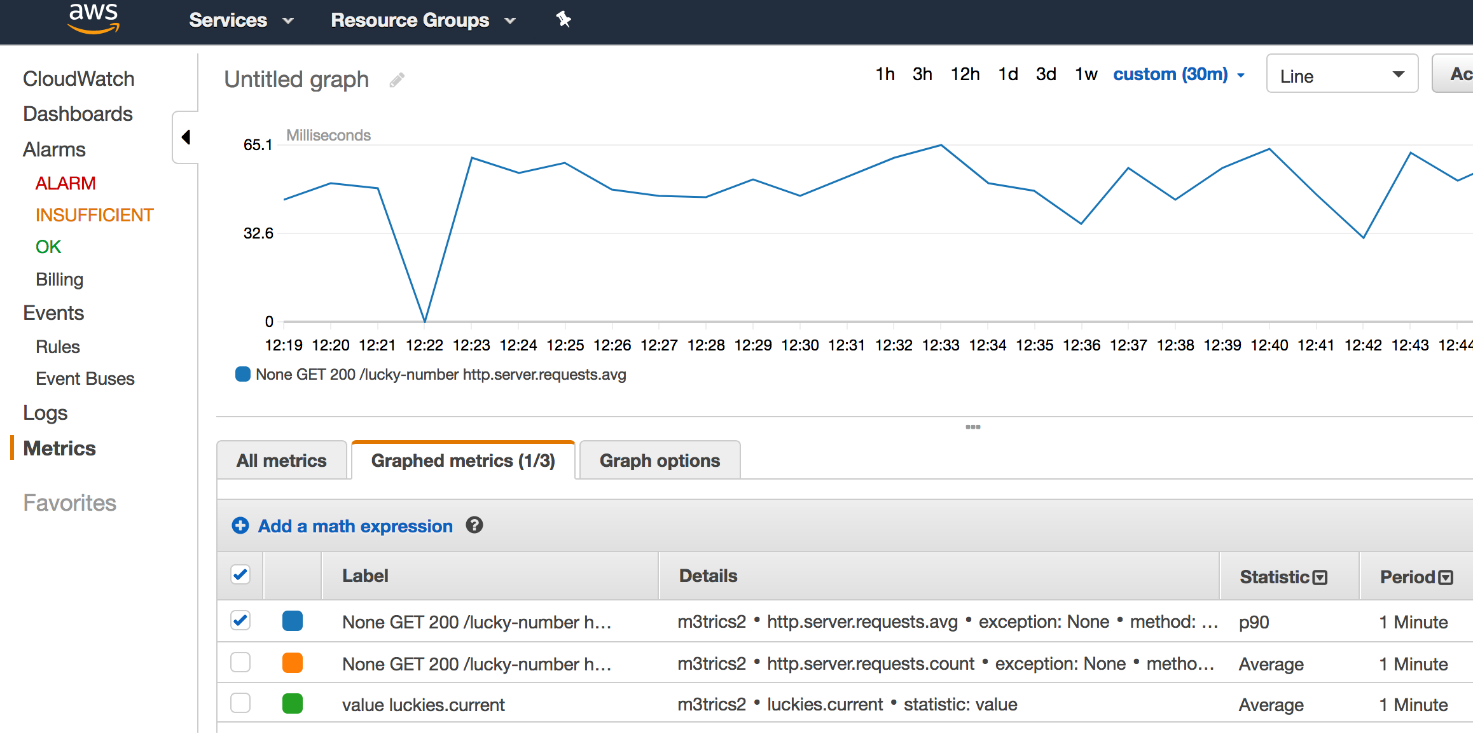


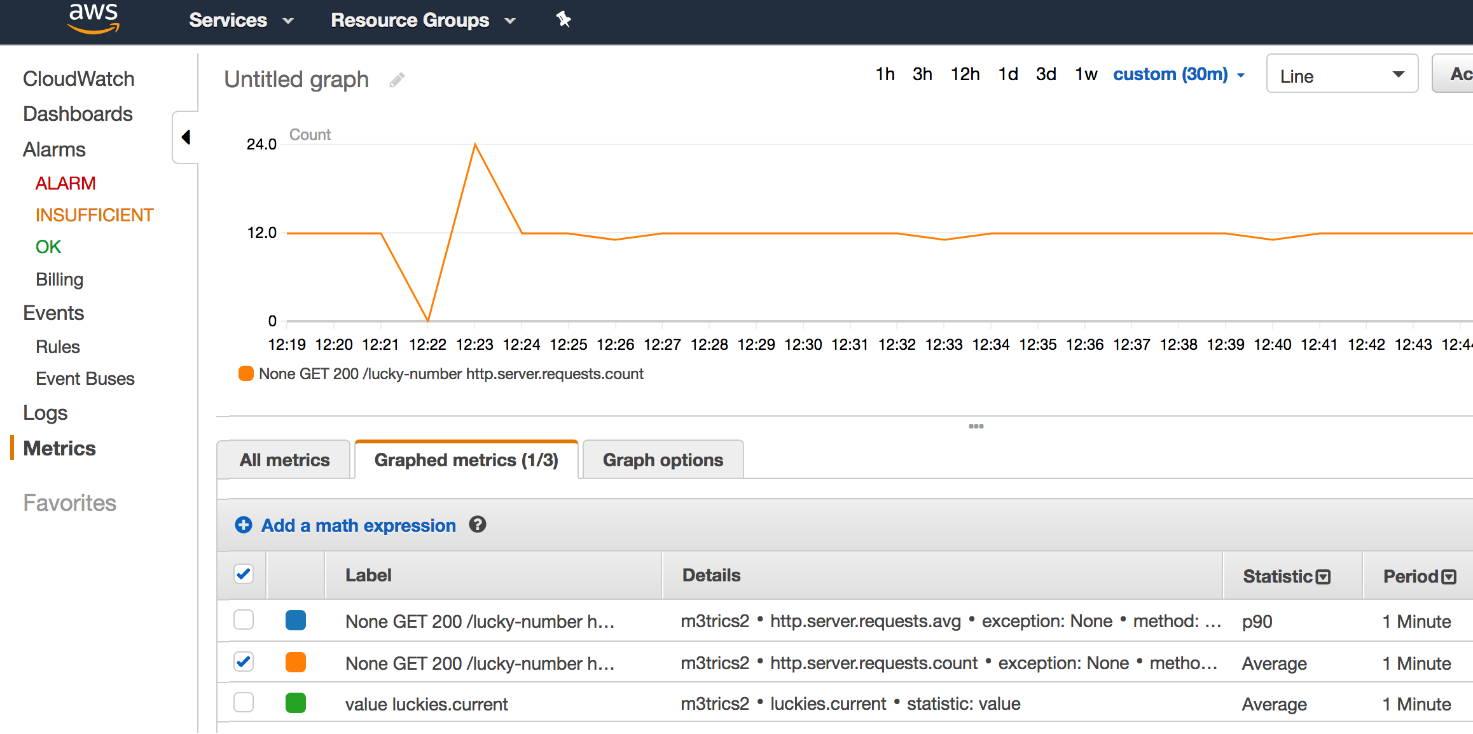
Sélectionner une base de données puis l'onglet **"Surveillance"** pour voir le tableau de monitoring de la base choisie.



## CloudWatch Metrics

L'application utilise **Spring Boot Actuator** et le starter **Spring Cloud AWS** pour récupérer les métriques sur un tableau de monitoring d'**AWS**. Se connecter sur l'interface d'**AWS** des instances des microservices et sélectionner l'écran **"Metrics"**.





## Analyse des logs

La seule façon de se rendre compte qu'une application fonctionne est d'analyser les logs qu'elle produits en temps réel. On peut suivre les temps de réponses, les codes de retours et le trafic sur un dashboard.

* **Elasticsearch** est la base de données **NoSQL** et moteur de recherche.
* **Logstash** est l'interface entre les données de logs générées et **Elasticsearch**.
* **Kibana** est le tableau de bord qui filtre les données venant **d'Elasticsearch** en temps réel.

Ce service est hébergé **logit.io** pour être découplé et indépendant de notre système applicatif.

