KUBERNETES

SMILE

I.T IS OPEN

Table des matières

I. Préambule

I.1. Smile

Avec une **présence dans 7 pays**, Smile est le **leader européen du digital ouvert**, expert du digital et de l'open source (conseil, intégration et infogérance). **Près de 1 700 passionnés** contribuent chaque année à plusieurs centaines de projets digitaux stratégiques pour les plus grands comptes français et européens sur la base de **solutions et concepts les plus innovants**. Maîtrisant autant les meilleurs produits, composants et frameworks open source que les enjeux Business, Smile accompagne ses clients à chaque étape de leur transformation digitale à travers quatre offres verticales (**Digital/Ebusiness, Business Apps, Infrastructure, Embedded/IoT**) et une ligne complète de services intégrés (**conseil, agence digitale, formation, développement et intégration, maintenance et infogérance**). En 2018, Smile a réalisé un chiffre d'affaires de 102.5 millions d'euros. La mission de Smile ? Faire de l'open source, le principal vecteur de digitalisation en Europe.

Smile est membre de l'**APRIL**, l'association pour la promotion et la défense du logiciel libre, du **PLOSS** – le réseau des entreprises du Logiciel Libre en Ile-de-France.

Depuis 2000 environ, **Smile mène une action active de veille technologique** qui lui permet de découvrir les produits les plus prometteurs de l'open source, de les qualifier et de les évaluer, de manière à proposer à ses clients les produits les plus about is, les plus robustes et les plus pérennes. Cette démarche a donné lieu à **toute une gamme de *livres blancs & mini-books*** couvrant différents domaines d'application. Chacun des **ouvrages présente une sélection des meilleures solutions open source** dans le domaine considéré, leurs qualités respect ives, ainsi que des retours d'expérience opérationnels.

Au fur et à mesure que des solut ions open source solides gagnent de nouveaux domaines, Smile sera présent pour proposer à ses clients d'en bénéficier sans risque.

Ces dernières années, Smile a également étendu la gamme des services proposés. Depuis 2005, un département **consulting** accompagne nos clients, tant dans les phases d'avant-projet, en recherche de solut ions, qu'en accompagnement de projet.

Depuis 2000, Smile dispose d'un **studio graphique**, devenu en 2007 Smile Digital – agence interactive, proposant outre la création graphique, une expertise e-marketing, éditoriale, et interfaces riches. Smile dispose aussi **d'une agence spécialisée dans la TMA** (support et l'exploitation des applications) et d'un **centre de formation** complet, Open Source School.

Enfin, Smile est implanté à Paris, Lille, Lyon, Grenoble, Nantes, Bordeaux, Marseille, Toulouse et Montpellier. Et présent également en Belgique, en Suisse, au Luxembourg, aux Pays-Bas, en Ukraine ou encore au Maroc.

I.2. Pourquoi ce livre blanc ?

La popularité de Kubernetes

Kubernetes est devenu un outil indispensable pour les applications Cloud-Native, conçues pour être hébergées dans le cloud, ainsi que les microservices. Ses avantages sont les suivants :

- Le design Cloud-Native : Kubernetes encourage l'implémentation des architectures modulaires et distribuées, ce qui augmente l'agilité, la disponibilité, et la scalabilité de l'application.
- La portabilité : Kubernetes fonctionne de la même manière, en utilisant la même image et les mêmes configurations, quelque soit le cloud provider ou l'environnement utilisé.
- L'open source : Kubernetes est un projet open source disponible sur GitHub. Grâce à sa large communauté, Kubernetes est devenu aujourd'hui l'un des projets les plus forkés sur GitHub.

Les objectifs de ce livre blanc

Ce livre blanc n'est pas un guide d'utilisation et de déploiement de Kubernetes et/ou d'applications sur Kubernetes. Le contenu de ce livre n'a pas vocation à remplacer la documentation https:/ kubernetes.io que nous considérons comme suffisamment complète sur ces sujets.

Ce livre blanc consiste à présenter l'outil Kubernetes, et de fournir un retour d'expérience sur sa mise en place dans le cadre d'un lab.

I.3. Auteur

Ce livre blanc a été rédigé par **Ismail KABOUBI** (Ingénieur Système DevOps à Smile) :

- CKA (Certified Kubernetes Administrator)
- CKAD (Certified Kubernetes Application Developper.)
- Formateur Kubernetes Fondamentales (Formation Certifiante CKA et CKAD)
- Contributeur sur le projet Charts de Kubernetes
- Développeur Python et Go

II. Introduction

Kubernetes est un orchestrateur de conteneurs open source créé par Google en 2014 et actuellement maintenu par la CNCF (Cloud Native Computing Foundation). Kubernetes a été inspiré par le projet Borg qui est un orchestrateur de conteneurs et également de machines virtuelles développé en interne par Google.

Borg est le fruit d'un travail acharné d'une dizaine d'années qui est aujourd'hui la base des travaux effectués par les ingénieurs Google et la communauté open source sur le projet Kubernetes.

Kubernetes est le fruit de l'expérience Google mais aussi d'une communauté open source très importante. Son intégration peut se faire aussi bien sur un Raspberry Pi que sur un parc de serveurs hébergés dans un datacenter. Il fournit un environnement adéquat pour déployer des applications fiables, scalables et distribuées.

Kubernetes est un mot grec qui signifie "capitaine" ou "pilote". Il est écrit en Go, un langage de programmation compilé et statiquement lié, conçu par Google en 2009 qui favorise la simplicité, la haute performance, et le parallélisme.

Kubernetes permet de gérer des applications conteneurisées dans un environnement distribué. Il simplifie les tâches de déploiement, de mise en échelle, et de configuration avec un "zero downtime".

Un conteneur fournit un contexte isolé dans lequel une application ou un microservice peut être exécuté avec ses dépendances. Toutefois, les conteneurs doivent être gérés en externe et être planifiés, distribués pour répondre aux besoins des applications et des infrastructures modernes. Parallèlement à cela, la persistance des données et la configuration du réseau compliquent la gestion des conteneurs.

Kubernetes fournit une couche sur l'infrastructure capable de relever ces défis. Il utilise des étiquettes (labels) en tant que balises de nom pour identifier ses objets, et il peut effectuer des requêtes sur la base de ces étiquettes. Les étiquettes peuvent être utilisées pour indiquer un rôle, un nom, ou d'autres attributs importants.

Dans ce livre blanc, nous allons présenter les différentes méthodes d'installation de Kubernetes, les outils permettant de déployer des applications résilientes et scalables, ainsi qu'un retour d'expérience sur l'intégration et l'utilisation de Kubernetes dans le contexte d'un lab.

III. Vue d'ensemble

III.1. Introduction

Kubernetes est devenu le leader dans le déploiement des applications Cloud-Native.

La conteneurisation aide les développeurs à packager leurs applications avec leurs dépendances, leur permettant d'être déployées facilement et rapidement. Kubernetes permet de fournir le contexte et les ressources nécessaires au bon fonctionnement des applications.

III.2. Autres solutions

Il existe d'autres solutions qui permettent l'orchestration des conteneurs.

Nous pouvons citer à titre d'exemple :

- **Docker Swarm**

Figure III.1 : Docker Swarm

C'est l'outil proposé par Docker pour assurer la gestion des clusters Docker. Comme Kubernetes, Docker Swarm fournit un control plane basé sur une API REST qui gère un certain nombre de ressources comme le cycle de vie des conteneurs, la création des services, des volumes, etc.

Pour plus de détails, voir la référence de l'API.

- **Nomad**

Figure III.2 : Nomad

Nomad est une solution open source proposée par Harshicorp, permettant la gestion et l'orchestration des clusters. Bien que le catalogue de fonctionnalités proposé soit moins complet que celui de Kubernetes, Nomad supporte aussi bien des applications conteneurisées que des applications virtualisées.

Nomad dispose d'une architecture client-serveur très simple, illustrée par la figure I.3.

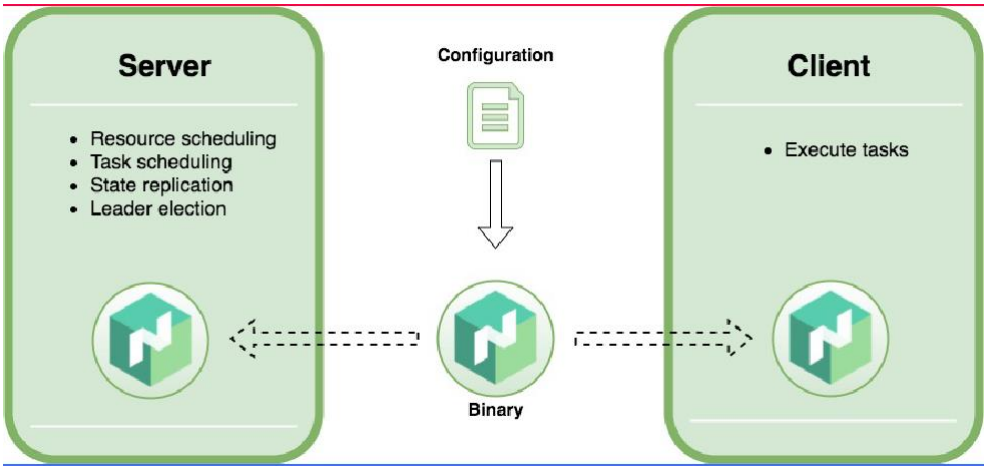

Figure III.3 : Architecrture de Nomad

Un serveur est responsable de :

- L'orchestration des tâches
- L'élection du leader pour la haute disponibilité du control plane
- La réplication des tâches

Et le client exécute les tâches planifiées.

