Rapport Kubernetes :
Déploiement d'une solution Wordpress élastique

Aiman Berrichi, Yann Jaulin, Clément Riehm, Amadou Moctar Thiam

21 mars 2022

Table des matières

Chapitre 1

Remerciements

Avant de commencer le rapport, nous souhaitons grandement remercier notre tuteur monsieur **Mathieu Goulin** pour l'aide qui nous a apporté durant le sujet et les nombreuses pistes qu'il nous a données afin de le réussir.

De plus, nous remercions **les membres du jury** pour leur écoute durant nos soutenances et leur précieux conseil pour nous améliorer.

Enfin, nous remercions tout autant monsieur **Philippe Dosch**, le responsable de la licence, pour la gestion de celle-ci et sans qui nous ne serons pas là aujourd'hui.

Chapitre 2

Introduction

Dans notre société actuelle, la technologie ainsi que l'informatique évolue de manière exponentielle à travers plusieurs procédés et protocoles qui nous ont permis d'arriver à ce que nous sommes aujourd'hui en termes d'informatique.

À partir de cette évolution, nous sommes capables de mettre en place des solutions qui nous permettent d'évoluer à travers plusieurs procédés, mais il existe un procédé unique en son genre : l'Automatisation.

Grâce à l'automatisation, nous sommes capables de déployer plusieurs solutions comme Kubernetes et c'est ce que nous allons vous montrer dans ce rapport.

2.1 Choix du sujet

Notre choix s'est porté sur "Le déploiement d'une solution Wordpress Élastique", on a délibérément choisi ce sujet, car nous avons voulu découvrir une nouvelle technologie, encore inconnu aujourd'hui aux yeux de tous.

2.2 Contexte et Problématique

La définition de l'architecture d'un service applicatif est une étape structurante pour tout projet d'hébergement. En 2022, avec la dématérialisation des ressources (cloud), et les nouvelles technologies, nous concevons des architectures dites élastiques.

L'élasticité, c'est disposer d'une quantité de ressources variables (Compute, Stockage) en fonction du besoin. Cela permet de ne réserver que le minimum de ressource au projet et d'optimiser grandement les coûts d'exploitations.

La containérisation des services est aujourd'hui l'une des solutions pour obtenir cette élasticité et l'orchestrateur Kubernetes, c'est aujourd'hui imposé comme orchestrateur de containers.

2.3 Motivation

Notre motivation s'est manifestée lorsque nous avons voulu apprendre cette nouvelle technologie, qui était étrangère et méconnue lorsque nous avons pris connaissance de Kubernetes.

Au fur et à mesure de notre progression, nous avons découvert plusieurs outils qui nous ont fait progresser dans le thème de l'automatisation comme les volumes, par exemple.

À partir de ces nouveaux éléments pris en considération, nous avons voulu approfondir les connaissances sur Kubernetes et à partir de cette profonde analyse que de nouveaux éléments se sont manifestés et cette profonde analyse a permis de naître notre motivation.

2.4 Objectif

Nous allons devoir, dans ce sujet étudié implémenter une application sous Kubernetes en prenant en compte les principes d'exploitation : plan de continuité, sauvegardes, archivages et surtout élasticité (capacité à monter en charge).

Avec en plus des scripts d'installation permettant de reproduire nos travaux sur d'autres clusters Kubernetes, nous devons valider notre architecture via des tests (crash d'un nœud, test de charge, déploiements de nouvelles versions)

Chapitre 3

Les composants de ce projet

3.1 Wordpress

WordPress est ce qu'on appelle un système de gestion de contenu ou CMS (Content Management System), c'est-à-dire un système permettant de créer et de concevoir des sites web. Il est relativement simple à prendre en main.

WordPress à ses débuts était plus orienté vers les blogs, car en quelques clics, un article peut être publié. Mais aujourd'hui, il s'adapte à tout. Avec Wordpress, nous pouvons créer plusieurs sites comme :

— Des Sites Web d'entreprises
— Des Boutiques e-commerce
— Des Blogs
— Des Portfolios
— Des CV
— Des Forums
— Les réseaux sociaux

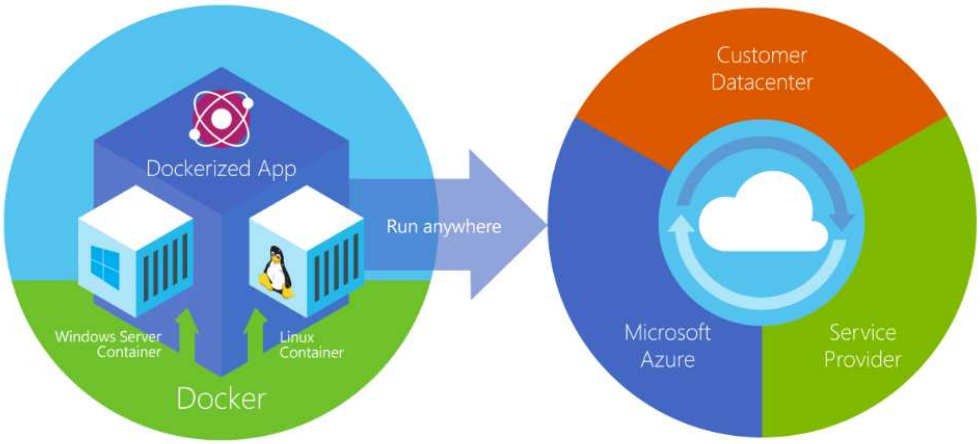
3.2 Qu'est-ce qu'un conteneur ?

En informatique, un conteneur est utilisé pour bénéficier d'un espace d'exécution dédié à une application logicielle. Il peut se présenter sous la forme d'un type de données abstrait, d'une structure d'informations ou encore d'une classe.

L'usage d'un conteneur permet de tester des applications en développement, ainsi que des logiciels. L'intérêt est de disposer d'un environnement isolé pour réaliser tous les contrôles nécessaires avant le déploiement. Il est donc plus facile de distinguer les dysfonctionnements, les failles de sécurité et d'éventuels problèmes de stabilité.

3.3 Docker

Docker est un projet open source permettant d'automatiser le déploiement d'applications en tant que conteneurs portables et autonomes exécutables sur le cloud ou localement. L'éditeur est également une entreprise qui promeut et met en œuvre cette technologie, en collaboration avec les fournisseurs cloud, Linux et Windows, y compris Microsoft.



3.4 Kubernetes

Kubernetes est une plateforme open-source extensible et portable pour la gestion de charges de travail (workloads) et de services conteneurisés. Elle favorise à la fois l'écriture de configuration déclarative (declarative configuration) et l'automatisation. C'est un large écosystème en rapide expansion. Les services, le support et les outils Kubernetes sont largement disponibles.

Google a rendu open-source le projet Kubernetes en 2014. Le développement de Kubernetes est basé sur une décennie et demie d'expérience de Google avec la gestion de la charge et de la mise à l'échelle (scale) en production, associée aux meilleures idées et pratiques de la communauté.

Pourquoi utiliser Kubernetes ?

Kubernetes a un certain nombre de fonctionnalités. Il peut être considéré comme :

— Une plateforme de conteneurs
— Une plateforme de micro-services
— Une plateforme cloud portable et beaucoup plus.

Kubernetes fournit un environnement de gestion focalisée sur le conteneur (container-centric). Il orchestre les ressources machines (computing), la mise en réseau et l'infrastruc-

ture de stockage sur les workloads des utilisateurs. Cela permet de se rapprocher de la simplicité des Platform as a Service (PaaS) avec la flexibilité des solutions d'Infrastructure as a Service (IaaS), tout en gardant de la portabilité entre les différents fournisseurs d'infrastructures (providers).

Ce que Kubernetes ne fait pas

Kubernetes n'est pas une solution PaaS (Platform as a Service). Kubernetes opérant au niveau des conteneurs plutôt qu'au niveau du matériel, il fournit une partie des fonctionnalités des offres PaaS, telles que le déploiement, la mise à l'échelle, l'équilibrage de charge, la journalisation et la supervision. Cependant, Kubernetes n'est pas monolithique. Ces implémentations par défaut sont optionnelles et interchangeables. Kubernetes fournit les bases en laissant la possibilité à l'utilisateur de faire ses propres choix.

Voici une liste de ce que Kubernetes ne fait pas :

- Il ne limite pas les types d'applications supportées. Si l'application peut fonctionner dans un conteneur, elle doit fonctionner correctement sur Kubernetes.

- Il ne fournit pas nativement de services au niveau applicatif tels que des middlewares, des bases de données (comme mysql), ou systèmes de stockage (comme Ceph).