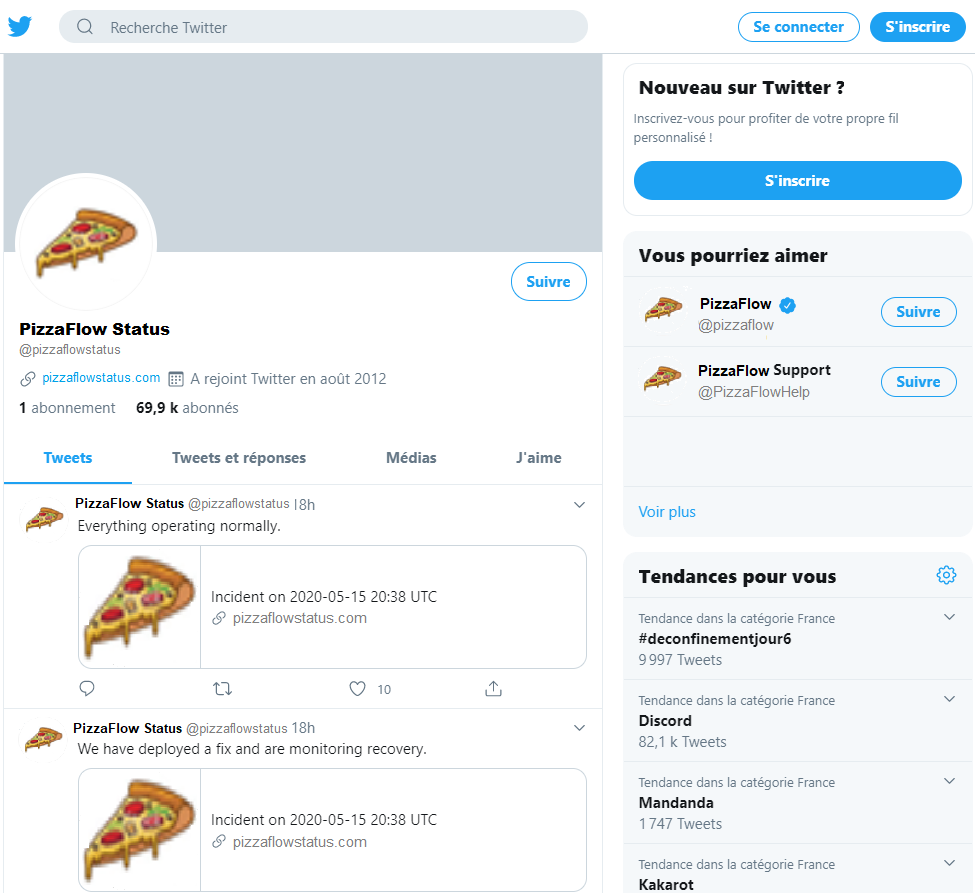


## Les sondes

Pour vérifier que l'application est toujours fonctionnelle et que la base de données n'a pas de problème, des batch effectuent des requêtes sur les microservices et comparent avec le résultat attendu.

La différence avec le résultat attendu est interprétée est une alarme est envoyées via **Twitter**. Un message de retour au service est envoyé sur **Twitter**.

Avec ce système de sonde, les responsables sont avertis dès qu'une défaillance apparait.

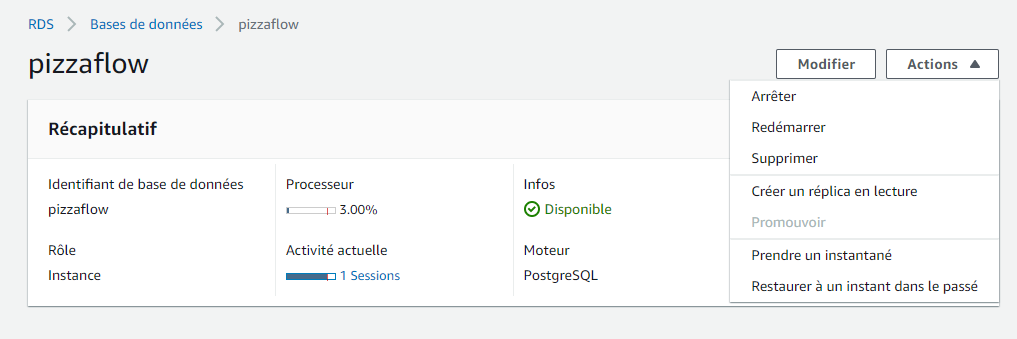


# Procédure de sauvegarde et restauration

## Base de données

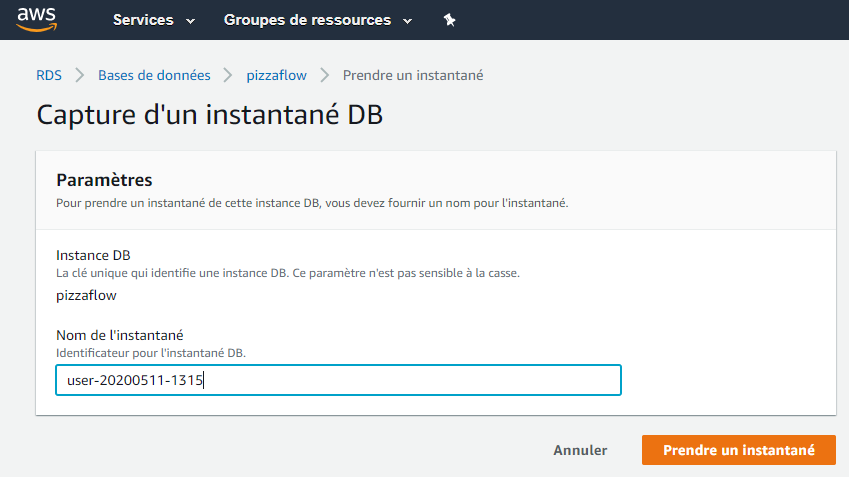
### Sauvegarde d'une base de données

Par sécurité il faut toujours faire une sauvegarde de l'état de la base de données pour pouvoir revenir en arrière en cas de problème de mise à jour. Depuis le menu "Action" de l'interface **Amazon RDS** de la base de données choisie sélectionner "Prendre un instantané".