- **Mesosphere DCOS**

Apache Mesos DCOS supporte plusieurs types de workload contrairement à Kubernetes qui supporte que les applications Cloud Native.

Figure III.4 : Mesosphere

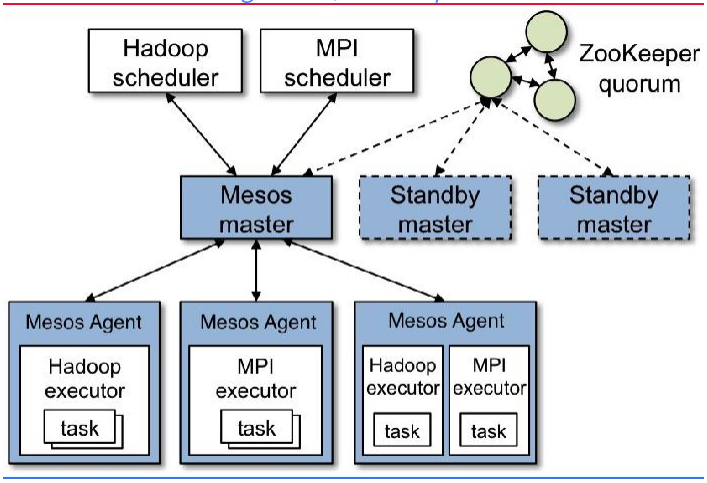

Figure III.5 : Architecture simplifiée de Mesosphere

Les composants majeurs d'un cluster Mesos DCOS sont :

- Nœuds Agents Mesos : ils permettent de lancer les tâches, tous les nœuds agents envoient au nœud master leurs ressources disponibles
- Nœud Master : le control plane du cluster
- Zookeeper : utilisé pour l'élection du master
- Frameworks : coopère avec le master pour planifier les tâches au niveau des nœuds agents

III.3. Héritage

Kubernetes se différencie des autres systèmes par son héritage. Il est inspiré du projet Borg, le système utilisé par Google en interne pour gérer leurs infrastructures. Borg a été le secret de Google pour une longue durée.

Le projet a été initié en 2003, par des développeurs chez Google. Aujourd'hui, toutes les applications Google s'exécutent dans Borg, y compris Google Cloud Plateforme.

L'architecture de Borg, illustrée dans la figure I.6, est très similaire à celle de Kubernetes.

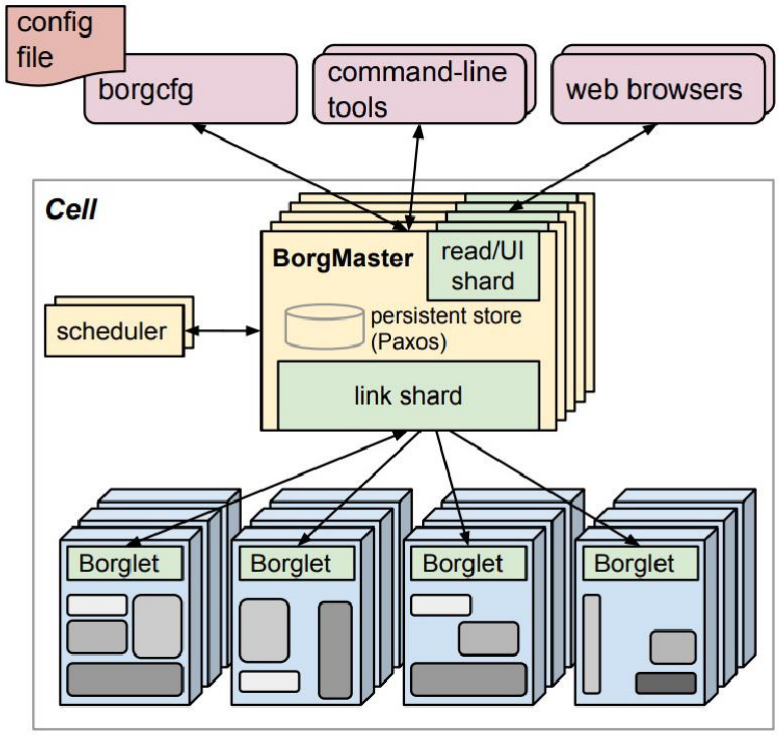

Figure III.6 : Architecture simplifiée de Borg

Une cell Borg est composée de :

- Un ou plusieurs masters qui regroupent les composants du control plane de la cell
- Une base de données pour stocker l'état de la cell
- Un ou plusieurs schedulers
- Des borglets qui sont des machines permettant d'exécuter des jobs

Un whitepaper Borg a été publié en 2015. Nous vous invitons fortement à le lire.

Kubernetes a bénéficié des bonnes idées de Borg. Voici quelques analogies entre les ressources Kubernetes et les ressources Borg.

Kubernetes	Borg	Détails
Pod	Alloc	La plus petite entité "schedulable" sur Kubernetes. groupe un ou plusieurs conteneurs qui partagent l'interface réseau.
Service	Service	Couche d'abstraction sur les pods qui fournit des services de répartition de charge
Labels	-	Les labels sont des métadonnées à base de clé valeurs qui peuvent se rajouter à tous les types de ressources sur Kubernetes. Borg ne fournit pas cette fonctionnalité.
IP-per-Pod	-	Dans Borg, toutes les tasks dans la même machine utilisent la même adresse IP. Grâce au SDON (Sofware Defined Overlay Network), Kubernetes est capable d'allouer une IP par pod et par service.

Tableau III.1 : Tableau comparatif entre Kubernetes et Borg

III.4. Cloud Native Computing Foundation

Figure III.7 : Cloud Native Computing Foundation

La CNCF (Cloud Native Computing Foundation) a repris la maintenance du projet Kubernetes en juillet 2015 après que Google l'ait initié en 2014. La CNCF est une fondation qui gère des projets open source, dédiés à rendre le "Cloud Native Computing" universel et durable.

Aujourd'hui la CNCF gère la gouvernance de plusieurs projets open source y compris Kubernetes. La CNCF fournit des certifications professionnelles pour les administrateurs et les développeurs :

- **CKAD (Certified Kubernetes Application Developer)** : Dédiée aux développeurs qui souhaitent certifier leurs connaissances de déploiement des applications sur Kubernetes
- **CKA (Certified Kubernetes Administrator)** : Dédiée aux administrateurs système qui souhaitent certifier leur savoir-faire autour de l'administration de Kubernetes

Les examens pour ces deux certifications sont totalement pratiques sous forme de lab, permettant de vérifier les connaissances pratiques du candidat.

III.5. Architecture

Kubernetes dispose d'une architecture client/serveur. Il est possible d'avoir une architecture avec plusieurs masters pour la haute disponibilité du control plane. Le serveur master regroupe plusieurs composants comme le kube-apiserver, une base de données etcd, le kube-controller-manager, le kube-scheduler, et un serveur DNS pour les services Kubernetes.

Les clients sont appelés minions et sont composés d'un agent kubelet, d'un agent kube-proxy, et d'un environnement d'exécution de conteneurs, comme par exemple Docker.

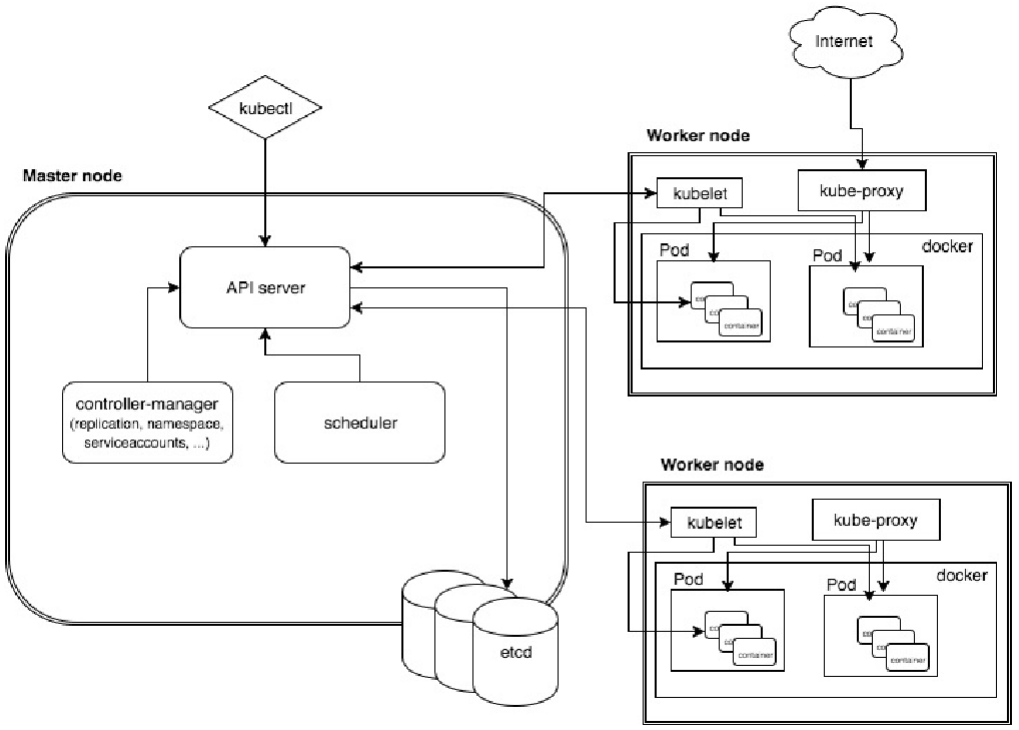

Figure III.8 : Architecture simplifiée de Kubernetes

Les nœuds master

Le nœud master est considéré comme le point d'entrée pour toutes les tâches d'administration.