- Il n'impose pas de solutions de journalisation ou de supervision. Kubernetes fournit quelques intégrations primaires et des mécanismes de collecte et export de métriques.

- Il ne fournit ou n'adopte aucune mécanique de configuration des machines, de maintenance, de gestion ou de contrôle de la santé des systèmes.

Kubernetes est composé d'un ensemble de processus de contrôle qui pilotent l'état courant vers l'état désiré. Peu importe comment on arrive du point A au point C. Un contrôle centralisé n'est pas requis. Cela aboutit à un système plus simple à utiliser et plus puissant, robuste, résiliant et extensible.

Pourquoi Kubernetes utilise-t'il des conteneurs ?

La nouvelle façon de déployer une application consiste à déployer des conteneurs basés sur une virtualisation au niveau du système d'opération plutôt que de la virtualisation hardware. Ces conteneurs sont isolés les uns des autres et de l'hôte. Ils sont aussi portables entre les différents fournisseurs de Cloud et les OS.

Étant donné que les conteneurs sont petits et rapides, une application peut être packagée dans chaque image de conteneurs. Cette relation application-image tout-en-un permet de bénéficier de tous les bénéfices des conteneurs.

Avec les conteneurs, des images immuables de conteneurs peuvent être créées au moment du build/release plutôt qu'au déploiement, vu que chaque application n'est pas liée à l'environnement de production. La génération d'images de conteneurs au moment du build permet d'obtenir un environnement constant. Avec une application par conteneur, gérer ces conteneurs équivaut à gérer le déploiement de son application.

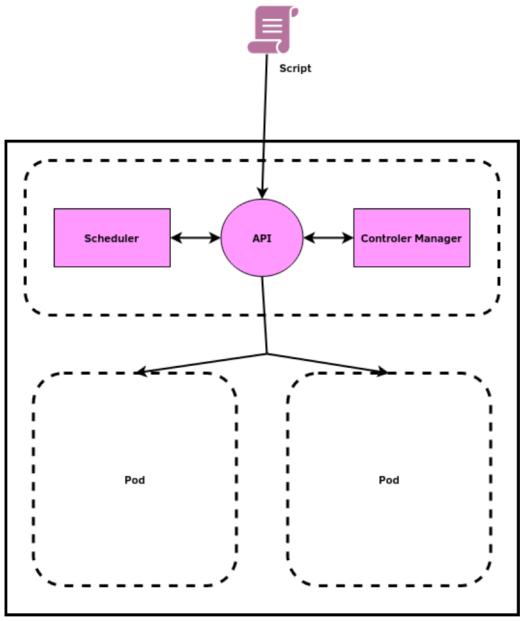
Chapitre 4

Comprendre Kubernetes

4.1 Les objets Kubernetes

4.1.1 L'API Kubernetes

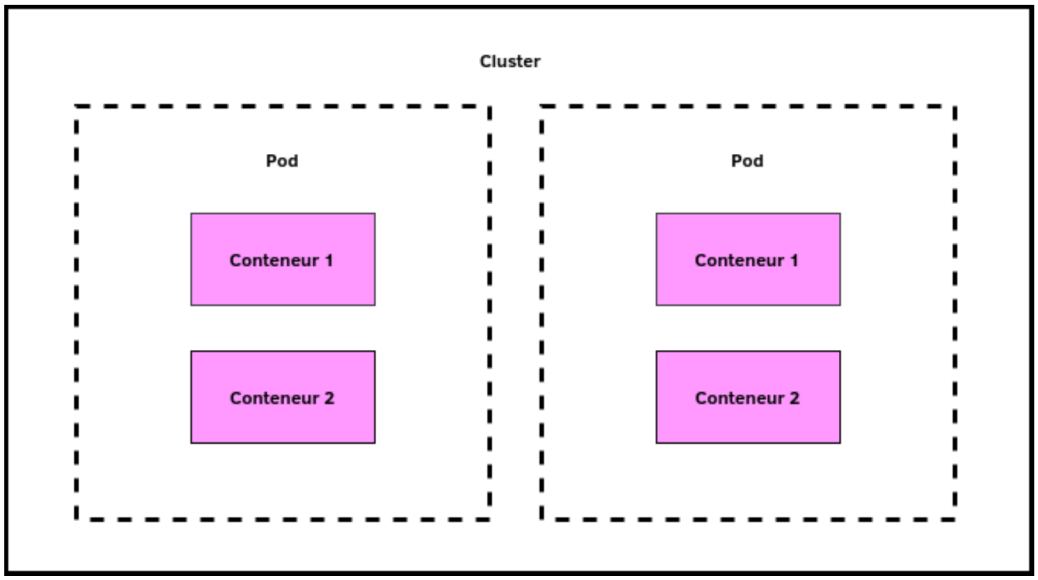
Une (Application Programming Interface) ou Interface de Programmation d'Application est utilisée par Kubernetes et est l'unique moyen pour nous administrateur d'interagir avec le cluster et également à Kubernetes d'interagir avec les éléments, les objets et les ressources dans le cluster. L'API RESTful (REST ou RESTful est un ensemble de contraintes architectural qui ne sont ni des protocoles ni des normes) s'exécute sur HTTP ou HTTPS à l'aide de JSON. Les objets de l'API Kubernetes entre autres qui sont les acteurs clé sont les pods, les contrôleurs, les services et les stockages ou volumes.



4.1.2 Les pods

Ce sont les unités de travail les plus élémentaires de Kubernetes. Un pods est également une représentation d'un ou de plusieurs conteneurs à l'intérieur du cluster. Kubernetes est chargé de la gestion de ces pods et non pas les conteneurs. De manière générale les pods représentent nos applications et ces pods n'ont pas de cycles de vie et ils sont éphémères et uniques. Kubernetes se charge de faire fonctionner nos pods et de s'assurer qu'ils soient déployés dans un état souhaité.

Il est également possible à l'aide de sonde d'activité d'avoir des informations sur l'état de santé d'un pod et de permettre par conséquent à Kubernetes d'agir de manière adéquate soit par un redémarrage, un arrêt, etc.



4.1.3 Les contrôleurs

Ils sont chargés de création et de la gestion du cluster à notre place dans nos déploiements en allant vers **l'état souhaité**.

La documentation Kubernetes les compare à un thermostat. On va donner un état désiré (la température) et le thermostat va essayer d'aller au plus proche de cet état.

Ils répondent également à l'état et à la santé du pod qu'ils soient opérationnels et prêts du point de vue de l'application.

Déploiements L'état désiré de pods est décris dans un deployment. Le **deployment controller** se charge ensuite d'aller de l'état actuel a l'état désiré.

Un exemple de configuration de deployment :

```
kind: Deployment
```

```
metadata:
  name: wordpress-deployment
```

L'état est spécifié dans `spec:`

```
spec:
```

`replicas:` indique le nombre de pods à créer.

```
  replicas: 3
  template:
```

Le `spec:` suivant décris les spécificités du pod, comme le ou les conteneurs utilisés.

```
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

StatefulSet Le statefulset est similaire au deployment. Mais contrairement au deployment le statefulset donne une identité propre a chacun des pods. Leurs spécifications sont les même mais les pods ne peuvent pas êtres interchangés.

4.1.4 configmap

Le configmap est utilisé pour stocker des données. Les pods peuvent les utiliser comme variables d'environnement ou de fichier de configuration dans un volume.