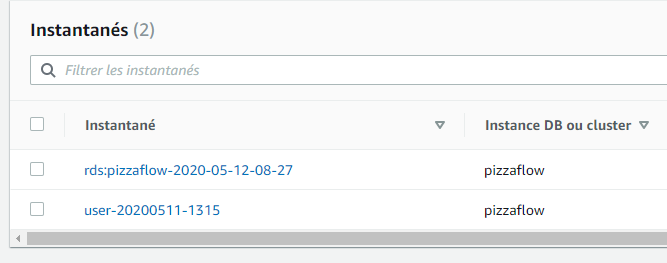


Dans la fenêtre qui s'ouvre on spécifie le nom de l'instantané en respectant l'expression :

**[nom-base]YYYYMMDD-HHMM**



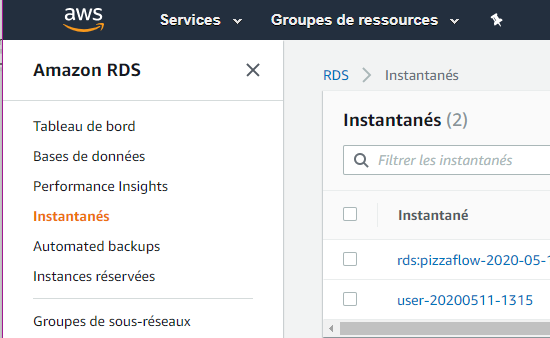
Il apparait ensuite dans la liste des instantanés de la base.



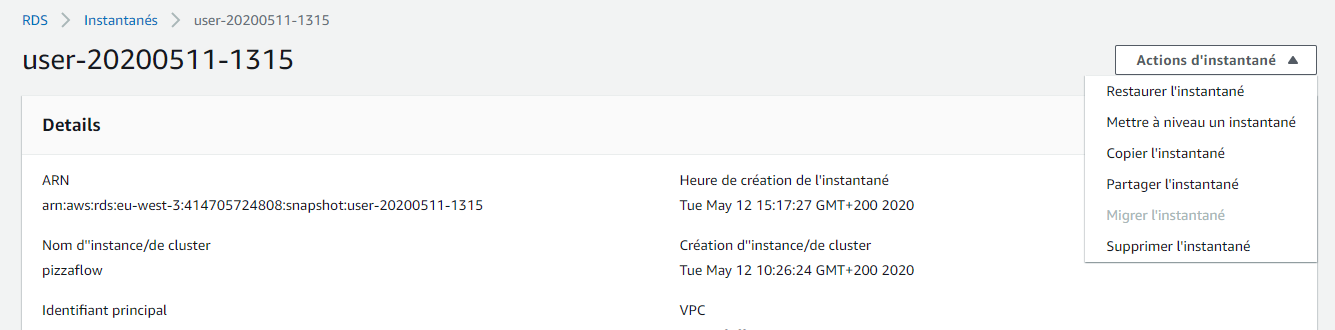
Il suffira de le sélectionner l'instantané pour restaurer la base.

### Restauration d'une base de données

Dans l'arborescence à gauche de l'interface **Amazon RDS** de la base de données sélectionner **"Instantanés"**.



Sélectionner ensuite l'instantané de la base voulu et choisir **"Restaurer l'instantané"** dans le menu **"Action d'instantané"**.



La base de données est restaurée suivant l'instantané choisi.

## Microservices

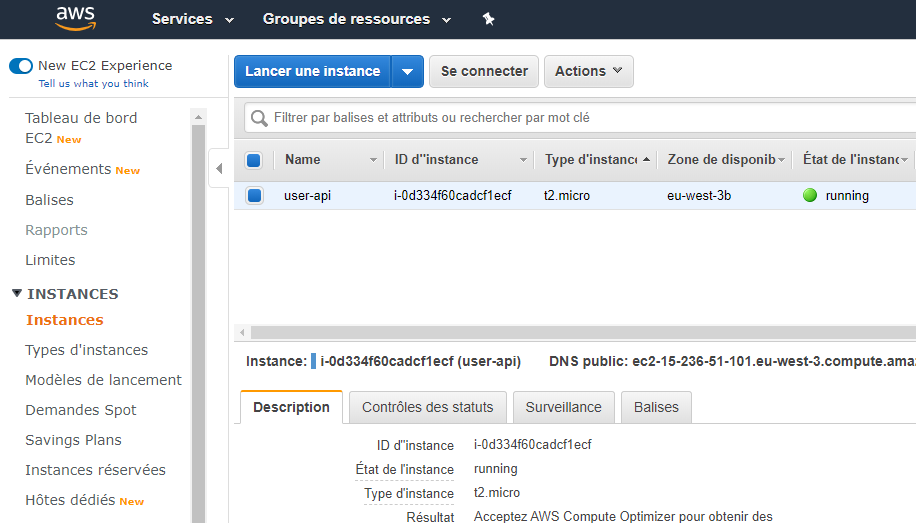
### Sauvegarde

La sauvegarde d'un microservice se fait en deux temps :