Ci-dessous les composants principaux du serveur master :

- **etcd** : Base de données clé-valeur distribuée utilisée pour enregistrer les données du cluster Kubernetes (comme le nombre de pods, leur état, les namespaces, etc.). Elle est seulement accessible via l'API server par mesure de sécurité.
- **API Server** : Traite les requêtes REST/HTTP, les valide, et met à jour les objets dans la base etcd. La logique business du cluster est implémentée séparément pour des raisons de simplicité.
- **Scheduler** : Permet de placer les pods dans les minions en fonction de plusieurs critères, comme par exemple les ressources disponibles sur chaque nœud.
- **Controller Manager** : Regroupe plusieurs controllers. Ce daemon écoute en permanence l'API server afin de détecter les changements effectués par les administrateurs afin de modifier l'état du cluster. Par exemple le replication controller assure qu'un certain nombre de réplicas déclarés est toujours disponible dans le cluster.

Les nœuds minions ou workers

Les workloads lancés sur le cluster Kubernetes sont planifiés sur les nœuds minions. Ces nœuds contiennent tous les services nécessaires pour interconnecter les conteneurs et communiquer avec le nœud master.

Les différents services installés sur ces nœuds sont les suivants :

- **Docker** : S'exécute sur tous les nœuds minion, permet de télécharger les images et lancer les conteneurs.
- **kubelet** : Agent qui permet de communiquer avec les serveurs master. Récupère la configuration des pods depuis l'API server et assure que tous les conteneurs décrits sont toujours fonctionnels.
- **kube-proxy** : Considéré comme un proxy et un load balancer pour les services qui sont dans le nœud. Il permet de faire le routage des paquets TCP et UDP.

III.6. Conclusion

Kubernetes n'est pas le seul système permettant d'orchestrer des conteneurs. Dans ce chapitre, nous avons présenté d'autres solutions, y compris Docker Swarm, Mesosphere, et Nomad. Kubernetes est gouverné par l'organisation Cloud Native Computing Foundatio_, qui maintient le projet et propose des certifications techniques pour les administrateurs et les développeurs.

Nous avons également présenté une vision simplifiée de l'architecture Kubernetes. Dans le prochain chapitre, nous évoquerons les différentes manières de mettre en œuvre Kubernetes.

IV. Installation

IV.1. Introduction

Kubernetes est un système complexe à mettre en place et qui nécessite des connaissances diverses. Pour l'installer, il est nécessaire de définir les besoins de haute disponibilité. Les différentes configurations d'installation sont :

- **Mono Serveur** : Ce mode consiste à déployer tous les composants du cluster sur la même machine, physique ou virtuelle. Il permet la mise en place aisée d'un environnement de développement. **Attention, ce mode est déconseillé pour les environnements de production.**
- **Mono Master - Multi Minions** : Ce mode consiste à installer les composants master et la base de données etcd sur un même serveur, physique ou virtuel, et d'installer les composants minion sur des serveurs distincts.
- **Multi Master - Multi Minions** : Ce mode consiste à mettre en place un load balancer devant l'API server. Le scheduler et le controller manager choisiront un leader configuré via des flags. La base de données etcd est installée sur un seul serveur.
- **Multi Master - Multi Minions - HA etcd** : Ce mode est identique au précédent, à la différence de la base de données etcd, qui est en mode cluster. Il consiste donc à séparer les processus etcd des serveurs master.

Nous parcourrons par la suite les différents outils permettant d'implémenter ces modes.

IV.2. Minikube

Minikube est un outil développé en Go qui permet l'exécution locale d'un cluster Kubernetes. Minikube s'exécute en mode déploiement mono serveur dans une machine virtuelle. Minikube est l'outil idéal pour entrer dans le monde de Kubernetes et s'initier.

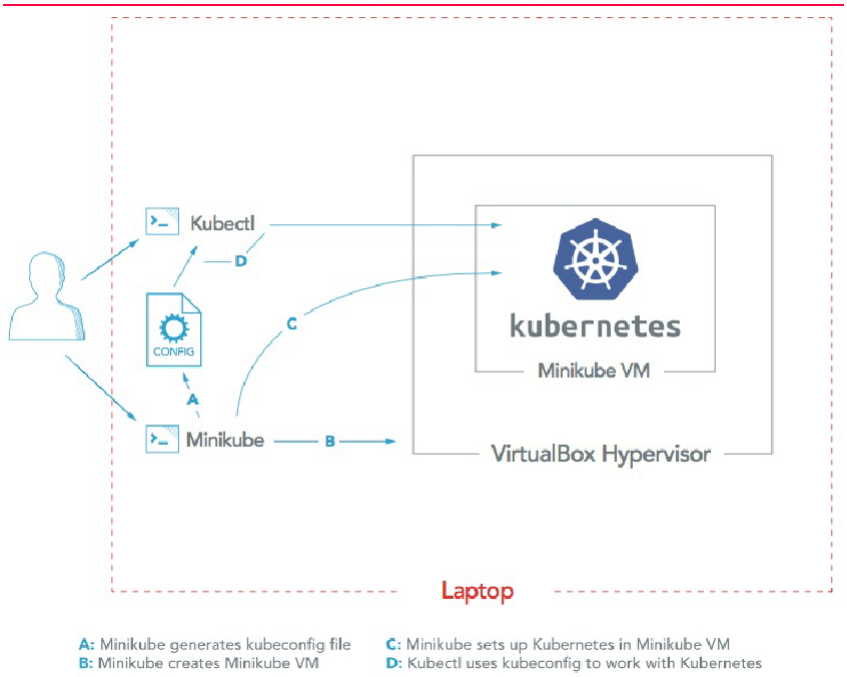

Figure IV .1 : Architecture de Minikube

Fonctionnalités

Minikube inclut les fonctionnalités suivantes :

- **DNS** : Un serveur de nom de domaine interne pour le cluster
- **Ingress** : Un reverse proxy pour accéder aux applications déployées dans Minikube
- **ConfigMaps** et **Secrets** : Permet de charger la configuration d'une application
- **Dashboard** : Tableau de bord pour administrer Kubernetes
- **Docker**, **rkt**, ou **CRI-O** : Le runtime qui lance les conteneurs

Nous détaillerons ces éléments dans les prochains chapitres.

Installation et lancement

Minikube supporte un certain nombre d'hyperviseurs, notamment VirtualBox, VMware Fusion, KVM, Hyperkit, xhyve (déprécié). L'installation de Minikube nécessite la présence d'un de ces hyperviseurs sur votre machine. Il s'agit ensuite de lancer cette commande :

```
$ ➜ curl -Lo minikube
https://github.com/kubernetes/minikube/releases/download/v0.27.0/minikube-linux-
amd64 && chmod +x minikube && sudo mv minikube /usr/local/bin/
```

Après l'installation, exécuter Minikube avec la commande suivante :

```
$ ➜ minikube start --vm-driver=<votre-driver>

Starting local Kubernetes v1.10.0 cluster...
Starting VM...
Getting VM IP address...
Moving files into cluster...
Downloading kubeadm v1.10.0
Downloading kubelet v1.10.0
Finished Downloading kubelet v1.10.0
Finished Downloading kubeadm v1.10.0
Setting up certs...
Connecting to cluster...
Setting up kubeconfig...
Starting cluster components...
Kubectl is now configured to use the cluster.
Loading cached images from config file.
```

À ce stade, Minikube télécharge l'ISO de la machine virtuelle, copie les fichiers de configuration nécessaires dans la VM, télécharge kubelet et kubeadm, initialise les certificats SSL pour la communication inter- composants, et configure la ligne de commande locale kubectl pour pouvoir interagir avec Minikube.

Une fois l'opération terminée, Minikube est bien configuré sur la machine et le cluster est disponible pour des tests.