```
kind: ConfigMap
metadata:
  name: mysql
  labels:
    app: mysql
data:
  primary.cnf: |
    [mysqld]
    log-bin
```

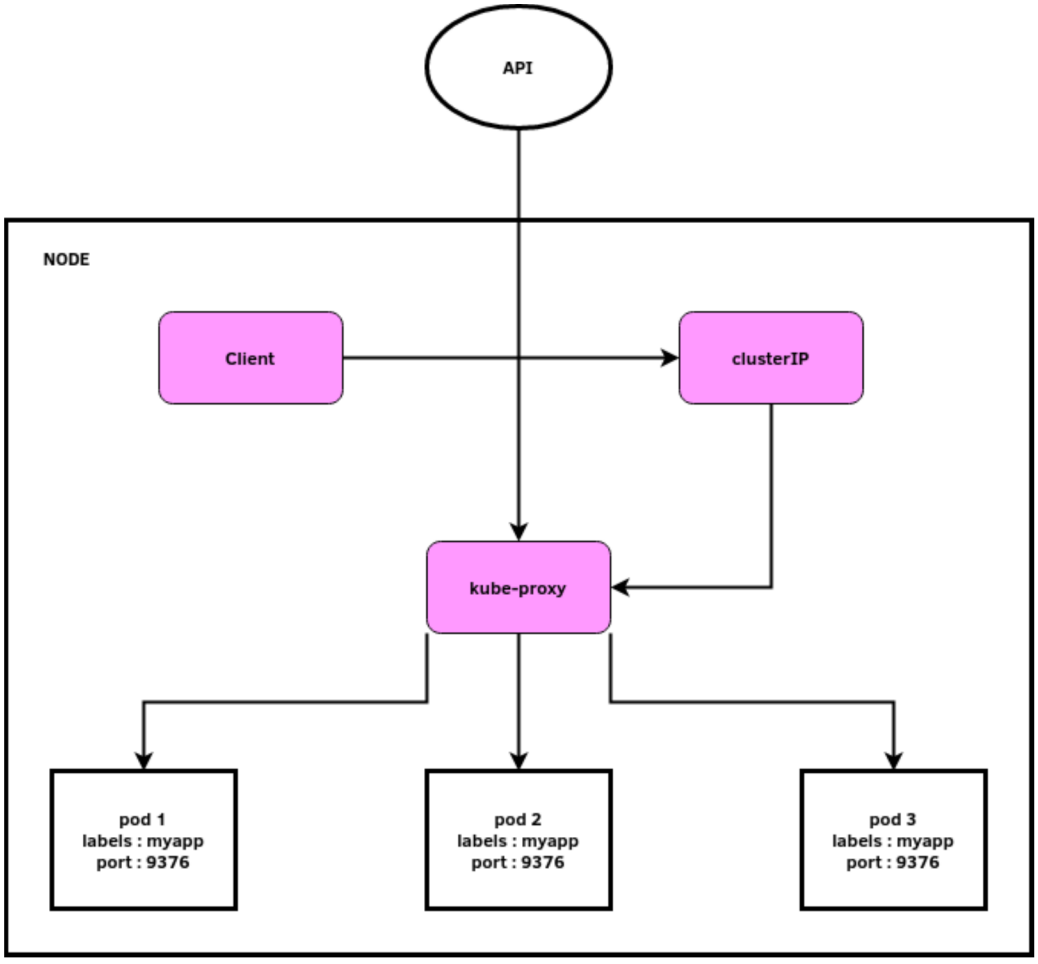
Ce configmap peut être utilisé par un pod pour avoir le fichier primary.cnf.

4.1.5 Les services

Les services ont pour objectif :

- D'ajouter de la persistance dans nos déploiements jusque-là éphémères ;
- De faire la mise en place d'une abstraction du réseau permettant l'accès au pod ;

- D'attribuer des adresses IP et de gérer la résolution de nom (DNS) ;
- De s'assurer du redéploiement automatique des pods ;
- De permettre l'exposition des pods au monde extérieur ;
- De gérer l'équilibrage de charge ;
- D'assurer le NATting des ports sur les pods ;



4.1.6 Le stockage

Par défaut, les données ne sont pas stockées d'où cet aspect éphémère des pods précédemment expliqué, alors est venu le concept de stockage persistant ou Persitant Volume (PV)

Le volume persistant est un stockage indépendant du pod défini par l'administrateur au niveau du cluster pour le stockage des données des pods. L'accès au stockage par les pods se fait par demande appelée Persistant Volume Claim (PVC). Il est possible d'avoir plusieurs PV de différents types (local, NFS, etc) et également d'avoir plusieurs PVC reliés à un seul PV.

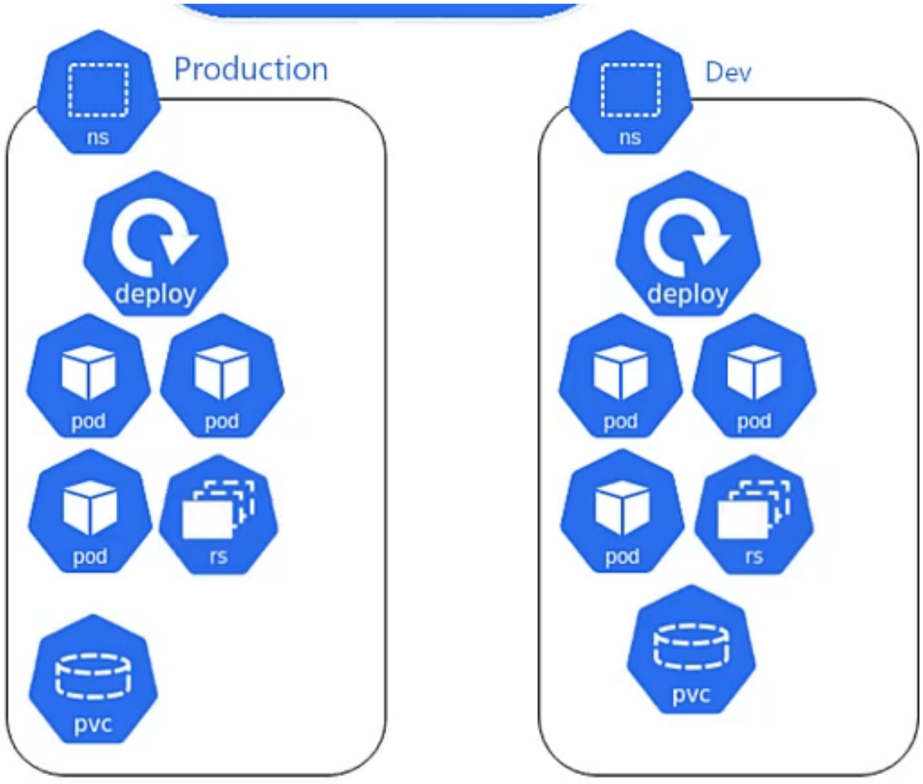
4.1.7 Les espaces de noms ou Namespace

Lorsqu'on se trouve dans un environnement de production où très facilement, on a des dizaines de pods qui sont déployés et à des objectifs et des utilisations différents, il est nécessaire de pouvoir dans l'idéal les séparer ou d'ajouter des moyens de cloisonnement pour que chaque service puisse s'exécuter dans son environnement avec ses particularités et besoins spécifiques. C'est dans ce contexte que dans Kubernetes, il existe ce que l'on appelle des espaces de noms ou communément les namespaces.

Les espaces de noms permettent d'isoler les ressources (pods, replicaset, pvc, deployment) dans un espace, par exemple dans notre cas où on a un espace par défaut qui contient nos pods qui seront déployés et un espace où se trouvent les pods utilisés pour la supervision de nos clusters (Grafana et Prometheus).

La création de ces espaces de noms nous permet de pouvoir gérer l'allocation de ressources, l'allocation de ressources par types, elle permettent également de pouvoir isoler les utilisateurs (l'utilisateur ne verra que ce qui est dans son espace dédié).

Cependant certains objets comme les PV (Persistent Volumes) et les nœuds ne font pas partie des espaces de noms. La commande `kubectl get-ressources` permet de pouvoir lister les objets qui font partie ou noms des namespaces avec des attributs `false` ou `true`.



4.1.8 Kubeconfig

Kubeconfig est un ensemble de fichiers qui permettent de définir les informations de connexion au cluster et les informations sur le (information d'authentification). Ces fichiers sont créés dans l'emplacement `/etc/kubernetes` du cluster et parmi ces fichiers entre autres, on a :

- Le fichier `admin.conf` : c'est le compte administratif de Kubernetes (le super-utilisateur à l'intérieur du cluster) ;
- Le fichier `Kubelet.conf` : permet d'aider le service kubelet à la localisation de l'API
- Le fichier `controler-manager.conf` : qui est le gestionnaire de contrôle ;
- Le fichier `scheduler.conf` : le planificateur.

Il y a également les fichiers de description des pods qui sont appelés les manifest, ils contiennent des informations de description des pods créés.

4.1.9 Déploiement du réseau

L'idée du réseau dans kubernetes est qu'il n'y a pas NAT par pod, mais une adresse IP unique pour chaque pod, la mise en place d'un routage direct pour la communication et le plus important est de permettre l'accessibilité entre les pods sur les nœuds.

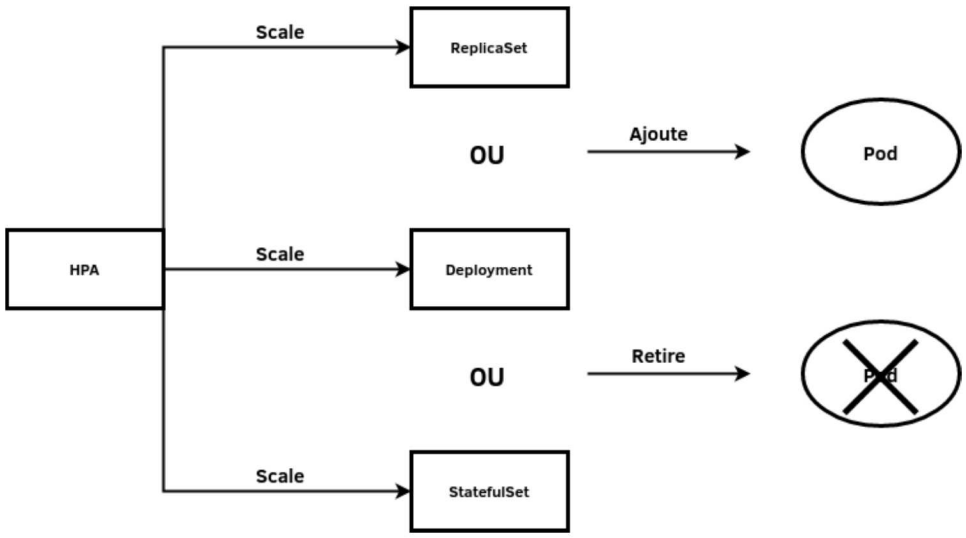
La gestion du réseau est possible avec des solutions parmi celles-ci, il y a Flannel, Calico et Weave. Cette gestion se fait par des fichiers manifest, un pod est créé sur chaque qui sera exclusivement chargé de la gestion de la communication réseau via le kubeproxy. Ces pods se trouvent généralement dans le namesapce appelé **kube-system**.

L'idée, c'est de créer entre tous les nœuds un réseau appelé overlay (Overlay Network), c'est un réseau qui se rajoute à la couche 3, il n'y a pas besoin que nos nœuds soient dans le même emplacement physique pour communiquer (principe de tunnel).

4.1.10 Kubectl

C'est l'outil en ligne de commande (CLI) principal utilisé pour contrôler les clusters Kubernetes. Il permet d'effectuer des opérations (création, démarrer, documenter, exécution ou suppression de ressources) sur des ressources (objets d'API) et de créer des sorties sur des formats de fichier json ou yaml.

4.1.11 HPA



Dans Kubernetes, un HorizontalPodAutoscaler met automatiquement à jour une ressource de charge de travail (comme un Deployment ou un StatefulSet), dans le but d'adapter automatiquement la charge de travail à la demande.

La mise à l'échelle horizontale signifie que la réponse à une augmentation de la charge consiste à déployer davantage de pods. C'est différent de la mise à l'échelle verticale, qui, dans le cas de Kubernetes, consisterait à attribuer davantage de ressources (par exemple, de la mémoire ou du processeur) aux pods déjà en cours d'exécution pour la charge de travail.

Si la charge diminue et que le nombre de pods est supérieur au minimum configuré, l'HorizontalPodAutoscaler demande à la ressource de la charge de travail (le déploiement, le StatefulSet ou toute autre ressource similaire) de réduire sa taille.

L'autoscaling horizontal des pods ne s'applique pas aux objets qui ne peuvent pas être mis à l'échelle (par exemple, un DaemonSet).

