Créer une application Scalable

Avec Spring-Boot / Hazelcast / Docker et Kubernetes

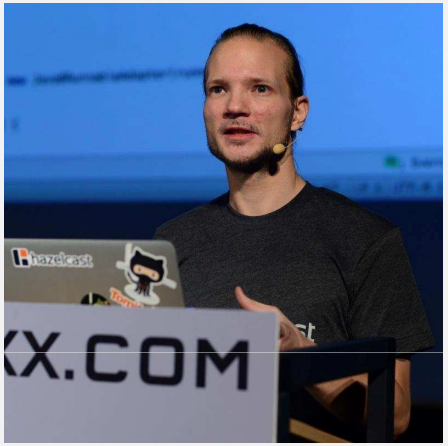
Corentin Azelart

@CooRentin_

Merci !

Ray Tsang

Google Cloud Developer



Christoph Engelbert

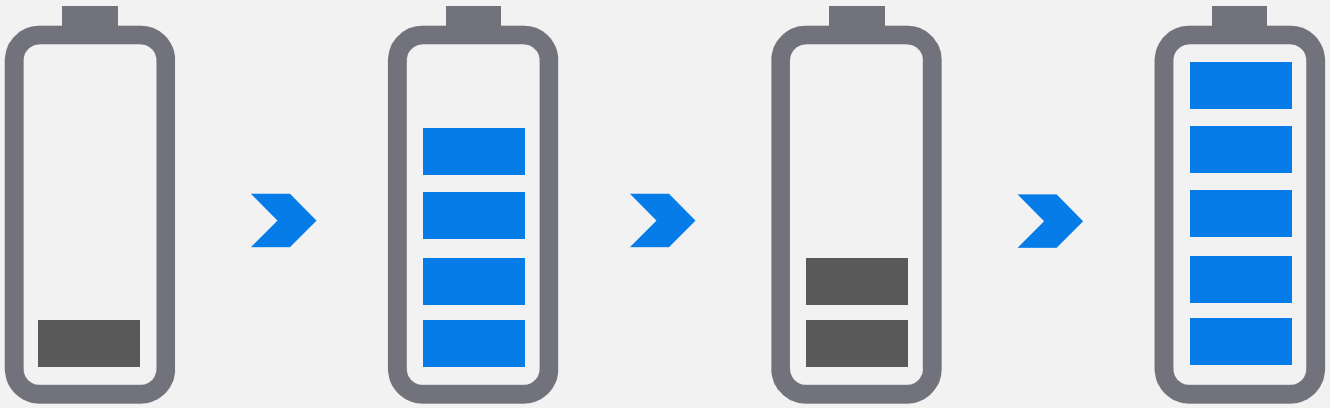
Hazelcast Evengelist / Apache Commiter

Plan

- Scalability ?
- Une application et une architecture
- Une demonstration

Scalability

« Désigne la capacité d'un produit à s'adapter à un changement d'ordre de grandeur de la demande, en particulier sa capacité à maintenir ses fonctionnalités et ses performances en cas de forte demande »



100
Users

500.000
Users

100.000
Users

700.000
Users

Scalability

Administrative

Donner l’impression à l’utilisateur de ne partager qu’un seul système.

Functional

Capacité à ajouter des fonctionnalités en minimum d’effort.

Load

La simplicité d’ajouter, supprimer, modifier un composant pour s’accomoder à la charge

Verticale vs Horizontale

- Augmenter la puissance
- Augmenter le nombre d’instances

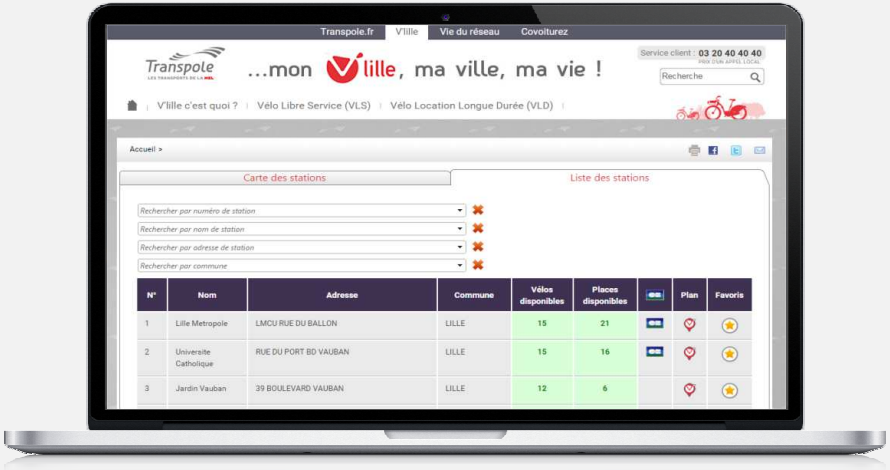
Par l’exemple

Le V’Lille comment ça marche ?

Bonne Question

How it work ?

La web-app V’Lille

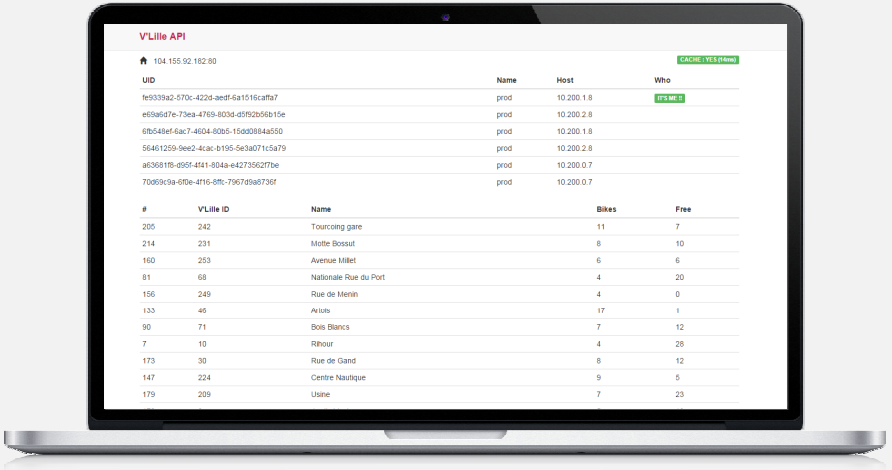


Donner de l’information sur le réseau

Cette page donne des informations sur l’état des stations du V’Lille

Notre application

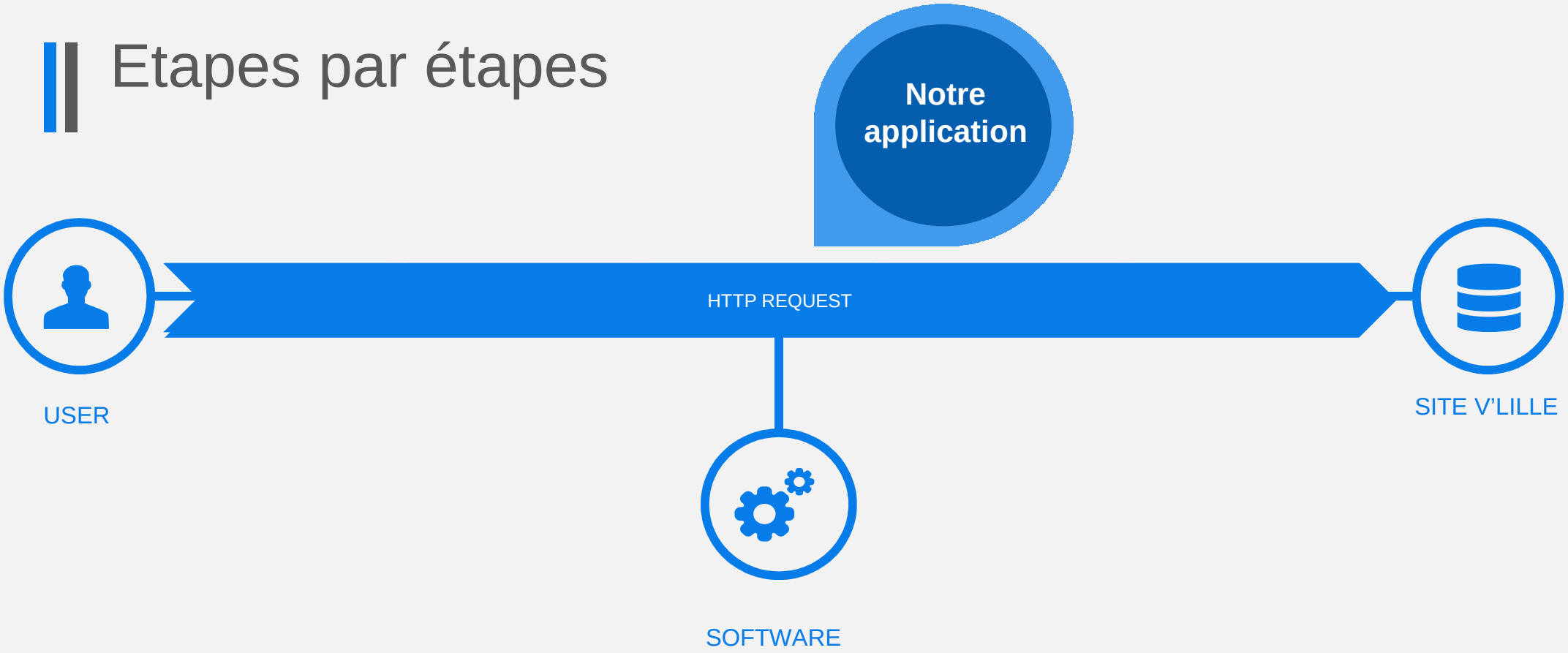
Réaliser la même chose mais à travers notre IHM



Au travers d’une application scalable

Réaliser le même service quelque soit l’état de l’API V’Lille et celà à un grand nombre d’utilisateurs

Etapes par étapes



En avant ! Rien que l’essentiel !

