Projet 8

ocpizza.com

Dossier d'exploitation

Version 1.0

Auteur
Thibaut Vuillaume
Développeur junior

TABLE DES MATIERES

1 - VERSIONS

Auteur	Date	Description	Version
Thibaut Vuillaume	09/10/2019	Finalisation du document	1.0
Thibaut Vuillaume	02/10/2019	Rédaction du document	05
Thibaut Vuillaume	25/09/2019	Création du document	0.1

2 - INTRODUCTION

2.1 - Objet du document

Le présent document constitue le dossier d'exploitation de l'application ocpizza.com

Objectif du document : préciser la démarche de déploiement de l'application ocpizza.com.

2.2 - Références

Pour de plus amples informations concernant l'application, se référer à :

1. **ocpizza.com - dossier de conception fonctionnelle.docx**
2. **ocpizza.com - dossier de spécifications techniques.pdf**

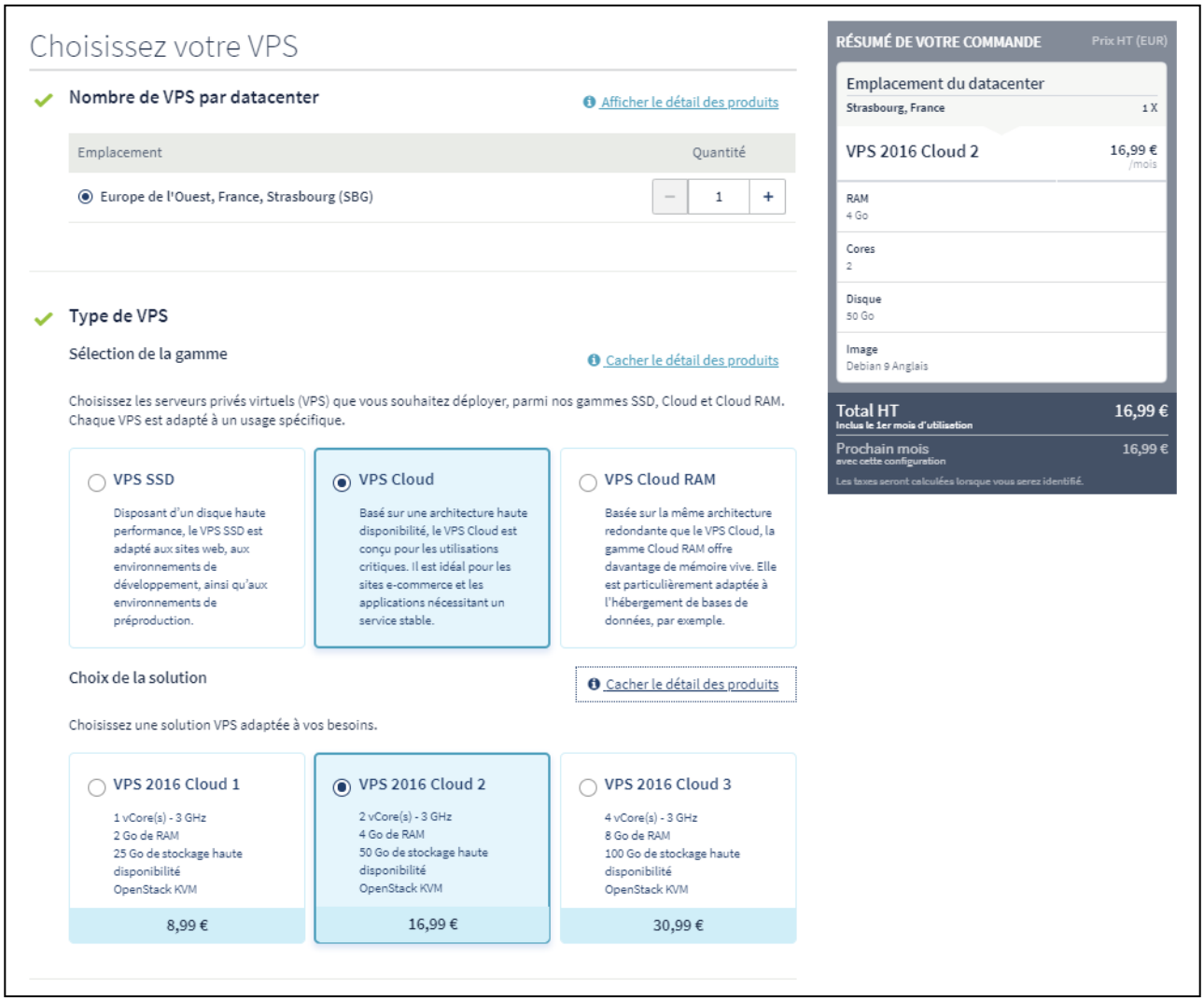
3 - PRÉ-REQUIS

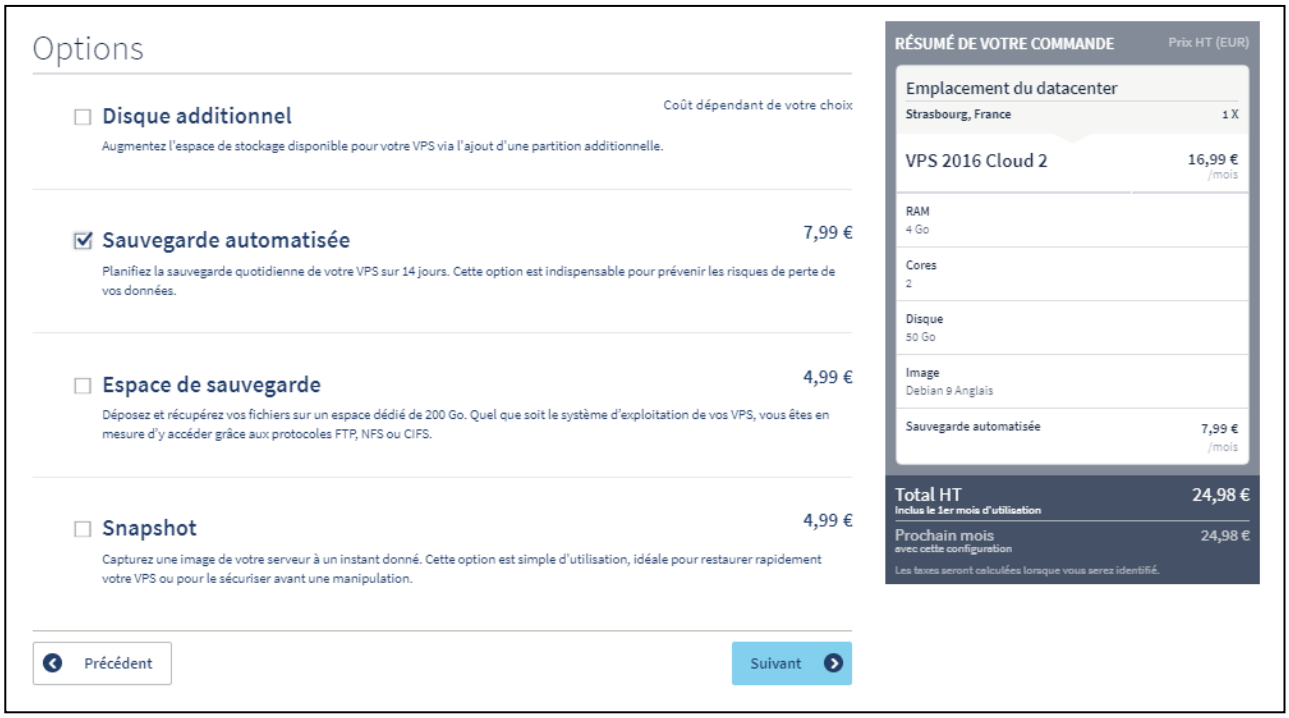
3.1 - Système

3.1.1 - Fournisseur de service

Le serveur qui hébergera l'application ocpizza.com et ses composants [1]sera fourni par ovh.com.

L'offre retenue sera VPS Cloud 2, proposée à 16.99€ HT/mois.





Une telle infrastructure sera suffisante pour garantir un bon niveau de performance de l'application, avec une disponibilité garantie à 99.99%. La sauvegarde automatisée préviendra tout risque de perte de données.