L'HorizontalPodAutoscaler est implémenté comme une ressource de l'API Kubernetes et un contrôleur. La ressource détermine le comportement du contrôleur. Le contrôleur d'autoscaling des pods horizontaux, qui s'exécute dans le plan de contrôle Kubernetes, ajuste périodiquement l'échelle souhaitée de sa cible (par exemple, un déploiement) pour correspondre aux métriques observées telles que l'utilisation moyenne du CPU, l'utilisation moyenne de la mémoire ou toute autre métrique personnalisée que vous spécifiez.

4.1.12 YAML

YAML est un langage qui est souvent utilisé pour écrire des fichiers de configuration. Selon la personne à qui vous vous adressez, YAML signifie "yet another markup language" (encore un autre langage de balisage) ou "YAML ain't markup language" (YAML n'est pas un langage de balisage), ce qui souligne que YAML est destiné aux données et non aux

documents.

YAML est un langage populaire, car il est lisible par l'homme et est facile à comprendre. Il peut également être utilisé en conjonction avec d'autres langages de programmation. En raison de sa flexibilité et de son accessibilité, YAML est utilisé par l'outil d'automatisation Ansible pour créer des processus d'automatisation, sous la forme de Playbooks Ansible.

Voici la syntaxe d'un fichier YAML :

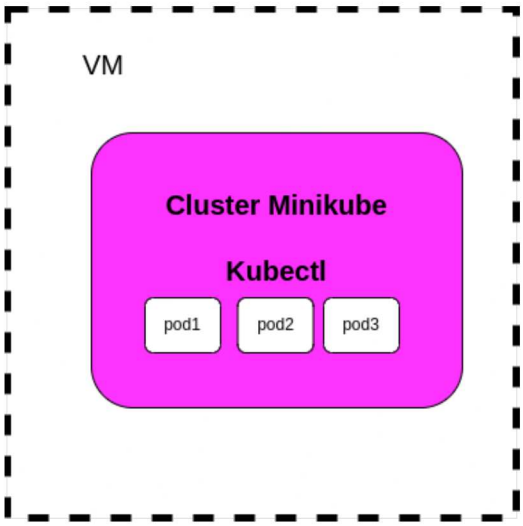
```yaml
---
# An employee record
name: Martin D'vloper
job: Developer
skill: Elite
employed: True
foods:
  - Apple
  - Orange
  - Strawberry
  - Mango
languages:
  perl: Elite
  python: Elite
  pascal: Lame
education: |
  4 GCSEs
  3 A-Levels
  BSc in the Internet of Things
```

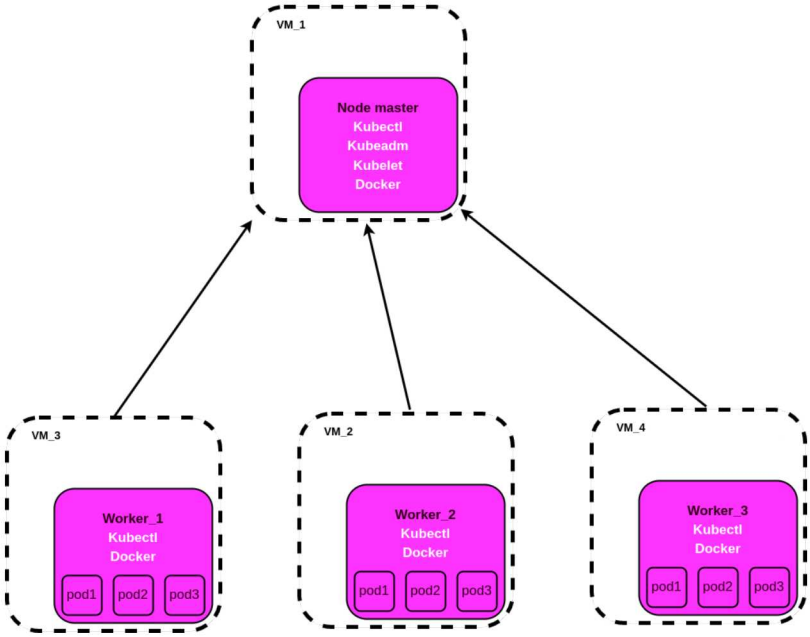
4.2 Les environnements de déploiement possible de Kubernetes

Après s'être assez documenté sur ce qu'est Kubernetes, il est important de maintenant pouvoir le déployer. En fonction de nos besoins et de l'environnement de déploiement, plusieurs solutions s'offrent à nous à savoir :

Minikube

Elle est la première technologie Kubernetes trouvée lorsqu'on essaie de déployer notre environnement Kubernetes. Elle est simple d'installation sur un quelconque ordinateur, car Minikube est particulièrement conçu à des fins de tests ou pour l'apprentissage et donc le plus simple moyen pour se familiariser avec l'utilitaire kubectl. Il est installé avec un cluster à nœud unique, avec l'utilitaire de ce qui la plus petite taille possible, d'où son aspect aussi pratique. En effet, il ne nécessite pas d'énorme ressource de la machine hôte, un minime d'une machine hôte avec deux processeurs, 2Go de mémoire vive et un espace de stockage de 20Go suffisent pour l'installer. Bien qu'il soit léger et assez facile à déployer, son principal inconvénient est qu'il n'est pas possible d'avoir des nœuds supplémentaires pour pouvoir profiter pleinement de la puissance de Kubernetes.

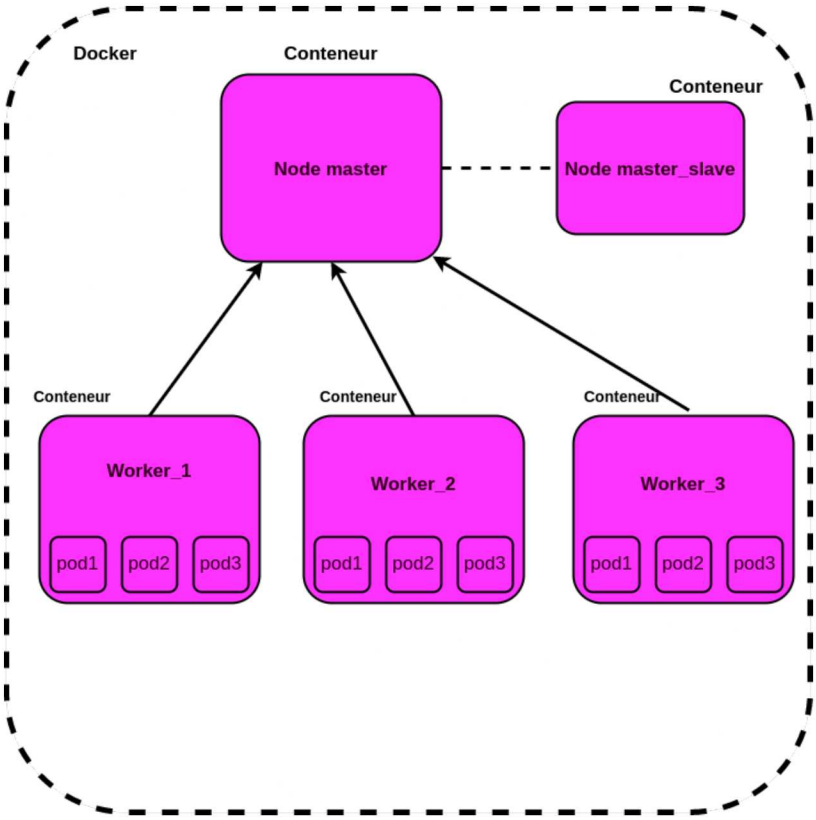




Kubeadm

Communément appelé la voie difficile pour débuter avec Kubernetes. Kubeadm permet d'avoir un cluster minimum de travail avec au minimum un cluster composé de deux nœuds à savoir un nœud de maître (master) et un nœud de travail (worker). Il est également possible d'ajouter autant de nœuds de travail souhaité. Il est important de savoir que c'est une solution assez gourmande en ressource et que chaque nœud doit être déployé sur une machine physique ou être visualisé avec des ressources minimales de 2Go de mémoire vive pour chaque nœud et d'au moins 2 CPU pour le nœud maître (master). Avec Kubeadm, on est sûr de pouvoir bénéficier de tout le potentiel qu'offre Kubernetes.

Il n'est possible que d'avoir un seul nœud maître (master).



Kind

C'est un outil permettant de déployer un cluster local à l'intérieur d'un conteneur docker. Sa particularité comparée autre solution est qu'il est possible d'avoir plusieurs nœuds maîtres (master) en plusieurs nœuds de travail comme Kubeadm. Avec cette solution, il est possible de déployer quasiment tout type de clusters.

Chapitre 5

Travail effectué

5.1 Architecture matérielle

Nous avons choisi d'utiliser Kubeadm pour l'environnement de test. Nous avons utilisé kubeadm sur quatre machines virtuelles avec un maître et trois nœuds.

5.2 Architecture du cluster

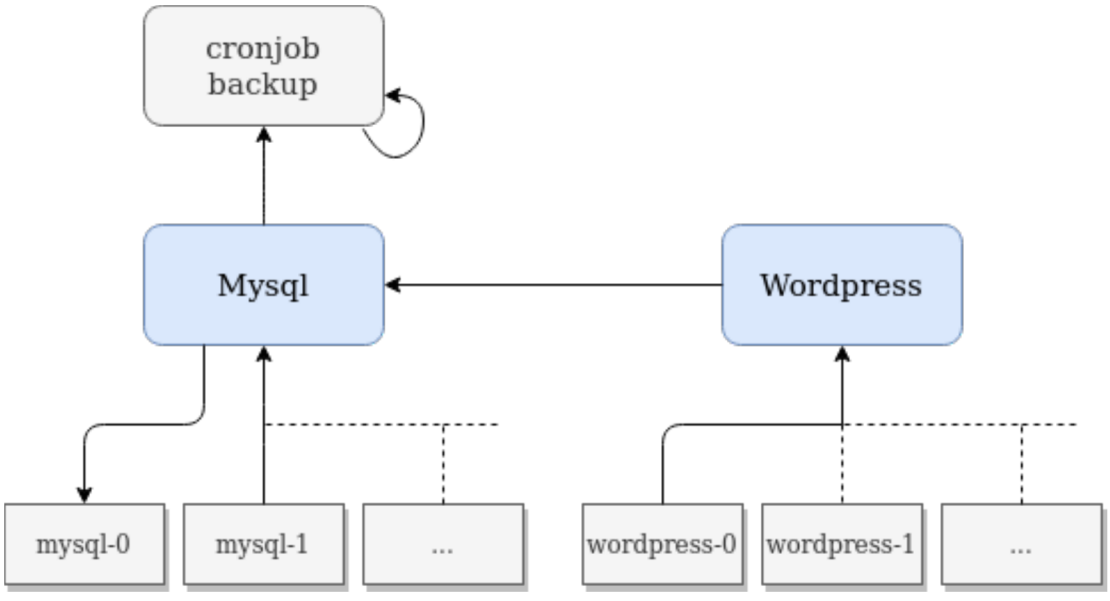


FIGURE 5.1 – Schéma simplifié du cluster

Wordpress stoque les données du site sur une base de donnée et uniquement sur celle-ci. Nous avons fais le choix de séparer les deux applications pour gérer plus facilement et

plus efficacement la mise a l'échelle.

L'application se divise alors en deux grandes parties. L'application **wordpress**, accessible depuis l'extérieur. Et la base de données **Mysql** sur laquelle se repose wordpress. Ces deux services sont scalés en fonction des besoins.

En plus de ces deux parties, un **cronjob** permet de sauvegarder régulièrement la base de données.

5.3 Wordpress

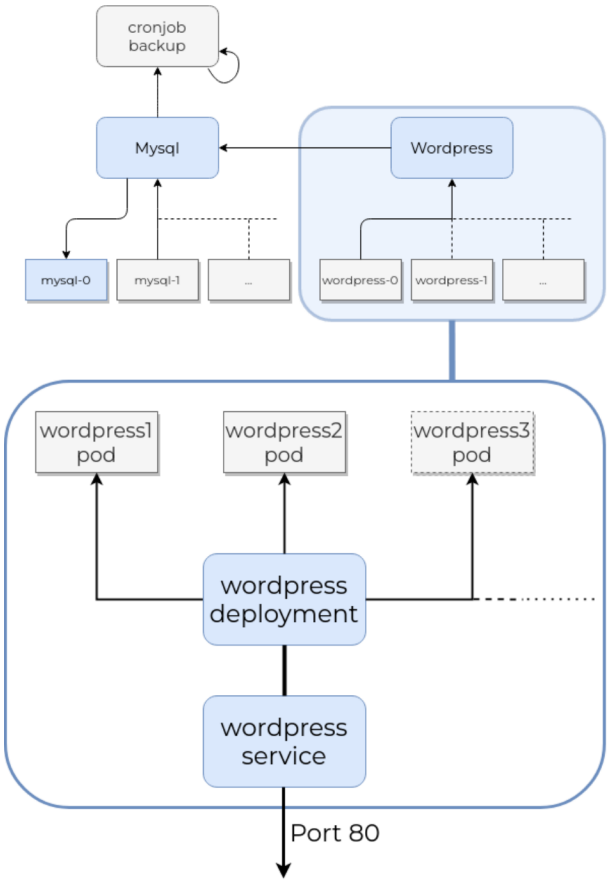


FIGURE 5.2 – Schéma du service wordpress

La partie wordpress est celle qui est **accessible depuis l'extérieur** car elle héberge le serveur web.

Le conteneur utilisé a wordpress déjà installé. Une variable d'environnement permet de spécifier le host de la base de données.

Le service wordpress permet d'**exposer** wordpress sur le port 80.

Un deployment wordpress est utilisé pour pouvoir rendre **élastique** le service. Le deployment permet de créer des pods. Il modifie la variable d'environnement du host de la base de données. Ce déployement va ensuite créer un certain nombre de pods avec les conteneurs wordpress.

Configuration YAML

La configuration est plutôt simple. Un service et deployment en plus du stockage suffisent.

Voici une partie de la configuration du deployment :