Pas de serveur d’application

Pas d’hebergement traditionnel

Je ne veux payer que ce que je consomme

Pas de WAR (POC)

“Make the JAR ! Not the WAR”

Posons les briques

Spring-Boot

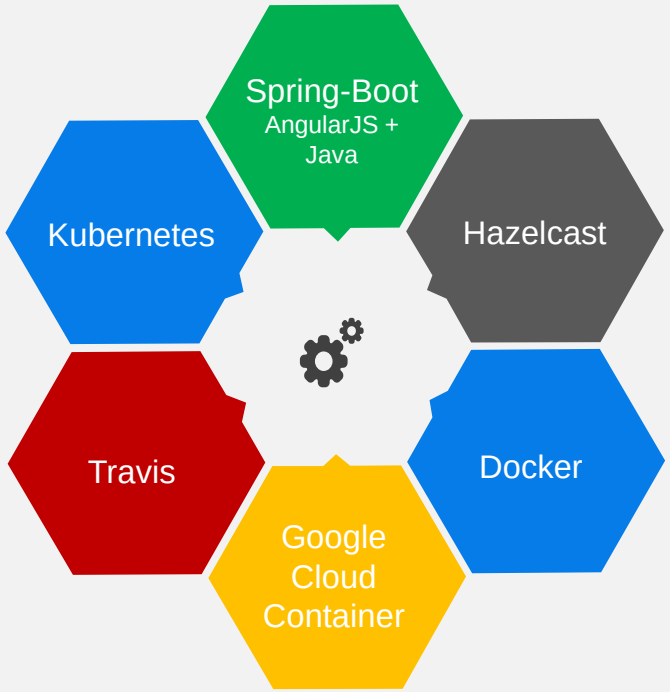
Container léger proposant une approche production ready.

Kubernetes

Orchestrateur pour le deploiement de containers

Travis-CI

Service d'intégration continue en ligne.