Pour vérifier le bon fonctionnement, il est possible de lancer ces quelques commandes : `kubectl get nodes` qui permet de récupérer les nœuds du cluster, `kubectl get cs` qui affiche l'état des composants tels que le scheduler, le controller manager, et la base de donnée etcd, et enfin `kubectl cluster-info` qui affiche des informations d'accessibilité sur le cluster.

```
$ ➜ kubectl get nodes && kubectl get cs && kubectl cluster-info
NAME       STATUS    ROLES     AGE       VERSION
minikube   Ready     master    22d       v1.10.0

NAME                 STATUS    MESSAGE              ERROR
scheduler            Healthy   ok
controller-manager   Healthy   ok
etcd-0               Healthy   {"health": "true"}

Kubernetes master is running at https://192.168.99.100:8443
KubeDNS is running at https://192.168.99.100:8443/api/v1/namespaces/kube-
system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

Pour plus d'informations sur Minikube, vous pouvez consulter la documentation officielle sur kubernetes.io.

IV.3. kubeadm

kubeadm est un outil, conçu par Lucas Kaldstrom, un des contributeurs du projet Kubernetes, qui permet de créer facilement des clusters Kubernetes. L'outil permet d'installer les composants du nœud master en premier lieu, générer les certificats et les clés SSL, et lance tous les composants dans des conteneurs. L'outil permet également de rattacher des nœuds minion au master en utilisant des jetons.

Pour en savoir plus sur kubeadm, vous pouvez consulter la documentation officielle de l'outil.

IV.4. Kubernetes The Hard Way

Kubernetes The Hard Way est un tutoriel créé par Kelsey Hightower qui illustre étape par étape la mise en place des composants du control plane du projet Kubernetes.

Le tutoriel couvre :

1. Les prérequis nécessaires
2. L'installation des outils client
3. L'approvisionnement des machines virtuelles sur "Google Cloud Platform"
4. L'approvisionnement des certificats TLS
5. La génération des fichiers de configuration pour l'authentification inter-composants
6. La génération de la configuration de chiffrement des données du cluster
7. La mise en place du cluster etcd
8. La mise en place du control plane
9. La mise en place des minions
10. L'accès distant
11. La mise en place du réseau
12. Le déploiement du DNS

Ces étapes peuvent changer en fonction des nouveautés de Kubernetes, le dépôt est activement maintenu par Kelsey et les différents contributeurs.

IV.5. Autres mécanismes d'installation

Il existe plusieurs autres mécanismes d'installation de Kubernetes, en fonction de l'environnement qui va l'accueillir. En voici certains :

Solution	Environnement	Description
Ubuntu on LXD	Machines locales	Environnement de développement sur Ubuntu LXD
GKE (Google Kubernetes Engine)	Cloud	Déploiement managé par Google
EKS (Amazon Elastic Kubernetes Service)	Cloud	Déploiement managé par AWS
AKS (Azure Kubernetes Service)	Cloud	Déploiement managé par Microsoft Azure
Openshift Dedicated	Cloud	Déploiement géré par RedHat OpenShift
IBM Cloud Kubernetes Service	Cloud	Déploiement managé par IBM Cloud
KubeSpray	Cloud + On-Premise	Déploiement sur le Cloud ou On-Premise

Cette liste n'est pas exhaustive, il existe bien évidemment d'autres implémentations de Kubernetes.

IV.6. Conclusion

Dans ce chapitre, nous avons présenté les différents moyens d'installation des clusters Kubernetes. Il existe cependant d'autres outils et produits qui n'ont pas été évoqués.

Nous avons détaillé l'usage de l'outil Minikube qui permet de lancer tous les composants en local afin de disposer d'un environnement de développement. Nous avons également évoqué les principaux outils et produits qui existent pour la mise en place de Kubernetes dans différents environnements.

Le prochain chapitre détaillera les différents concepts de Kubernetes.

V. Concepts de base

V.1. Introduction

Kubernetes est un projet assez complexe à mettre en place comme nous l'avons vu dans le précédent chapitre. Kubernetes repose sur quelques concepts de base qui lui sont propres.

Toutes les requêtes REST passent par le composant API server et sont ensuite traitées par les différents contrôleurs. Ces requêtes sont en réalité des appels HTTP sur des ressources REST fournies par Kubernetes.

Parmi ces ressources, nous pouvons citer les pods, les services, les ReplicaSets, les deployments ainsi que d'autres ressources que nous allons détailler dans ce chapitre.

V.2. Pods

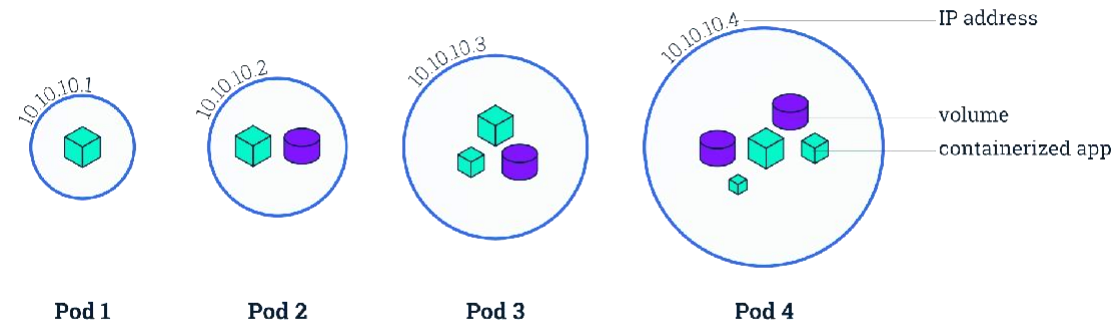

Figure V.1 : Pods

Le pod est la plus petite ressource orchestrable par Kubernetes, il représente un ou plusieurs conteneurs qui partagent le même namespace réseau. L'ensemble des conteneurs dans le même pod peut donc communiquer via localhost ou 127.0.0.1.

Les conteneurs dans le même pod ont le même cycle de vie. Par exemple lorsqu'un pod s'arrête, tous les conteneurs dans ce pod s'arrêteront.

Kubernetes supporte plusieurs runtime : Docker, rkt, containerd, CRI-O, etc.

Durabilité

Les pods ne doivent pas être traités comme des entités durables. Ils ne sont pas des ressources protégées contre les erreurs de scheduling, les pannes de machines ou d'autres types de défaillance. En général, les utilisateurs ne sont pas censés créer des pods directement.

La création des pods se fait via des contrôleurs spécifiques comme les replication controllers ou les deployments. Nous détaillerons ces ressources par la suite dans ce chapitre.

Exemple

Voici un exemple d'une spécification d'un pod avec un nom `demo` qui contient un conteneur nommé `demo-container` avec une limitation de ressources sur la mémoire et le CPU.

```
apiVersion: v1
kind: Pod
metadata:
 name: demo
spec:
containers:
 - name: demo-container
  image: demo:1.0 resources:
    limits:
     memory: "200Mi"
    requests: memory:
    "100Mi"
```

Une fois envoyée à l'API server avec la commande `kubectl`, cette configuration permettrra de créer un pod dans le cluster Kubernetes.

V.3. Services

Les pods sont des ressources temporaires, ils peuvent à tout moment être perdus, et c'est bien un comportement désiré par les développeurs de Kubernetes, d'où l'utilité des services.

Un service est une abstraction qui définit un ensemble logique de pods et une politique d'accès. Les pods reliés à un service sont définis via un sélecteur de labels. Par exemple, dans un scénario où nous aurions 3 réplicas de pods backend avec 3 adresses IPs différentes, l'accès à ces pods via d'autres pods frontend se ferait via la VIP (Virtual IP) d'un service.

Les services permettent aussi de distribuer la charge du trafic sur ses différents pods.

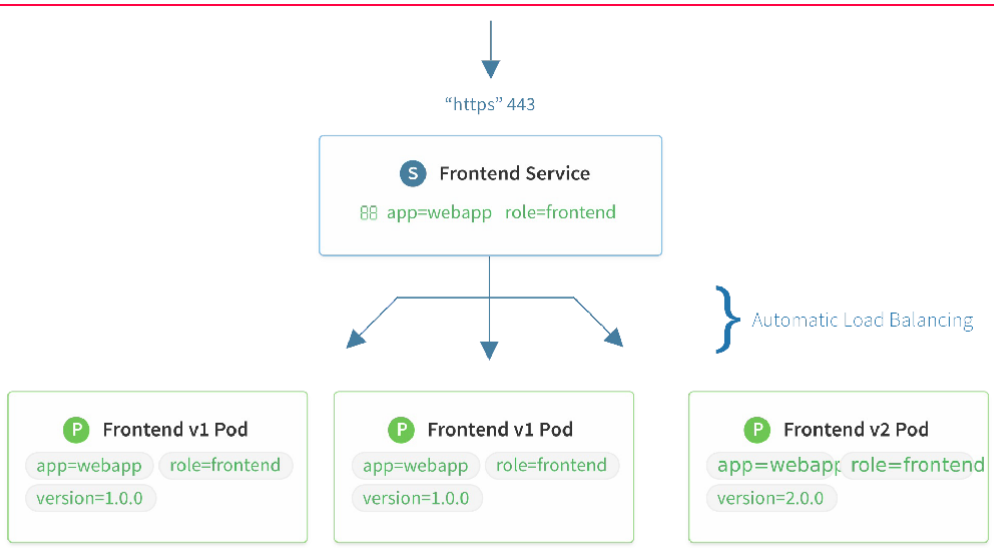

Figure V.2 : Services

La figure III.2 illustre un service frontend qui se rattache à des pods labélisés app=webapp et `role=frontend`.

Il existe 3 types de service :

1. **ClusterIP** : C'est le type par défaut, il fournit un accès interne au cluster.
2. **NodePort** : Permet d'ouvrir un port entre 3000 et 32000 sur tous les nœuds minions pour accéder au pod. N'est pas recommandé pour l'utilisation dans des environnements de production.
3. **LoadBalancer** : Seulement implémenté dans les clusters déployés sur des cloud providers, fournit un load balancer (ELB pour AWS par exemple) attaché à tous les minions et qui redirige le trafic vers les pods sélectionnés.

Exemple

Comme toutes les ressources Kubernetes, les services sont configurés à l'aide d'un fichier YAML.

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector: app:
  MyApp ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

Ce service portant le nom `my-service` sélectionne tous les pods qui ont un label `app=MyApp` et expose le port 80 en TCP.

V.4. ReplicaSets

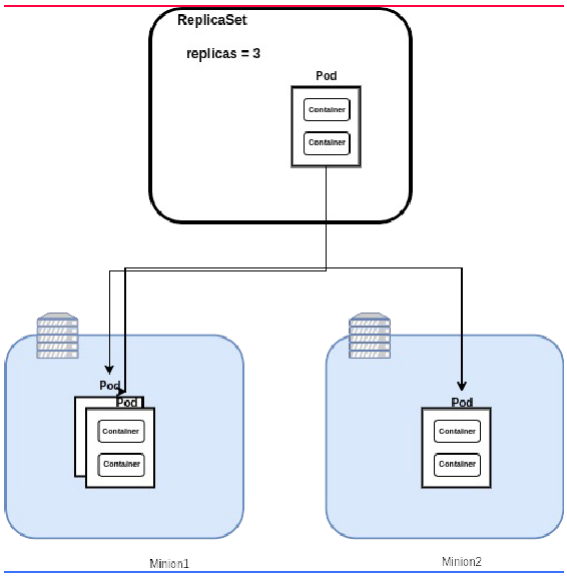

Figure V.3 : ReplicaSets

Un ReplicaSet est une ressource Kubernetes permettant d'assurer la présence d'un certain nombre de réplicas sur le cluster. Les ReplicaSets sont généralement utilisés à travers les deployments.

Exemple

L'exemple ci-dessous spécifie un ReplicaSet `frontend` gérant 3 réplicas de pods labélisés `tier=frontend` ou bien `tier In [frontend]`. Le ReplicaSet utilisera la spécification du pod contenue dans la section `templates`.

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: frontend
 labels:
  app: demo
  tier: frontend
spec:
 replicas: 3
 selector:
 matchLabels:
    tier: frontend
 template:
 metadata:
    labels:
     app: demo
     tier: frontend
  spec:
    containers:
    - name: demo
     image: demo:1.3
     resources:
     requests:
        cpu: 100m
        memory: 100Mi
     ports:
     - containerPort: 80
```

Il est recommandé d'utiliser les ReplicaSets dans le cadre de deployments. Nous détaillerons cette ressource dans la prochaine section.