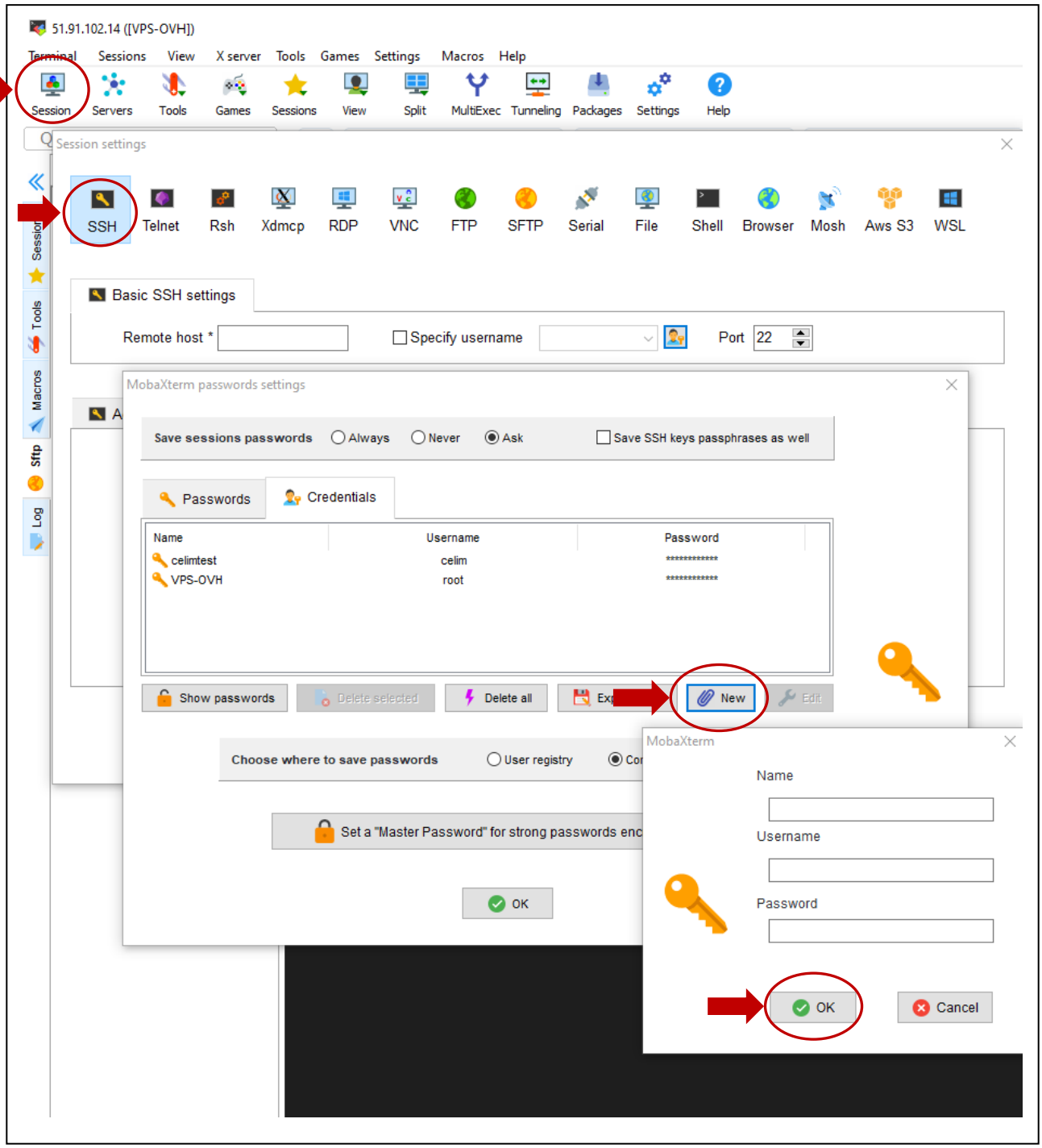
3.1.2 - Administration du serveur hôte

3.1.2.1 - Installation de MobaXterm

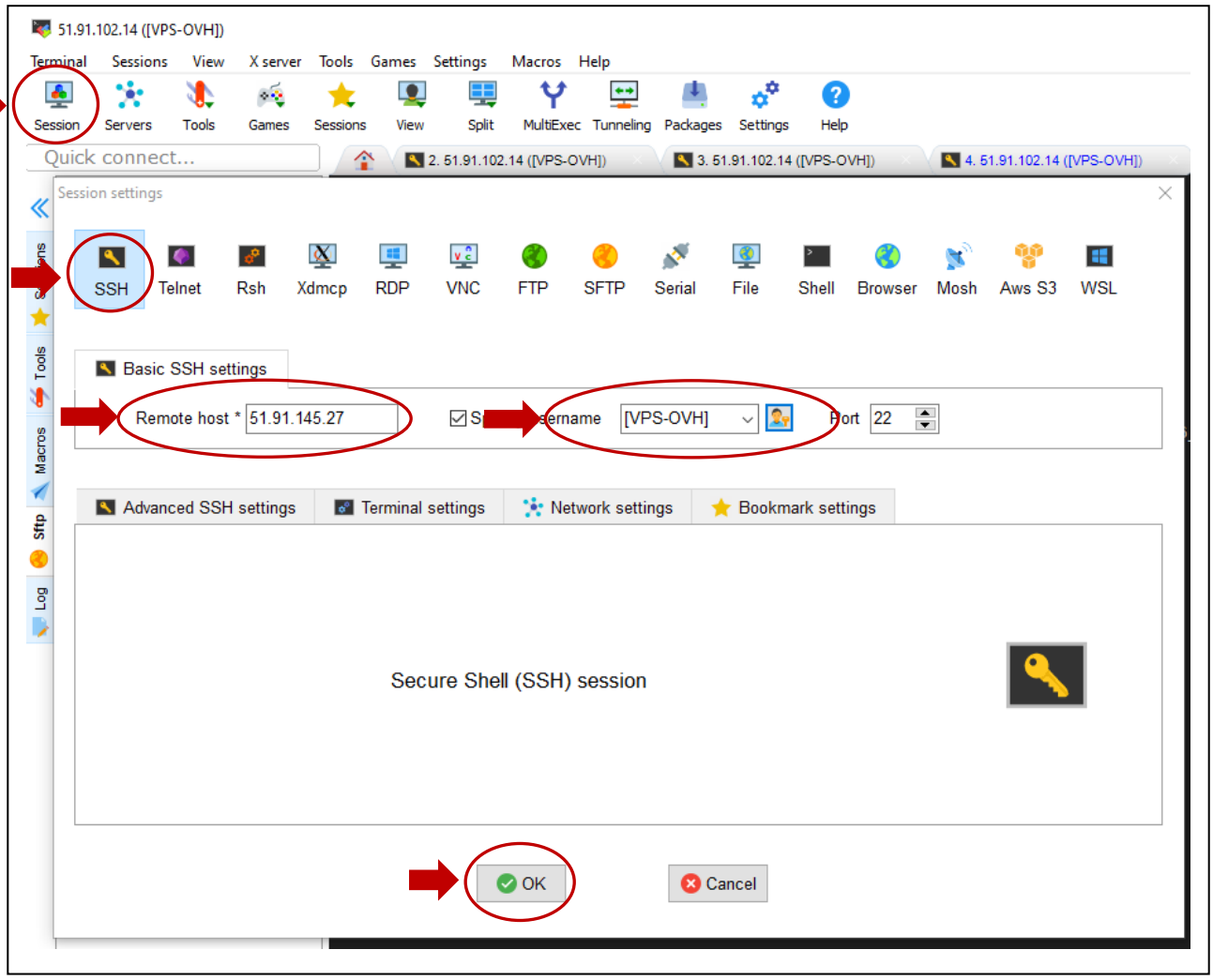
Télécharger puis installer MobaXterm :

https://mobaxterm.mobatek.net/download.html

3.1.2.2 - Paramétrage de MobaXterm

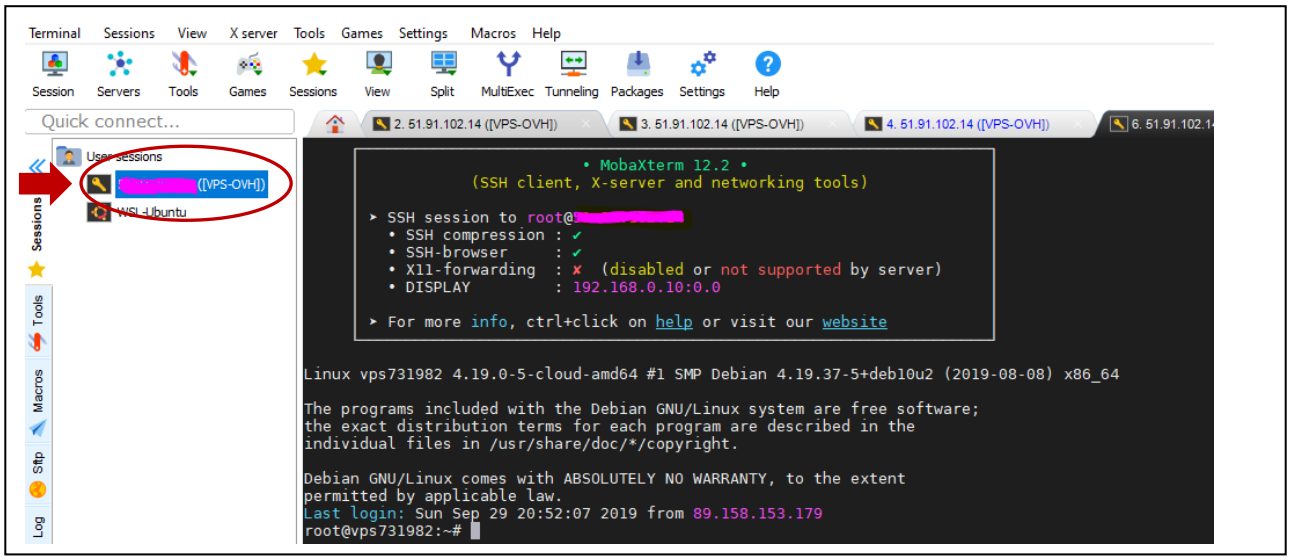


Suivre la séquence ci-dessus et entrer le Username et le Password fourni par le fournisseur de service.



Suivre la séquence ci-dessus : saisir l'adresse IP fourni par le fournisseur de service, sélectionner le Username précédemment enregistré puis valider.

Accéder ensuite au terminal en double cliquant sur la session qui vient d'être créée.