Hazelcast

Un cache répliqué et partagé entre toutes nos instances

Docker

Empaqueteur pour notre application

Google Cloud Container

Hebergement et création dynamique d'images

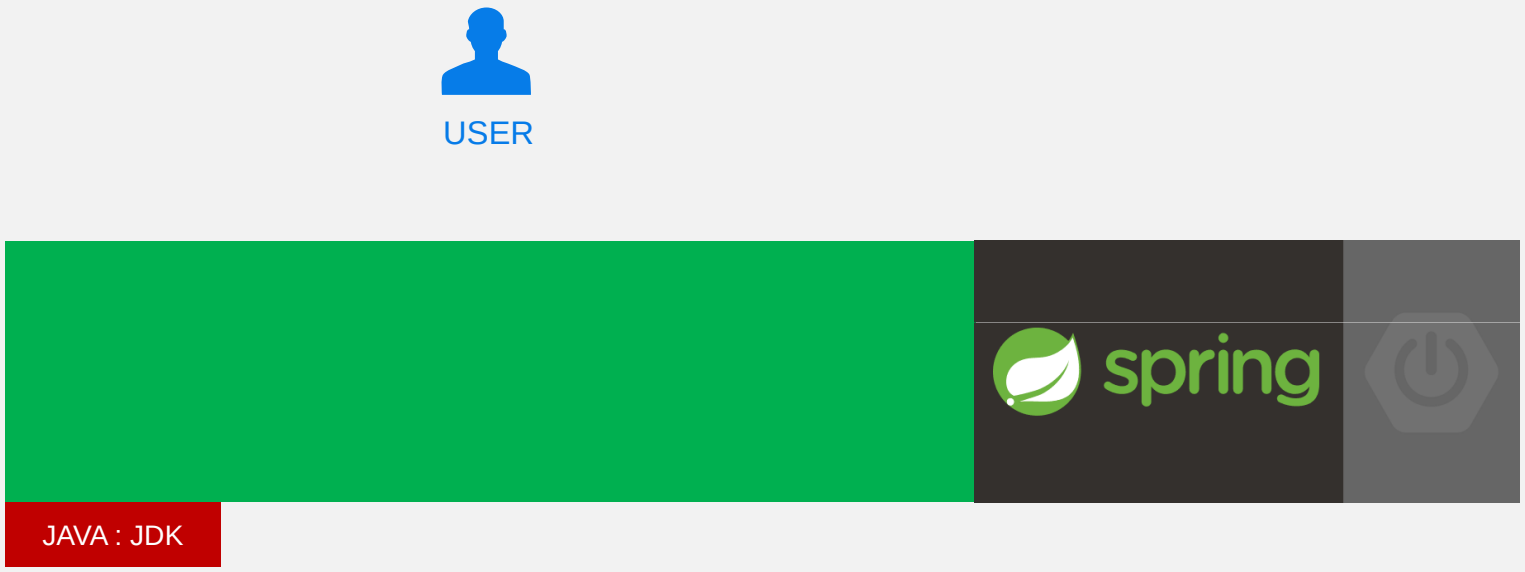
Posons les briques

USER

JAVA : JDK

V'LILLE API

Posons les briques

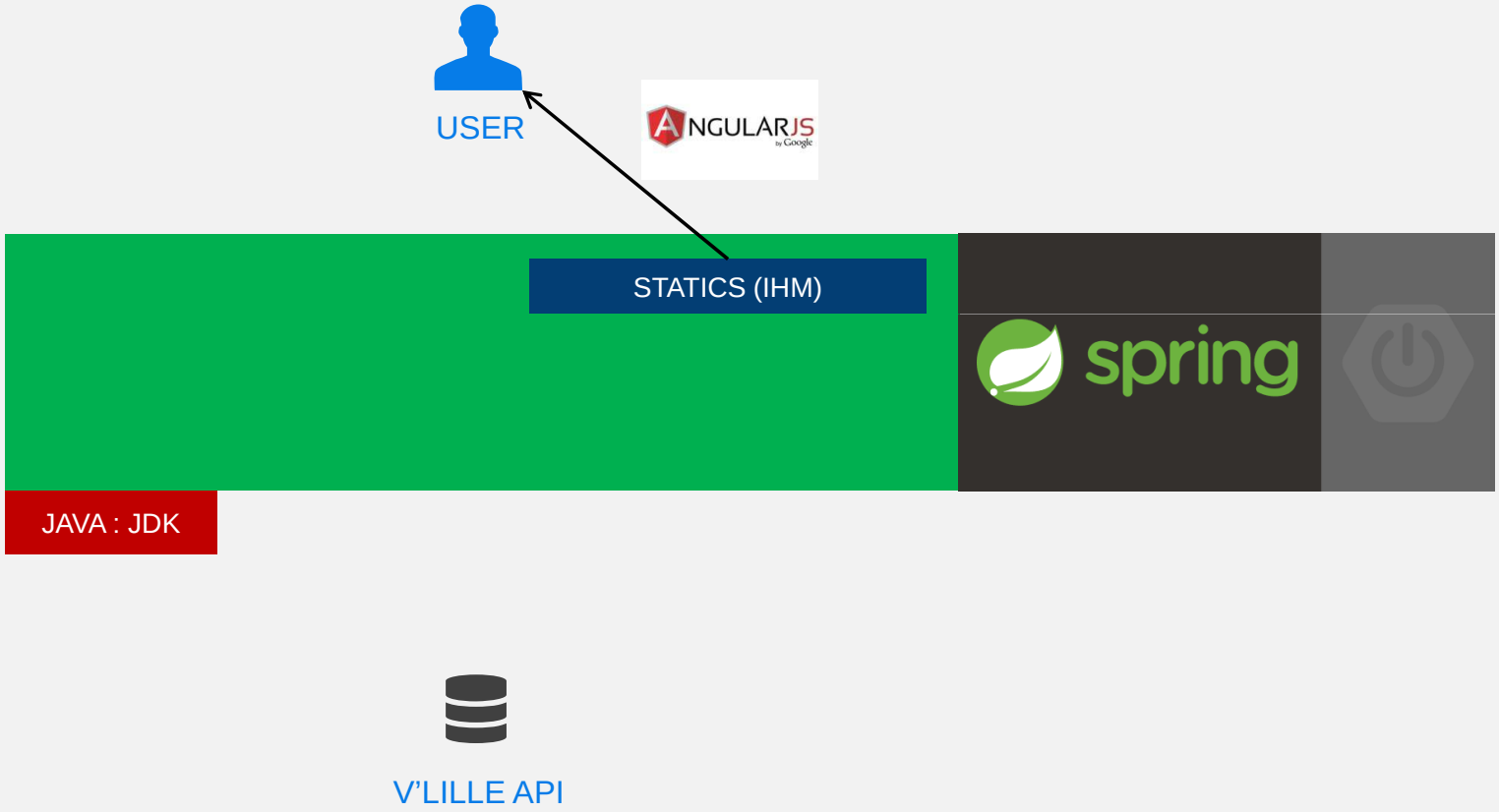


Pour les connaisseurs

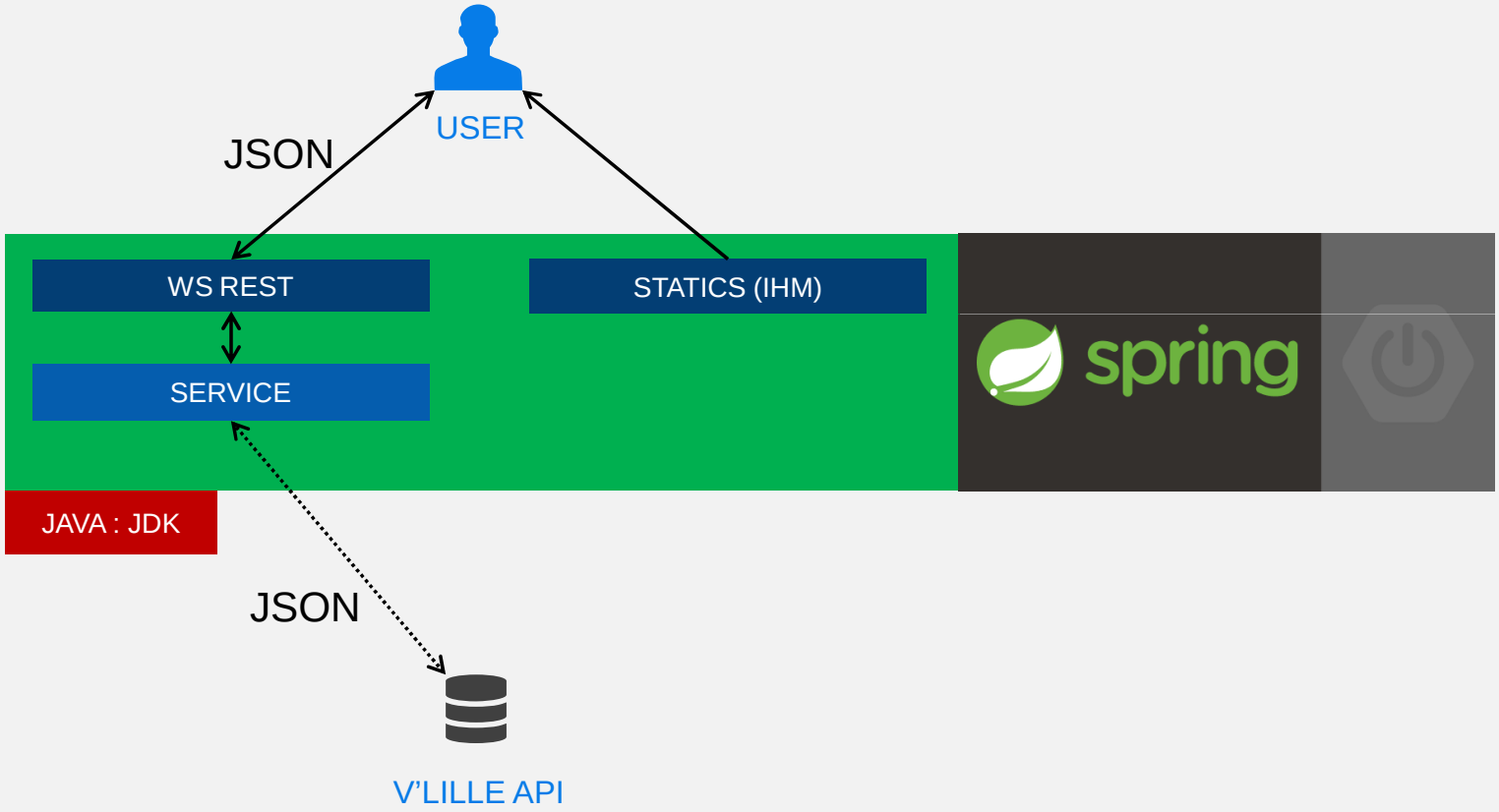
pom.xml

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>1.3.0.RELEASE</version>
  <relativePath/> <!-- lookup parent from repository -->
</parent>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-ws</artifactId>
  </dependency>
</dependencies>

 <build>
  <plugins>
    <plugin>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-maven-plugin</artifactId>
    </plugin>
  </plugins>
</build>
```

Posons les briques



Posons les briques

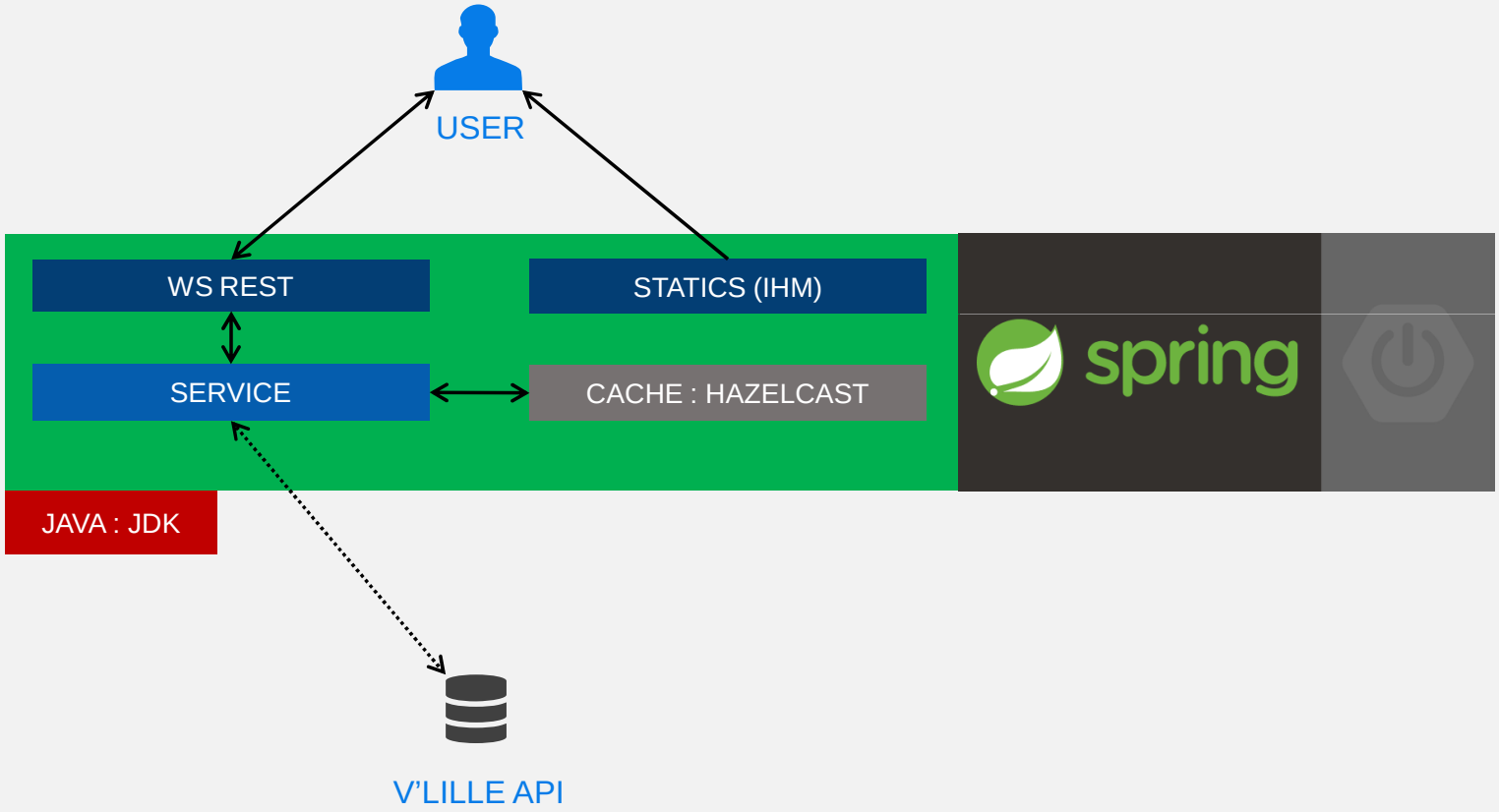


Pour les connaisseurs

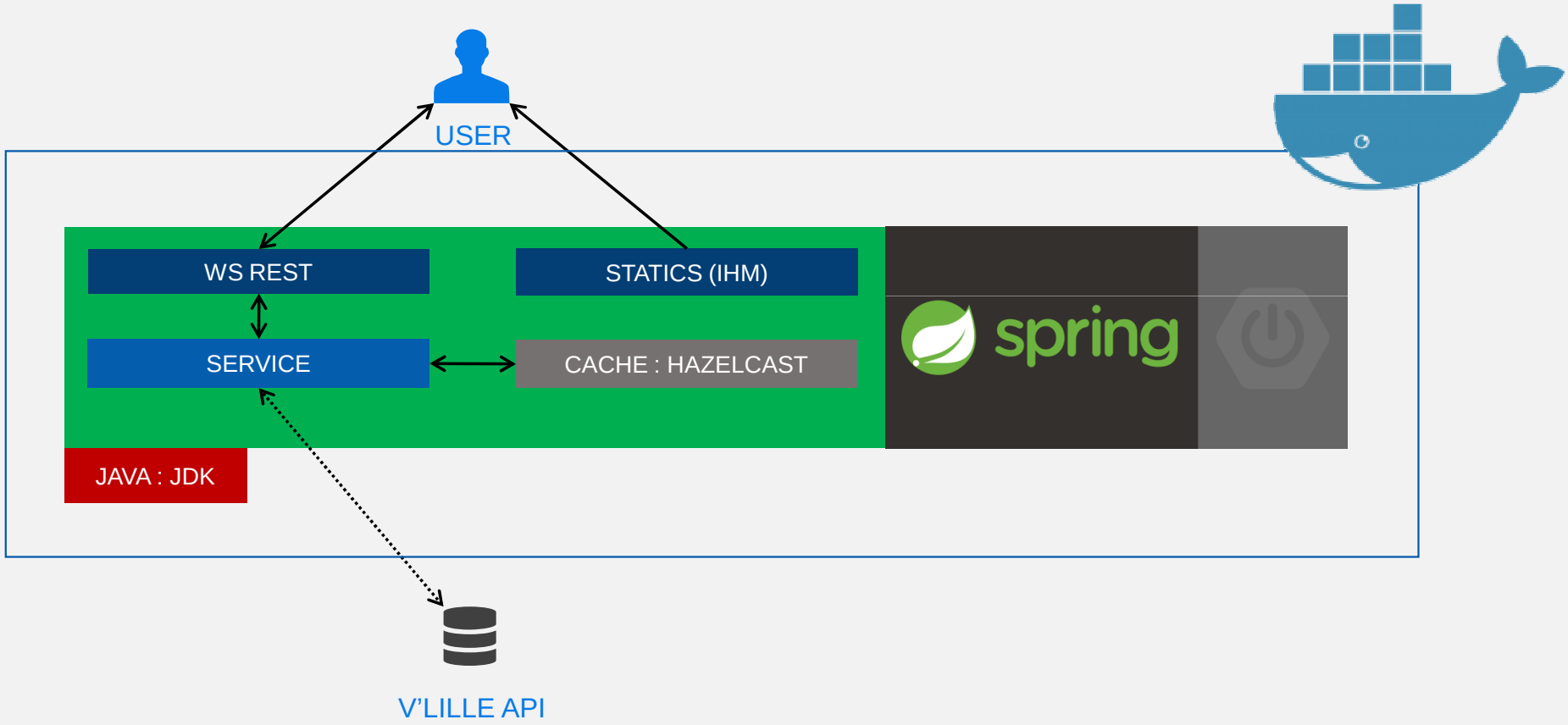
VLilleWS.java