```
kind: Deployment
spec:
  strategy:
    type: Recreate
  template:
    spec:
      containers:
      - image: wordpress:4.8-apache
        name: wordpress
        env:
        - name: WORDPRESS_DB_HOST
          value: mysql
        ports:
        - containerPort: 80
          name: wordpress
        volumeMounts:
        - name: wordpress-persistent-storage
          mountPath: /var/www/html
```

La partie `env:` met mysql dans la variable d'environnement `WORDPRESS_DB_HOST` pour que wordpress situ la base de données.

5.4 Mysql

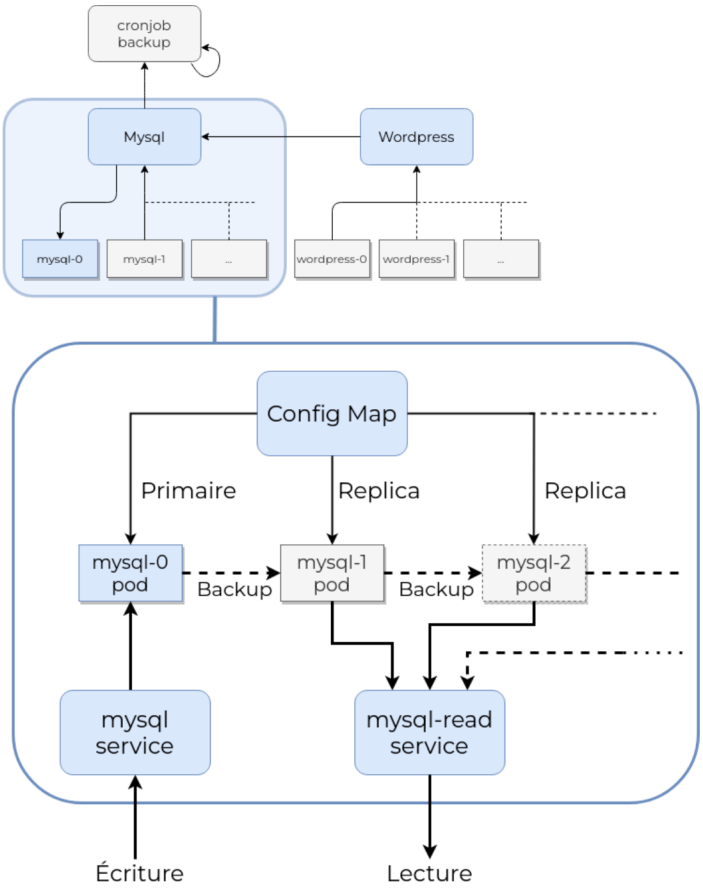


FIGURE 5.3 – Schéma du service mysql

Une architecture a dû être mise en place pour rendre **élastique** mysql. Ainsi, plusieurs pods peut-être mis en service. Mais **seul le premier pod** est accessible en **écriture**. Les pods ont un **nom unique** en fonction de l'ordre d'initialisation : mysql-**0**, mysql-**1**, mysql-**2**...

Initialisation des pods mysql

Avant la création du conteneur MySQL, le pod lance plusieurs conteneurs. Le premier est **init-mysql**. Il **va appliquer le configmap** approprié en fonction de si le pod est le primaire ou un réplica.

Un second conteneur appelé **mysql-clone** s'occupe de **cloner les données**. Chaque pod clone les données à partir du pod précédent. Par exemple, le pod mysql-3 clone les données du pod mysql-2. Pour créer un nouveau pods, le statefulset s'assure donc que le pod précédent a fini d'être créé. Il n'est **pas possible de créer plusieurs pods en parallèle**.

L'outil open source **xtrabackup** est utilisé pour cloner les données.

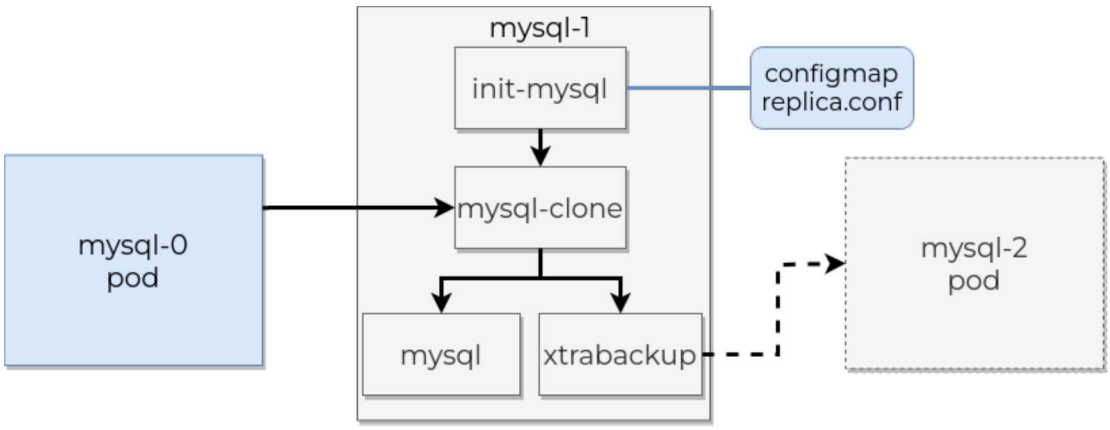


FIGURE 5.4 – Initialisation d'un pod mysql

Le pod final a deux conteneurs. Un conteneur MySQL qui s'occupe de la base de données et un conteneur xtrabackup. Ce dernier permet au pod suivant de cloner les données.

Configuration YAML

Pour réaliser cette architecture logicielle, nous avons utilisé un configmap et un statefulset.

Le configmap nous permet de générer un petit stockage avec des fichiers de configuration. Une configuration primary.cnf pour le premier pod mysql et replica.cnf pour les autres. La configuration va surtout permettre de configurer la base de données des réplicas en lecture seule.

Un statefulset défini l'état souhaité des pods. On y définit dans `initContainers` init-mysql et clone-mysql.

init-mysql utilise un conteneur mysql et clone-mysql un conteneur xtrabackup. initmysql a le rôle d'appliquer la bonne configuration sql. clone-mysql va copier la base de données dans le pod.

Le statefulset va faire exécuter ce script bash a init-mysql :