3.1.3 - Installation de Docker et Docker Compose sur le serveur hôte

3.1.3.1 - Docker[2]

Accéder au terminal du serveur hôte[3], et exécuter les commandes suivantes :

```
# mise à jour de la liste des packages
$ sudo apt update

# installation des packages prérequis qui autorisent apt à utiliser des packages via HTTPS
$ sudo apt install apt-transport-https ca-certificates curl gnupg2 software-properties-common

# ajout de la clé GPG pour le dépôt Docker officiel sur le serveur hôte
$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add –

# ajout du dépôt Docker aux sources APT
```

```
$ sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/debian $(lsb_release -cs) stable"

# mise à jour de la base de données des packages avec les packages Docker depuis le dépôt nouvellement ajouté
$ sudo apt update

# garantie une installation de Docker depuis le dépôt Docker et non depuis le dépôt Debian par défaut
$ apt-cache policy docker-ce
```

```
# output attendu
> docker-ce:
  Installed: (none)
  Candidate: 18.06.1~ce~3-0~debian
  Version table:
     18.06.1~ce~3-0~debian 500
        500 https://download.docker.com/linux/debian stretch/stable amd64 Packages

# installation de Docker
$ sudo apt install docker-ce

# verification de l'installation
$ sudo systemctl status docker
```

```
# output attendu
> docker.service - Docker Application Container Engine
  Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
  Active: active (running) since Thu 2018-07-05 15:08:39 UTC; 2min 55s ago
    Docs: https://docs.docker.com
 Main PID: 21319 (dockerd)
  CGroup: /system.slice/docker.service
          ├─21319 /usr/bin/dockerd -H fd://
          └─21326 docker-containerd --config /var/run/docker/containerd/containerd.toml
```

```
# ajout du username  au groupe docker afin de ne plus avoir besoin de la commande sudo
$ sudo usermod -aG docker ${USER}

# application des changements
$ su - ${USER}

# verification
$ id -nG

# output attendu
> root docker
```

3.1.3.2 - Docker Compose

Accéder au terminal du serveur hôte, et exécuter les commandes suivantes :

```
# téléchargement de la release actuelle stable de Docker Compose
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

# application des permissions executables
$ sudo chmod +x /usr/local/bin/docker-compose
```

3.1.4 - Serveur de base de données

La base de données sera sous PostgreSQL 11.

En vue des 3 environnements Dev, Test et Prod, 3 schémas seront créés :

- ocpizza_dev
- ocpizza_test
- ocpizza

L'insertion des données sera faite par le client dans l'interface prévue à cet effet. Cette interface ne sera accessible que par les collaborateurs dont le compte dispose du rôle « admin ».

3.1.4.1 - Caractéristiques techniques

La base de données sera déployée dans un conteneur Docker[4].

L'image Docker [5]qui sera construite est décrite dans un dockerfile : Dockerfile-db.

L'image utilise une distribution Linux : Debian GNU/Linux 9.11 (stretch).

L'allocation des ressources du serveur hôte au conteneur est gérée automatiquement par Docker. On peut toutefois modifier les paramètres par défauts, en se référant à cette documentation : https://docs.docker.com/config/containers/resource_constraints/.

3.1.4.2 - Configuration

Nom de la base de données : db_ocpizza.

Host : xxx-xxx-xxx-xxx.

Port : 5432.

URL : jdbc:postgresql:// xxx-xxx-xxx-xxx:5432/db_ocpizza

3.1.5 - Serveur Backend

Le Backend[6] de l'application est développé en Java 8. Le framework Springboot 2 est utilisé, avec une architecture REST.

Le backend de l'application sera packagé dans un fichier .war. Ce dernier sera à importer sur le serveur hôte, à l'emplacement : /opt/ocpizza/docker/build/back. Il sera déployé dans un conteneur Docker.

3.1.5.1 - Caractéristiques techniques

L'image Docker qui sera construite est décrite dans un dockerfile : Dockerfile-back.

L'image utilise une distribution Linux : Debian GNU/Linux 9 (stretch).

L'allocation des ressources du serveur hôte au conteneur est gérée automatiquement par Docker.

3.1.6 - Serveur Frontend

Le Frontend [7]de l'application est développé en Angular 7. Le contenu de l'archive « ocpizza-front.zip » sera à importer sur le server host, à l'emplacement : /opt/ocpizza/docker/build/front/. Il sera déployé dans un conteneur Docker.

3.1.6.1 - Caractéristiques techniques

L'image Docker qui sera construite est décrite dans un dockerfile : Dockerfile-front.

L'image utilise une distribution de Linux : Debian GNU/Linux 9.9 (stretch)

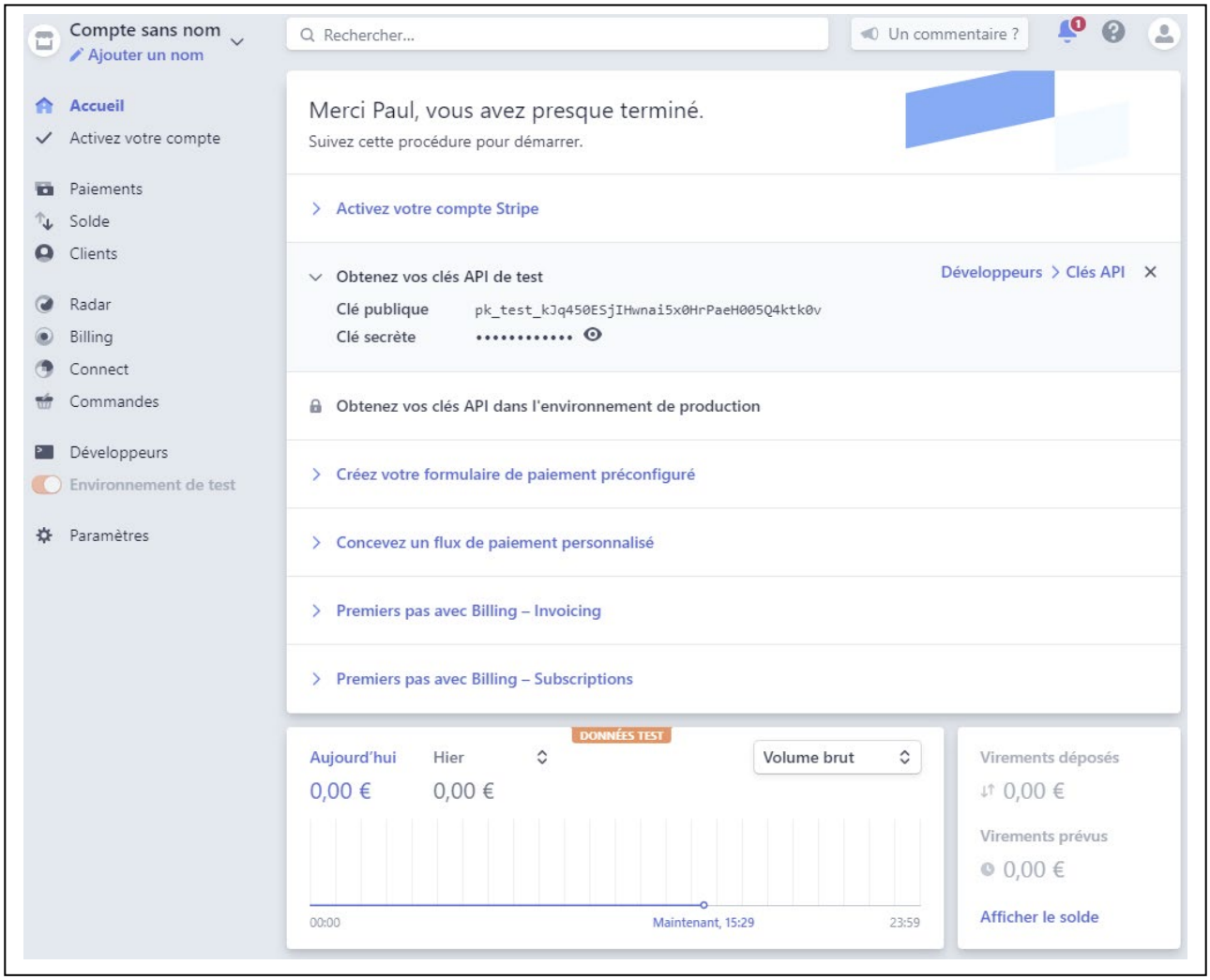
L'allocation des ressources du serveur hôte au conteneur est gérée automatiquement par Docker.