V.5. Deployments

Le deployment est une ressource Kubernetes permettant de manipuler des ReplicaSets. Parmi

les actions que les deployments peuvent effectuer :

- **Le rollout**: Mise à jour de la spécification du deployment, entraînant la création des pods en arrière-plan
- **Le rollback**: Permet de revenir à une ancienne version des ReplicaSets
- **La scalabilité horizontale**: Permet de mettre en échelle l'application horizontalement

La figure III.4 illustre 3 versions d'une application gérée par un deployment.

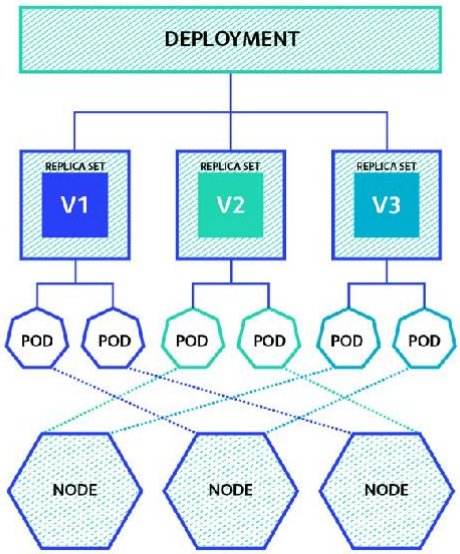

Figure V.4 : Deployments

Exemple

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: nginx-deployment
 labels:
  app: nginx
spec:
 replicas: 3
 selector:
 matchLabels:
    app: nginx
 template:
 metadata:
    labels:
     app: nginx
  spec:
  containers:
   - name: nginx
     image: nginx:1.7.9
     ports:
     - containerPort: 80
```

Cette spécification permet de créer un deployment `nginx-deployment` qui gère 3 réplicas de pod Nginx.

V.6. Autres ressources

Kubernetes fournit d'autres ressources qui permettent de déployer et de configurer des applications Cloud Native.

StatefulSets

Comme les deployments, les StatefulSets anciennement appelés PetSets sont des ressources Kubernetes qui donnent la possibilité d'installer des applications stateful, comme les bases de données sur Kubernetes. Les pods rattachés à un StatefulSet auront des noms d'hôte fixes, ce qui permettra de conserver un état pour ces pods.

Les StatefulSets sont généralement utilisés avec des services headless, ce qui permet de s'adresser directement à un pod avec un nom de domaine unique. Exemple : `database-pod-1.service-db.database-namespace.svc.cluster.local`. Cette combinaison permet de mettre en place, par exemple, des clusters de bases de données ou des applications dont on souhaite conserver l'état.

DaemonSets

Cette ressource Kubernetes permet d'assurer qu'un pod est présent sur chaque nœud du cluster. Il est alors possible grâce à elle, par exemple, de mettre en place des plugins réseaux SDN fournissant une interconnexion inter-pod. Les DaemonSets peuvent être utilisés aussi pour collecter les logs des conteneurs du cluster.

Persistent Volumes et Persistent Volume Claims

Les persistent volumes sont une abstraction sur les différents types de volume qui peuvent être utilisés dans un cluster. Kubernetes fournit deux ressources responsables de l'instanciation des volumes : Persistent Volume et Persistent Volume Claim.

Les Persistent Volumes (PV) sont un composant de stockage provisionné par un administrateur ou automatiquement via des StorageClass. Ce sont des plugins qui ont des cycles de vie indépendants des autres ressources.

Les Persistent Volume Claims permettent à l'utilisateur de consommer des ressources de stockage au niveau de conteneurs. C'est la ressource qui va être montée dans les pods.

ConfigMaps et Secrets

La configuration des applications Twelve-Factor est injectée au moment du runtime. Kubernetes supporte cette méthodologie et met à disposition deux ressources :

- **ConfigMaps** : Permettent de créer des objets Kubernetes pouvant être utilisés dans un pod au moment du runtime. Tout type de fichier de configuration est supporté par Kubernetes.
- **Secrets** : Permettent de créer des secrets dans Kubernetes. Par exemple, des mots de passe qui seraient encodés en Base64 et chiffrés dans la base de données etcd.

V.7. Conclusion

Kubernetes dispose de plusieurs ressources afin de déployer et mettre en échelle une application Cloud Native. Dans ce chapitre, nous avons évoqué plusieurs ressources, tels que les pods, les services, les ReplicaSets, et les deployments pour gérer la configuration et les volumes au niveau des pods. Dans le prochain chapitre nous aborderons les ressources Ingress qui permettent de fournir la fonctionnalité reverse proxy pour les services Kubernetes.

VI. Ingress Controllers

VI.1. Introduction

Comme précisé dans les chapitres précédents, Kubernetes fournit 3 types de services, y compris le service de type LoadBalancer.

Ce dernier crée un LoadBalancer (ELB sur AWS par exemple) sur le cloud provider, qui sera rattaché à tous les nœuds worker sur un port spécifique.

L'évolution du prix des load balancers sur AWS est linéaire, comme l'illustre la figure IV.1. D'où l'utilité des Ingress controllers.

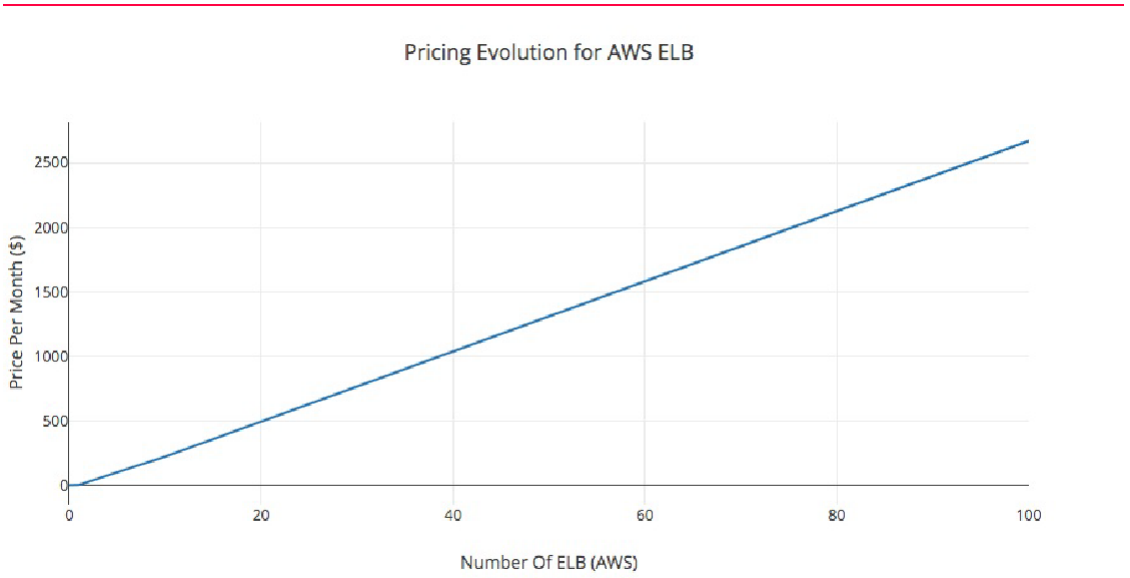

Figure VI.1 : L'évolution du prix ELB sur AWS

Un Ingress controller est un composant déployé dans le cluster Kubernetes. Il permet de détecter les requêtes sur l'API server qui ont un `kind: Ingress` et de configurer un reverse proxy pour ajouter la règle définie dans l'objet Ingress.

Dans ce contexte, nous créons un seul service, de type LoadBalancer, qui sera le service Ingress controller.

VI.2. Custom Resource Definition

Les Custom Resource Definitions (CRD) sont des objets qui étendent l'API de Kubernetes avec de nouvelles ressources. Ces objets permettent de définir de nouveaux types dans un namespace ou dans le cluster.

D'une certaine manière, le serveur API Kubernetes peut être considéré comme une base de données. Il permet le stockage et la récupération des ressources Kubernetes.

Si l'API server est une base de données, les CRD sont des définitions de schéma fournies par l'utilisateur. Une fois que vous avez une méthode pour définir votre propre schéma, vous pouvez lui appliquer les mêmes mécanismes de contrôleur que ceux utilisés pour les ressources natives.

Comme indiqué ci-dessus, les CRD ne remplacent pas complètement les ressources Kubernetes natives, ils ne le feront probablement jamais. Il est difficile, voire impossible, de concevoir un moyen aussi souple que l'écriture de code arbitraire pour qualifier de manière déclarative des ressources.

Exemple

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition metadata:
 name: database.stable.example.com
spec:
 group: stable.example.com
 version: v1
 scope: Namespaced names:
  plural: databases
  singular: database
  kind: DataBase
  shortNames:
  - db
```

Dans cet exemple, nous définissons un nouveau type `DataBase` dans le groupe `stable.example.com`.