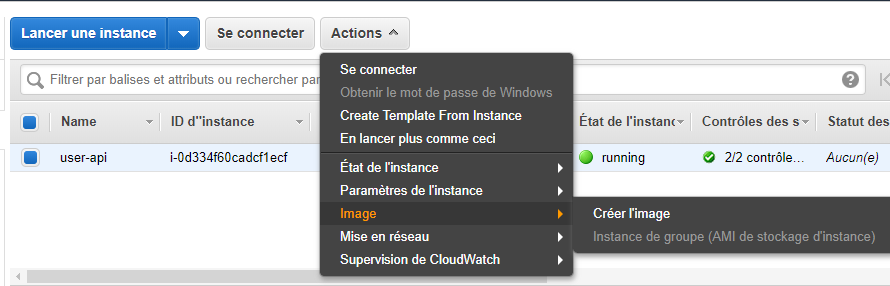
* Prise d'un instantané de l'instance EC2.
* Backup des fichiers du microservice.

#### Sauvegarde sur AWS.

Une sauvegarde des instances sous forme d'image est effectuée tous les jours avec une profondeur de 10 jours. Pour effectuer une sauvegarde manuelle, se rendre sur l'application de gestion des instances **EC2** sur **AWS** et sélectionner l'écran **"INSTANCES/Instances"** puis l'instance à sauvegarder dans l'écran de gauche.

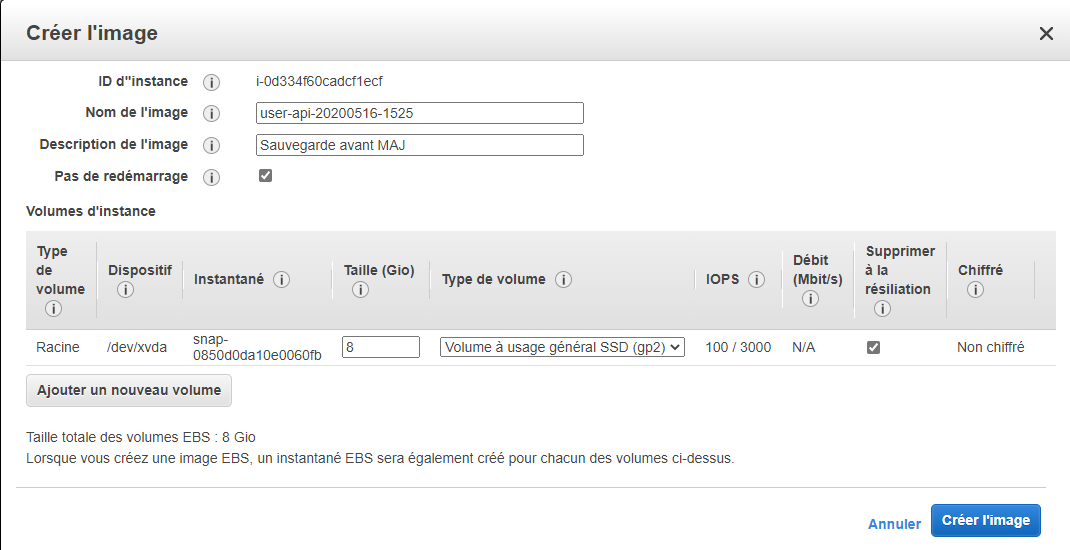


Sélectionner dans le menu **"Actions"** le sous-menu **"Image/Créer l'image"**.