3.2 - Service de paiement externe : Stripe

Un service de paiement externe est utilisé. Il s'agit de Stripe.

Un compte test a été créé. Il est accessible à cette adresse :
https://dashboard.stripe.com/test/dashboard.

Identifiant : tvuillaume10@gmail.com
Mot de passe : 71eult1IZKuc



4 - Procédure de déploiement

4.1 - Import des différents composants sur le serveur hôte

4.1.1 - Artefacts

- **ocpizza-front.zip** : fichier obtenu en archivant le Frontend de l'application avec Winzip. Extraire et importer à l'emplacement :

  < /opt/ocpizza/docker/build/front >.

- **ocpizza-back.war** : fichier obtenu en packageant le Backend de l'application avec Maven. Importer à l'emplacement : < /opt/ocpizza/docker/build/back >.

4.1.2 - Dockerfiles

- Dockerfile-back : fichier qui décrit l'image Docker utilisée par le conteneur du Backend. Importer à l'emplacement < /opt/ocpizza/docker/build/back >.
- Dockerfile-front : fichier qui décrit l'image Docker utilisée par le conteneur du Frontend. Importer à l'emplacement < /opt/ocpizza/docker/build/front >.
- Dockerfile-db : fichier qui décrit l'image Docker utilisée par le conteneur de la base de données. Importer à l'emplacement < /opt/ocpizza/docker/db >.

4.1.3 - Docker Compose

- docker-compose.yml : ce fichier définit les services et le réseau qui seront créés par Docker. Importer à l'emplacement < /opt/ocpizza/docker >

```
version: '3.7'

services:
    db-ocpizza:
        build:
```

```
            context: .
            dockerfile: db/Dockerfile-db
        container_name: cont-ocpizza-db
        restart: unless-stopped
        ports:
            - 5432:5432
        expose:
            - 5432
        environment:
            POSTGRES_USER: admin_ocp
            POSTGRES_PASSWORD: 123
            POSTGRES_DB: db_ocpizza
        volumes:
            - db_data:/var/lib/postgres/data
        networks:
            - ocp-network

    back-ocpizza:
        depends_on:
            - db-ocpizza
        build:
            context: .
            dockerfile: build/back/Dockerfile-back
        container_name: cont-ocpizza-back
        ports:
            - 8080:8080
        expose:
            - 8080
        environment:
            SPRING_DATASOURCE_USERNAME: admin_ocp
            SPRING_DATASOURCE_PASSWORD: 123
            SPRING_DATASOURCE_URL: jdbc:postgresql://db-ocpizza:5432/db_ocpizza
        networks:
            - ocp-network

    front-ocpizza:
        build:
            context: .
            dockerfile: build/front/Dockerfile-front
        container_name: cont-ocpizza-front
        volumes:
            - 'front:/app'
            - '/app/node_modules'
        ports:
            - 4200:4200
        networks:
            - ocp-network

networks:
    ocp-network:

volumes:
    db_data:
        driver: local
        driver_opts:
```

```
            type: 'none'
            o: 'bind'
            device: '/opt/ocpizza/docker/volumes/postgres'
    front:
```

4.1.4 - Scripts

Scripts SQL :

- create_db_ocpizza : ce fichier contient le script SQL de création du schéma de destiné à l'environnement de production.
- create_db_ocpizza_test : ce fichier contient le script SQL de création du schéma de destiné à l'environnement de test.
- create_db_ocpizza_dev : ce fichier contient le script SQL de création du schéma de destiné à l'environnement de développement.

Ces fichiers sont à importer à l'emplacement < /opt/ocpizza/docker/db >.

Scripts Shell :

- backup-db.sh : ce fichier contient le script Shell qui lancera la commande de backup de la base de données PostgreSQL.
- extract-backup.sh : ce fichier contient le script Shell qui lancera la commande de récupération du backup de la base de données depuis son conteneur.

Ces fichiers sont à importer à l'emplacement < /opt/ocpizza/docker/sh >.

4.1.5 - Variables d'environnement

Le serveur hôte dispose des variables d'environnement suivantes :

NOM DE VARIABLE	CONTENU
SHELL	/bin/bash
PWD	/opt/ocpizza
LOGNAME	root
XDG_SESSION_TYPE	tty

HOME	/root
LANG	en_US.UTF-8
SSH_CONNECTION	xxx.xxx.xxx.xxx 63434 xxx.xxx.xxx.xxx
XDG_SESSION_CLASS	user
TERM	xterm
USER	root
SHLVL	1
XDG_SESSION_ID	6921
XDG_RUNTIME_DIR	/run/user/0
SSH_CLIENT	xxx.xxx.xxx.xxx 63434 22
PATH	/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
MAIL	/var/mail/root
SSH_TTY	/dev/pts/4
_	/usr/bin/printenv
OLDPWD	/root

4.2 - Déploiement du Backend de l'application (service)

4.2.1 - Artefacts[8]

Le Backend de l'application est packagée dans un fichier .war : ocpizza-back.war.