```java
@RestController
@RequestMapping("/api/stations")
public class VLilleWS {

    /**
     * VLille Service.
     */
    @Autowired
    private VLilleService vLilleService;

    /**
     * Return a list of stations.
     * @return a list of stations.
     */
    @RequestMapping(method = RequestMethod.GET)
    public StationResponseDTO findAll() throws SynchronisationException {
        final Long startTime = System.currentTimeMillis();
        final StationResponseDTO stationResponseDTO = vLilleService.findAll();
        stationResponseDTO.setTime(System.currentTimeMillis() - startTime);
        return stationResponseDTO;
    }
}
```

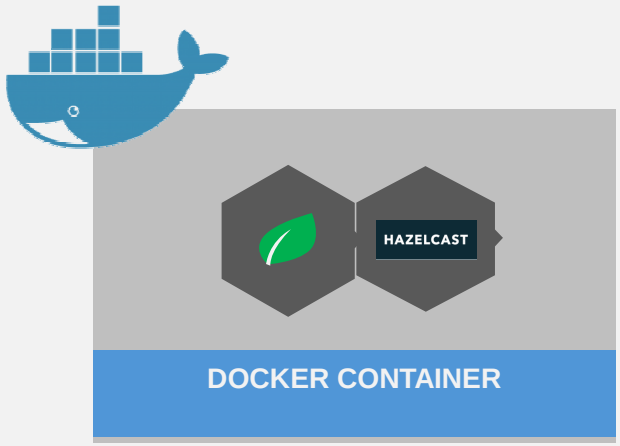
Posons les briques



Posons les briques



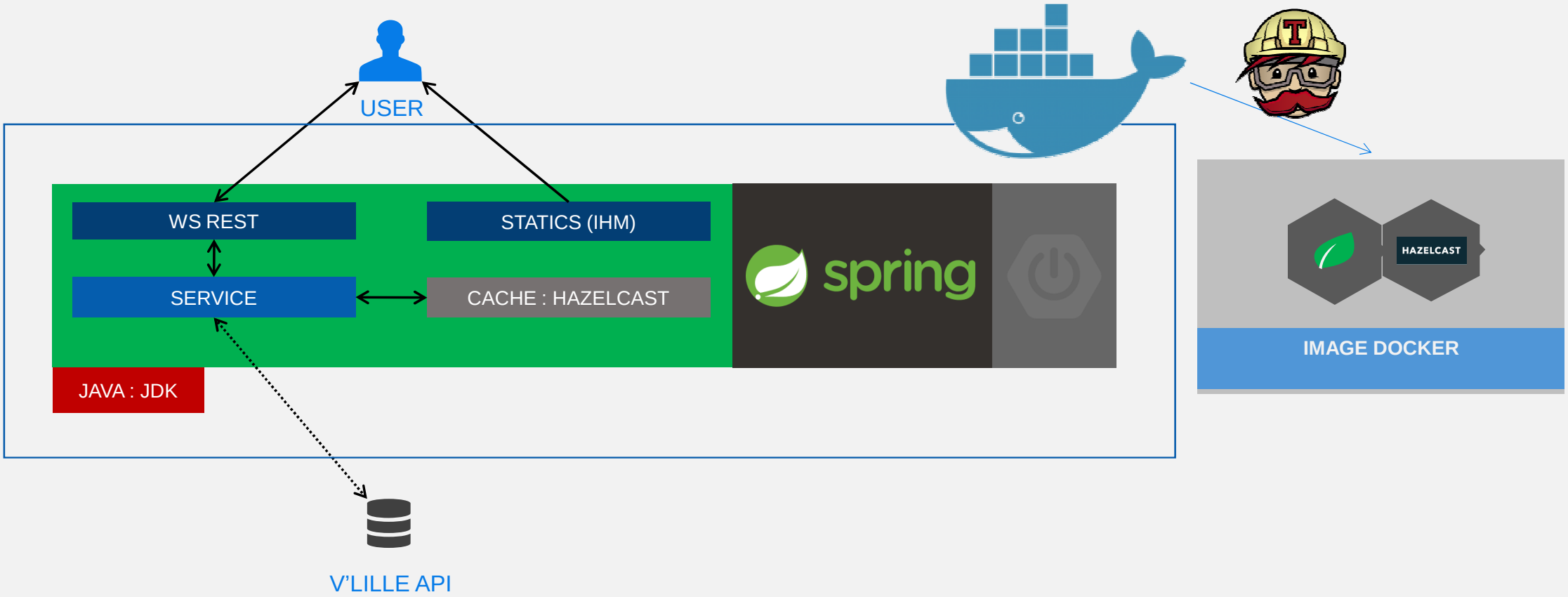
Pour les connaisseurs



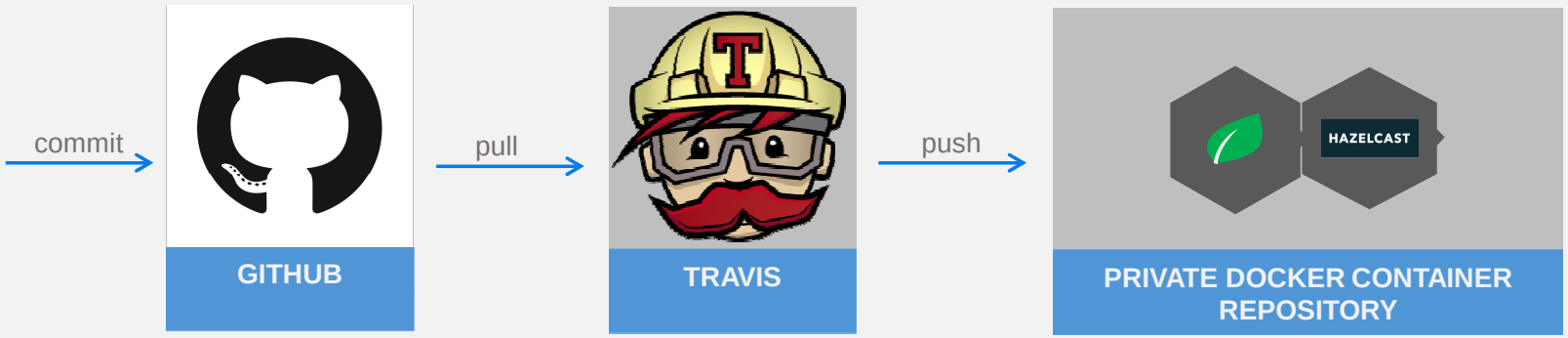
Dockerfile