```
[[ `hostname` =~ -([0-9]+)$ ]] || exit 1
ordinal=${BASH_REMATCH[1]}
# Ajout de 100 pour eviter server-id=0 qui est indisponible.
echo server-id=$((100 + $ordinal)) >> /mnt/conf.d/server-id.cnf
```

Ces lignes génèrent le server-id pour mysql en fonction du nom du pod.

Le script applique ensuite la configuration primary.cnf ou replica.cnf en fonction de si c'est le pod 0 (ordinal=0) ou non.

```
if [[ $ordinal -eq 0 ]]; then
    cp /mnt/config-map/primary.cnf /mnt/conf.d/
else
    cp /mnt/config-map/replica.cnf /mnt/conf.d/
fi
```

Le conteneur clone-mysql va lui aussi exécuter un script qui va cloner la base de données du pod précédent.

On définit ensuite dans `containers:` les conteneurs du pod final.

D'abord le conteneur MySQL. Les ressources cpu et mémoire sont définis :

```
        resources:
          requests:
            cpu: 500m
            memory: 1Gi
```

Les commandes pour connaître l'état du conteneur MySQL sont redéfinies. On veut connaître l'état de la base de données et que de celle-ci. On vérifie son état avec un ping de la base.

```
livenessProbe:
  exec:
    command: ["mysqladmin", "ping"]
  initialDelaySeconds: 30
  periodSeconds: 10
  timeoutSeconds: 5
```

```
readinessProbe:
  exec:
    command: ["mysql", "-h", "127.0.0.1", "-e", "SELECT 1"]
  initialDelaySeconds: 5
  periodSeconds: 2
  timeoutSeconds: 1
```

5.5 Sauvegardes

Un conteneur est construit pour réaliser les sauvegardes. Ce conteneur a une base alpine Linux sur laquelle est installé MySQL. Le conteneur lance un script au démarrage qui permet de faire les sauvegardes.

Un cronjob est utilisé pour appeler le conteneur deux fois par jour.

5.6 Élasticité

Pour augmenter le nombre de pods en cas de charge trop importante, il faut d'abord connaître la charge des pods. Kubernetes ne le fait pas de base. Il faut installer un metrics-server. On peut en installer un avec cette commande :

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/
    latest/download/components.yaml
```

Ensuite, c'est encore plus simple. Une commande kubectl permet de créer un auto-scaler pour un deployment ou autre. Exemple pour le deployment wordpress :

```
kubectl autoscale deployment wordpress --cpu-percent=70 --min=1 --max=10
```

5.7 Supervision

5.7.1 Prometheus

Prometheus est un framework de surveillance open-source. Il fournit des capacités de surveillance prêtes à l'emploi pour la plateforme d'orchestration de conteneurs Kubernetes.