4.2.2 - Dockerfile

```
FROM openjdk:8
ADD build/back/ocpizza-back.war ocpizza-back.war
ENTRYPOINT ["java", "-jar", "ocpizza-back.war"]
```

4.2.3 - Variables d'environnement

Le conteneur Docker contiendra les variables d'environnements suivantes :

NOM DE VARIABLE	CONTENU
LANG	C.UTF-8
HOSTNAME	a53f91a1da6b
SPRING_DATASOURCE_URL	jdbc:postgresql://db-ct:5432/db_contacts
JAVA_HOME	/usr/local/openjdk-8
JAVA_VERSION	8u222
PWD	/
HOME	/root
SPRING_DATASOURCE_PASSWORD	123
TERM	xterm
JAVA_BASE_URL	https://github.com/AdoptOpenJDK/openjdk8-upstream-binaries/releases/download/jdk8u222-b10/OpenJDK8U-jdk_
SHLVL	1
JAVA_URL_VERSION	8u222b10
SPRING_DATASOURCE_USERNAME	admin_ct
PATH	/usr/local/openjdk-8/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
_	/usr/bin/printenv

4.3 - Déploiement du Frontend de l'application (client)

4.3.1 - Artefacts

Le Frontend de l'application est packagée dans l'archive « ocpizza-front.zip ». Son contenu doit être uploadé sur le serveur hôte à l'emplacement </opt/ocpizza/front>

4.3.2 - Dockerfile

```
# image de base
FROM node:latest

# installe chrome pour les tests protractor
RUN wget -q -O - https://dl-ssl.google.com/linux/linux_signing_key.pub | apt-key add -
RUN sh -c 'echo "deb [arch=amd64] http://dl.google.com/linux/chrome/deb/ stable main"
>> /etc/apt/sources.list.d/google.list'
RUN apt-get update && apt-get install -yq google-chrome-stable

# définit le répertoire de travail
WORKDIR /app

# ajoute `/app/node_modules/.bin` au $PATH
ENV PATH /app/node_modules/.bin:$PATH

# installe et met en cache les dépendances de l'application
COPY build/front/package.json /app/package.json
RUN npm install
RUN npm install -g @angular/cli@7.3.9

# copie app
COPY build/front/. /app

# démarre l'application
CMD ng serve --host 0.0.0.0
```

4.3.3 - Variables d'environnement

Le conteneur Docker contiendra les variables d'environnements suivantes :

NOM DE VARIABLE	CONTENU
YARN_VERSION	1.15.2
HOSTNAME	a057d3e67fd0
PWD	/app
HOME	/root
NODE_VERSION	12.2.0
TERM	xterm
SHLVL	1
PATH	/app/node_modules/.bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

_	/usr/bin/printenv

4.4 - Déploiement de la base de données

4.4.1 - Dockerfile

```
FROM postgres:latest
COPY db/create_db_ocpizza.sql /docker-entrypoint-initdb.d/init.sql
COPY db/insert_data_ocpizza.sql /opt/insert_data_ocpizza.sql
COPY sh/backup-db.sh /opt/backup-db.sh
RUN apt-get update \
&& apt-get install vim -y \
&& export EDITOR=/usr/bin/vim \
&& chmod +x /opt/backup-db.sh \
&& cd /opt \
&& mkdir db_backup \
```

4.4.2 - Création de la base de données et des schémas

La base de données et les schémas sont construits lorsque le conteneur Docker démarre.

Cela est rendu possible par la commande du Dockerfile :

```
COPY db/create_db_ocpizza.sql /docker-entrypoint-initdb.d/init.sql
```

Cette commande importe le fichier < create_db_ocpizza.sql > dans le conteneur. Il sera exécuté à la construction de l'image.

4.4.3 - Variables d'environnement

Le conteneur Docker contiendra les variables d'environnements suivantes :

NOM DE VARIABLE	CONTENU
LANG	en_US.utf8
HOSTNAME	f9e4161aef0a
PG_MAJOR	11

PWD	**/**
HOME	**/root**
PG_VERSION	**11.5-1.pgdg90+1**
GOSU_VERSION	**1.11**
PGDATA	**/var/lib/postgresql/data**
POSTGRES_DB	**db_contacts**
TERM	**xterm**
POSTGRES_PASSWORD	**123**
POSTGRES_USER	**admin_ct**
SHLVL	**1**
PATH	**/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/lib/postgresql/11/bin**
_	**/usr/bin/printenv**

4.5 - Vérification

Afin de vérifier le bon déploiement de l'application, accéder au terminal du serveur hôte et exécutez la commande <docker-compose ps>.

Output attendu :

```
Name                  Command                  State          Ports
------------------------------------------------------------------------------
cont-ocpizza-back     java -jar ocpizza-back.jar     Up      0.0.0.0:8080->8080/tcp
cont-ocpizza-front    /bin/sh -c ng serve --host ...  Up      0.0.0.0:4200->4200/tcp
cont-ocpizza-db   docker-entrypoint.sh postgres     Up      0.0.0.0:5432->5432/tcp
```

5 - Procédure de démarrage / arrêt

5.1 - Démarrage des composants

Accéder au terminal du serveur hôte, et exécuter la commande suivante :

```
$ docker-compose up
```

Cela construira toutes les images en suivant le fichier de configuration < docker-compose.yml > puis démarrera tous les conteneurs et services décrits dans ce dernier.

5.2 - Arrêt des composants

Accéder au terminal du serveur hôte, et exécuter la commande suivante :

```
$ docker-compose down
```

Cela arrêtera tous les services du < docker-compose.yml > et donc tous les composants.

5.3 - Arrêt d'un conteneur

Accéder au terminal du serveur hôte, et exécuter les commandes suivantes :

```
$ docker-compose stop <nom-du-service>
```

Cela arrêtera le conteneur Docker désigné par < nom-du- service >.

5.4 - Redémarrage d'un conteneur

Accéder au terminal du serveur hôte, et exécuter les commandes suivantes :

```
$ docker-compose restart <nom-du-service>
```

Cela redémarrera le conteneur Docker désigné par < nom-du- service >.

6 - PROCEDURE DE MISE A JOUR

Pendant une procédure de mise-à-jour, un service Docker doit être stoppé pour être redémarré. Une interruption de service est donc ici nécessaire.
Il est indispensable de n'exécuter ces procédures que lors des périodes les plus creuses en sollicitation de ocpizza.com.

6.1 - Mise à jour d'un composant

Quel que soit le composant concerné, Docker facilite grandement les choses.
Il suffit de :

- Importer le composant mis-à-jour sur le serveur hôte (le .war pour le Backend, le contenu du projet Angular pour le Frontend, le script SQL pour la base de données). Si besoin, modifier le Dockerfile concerné.
- Arrêter le conteneur concerné (cf. partie 5.2).
- Reconstruire l'image du conteneur concerné :

```
$ docker-compose build <nom-du-service>
```

- Redémarrer le conteneur concerné (cf. partie 5.3).
- Mise à jour des artefacts sur github : https://github.com/thibaut54/ocpizza.com/tree/master/build

6.2 - Mise à jour de la config Docker (Dockerfile / docker-compose.yml)

Modifier le ou les fichiers concernés.
Saisir la commande suivante dans le terminal du serveur hôte :

```
$ docker-compose up --force-recreate --no-deps --build <nom-du-service>
```

Cela reconstruira l'image du service et relancera son conteneur.