VI.3. Ingress API Resource

Ingress est un objet de l'API server qui gère les accès externes aux services Kubernetes. Les Ingress peuvent fournir la terminaison SSL, la répartition de charge, ainsi que des virtualhosts basés sur des noms de domaine.

De base, les IPs des services et des pods sont seulement routables au sein-même du cluster. Un Ingress est un ensemble de règles permettant aux connexions entrantes d'atteindre les services du cluster.

La figure IV.2 ci-dessous illustre la façon dont l'Ingress controller redirige le trafic au sein du cluster Kubernetes.

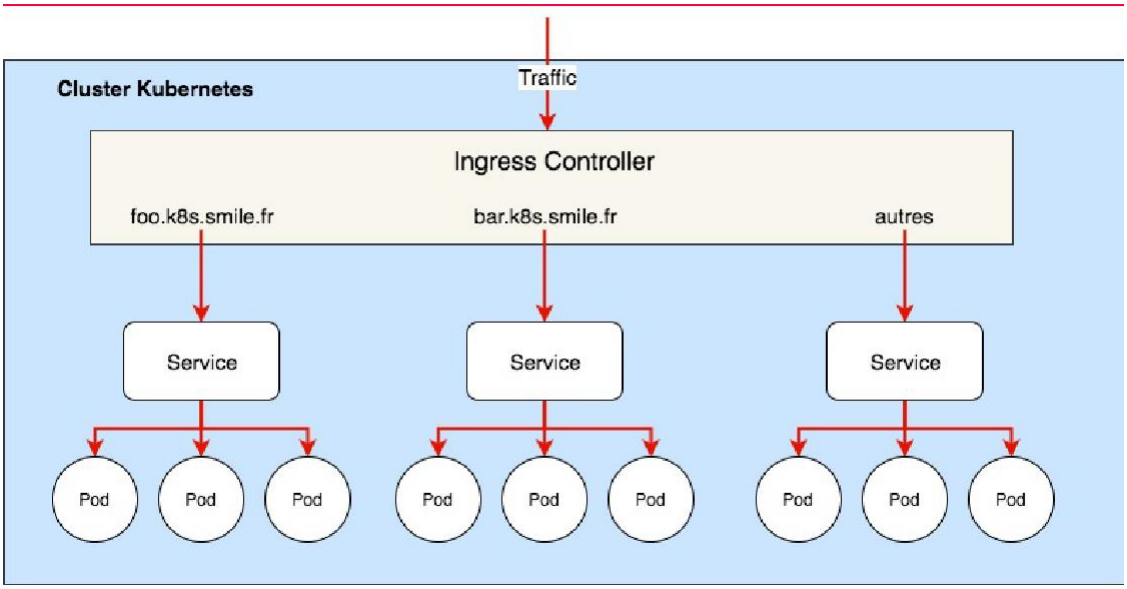

Figure VI.2 : Ingress Controller

Pour créer une ressource Ingress, l'utilisateur du cluster POST une spécification d'une ressource Ingress à l'API server, comme illustré par l'exemple ci-dessous. Un contrôleur d'entrée est chargé de configurer cette nouvelle entrée, généralement avec un équilibreur de charge.