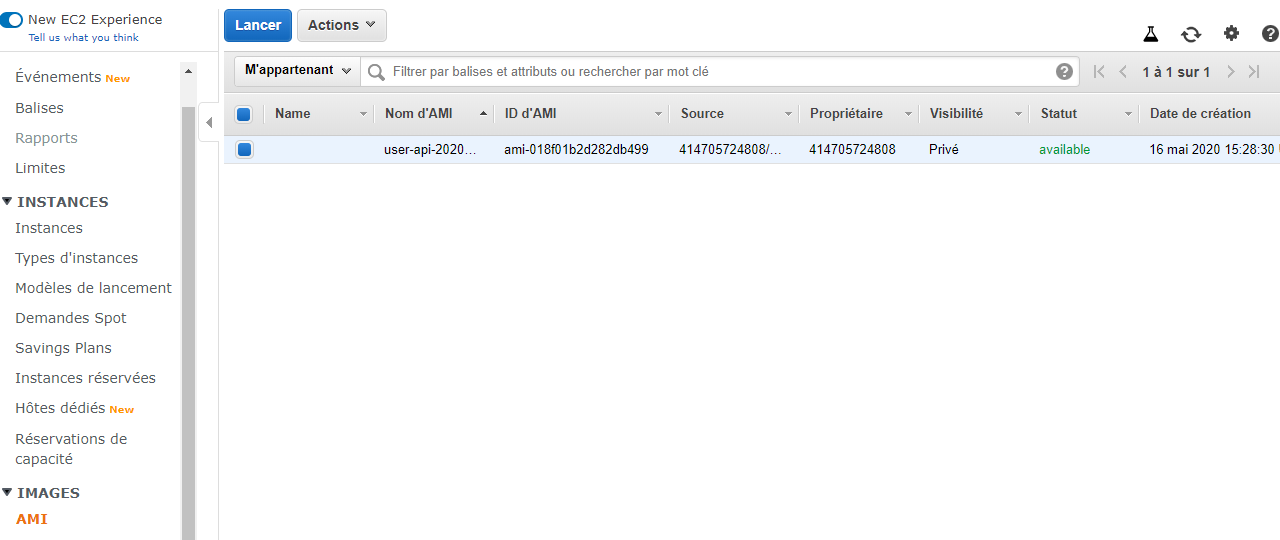


Indiquer le nom de l'image suivant la notation suivante et créer l'image :

**[nom-microservice]-YYYYMMDD-HHMM**



Vérifier que l'image a été créée et apparait dans la liste en sélectionnant dans l'arborescence de gauche le menu **"IMAGES/AMI"** avec son nom, son statut et sa date de création



#### Backup.

Se connecter à l'instance du microservice choisi avec **PuTTY**. Aller dans le répertoire de l'application avec la commande suivante en modifiant la valeur entre crochets par le nom du microservice et lancer le script **backup.sh**.

|  |
| --- |
| $cd /srv/[nom-microservice]  /srv/[nom-microservice]$sudo ./backup.sh |

Le script effectue une sauvegarde dans le répertoire dont le nom est au format :

**/srv/[nom-microservice]/backup/[nom-microservice]-YYYYMMDD-HHMM**

Et copie un instantané du microservice. Relancer le microservice avec la commande :

|  |
| --- |
| $cd /srv/[nom-microservice]  /srv/[nom-microservice]$sudo ./start.sh |

### Restauration ancienne configuration

Deux types de restaurations sont disponibles :

* Restauration des binaires de l'application à une version précédente.
* Restauration du système sous **AWS** à partir d'une image.

Une sauvegarde automatique journalière des instances est effectuée sur **AWS** avec un archivage des 10 dernières images. On peut ajouter des sauvegardes en créant des images manuellement.

#### Restauration des binaires

Se connecter à l'instance du microservice choisi avec **PuTTY**. Aller dans le répertoire de l'application et arrêter le microservice avec les commandes suivantes en modifiant la valeur entre crochets par le nom du microservice.

|  |
| --- |
| $cd /srv/[nom\_microservice]  /srv/[nom\_microservice]$sudo ./shutdown.sh |

Lancer le script de restauration en spécifiant la date **YYYYMMDD** et l'heure **HHMM** du système à restaurer avec la commande suivante :

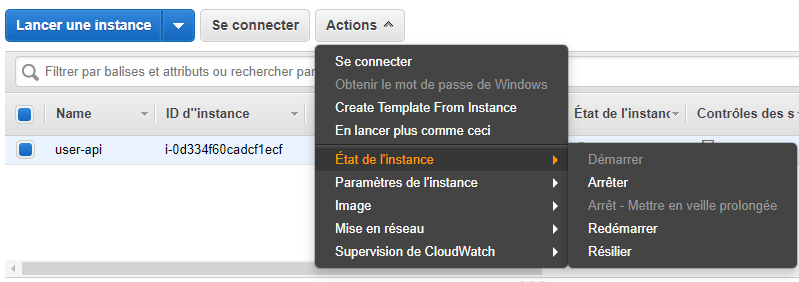
|  |
| --- |
| /srv/[nom\_microservice]$sudo ./restore.sh [YYYYMMDD]-[HHMM] |

La configuration dans le backup **nom-microservice-YYYYMMDD-HHMM** est restaurée. Relancer le microservice avec la commande :