7 - SUPERVISION/MONITORING

7.1 - Monitoring du serveur hôte

OVH fourni un service de monitoring, au niveau du serveur et des applications, intégrant une service d'alerte email et sms en cas de défaillance.

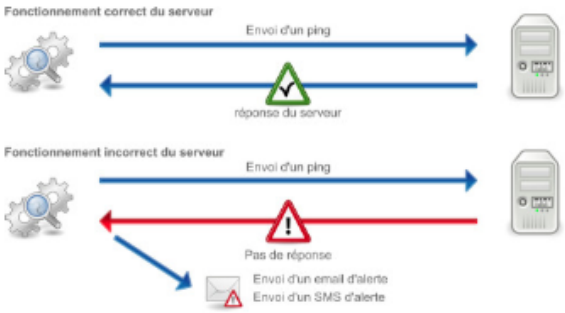
Les outils de monitoring

OVH fournit plusieurs outils pour vous permettre de suivre l'état de votre machine et de déclencher automatiquement l'intervention d'un technicien dans le datacentre.

Nouveau !

Recevez des notifications sur votre iPhone
Avec l'application Momi, les messages d'alerte de monitoring apparaissent sous forme de notifications. Téléchargez l'application dès maintenant

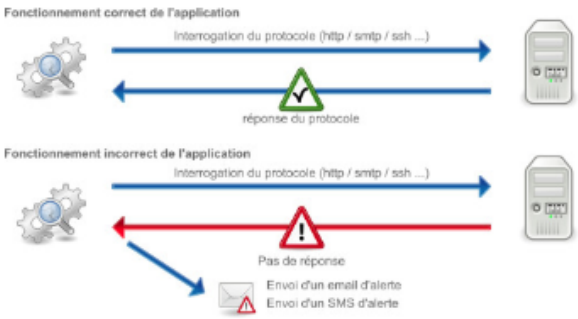
Vérification de l'état du serveur



Tous les serveurs de nos clients ainsi que l'ensemble du réseau sont surveillés 24h/24 et 7j/7 par les équipes techniques d'OVH.
OVH intervient dès le déclenchement d'une alerte (non réponse aux pings) afin de limiter au maximum le temps d'indisponibilité des serveurs et du réseau.
En cas de soucis technique, nous rebootons votre machine et nous vous faisons parvenir un diagnostic de l'incident afin que vous puissiez prendre les mesures nécessaires pour éviter toute panne du même type à l'avenir. Si besoin est, un technicien intervient sur votre machine et remplace le matériel défaillant. En cas de défaillance matérielle, nous remplaçons les éléments incriminés. Si le problème est d'origine logicielle, nous vous faisons parvenir un diagnostic afin que vous puissiez prendre les mesures correctives appropriées.

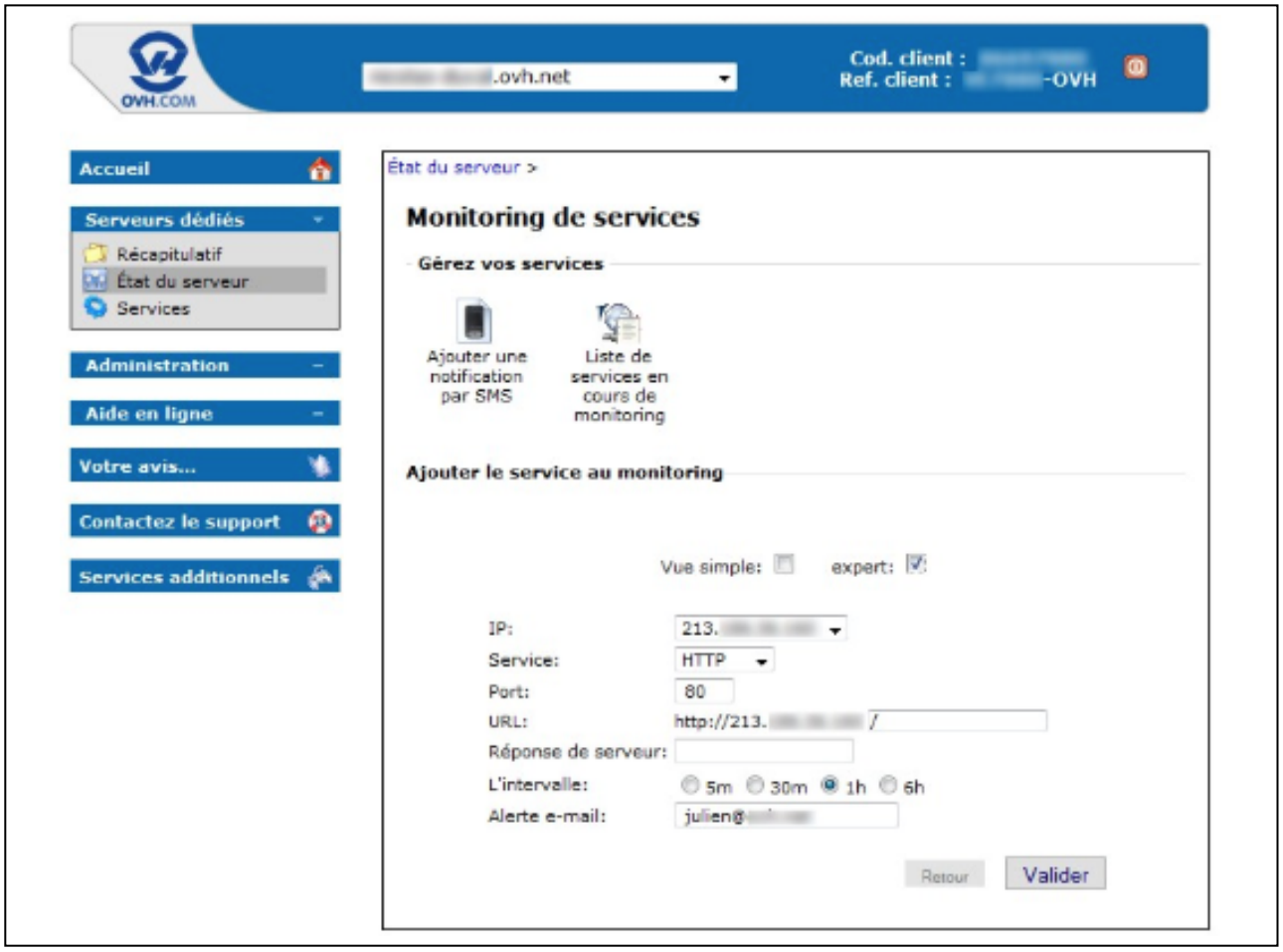
Restez informé du statut de votre serveur

Vérification du bon fonctionnement des applications de votre serveur



Vous pouvez spécifier les services à surveiller (http, FTP, etc...) et même les protocoles moins connus grâce à une sonde TCP.