Exemple

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
 name: kubernetes-dashboard-proxy
 namespace: kube-system
spec:
 rules:
 -host: kubernetes-dashboard.k8s.lab-si.eqx.intranet
  http:
   paths:
   - backend:
     serviceName: kubernetes-dashboard-proxy servicePort:
     3000
    path: /
 tls:
 - hosts:
  - kubernetes-dashboard.k8s.lab-si.eqx.intranet
  secretName: star.k8s.lab-si.eqx.intranet
```

Cet exemple illustre la création d'une nouvelle ressource Ingress, basée sur l'enregistrement kubernetes-dashboard.k8s.lab-si.eqx.intranet, qui permet de rediriger tout le trafic vers le service kubernetes-dashboard-proxy sur le port 3000, avec une terminaison TLS.

VI.4. Ingress Controller

Pour que la ressource Ingress s'applique sur le cluster, l'Ingress controller doit être préalablement déployé sur le cluster.

L'Ingress controller est une application qui s'exécute dans le cluster Kubernetes et qui configure un load balancer HTTP, par exemple Nginx ou HAProxy, en fonction des ressources Ingress fournies.

Le load balancer peut être un logiciel et exécuté au sein du cluster, il peut être matériel, ou il peut encore s'agir d'un service cloud.

La figure IV.3 illustre en détails les communications et les flux de l'Ingress controller.

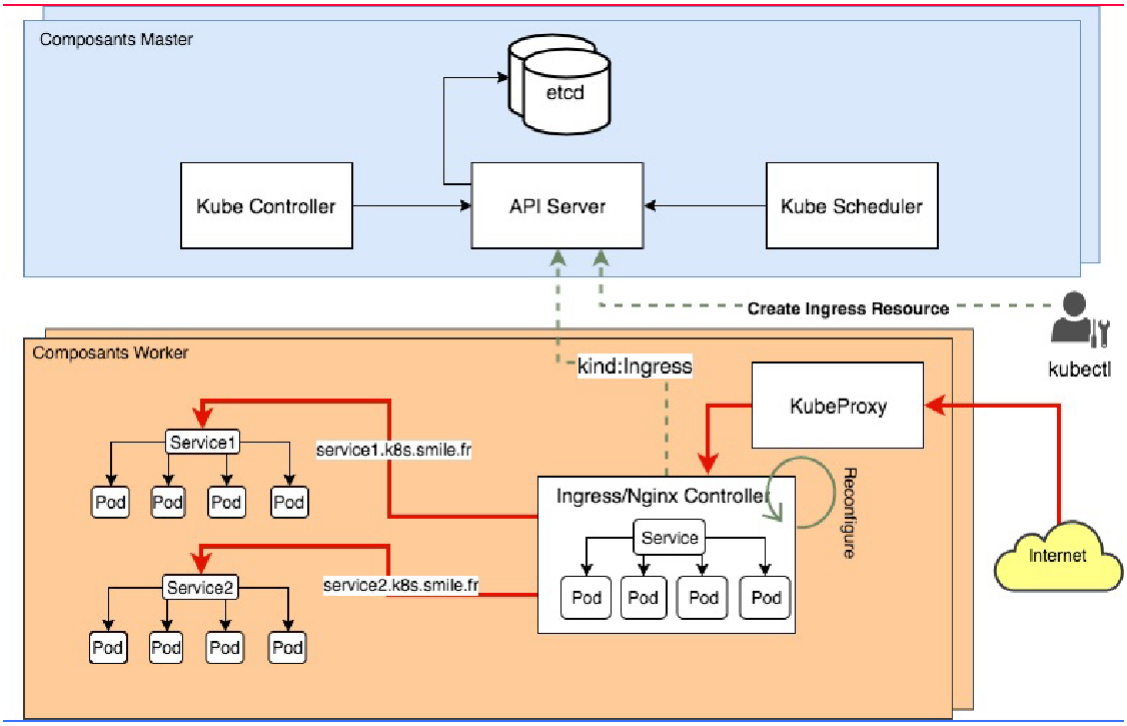

Figure VI.3 : Ingress Controller Details

1. Un administrateur crée une ressource Ingress en utilisant la commande `kubectl` qui communique en HTTPS avec l'API server
2. L'Ingress controller détecte qu'une nouvelle demande de création de ressource de type Ingress a été envoyée à l'API server
3. L'Ingress controller reconfigure le load balancer afin d'ajouter le virtualhost pour cet Ingress
4. Le trafic sera redirigé par le kube-proxy vers le service Kubernetes adéquat à travers l'Ingress controller

VI.5. Conclusion

Dans ce chapitre nous avons traité de la ressource Ingress sur Kubernetes, qui permet d'accéder aux services créés dans un cluster Kubernetes. Sans Ingress controller, les ressources Ingress ne valent rien dans le cluster. Nous avons également présenté en détails le fonctionnement de l'Ingress controller, qui permet d'intercepter les requêtes Ingress sur l'API server et de reconfigurer automatiquement un load balancer pour ajouter de nouvelles règles comme spécifié.

VII. Monitoring

VII.1. Introduction

Le monitoring de l'état actuel d'une application est l'un des moyens les plus efficaces pour anticiper les problèmes et détecter les goulots d'étranglement dans un environnement de production. C'est également l'un des plus grands défis de tous les éditeurs logiciel.

L'adoption croissante des microservices complique davantage le logging et le monitoring. Cette complexité est due aux communications entre un grand nombre d'applications, réparties et diversifiées.

Généralement, il existe plusieurs métriques Kubernetes qui sont à surveiller. Ces métriques peuvent être séparées en deux composants principaux :

- Le monitoring du cluster lui-même
- Le monitoring des pods exécutés sur le cluster

VII.2. Monitoring du cluster

L'objectif est de surveiller la santé de l'ensemble du cluster Kubernetes. Un administrateur souhaitera savoir si tous les nœuds du cluster fonctionnent correctement et à quelle capacité, ainsi que le nombre d'applications en cours d'exécution sur chaque nœud et l'utilisation des ressources de l'ensemble du cluster.

Dans ce contexte, nous pouvons distinguer certains paramètres mesurables :

- L'utilisation des ressources du nœud : Il existe de nombreuses mesures dans ce contexte, toutes liées à l'utilisation des ressources. La bande passante réseau, l'utilisation du disque, l'utilisation du processeur et de la mémoire en sont des exemples. A l'aide de ces mesures, il est possible de déterminer la nécessité d'augmenter ou de réduire le nombre et la taille des nœuds du cluster.
- Le nombre de nœuds : Le nombre de nœuds disponibles est une métrique importante à superviser. Cela vous permet de déterminer ce pour quoi vous payez (si l'infrastructure est hébergée dans le cloud) et de découvrir à quoi sert le cluster.
- Le nombre de pods actifs : Le nombre de pods en cours d'exécution indiquera si le nombre de nœuds disponibles est suffisant et s'ils seront capables de gérer la totalité de la charge de travail en cas de défaillance d'un nœud.

VII.3. Monitoring des pods

Le fait de contrôler un pod peut être détaillé en trois catégories :

Métriques de Kubernetes

Nous pouvons surveiller la manière dont un pod et son déploiement sont gérés par l'orchestrateur. Les informations suivantes peuvent être contrôlées :

- Le nombre d'instances actuellement disponibles sur un pod ainsi que le nombre d'instances qui était attendu (si le nombre est faible, le cluster est peut-être à court de ressources)
- Le déroulement du déploiement en cours (le nombre d'instances changées d'une ancienne version à une nouvelle)
- Des bilans de santé (health checks)

Métriques des conteneurs

Les mesures de conteneurs sont disponibles principalement par le biais de cAdvisor et exposées par Heapster, qui interroge chaque nœud sur les conteneurs en cours d'exécution. Dans ce cas, les métriques telles que l'utilisation du processeur, du réseau, et de la mémoire par rapport au maximum autorisé sont les points d'attention.

Métriques des applications

Ces métriques sont fournies par l'application elle-même et sont liées aux règles métier qu'elle adresse. Par exemple, une application de base de données exposerait probablement des mesures liées à l'état des index et des statistiques concernant les tables et les relations. De la même manière, une application de commerce électronique exposerait des données concernant le nombre d'utilisateurs en ligne et le gain généré par le logiciel au cours de la dernière heure.

VII.4. Méthodes de monitoring

Il existe deux approches principales pour la collecte des métriques du cluster et leur export vers un point de terminaison externe. Généralement, la collecte des mesures doit être gérée de manière cohérente sur l'ensemble du cluster. Le système doit gérer la collecte de métriques de la même manière et avec la même fiabilité, peu importe la localisation des nœuds.

DaemonSets

Une approche de monitoring de tous les nœuds du cluster consiste à créer un type spécial de pod Kubernetes appelé DaemonSet. Kubernetes veille à ce que chaque nœud créé dispose d'une copie du pod DaemonSet. Ceci permet à un déploiement de surveiller chaque machine du cluster. Lorsque les nœuds sont détruits, le pod est également terminé. De nombreuses solutions de monitoring utilisent la structure DaemonSet pour déployer un agent sur chaque nœud de cluster. Dans ce cas, il n'y a pas de solution générale, chaque outil aura son propre logiciel de monitoring de cluster.

Heapster

Heapster, en revanche, est une plateforme uniforme adoptée par Kubernetes pour envoyer des métriques de monitoring à un système. Heapster est un pont entre un cluster et un stockage conçu pour collecter des métriques. Les stockages pris en charge sont : ElasticSearch, GCM, Stackdriver, Hawkular, InfluxDB, Metric (memory), Kafka, OpenTSDB, Riemann, Graphite, Wavefront, Librato, Honeycomb, StatsD.

Contrairement aux DaemonSets, Heapster agit comme un pod normal et découvre chaque nœud du cluster via l'API Kubernetes. En utilisant Kubelet et cAdvisor (un outil de monitoring de conteneur qui collecte des métriques pour chaque conteneur en cours d'exécution), le pont peut stocker toutes les informations pertinentes sur le cluster et ses conteneurs.

Un cluster peut comporter des milliers de nœuds et un nombre encore plus important de pods. Il est pratiquement impossible d'observer chacun d'entre eux. De ce fait, il est important de créer plusieurs étiquettes pour chaque déploiement. Par exemple, la création d'une étiquette pour les pods intensifs en base de données permettra à l'opérateur d'identifier de potentiels problèmes au niveau du service de base de données.

VII.5. Opérateur Prometheus

Les opérateurs sont des applications (pods) spécifiques à Kubernetes qui configurent, gèrent, et optimisent automatiquement les autres déploiements. Ils sont implémentés en tant que contrôleurs personnalisés. Un opérateur Kubernetes englobe le savoir-faire du déploiement et de la mise à l'échelle d'une application, et exécute directement les décisions de l'algorithme communiquant avec l'API. Les opérateurs Kubernetes utilisent la ressource CRD (Custom Resource Definition) pour créer des entités et des objets spécifiques au contexte auquel on accédera et ce comme toute autre ressource de l'API Kubernetes.

L'opérateur Prometheus Kubernetes fournit des définitions de monitoring pour les services Kubernetes ainsi que le déploiement et la gestion d'instances Prometheus. L'opérateur agit sur les définitions des ressources personnalisées (CRD) suivantes :

- Prometheus : Définit le déploiement souhaité de Prometheus. L'opérateur s'assure à tout moment qu'un déploiement en cours d'exécution correspond à la définition de ressources.
- ServiceMonitor : Spécifie de manière déclarative la surveillance des groupes de service. L'opérateur génère automatiquement la configuration de scrap de Prometheus en fonction de la définition.
- PrometheusRule : Custom Resource qui définit les règles Prometheus souhaitées, pouvant être chargée par une instance de Prometheus. Cette ressource spécifie des règles d'alerte et d'enregistrement Prometheus
- AlertManager : Définit le déploiement souhaité du composant AlertManager. L'opérateur s'assure en permanence qu'un déploiement correspondant à la définition de ressources est en cours d'exécution.

Afin de comprendre le problème, nous devons d'abord comprendre le fonctionnement de Prometheus Operator. La figure VII.1 définit les composants de l'opérateur Prometheus.

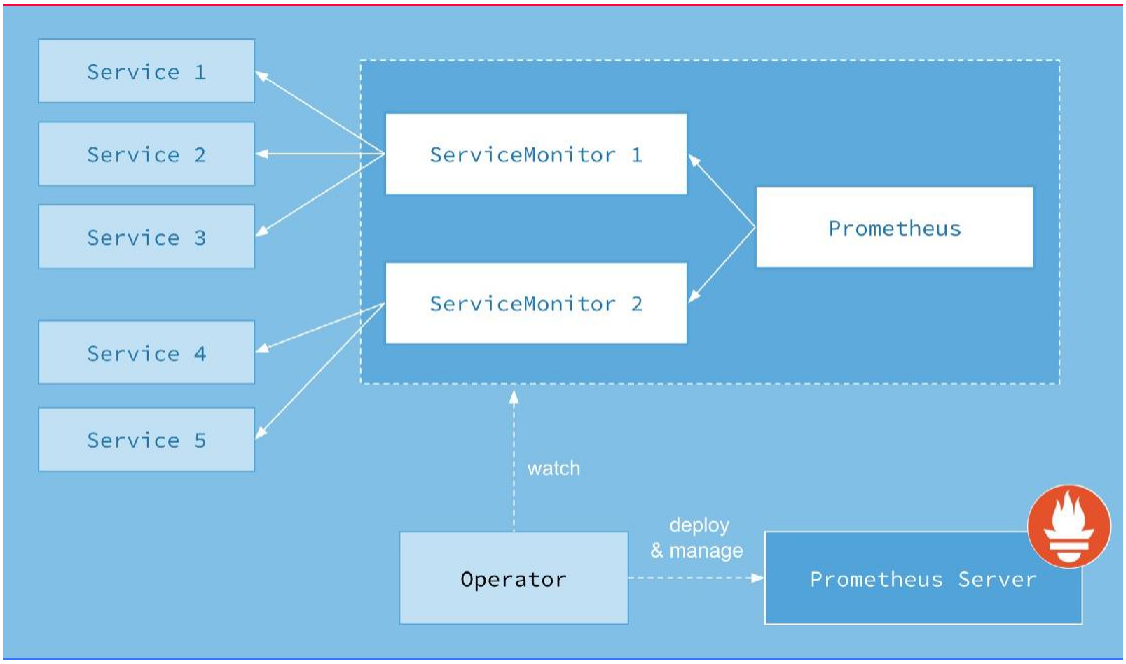

Figure VII.1 : L'architecture sipmlifiée de l'opérateur Prometheus

Lorsqu'une nouvelle version de votre service est mise à jour, un nouveau pod est créé. Prometheus surveille l'API Kubernetes. Ainsi, lorsqu'il détecte ce type de modification, il créé un nouvel ensemble de configuration pour ce nouveau service (pod).

ServiceMonitor

L'opérateur Prometheus utilise une CRD, nommée ServiceMonitor, qui permet de décrire l'ensemble des cibles à surveiller par Prometheus.

Voici un exemple d'un ServiceMonitor :

```
apiVersion: monitoring.coreos.com/v1alpha1 kind:
ServiceMonitor
metadata:
  name: api
  labels:
    prometheus: kube-prometheus
    tier: api
spec: selector:
    matchLabels:
      tier: api
  endpoints:
  - port: metric
    interval: 10s
```

Cette spécification de ServiceMonitor permet de configurer le serveur Prometheus dynamiquement afin qu'il surveille les services labélisés `tier: api` sur le port `metric`.

Déploiement

Pour déployer l'opérateur Prometheus ainsi que tous les composants nécessaires à la supervision du cluster et des applications qu'il héberge, nous allons utiliser helm qui est le gestionnaire de paquets officiel de Kubernetes.

1. Ajout du repository Helm de CoreOS

```
$ helm repo add coreos https://s3-eu-west-1.amazonaws.com/coreos-charts/stable/
"coreos" has been added to your repositories
```

2. Installation du composant prometheus-operator