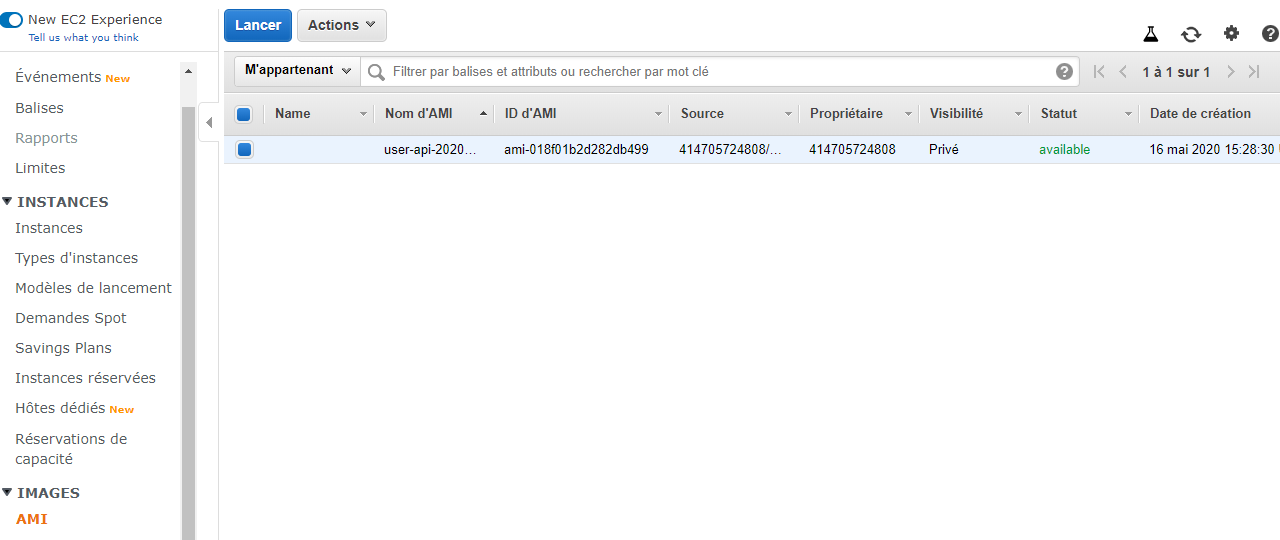
|  |
| --- |
| /srv/[nom\_microservice]$sudo ./start.sh |

#### Restauration d'une image d'instance sur AWS

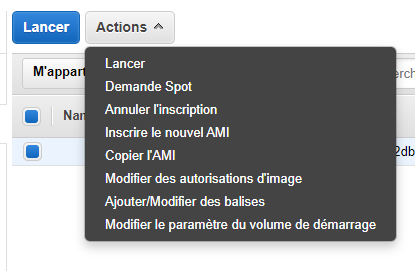
Pour effectuer une restauration manuelle d'une instance à l'aide de son image, se rendre sur l'application de gestion des instances **EC2** sur **AWS** pour arrêter l'instance en cours. Sélectionner l'instance puis dans le menu **"Actions"** le sous-menu **"État de l'instance/Arrêter"**.



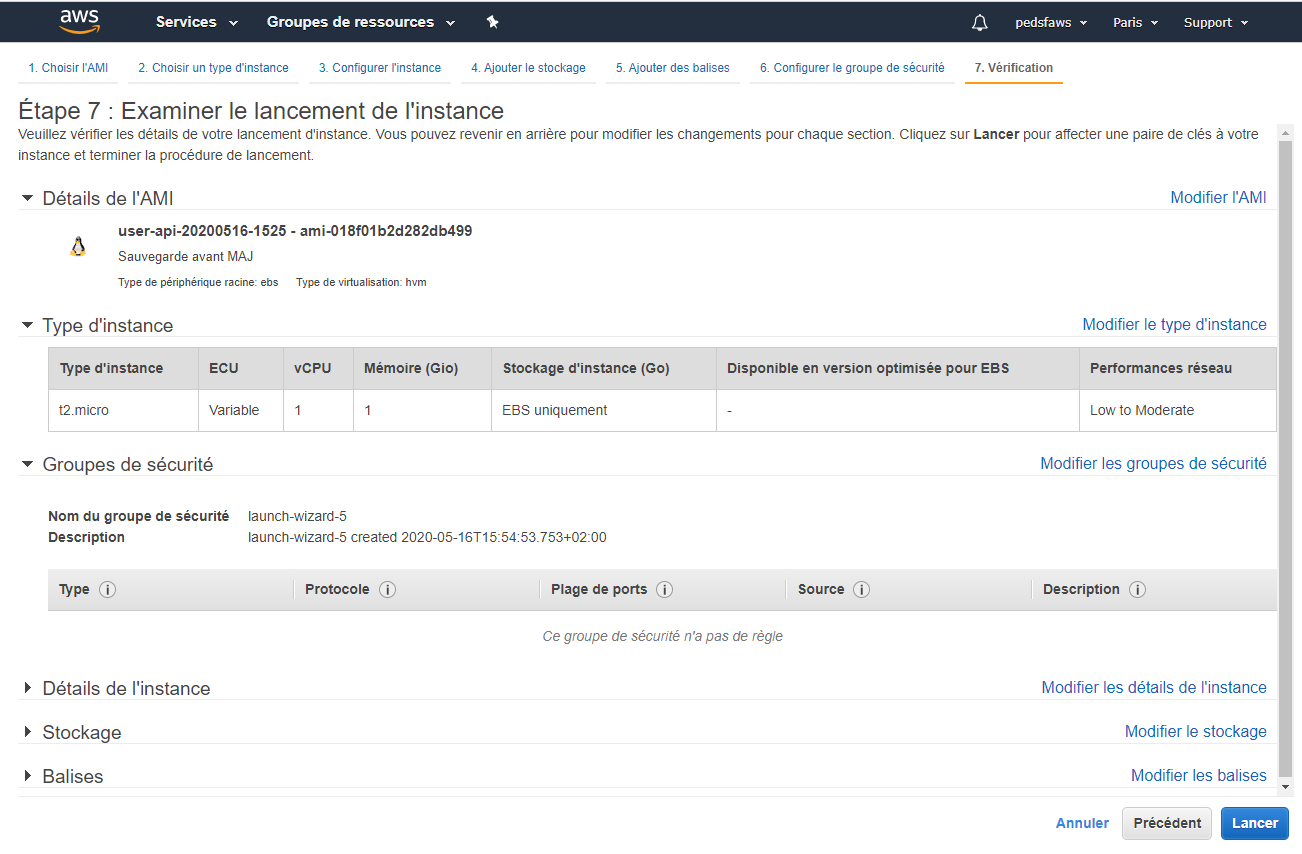
Sélectionner l'écran **"IMAGES/AMI"** puis l'image de l'instance à restaurer dans l'écran de gauche.