Parmi ses fonctionnalités, il y a :

— Collecte de métriques : Prometheus utilise le modèle pull pour récupérer les métriques via HTTP. Il existe une option permettant de pousser les métriques vers Prometheus à l'aide de Pushgateway pour les cas d'utilisation où Prometheus ne peut pas récupérer les métriques.
Un tel exemple est la collecte de mesures personnalisées à partir de tâches Kubernetes et de tâches Cronjobs de courte durée.
— Les métrics endpoint : les systèmes à surveiller à l'aide de Prometheus doivent exposer les métriques sur un point de sortie.
Prometheus utilise les metrics endpoints pour extraire les métriques à intervalles réguliers.
— PromQL : Prometheus est fourni avec PromQL, un langage de requête très souple qui peut être utilisé pour interroger les mesures dans le tableau de bord Prometheus. De plus, la requête PromQL sera utilisée par Prometheus UI et Grafana pour visualiser les métriques.
— Prometheus Exporters ou exportateurs Prometheus : ce sont des bibliothèques qui convertissent les métriques existantes des applications tierces au format des métriques Prometheus. Il existe de nombreux exportateurs Prometheus officiels et communautaires.
Un exemple est l'exportateur de nœuds Prometheus. Il expose toutes les métriques de niveau système de Linux au format Prometheus.
— TSDB (Time-Series Database ou base de données de séries temporelles) : Prometheus utilise TSDB pour stocker efficacement toutes les données. Par défaut, toutes les données sont stockées localement.

La pile de surveillance Prometheus de Kubernetes comprend les composants suivants :

— Prometheus server ;

— Alert Manager ;
— Grafana

5.7.2 Grafana

Grafana est un logiciel libre de visualisation de donnée. Il permet de faire des tableaux de bord à partir de plusieurs sources de données. Dans notre cluster grafana est connecté à prometheus pour visualiser la charge des nœuds.

Voici a quoi ressemble l'installation grafana qui monitor notre cluster wordpress sur un minikube :

5.8 Script d'installation

Un script bash a été créé pour facilement mettre en place le wordpress et appliquer les configurations YAML.

Le script va d'abord construire le conteneur avec docker à l'aide d'un dockerfile.

```
docker build -t wordback ./mysql-backup/
```

Puis configurer prometheus et grafana dans le namespace “monitoring”.

```
kubectl create namespace monitoring

#Installation de Prometheus
kubectl create -f clusterRole.yaml
kubectl create -f config-map.yaml
kubectl create -f prometheus-deployment.yaml
kubectl create -f prometheus-service.yaml --namespace=monitoring

#Installation de Grafana
kubectl create -f grafana-datasource-config.yaml
kubectl create -f grafana-deployment.yaml
kubectl create -f grafana-service.yaml
```

Mise en place du mysql puis de wordpress :

```
#Mise en place de mysql
kubectl apply -f ./mysql-configmap.yaml
kubectl apply -f ./mysql-services.yaml
kubectl apply -f ./mysql-statefulset.yaml

#Mise en place de wordpress
kubectl apply -f ./wordpress-deployment.yaml
```

Ensuite, est mis en place les horizontal pod autoscaler. C'est ici que l'on peut configurer quelques paramètres du hpa. Par exemple le seuil d'utilisation cpu pour la création d'un nouveau pod.

```
kubectl autoscale deployment wordpress --cpu-percent=70 --min=1 --max=10
kubectl autoscale StatefulSet mysql --cpu-percent=70 --min=2 --max=10
```

Enfin, le script donne des informations sur les objets créés pour vérifier qu'ils se lancent. La dernière ligne du script donne l'URL du wordpress.

```
kubectl get pvc
kubectl get pods
kubectl get services wordpress

echo "ip du wordpress:"
kubectl get service/wordpress -o jsonpath='{.spec.clusterIP}'
```

Liste des prérequis :
— Un cluster Kubernetes
— kubectl
— Docker pour construire le conteneur de sauvegarde
En plus des minimums matériels variables en fonction du site Wordpress.

5.9 Test de charge

Afin de tester la robustesse de notre cluster et sa capacité à s'adapter à un nombre important d'utilisateurs, nous avons effectué un test de charge sur celui-ci afin de savoir en combien de temps le site peut charger pour tous les utilisateurs et s'ils n'attendent pas trop, en outre savoir si notre cluster fait bien son boulot et rapidement.

Pour effectuer ce test, nous avons utilisé Gatling qui est un outil puissant de test de charge open-source.

À partir de scénario que nous créons nous-même avec le langage Java, Scala et tant d'autres, nous pouvons simuler automatiquement des centaines de milliers de requêtes par seconde sur notre Wordpress et obtenir des mesures de haute précision.

Une fois le test terminé, un rapport est généré et nous allons l'étudier dans cette partie.

Tout d'abord, voici le script que nous avons créé :

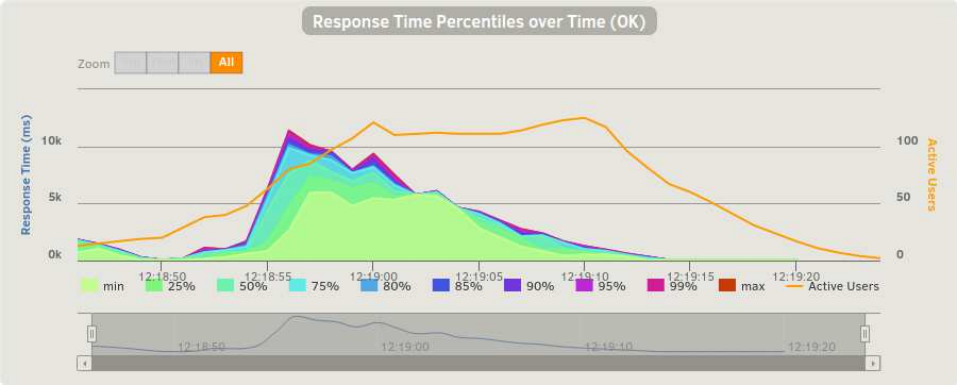
```
import io.gatling.javaapi.core.*;
import io.gatling.javaapi.http.*;

import static io.gatling.javaapi.core.CoreDsl.*;
import static io.gatling.javaapi.http.HttpDsl.*;
```

```
public class test extends Simulation { // 3

  HttpProtocolBuilder httpProtocol = http // 4
    .baseUrl("http://192.168.39.134:31286/") // 5
    .acceptHeader("text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q
    =0.8") // 6
    .doNotTrackHeader("1")
    .acceptLanguageHeader("en-US,en;q=0.5")
    .acceptEncodingHeader("gzip, deflate")
    .userAgentHeader("Mozilla/5.0 (Windows NT 5.1; rv:31.0) Gecko/20100101
    Firefox/31.0");

  ScenarioBuilder scn = scenario("BasicSimulation") // 7
    .exec(http("request_1") // 8
      .get("/")) // 9
    .pause(5); // 10

  {
    setUp(
  scn.injectOpen(
    nothingFor(4), // 1
    atOnceUsers(10), // 2
    rampUsers(10).during(5), // 3
    rampUsersPerSec(10).to(20).during(10).randomized(), // 4
    stressPeakUsers(100).during(20) // 5
  ).protocols(httpProtocol)
);
  }
}
```

Ce script simple en java explique qu'il va simuler sur le site (baseUrl) des arrivées d'utilisateurs sur Firefox avec des paramètres appliqués sur celui-ci (userAgentHeader, acceptLanguageHeader, etc).

Le scénario est simple (injectOpen) : 4 seconde où il ne se passe rien et ensuite 10 utilisateurs viennent d'un coup sur le site, ensuite 10 utilisateurs de plus vont venir en 5 seconde sur le site et entre 10 et 20 utilisateurs vont venir par seconde sur le site pendant 10 seconde, enfin, pendant 20 secondes, il va y avoir un pique de 100 utilisateurs sur le site.

Ce scénario simple et efficace va nous permettre de voir si notre infrastructure est parée à un tel pique d'utilisateurs, d'ailleurs voici les résultats :



STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	269	269	0	0%	6.897	45	1740	6529	9257	10237	11385	3481	3300
request_1	269	269	0	0%	6.897	45	1740	6529	9257	10237	11385	3481	3300

Comme on peut le voir, toutes les requêtes ont reçu leur réponse, mais malheureusement, seulement 86/250 l'ont reçu en moins de 800ms, ce qui est peut.