```
FROM java:8

MAINTAINER Corentin <corentin@azelart.fr>

COPY target/vlille.jar app.jar

EXPOSE 80
EXPOSE 5701

ENTRYPOINT ["java", "-jar", "app.jar", "--server.port=80"]
```

Posons les briques



Créer un container à chaque commit

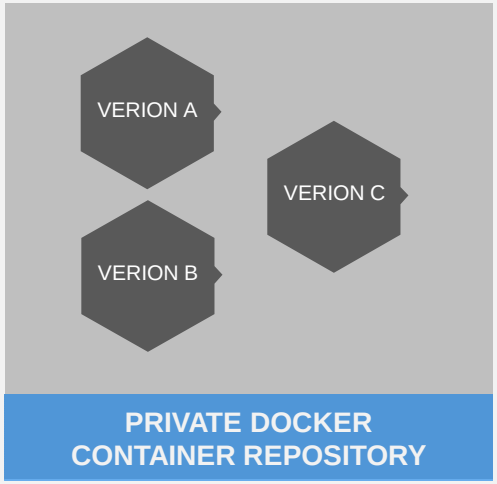


Pour les connaisseurs

.travis.yml

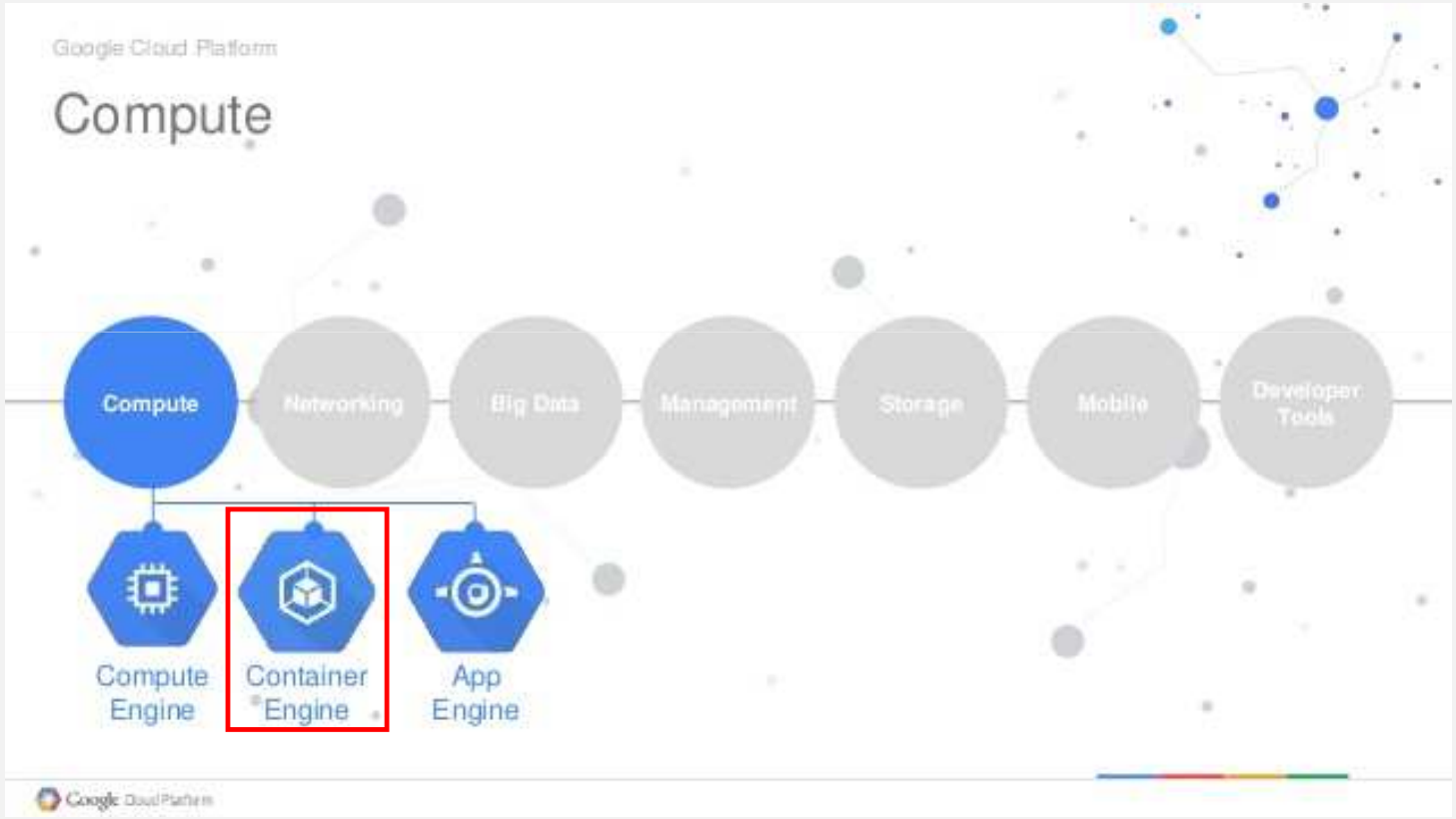
```
sudo: required
language: java
jdk:
services:
- docker
install:
- docker login --email=$DOCKER_HUB_EMAIL --username=$DOCKER_HUB_USERNAME --
password=$DOCKER_HUB_PASSWORD
before_script:
- mvn package spring-boot:repackage
script:
- docker build -t $DOCKER_IMAGE_NAME .
- if [ ! -z "$TRAVIS_TAG" ]; then docker tag $DOCKER_IMAGE_NAME:latest