Restez informé de l'état des protocoles de votre serveur



7.2 - Supervision des conteneurs Docker

Accéder au terminal du serveur hôte, et exécuter les commandes suivantes :

```
$ docker stats <OPTIONS> <CONTAINER…>
```

Cela affichera différents indicateurs relatifs aux conteneurs démarrés sur le serveur :

- ID du conteneur
- Nom du conteneur
- % d'utilisation du CPU
- Utilisation de RAM - Total disponible - % d'utilisation
- NET – I/O

- BLOCK – I/O
- PIDS

7.3 - Supervision du Backend

Springboot Actuator est implémenté. Il fournira les informations ci-dessous :

The metrics endpoint publishes information about OS, JVM as well as application level metrics. Once enabled, we get information such as memory, heap, processors, threads, classes loaded, classes unloaded, thread pools along with some HTTP metrics as well.

Here's what the output of this endpoint looks like out of the box:


```
{
    "mem" : 193024,
    "mem.free" : 87693,
    "processors" : 4,
    "instance.uptime" : 305027,
    "uptime" : 307077,
    "systemload.average" : 0.11,
    "heap.committed" : 193024,
    "heap.init" : 124928,
    "heap.used" : 105330,
    "heap" : 1764352,
    "threads.peak" : 22,
    "threads.daemon" : 19,
    "threads" : 22,
    "classes" : 5819,
    "classes.loaded" : 5819,
    "classes.unloaded" : 0,
    "gc.ps_scavenge.count" : 7,
    "gc.ps_scavenge.time" : 54,
    "gc.ps_marksweep.count" : 1,
    "gc.ps_marksweep.time" : 44,
    "httpsessions.max" : -1,
    "httpsessions.active" : 0,
    "counter.status.200.root" : 1,
    "gauge.response.root" : 37.0
}
```


Pour plus d'information, une documentation détaillée est disponible : https://docs.spring.io/spring-boot/docs/2.1.8.RELEASE/actuator-api/html/.

8 - PROCEDURE DE SAUVEGARDE ET RESTAURATION

8.1 - Mise en place du backup automatique périodique

Accéder au terminal du serveur hôte, et exécuter les commandes suivantes :

```
$ docker exec -it cont-ocpizza-db /bin/bash
```

Cela donne accès au terminal du conteneur Docker « cont-ocpizza-db ».

Exécuter la commande :

```
$ crontab -e
```

Le fichier de CRON de l'OS est ouvert dans VIM.

Entrer en mode édition avec la touche <i>.

Ajouter la chaine de caractères suivante à la suite des commentaires :

```
0 23 * * * /opt/backup-db.sh
```

Quitter le mode édition avec la touche <Echap> puis quitter en sauvegardant avec la commande < :wq > puis la touche <Entrée>.

Le script shell suivant sera exécuté tous les jours à 23h :

```
#!/bin/bash

HOST="localhost"
DB="db_ocpizza"
USER="admin_ocp"
PGPASSWORD="admin"
DATE=`date +%Y%m%d`
FILENAME="/opt/db_backup/${DB}_${DATE}.tar"
#FILENAME="/opt/db_backup/${DB}_${DATE}.sql"

PGPASSWORD=${PGPASSWORD} pg_dump -U ${USER} -W -F t ${DB} > ${FILENAME}
#PGPASSWORD=${PGPASSWORD} pg_dump -Fc -U ${USER} -h ${HOST} ${DB} > ${FILENAME}
```

Accéder au terminal du serveur hôte, et exécuter les commandes suivantes :

```
$ apt-get update && apt-get install vim -y && export EDITOR=/usr/bin/vim

$ mkdir /opt/ocpizza && mkdir opt/ocpizza/db_backup

$ crontab -e
```

Le fichier de CRON de l'OS est ouvert dans VIM.

Entrer en mode édition avec la touche <i>.

Ajouter la chaine de caractères suivante à la suite des commentaires :

```
10 23 * * * /opt/extract-backup.sh
```

Quitter le mode édition avec la touche <Echap> puis quitter en sauvegardant avec la commande < :wq > puis la touche <Entrée>.

Le script shell suivant sera exécuté tous les jours à 23h10 :

```
#!/bin/bash

DB="db_ocpizza"
DATE=`date +%Y%m%d`
CONT_PATH="/opt/db_backup/${DB}_${DATE}.tar"
CONT_NAME="cont-ocpizza-db"
DESTINATION="/opt/ocpizza/db_ocpizza/${DB}_${DATE}.tar"
# -E UTF-8
docker cp ${CONT_NAME}:${CONT_PATH} ${DESTINATION}
```

A son exécution, le fichier de backup de la base de données est copié depuis le conteneur Docker <cont-ocpizza-db> vers le serveur hôte, à l'emplacement attribué à la variable < DESTINATION >.

8.2 - Restauration à partir d'une sauvegarde

Accéder au terminal du serveur hôte, et exécuter les commandes suivantes :

```
$ docker exec -it cont-ocpizza-db pg_restore -d db_contacts /opt/db_backup/<nom_du_fichier.tar> -c -U admin_ocp
```

Cela restaurera le contenu de la sauvegarde de l'archive TAR désignée dans la base de données du conteneur cible.

9 - GLOSSAIRE

[1] **Composant** : définit un élément constitutif d'une solution applicative.

[2] **Docker** : plateforme de conteneurisation utilisée pour déployer les composants de l'application sur le serveur hôte.

[3] **Serveur hôte** : définit l'entité physique qui hébergera les composants de l'application ocpizza.

[4] **Conteneur docker** : un conteneur enveloppe un composant dans une boîte virtuelle et isolée avec tout ce dont il a besoin pour s'exécuter.

[5] **Image docker** : définit une collection ordonnée de changements au niveau d'un système de fichier racine et des paramètres correspondant pour une utilisation à l'intérieur d'un conteneur au runtime.

[6] **Backend :** dans une solution applicative, désigne l'ensemble des composants logiciels responsables, directement ou non, des traitements effectués en arrière-plan par celle-ci.

[7] **Frontend :** dans une solution applicative, désigne l'ensemble des composants logiciels qui seront reçues par le client (souvent un navigateur internet tel que Chrome ou Firefox).

[8] **Artefact :** représente une partie physique de l'information, utilisée ou produite par le processus de développement d'un logiciel (ex : modèle, fichier, table, etc.).