La raison que nous avons est dans le graphique, comme il y a beaucoup d'utilisateurs qui viennent d'un coup sur le site, Kubernetes n'était pas préparé à ça et donc pour contrer cela, il crée des replicats afin d'alléger la charge de travail et comme on peut le voir, le temps de réponse baisse considérablement.

En outre, la grande partie du résultat est basée sur le moment où il crée des réplicats et donc au moment où le temps d'attente est long.

Au vu du graphique, on peut dire que Kubernetes fait bien son travail, car il a réussi à baisser le temps de réponse quand il y avait encore beaucoup d'utilisateurs, bien que cela ait pris du temps, une fois le réplicat fait, tout roule comme sur des roulettes.

En conclusion, notre infrastructure est parée à un pique d'utilisateurs élevés une fois le replicat créé.

Chapitre 6

Conclusion

Avec ce projet, nous avons appris ce que représente Kubernetes à travers sa documentation complète et fournie ainsi que des mises en pratique de cette dernière. Mais Kubernetes nécessite une maîtrise ainsi qu'une connaissance que nous n'avons pas eue au début, en ce qui concerne son environnement. On peut citer, par exemple, l'HPA (Horizontal Pod Autoscaling), le langage YAML ainsi que Minikube. À travers cette documentation, nous avons su résoudre des problèmes au sein de ce projet, mais avec cette documentation complète et fournie, il est également nécessaire de comprendre certains procédés et certains langages. On peut citer les volumes, par exemple. De plus, sur notre environnement, nous avons déployé qu'un nœud sur Minikube, alors que la perspective voulue est de vouloir déployer plusieurs nœuds, alors nous avons décidé de changer Minikube pour KubeAdm. Enfin, nous terminerons sur une citation qui s'est manifestée lors de ce projet : "Il faut toujours se renseigner pour mieux progresser".

Chapitre 7

Bibliographie

Voici les liens que nous avons utilisés pour mener à bien ce projet.

Pour comprendre kubernetes en genéral.

- https ://www.jesuisundev.com/comprendre-kubernetes-en-5-minutes/
- https ://kubernetes.io/docs/tutorials/kubernetes-basics/
- https ://www.mirantis.com/blog/introduction-to-yaml-creating-a-kubernetes-deployment/
- https ://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/

Pour comprendre le stockage dans kubernetes

- https ://akomljen.com/kubernetes-persistent-volumes-with-deployment-and-statefulset/
- https ://devopssec.fr/article/fonctionnement-manipulation-volumes-kubernetes
- https ://www.adaltas.com/fr/2017/10/28/methodes-de-stockage-persistees-dans-kubernetes/
- https ://kubernetes.io/docs/concepts/storage/volumes/
- https ://www.grottedubarbu.fr/kubernetes-volumes/

Réplication d'une base de donnée

- https ://kubernetes.io/docs/tasks/run-application/run-replicated-stateful-application
- https ://kubernetes.io/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume/
- https ://kublr.com/blog/setting-up-mysql-replication-clusters-in-kubernetes-2/
- https ://dev.to/preethamsathyamurthy/set-up-a-kubernetes-master-slave-architecture-using-kubeadm-9b3
- https ://github.com/nirgeier/KubernetesLabs/tree/master/Labs/09-StatefulSet
- https ://portworx.com/blog/ha-mysql-openshift/
- https ://stackoverflow.com/questions/69215635/scale-up-mysql-with-hpa-creates-an-empty-database-on-kubernetes

Supervision

- https ://computingforgeeks.com/setup-prometheus-and-grafana-on-kubernetes/

Sauvegarde d'une base de donnée dans kubernetes

- https ://fr.wordpress.org/support/article/wordpress-backups/
- https ://alexohneander.medium.com/backup-mysql-databases-in-kubernetes-e4a410160fe4

Chapitre 8

Annexes

8.1 Configuration YAML

8.1.1 Wordpress

wordpress-deployment.yaml qui définit le Service, le stockage et le déploiement de wordpress.

```yaml
kind: Service
metadata:
  name: wordpress
  labels:
    app: wordpress
spec:
  ports:
    - port: 80
  selector:
    app: wordpress
    tier: frontend
  type: LoadBalancer
```

```yaml
---
kind: PersistentVolumeClaim
metadata:
  name: wp-pv-claim
  labels:
    app: wordpress
spec:
  resources:
    requests:
      storage: 20Gi
```

```yaml
---
kind: Deployment
metadata:
  name: wordpress
spec:
  template:
    metadata:
```

```
      labels:
        app: wordpress
    spec:
      containers:
      - image: wordpress:4.8-apache
        env:
        - name: WORDPRESS_DB_HOST
          value: mysql
        ports:
        - containerPort: 80
          name: wordpress
        volumeMounts:
        - name: wordpress-persistent-storage
          mountPath: /var/www/html
      volumes:
      - name: wordpress-persistent-storage
        persistentVolumeClaim:
          claimName: wp-pv-claim
```

8.1.2 mysql

mysql-configmap.yaml définit le configmap.

```
kind: ConfigMap
metadata:
  name: mysql
data:
  primary.cnf: |
    [mysqld]
    log-bin
  replica.cnf: |
    [mysqld]
    super-read-only
```

mysql-statefulset.yaml définit le statefulset.

```
kind: StatefulSet
metadata:
  name: mysql
spec:
  serviceName: mysql
  replicas: 3
  template:
    metadata:
      labels:
        app: mysql
    spec:
```

```
      initContainers:
      - name: init-mysql
        image: mysql:5.7
        command:
```

```
      - bash
      - "-c"
      - |
        set -ex
        # Genere l'id mysql  selon l'index ordinal du pod.
        [[ `hostname` =~ -([0-9]+)$ ]] || exit 1
        ordinal=${BASH_REMATCH[1]}
        echo [mysqld] > /mnt/conf.d/server-id.cnf
        # Ajout de 100 pour eviter server-id=0 qui est
indisponible.
        echo server-id=$((100 + $ordinal)) >> /mnt/conf.d/
server-id.cnf
        # Applique le primary.cnf ou replica.cnf defini dans
        # le configmap en fonction de l'index ordinal
        if [[ $ordinal -eq 0 ]]; then
          cp /mnt/config-map/primary.cnf /mnt/conf.d/
        else
          cp /mnt/config-map/replica.cnf /mnt/conf.d/
        fi
      volumeMounts:
      - name: config-map
        mountPath: /mnt/config-map
```

```
    - name: clone-mysql
      image: gcr.io/google-samples/xtrabackup:1.0
      command:
      - bash
      - "-c"
      - |
        set -ex
        # Exit si une base existe deja.
        [[ -d /var/lib/mysql/mysql ]] && exit 0
        # Exit si c'est le premier pod.
        [[ `hostname` =~ -([0-9]+)$ ]] || exit 1
        ordinal=${BASH_REMATCH[1]}
        [[ $ordinal -eq 0 ]] && exit 0
        # Clone la base.
        ncat --recv-only mysql-$(($ordinal-1)).mysql 3307 |
xbstream -x -C /var/lib/mysql
        # Prepare the backup.
        xtrabackup --prepare --target-dir=/var/lib/mysql
      volumeMounts:
      - name: data
        mountPath: /var/lib/mysql
        subPath: mysql
      - name: conf
        mountPath: /etc/mysql/conf.d
```

```
    containers:
    - name: mysql
      image: mysql:5.7
      volumeMounts:
      - name: data
```

```
          mountPath: /var/lib/mysql
      - name: xtrabackup
        image: gcr.io/google-samples/xtrabackup:1.0
        volumeMounts:
        - name: data
          mountPath: /var/lib/mysql
          subPath: mysql
      volumes:
      - name: conf
        emptyDir: {}
      - name: config-map
        configMap:
          name: mysql
```

```
  volumeClaimTemplates:
  - metadata:
      name: data
    spec:
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 5Gi
```

mysql-services.yaml définit les services mysql et mysql-read.

```
kind: Service
metadata:
  name: mysql
spec:
  ports:
  - name: mysql
    port: 3306
  clusterIP: None
```

```
---
kind: Service
metadata:
  name: mysql-read
spec:
  ports:
  - name: mysql
    port: 3306
```

8.1.3 cronjob sauvegarde

cron-backup.sh

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: wordback
spec:
```

```
schedule: "0 * * * *"
jobTemplate:
  spec:
    template:
      spec:
        containers:
        - name: wordback
          image: wordback/wb
          env:
          - name: MYSQL_HOST
            value: "mysql"
          volumeMounts:
          - mountPath: "/pg_backup"
            name: backup-volume
          imagePullPolicy: IfNotPresent
        restartPolicy: OnFailure
        volumes:
          - name: backup-volume
            persistentVolumeClaim:
              claimName: pg-backup-pvc
```