$DOCKER_IMAGE_NAME:$TRAVIS_TAG; fi && docker push $DOCKER_IMAGE_NAME
env:
  - DOCKER_IMAGE_NAME=corentin59/hazelcast-with-docker-and-kubernetes
```

Aprés quelques commit…



Comment envoyer mes images en production ?

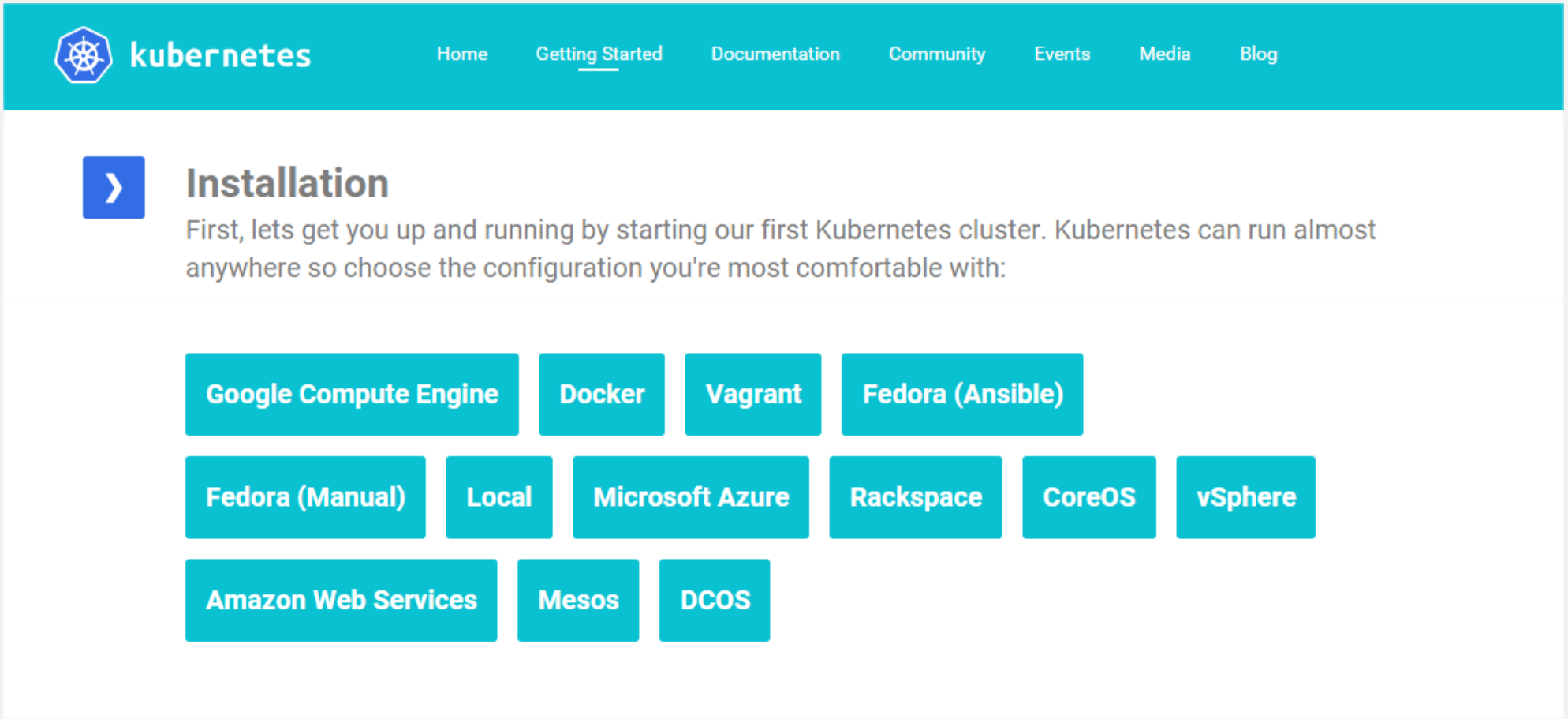
Hebergement :Google Cloud Container



Kubernetes : Orchestrateur

- Orchestrateur
- Automatiser le deploiement sur des milliers de serveurs
- Scaling à froid ou à chaud
- “Compatible” avec plusieurs providers/hebergeurs de containers

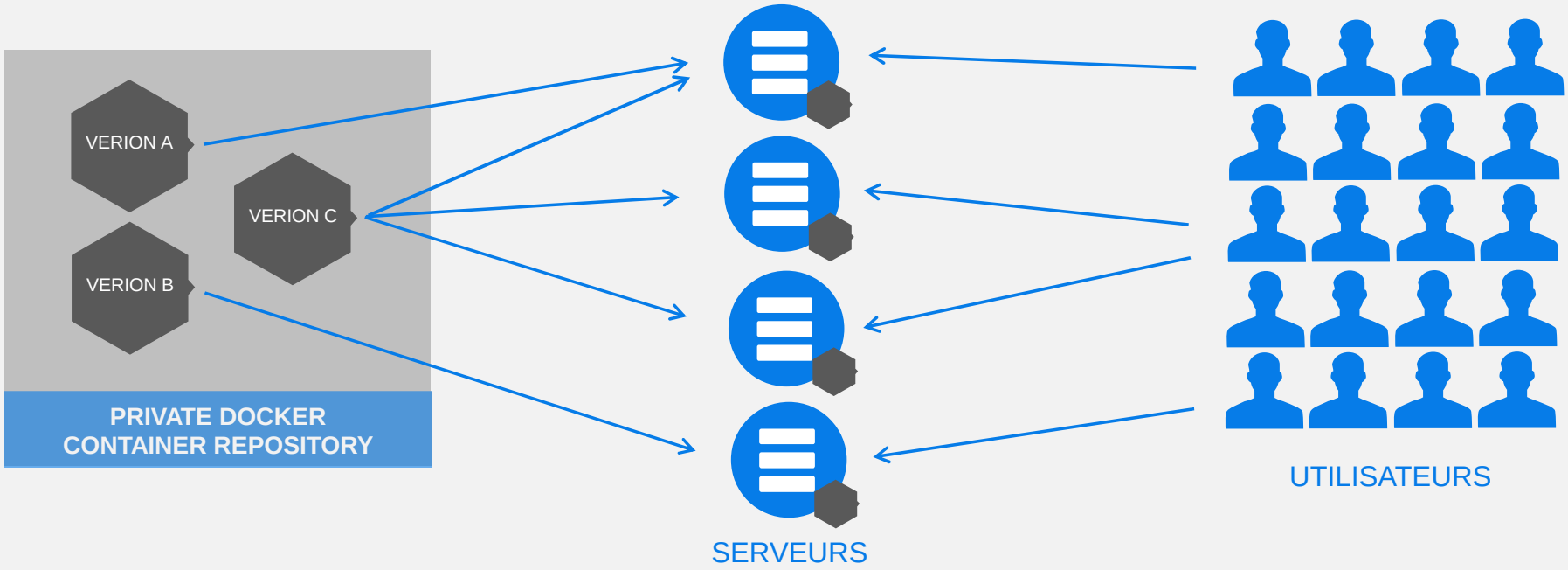
Kubernetes : confort



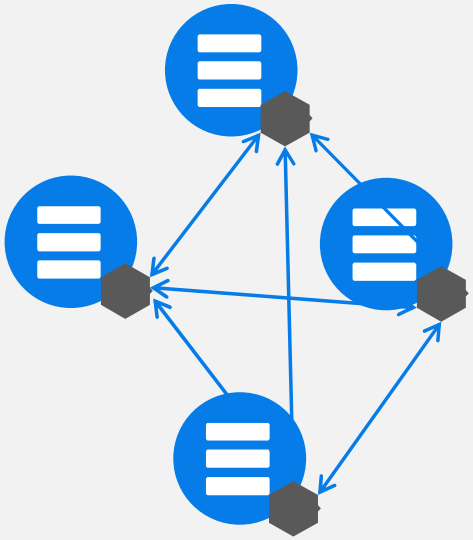
Avantages

- Un commit une version
- A/Z testing
- Tester trés rapidement des (micros) features sur les utilisateurs
- Deploiement continu