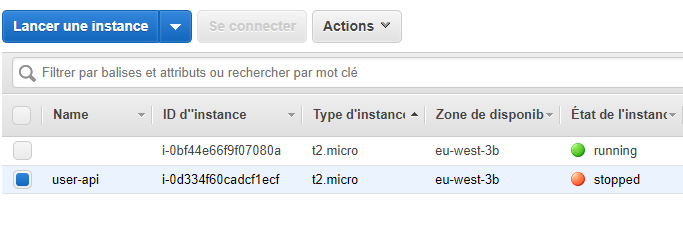
Dans le menu **"Action"** sélectionner **"Lancer"**.



Dans les écrans suivants vérifier et valider.



Vérifier les clés et valider pour afficher l'écran des instances.



Après vérification du fonctionnement de l'application, on peut supprimer l'ancienne instance qui arrêtée en choisissant **"État de l'instance/Résilier"** dans le menu **"Actions"**.

# Glossaire

|  |  |
| --- | --- |
| **AMI** | (**Amazon Machine Image**) logiciel d'exploitation Amazone de type linux. |
| **AWS** | (**Amazon Web Services**) service internet d'Amazon. |
| **CNIL** | (**Commission nationale de l'informatique et des libertés**). |
| **CSS** | (**Cascading Style Sheets**) fichier de style pour la présentation des pages **HTML**. |
| **DTO** | (**Data Transfer Object**) type d'objet permettant de transférer des données. |
| **EC2** | (**Elastic Cloud Compute**) serveur de base permettant d'intégrer de nombreux systèmes. |
| **IAM** | (**Identity and Access Management**) service d'Amazon pour gérer l'authentification des utilisateurs. |
| **Jasypt** | (**Java Simplified encryption**) librairie java qui permet d'effectuer un cryptage basic dans des fichiers de configuration. |
| **JDK** | (**Java Developer Kit**) outils de développement du langage **Java**. |
| **JRE** | (**Java Runtime Environment**) outils pour exécuter un exécutable **Java**. |
| **JS** | **Javascript** est un langage de script pour les pages web. |
| **JSON** | (**JavaScript Object Notation**) format léger d'échange de données facilement compréhensible par l'homme et manipulable par l'ordinateur. |
| **load-balancing** | Action de répartir la charge entre plusieurs instances d'une même application. |
| **NoSQL** | (**Not Only SQL**) Type de base de données qui n'utilise pas l'architecture classique des bases de données relationnelles **SQL**. |
| **pgAdmin** | (**PostgreSQL Admin**) logiciel d'administration de bases de données. |
| **PPK** | (**PuTTY Private Key**) Fichier de clé privée pour **PuTTY**. |
| **PuTTY** | Logiciel pour se connecter à distance sur une machine. |
| **RDS** | (**Relational Database Service**) Serveur avec un **SGBD-R** intégré. |
| **Repository** | Répertoire dans le cloud. |
| **S3** | (**Simple storage Service**) serveur de fichiers static pour les sites web. |
| **SGBD-R** | (**Système de Gestion de Bases de Données Relationnelles**). |
| **SLF4J** | (**Simple Loggin Facade for Java**) couche abstraite pour l'utilisation de différents loggers. |
| **VM** | (**Virtual Machine**) machine virtuelle qui contient un système d'exploitation pour faire fonctionner une application. |