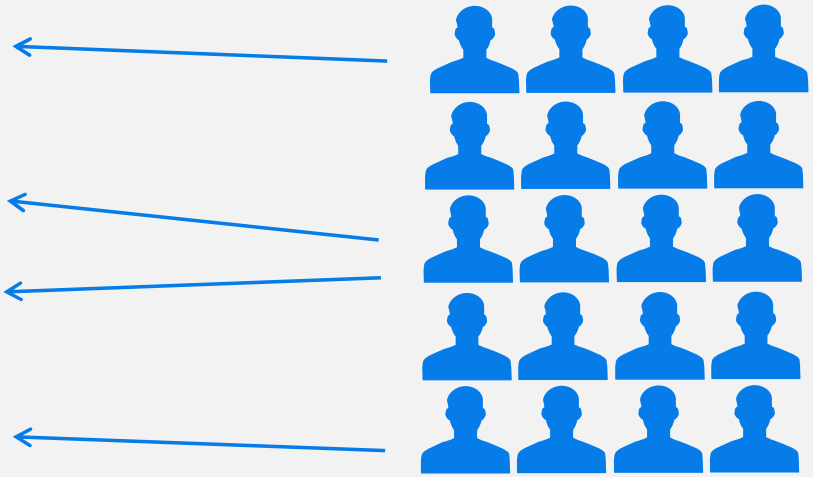
En route vers la prod…



Communication inter-containers

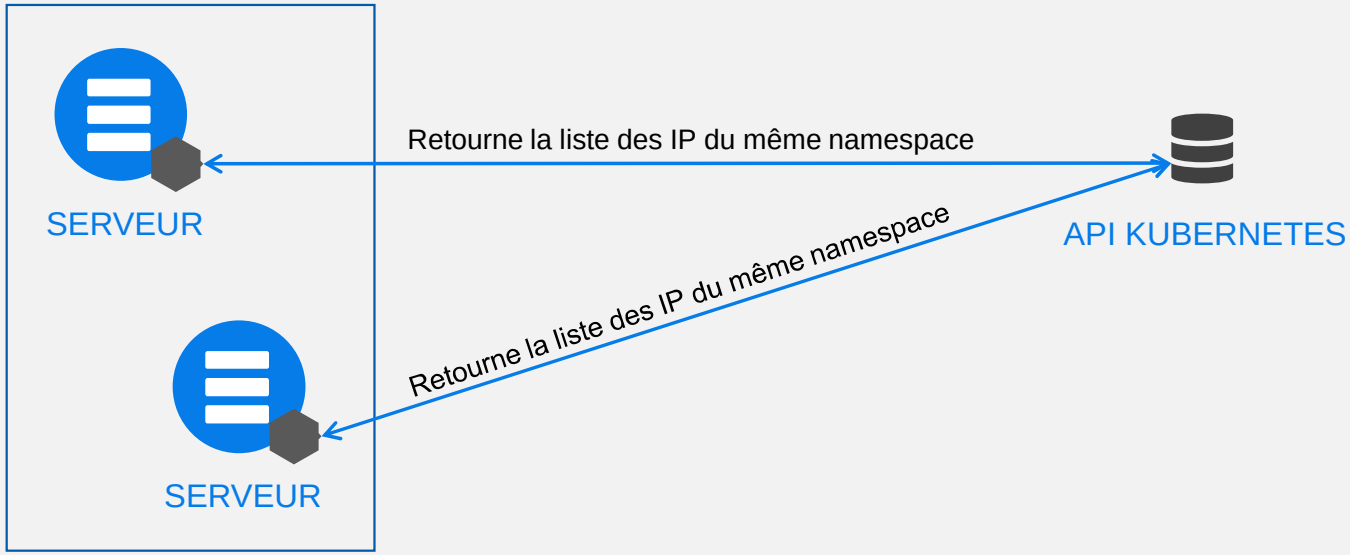


SERVEURS



UTILISATEURS

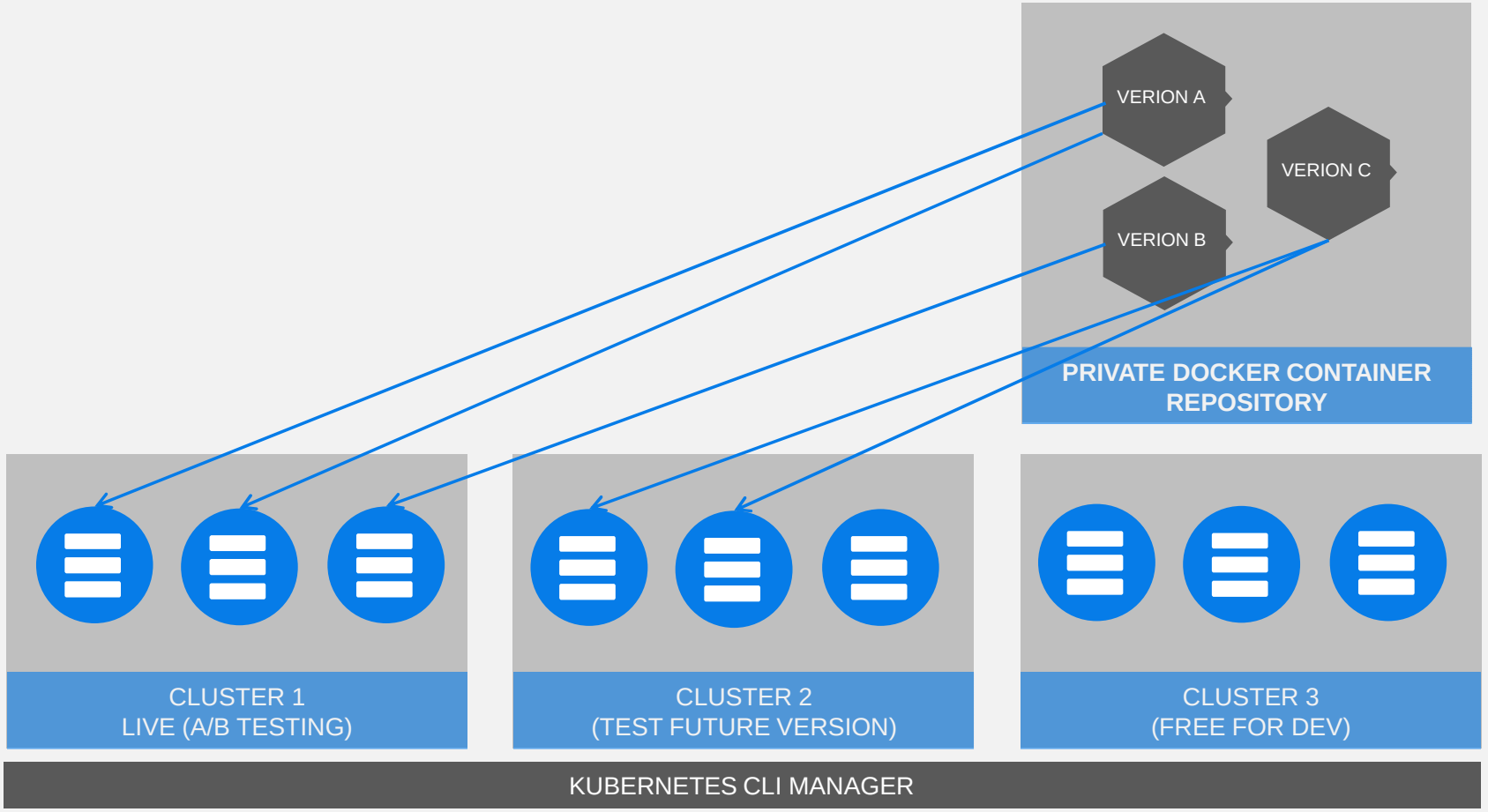
API Kubernetes



/var/run/secrets/kubernetes.io/serviceaccount/token

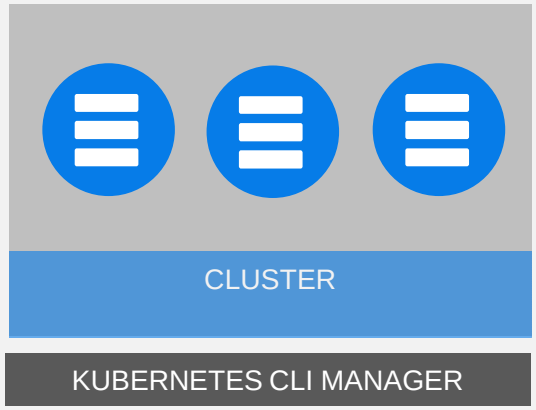
Architecture type avec Kubernetes

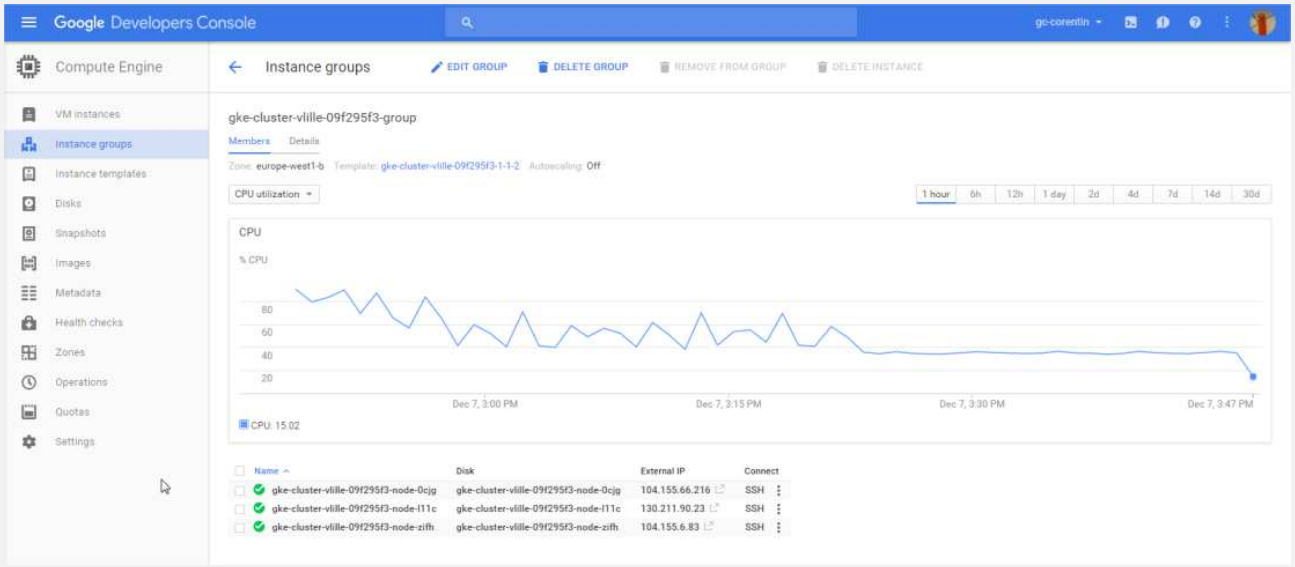
“Platform for automating deployment, scaling, and operations of application containers across clusters of hosts”



Mise en pratique : La console Google

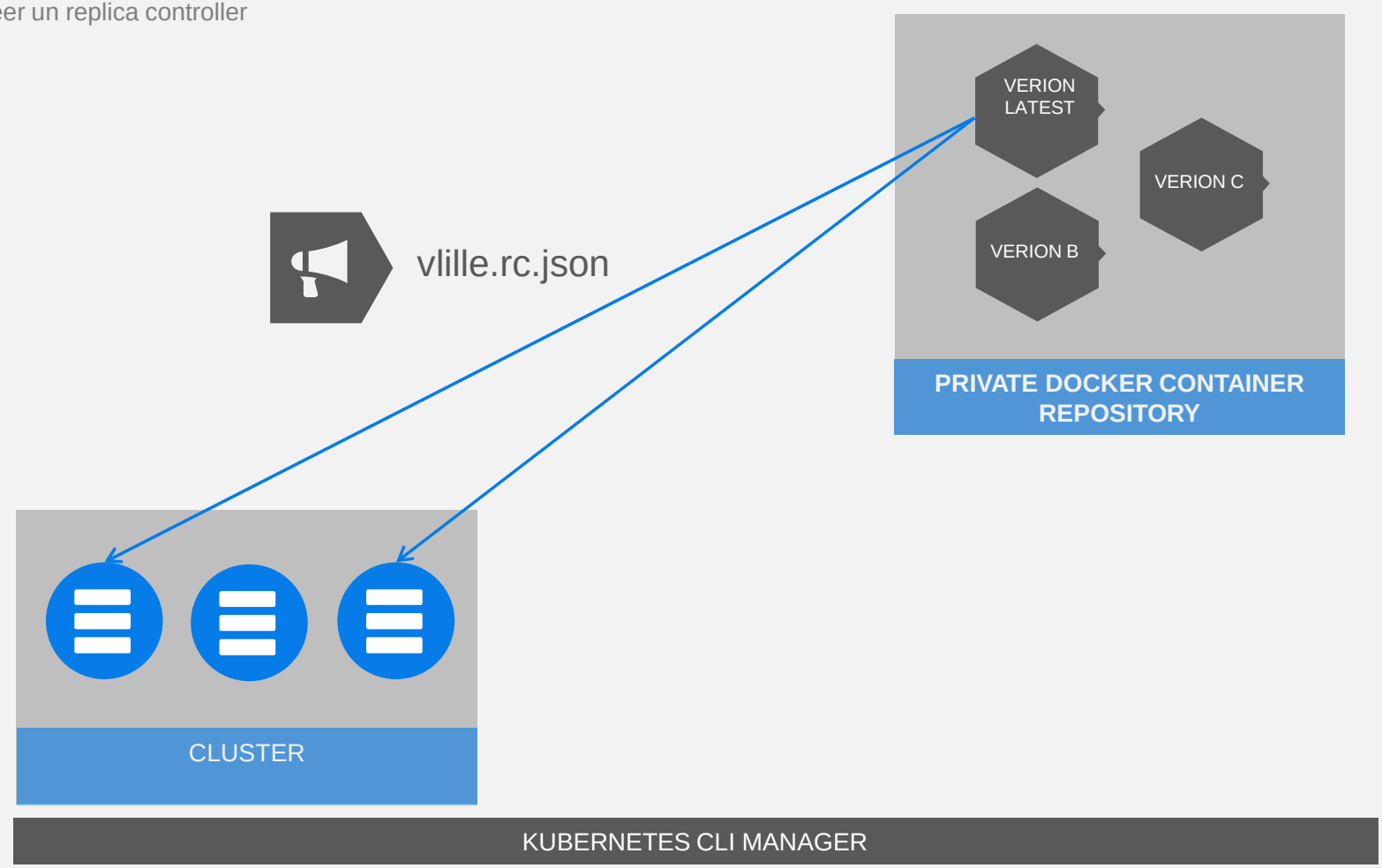
Créer un cluster sur Google Cloud Container





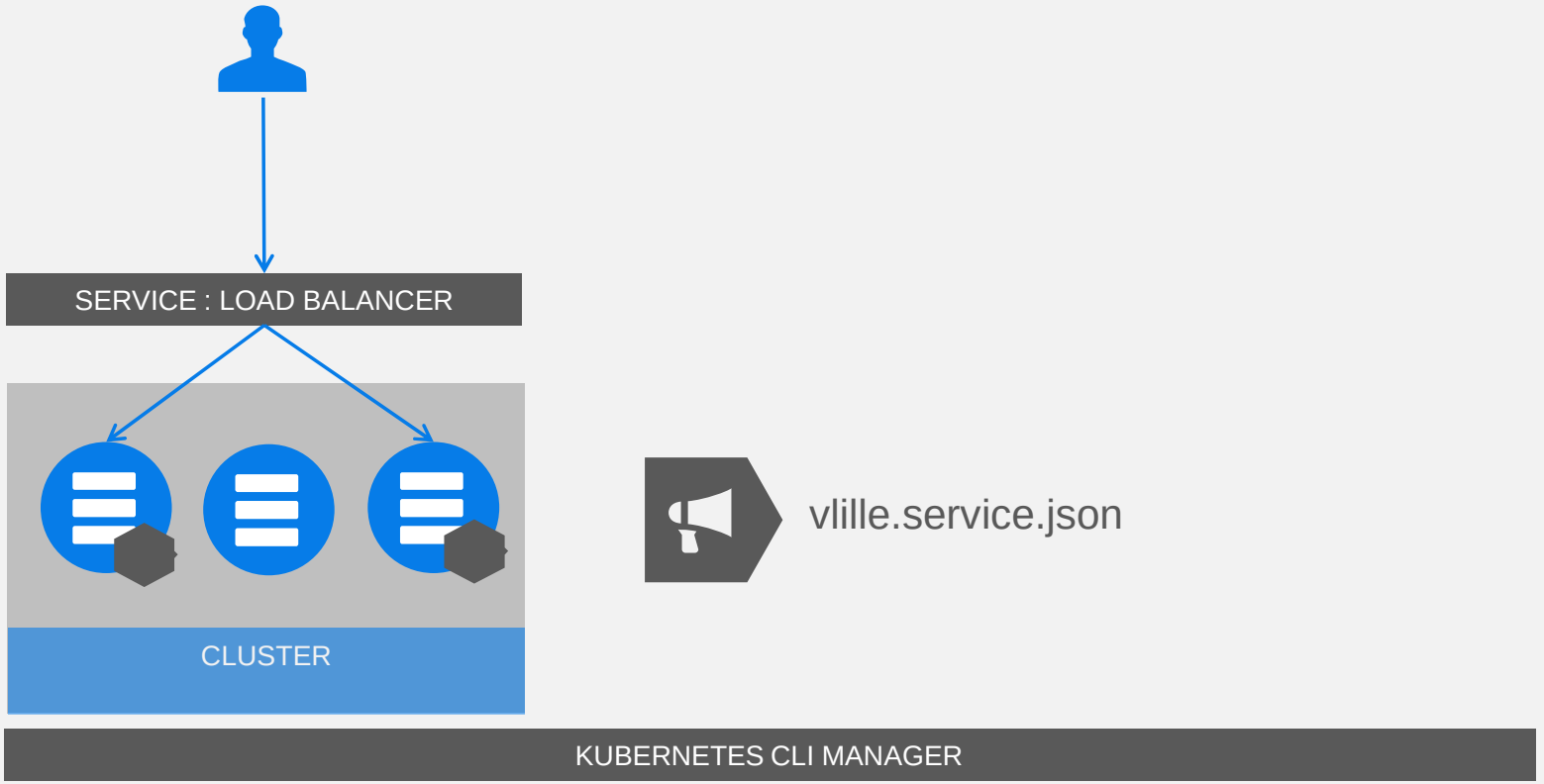
Mise en pratique : Replica Controller

Créer un replica controller



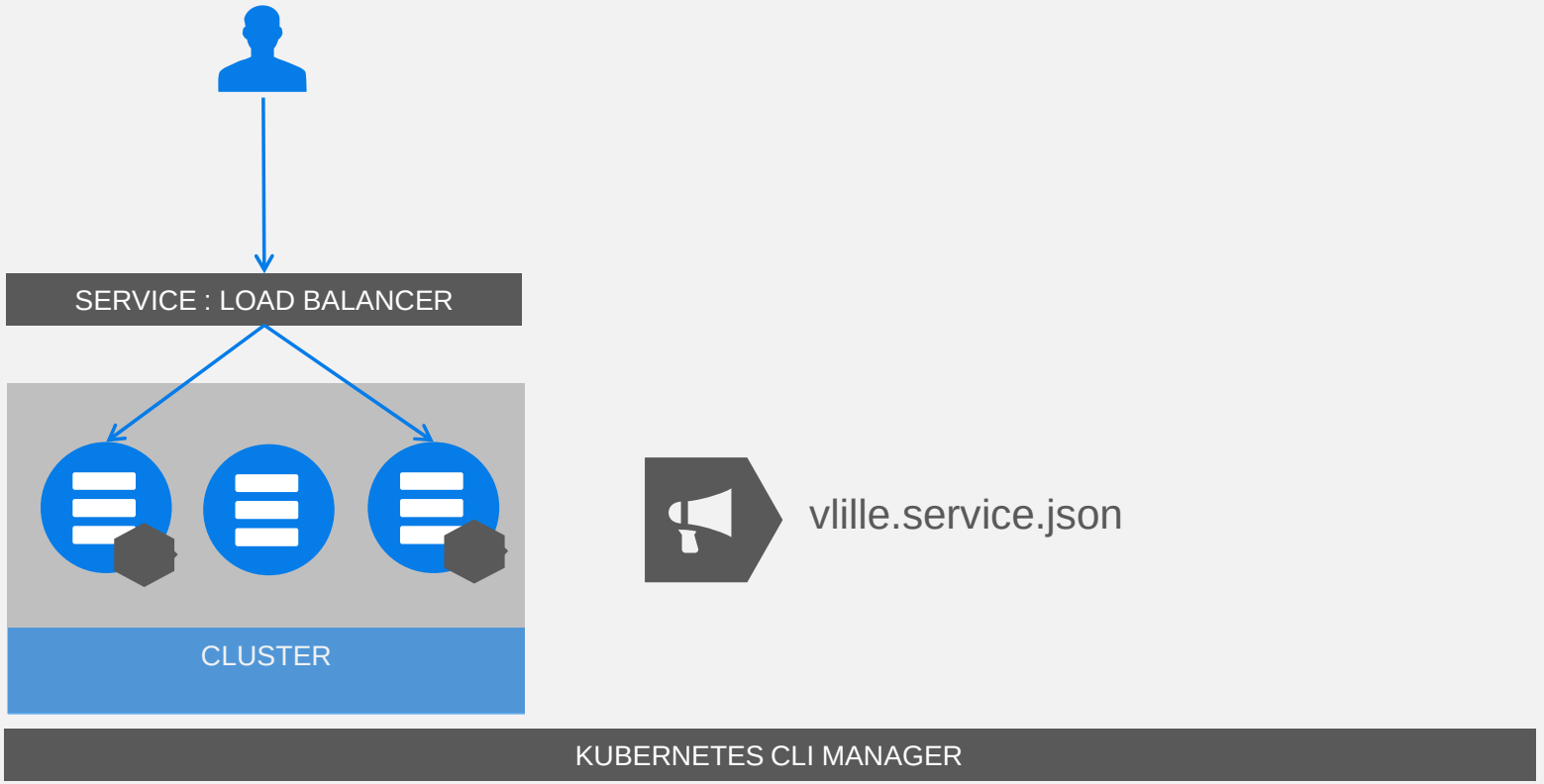
Mise en pratique : Service et Load Balancer

Créer un service : load balancer



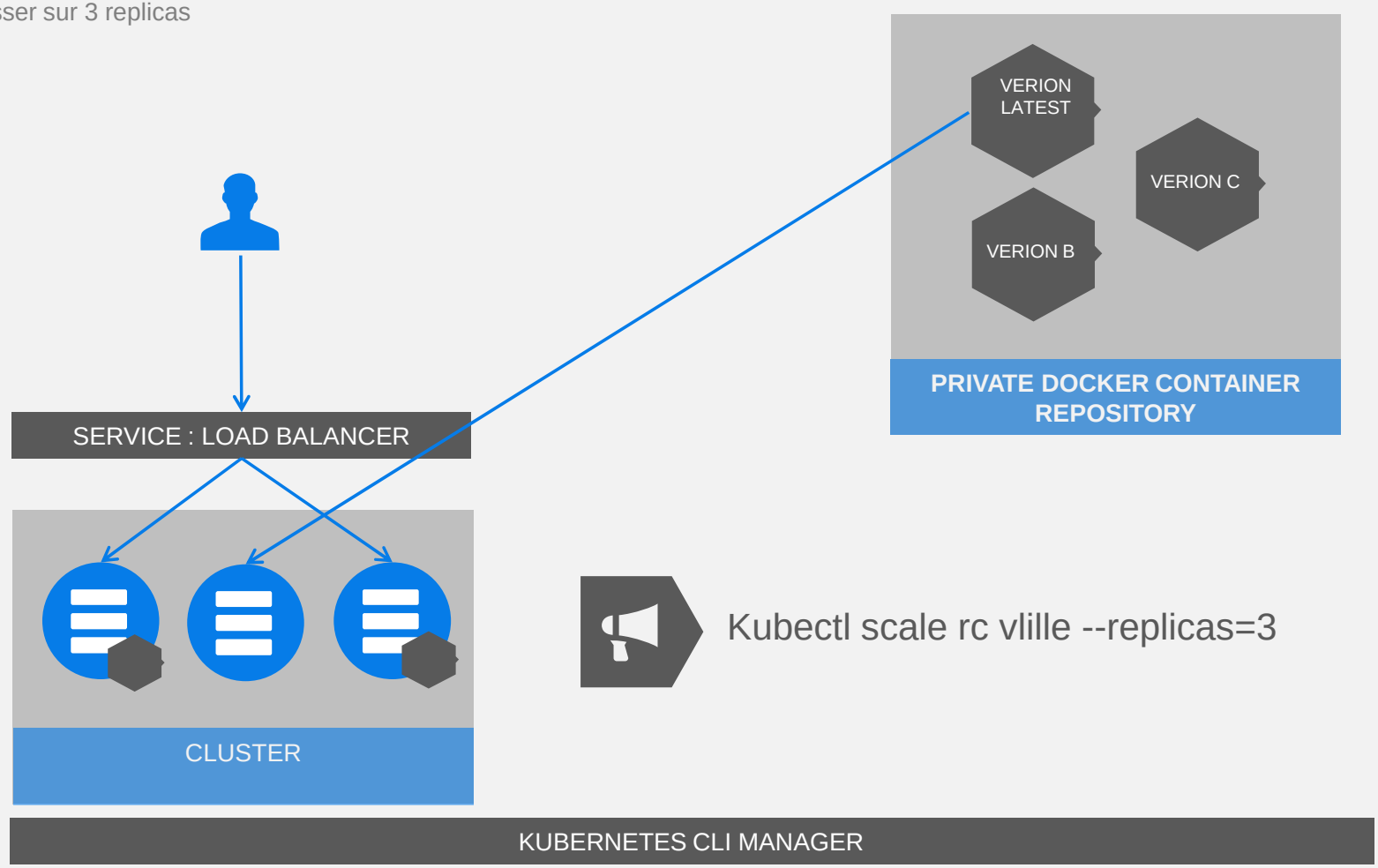
Mise en pratique : Re-Scale

Passer sur 2 replicas



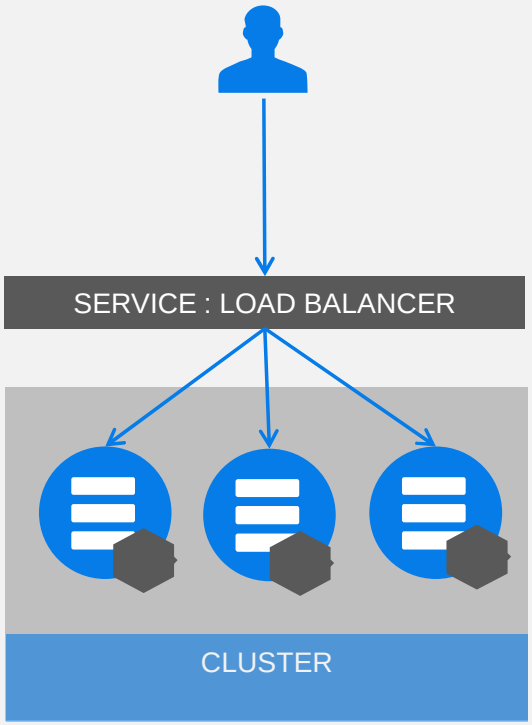
Mise en pratique : Re-Scale

Passer sur 3 replicas



Mise en pratique : Re-Scale

Passer sur 3 replicas

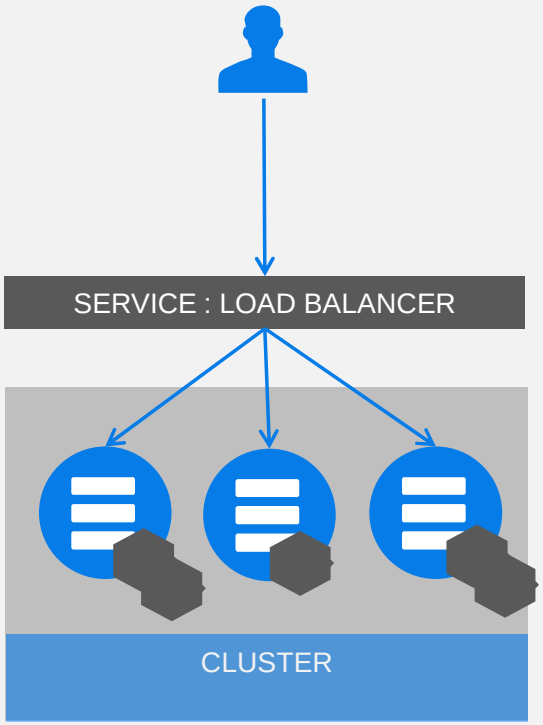


Kubectl scale rc vlille --replicas=3

KUBERNETES CLI MANAGER

Mise en pratique : Re-Scale

Passer sur 5 replicas



Kubectl scale rc vlille --replicas=5

KUBERNETES CLI MANAGER

Questions ?

Thanks for Coming

https://github.com/corentin59/hazelcast-with-docker-and-kubernetes