```
$ helm install coreos/prometheus-operator --name prometheus-operator --namespace monitoring
```

3. L'opérateur Prometheus et ses CRD sont installés, nous allons maintenant déployer Prometheus, AlertManager, et Grafana

```
$ helm install coreos/kube-prometheus --name kube-prometheus --namespace monitoring
```

4. Vérification de l'installation, et de l'état des pods qui doit être Running

```
$ kubectl get pods -n monitoring
```

Cette dernière commande permet de lister les pods installés dans le namespace monitoring.

Les deux charts helm deployées (coreos/kube-prometheus et coreos/prometheus-operator) permettent d'installer :

- Un DaemonSet node-exporter qui permet de fournir des métriques relatives à l'utilisation des ressources par nœud
- Une instance Grafana pour visualiser les métriques
- Une instance Prometheus
- Un opérateur Prometheus qui permet de configurer le serveur Prometheus avec de nouveaux targets

Nous avons créé 3 Ingress (voir le chapitre Ingress Controller) afin de pouvoir accéder aux interfaces :

1. alert.k8s.lab-si.eqx.intranet pour accéder au service AlertManager
2. grafana.k8s.lab-si.eqx.intranet pour accéder à l'interface Grafana
3. prometheus.k8s.lab-si.eqx.intranet pour accéder à l'interface Prometheus

Grâce à ces Ingress, nous pouvons accéder aux services de monitoring qui s'exécutent sur le cluster.

VII.6. Conclusion

La surveillance de Kubernetes est une étape, mais pour obtenir une vue complète, il est également nécessaire de surveiller les applications elles-mêmes.

Dans ce chapitre, nous avons expliqué l'importance de la surveillance pour un système distribué et avons examiné les types de métriques disponibles et leur utilisation avec différentes technologies. Le choix des technologies, qu'il s'agisse de solutions open source ou de solutions SaaS propriétaires, dépendra de vos besoins, du niveau d'expertise de vos ingénieurs et, bien entendu, de votre budget.

VIII. SmileLab

VIII.1. Introduction

Smile fournit à ses collaborateurs la possibilité de tester des nouvelles technologies dans un cadre de Lab.

Le Lab Kubernetes était une initiative de Smile pour mettre en place un environnement permettant aux collaborateurs de différentes agences de s'initier à l'orchestration de conteneurs.

Cette version du Lab est hébergée sur des serveurs bare metal en utilisant kubeadm. Kubeadm est une boîte à outils qui permet de créer des cluster Kubernetes sur une infrastructure existante en respectant les bonnes pratiques.

VIII.2. Architecture

Le Lab Kubernetes est déployé en mode **Mono Master - Multi Minions** sur un ensemble de machines virtuelles réservées pour des travaux de R&D au sein de la BU Système et Infrastructure.

Comme l'indique le schéma d'architecture de la figure VIII.1, le cluster Kubernetes est composé d'un master, de "n" minions, d'un nœud router, et d'un serveur de stockage NFS.

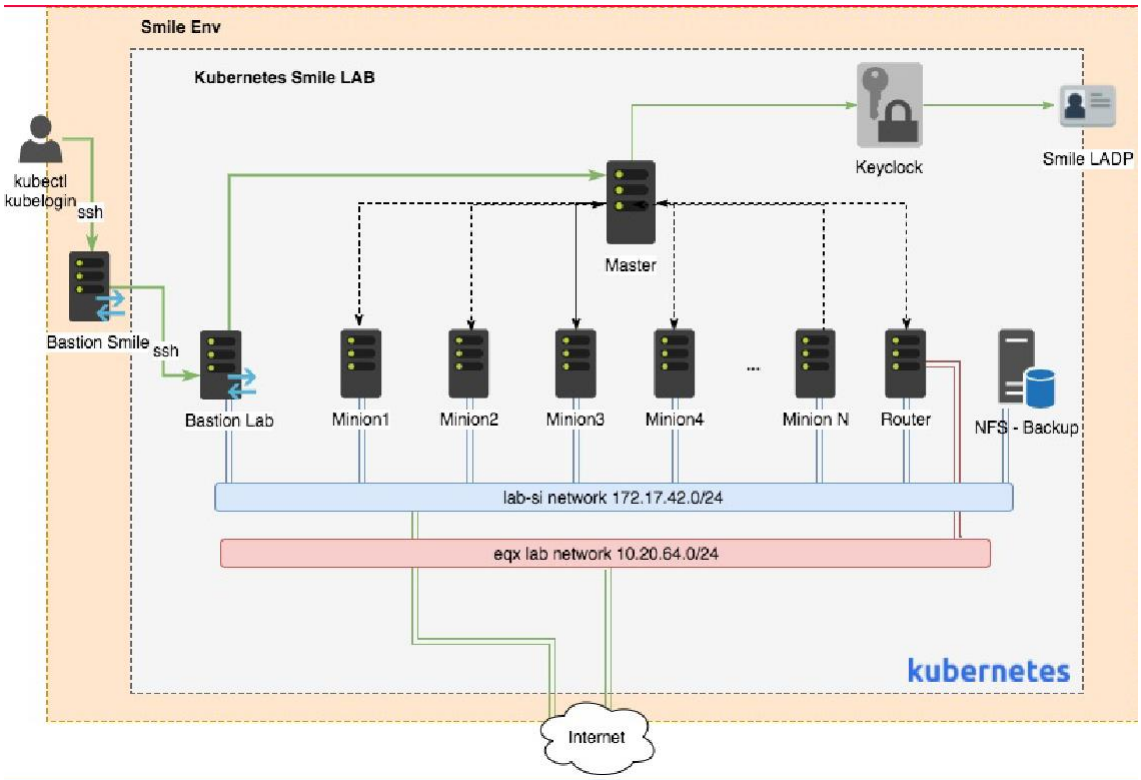

Figure VIII.1 : L'architecture du Lab Kubernetes

L'accès au cluster est possible à travers 2 bastions et protégé par une instance Keycloak reliée à l'annuaire OpenLDAP du groupe Smile en tant qu'identity provider.

VIII.3. Authentification et Authorization

L'authentification et l'autorisation sont deux exigences très importantes lors de la configuration d'un cluster Kubernetes.

Le flux d'authentification et d'autorisation de base dans un cluster Kubernetes est illustré par la figure VIII.2.

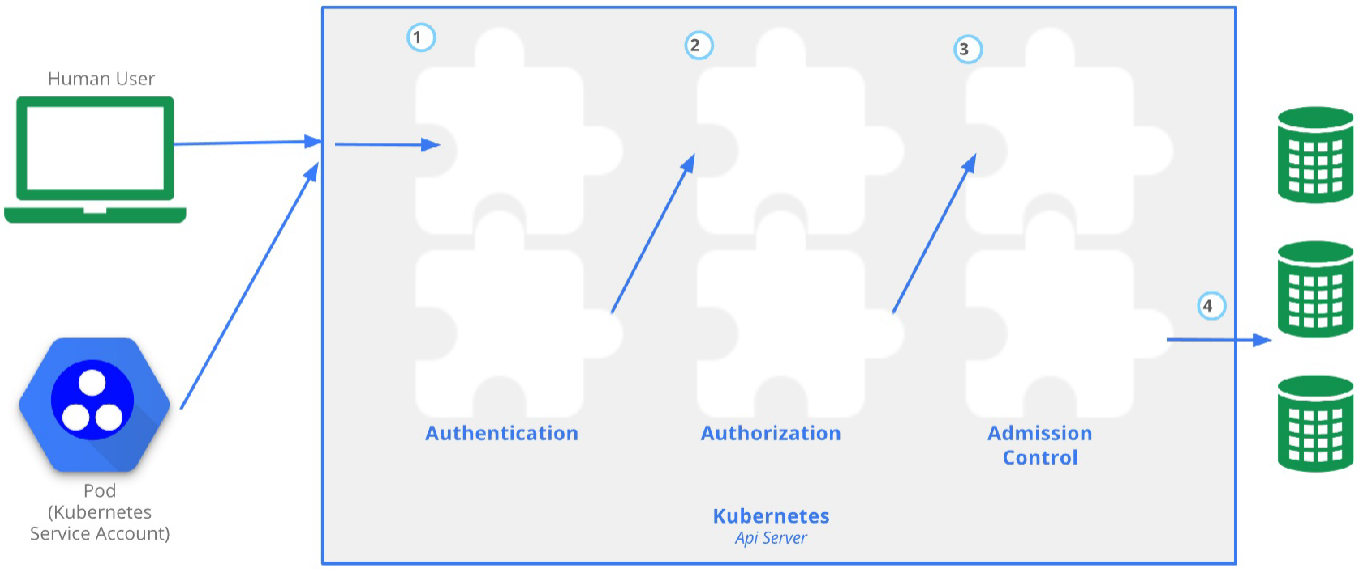

Figure VIII.2 : Authentification et Authorisation

Dans le cadre de ce Lab :

- **Authentification** : Une fois que la connexion TLS est établie, la demande HTTP passe à l'étape d'authentification. Ceci est indiqué à l'étape 1 du diagramme. L'administrateur du cluster configure l'API server pour exécuter un ou plusieurs modules d'authentification. Dans le cadre du Lab, nous utilisons Keycloak pour fournir une interface OpenIDConnect avec LDAP comme identity provider.
- **Autorisation** : Une fois que la requête est authentifiée par un utilisateur spécifique, elle doit être autorisée. Ceci est indiqué à l'étape 2 du diagramme. Dans le Lab, nous utilisons le RBAC de Kubernetes.
- **Contrôle d'admission** : Les modules de contrôle d'admission sont des modules logiciels pouvant modifier ou rejeter des demandes. Outre les attributs disponibles pour les modules d'autorisation, les modules de contrôle d'admission peuvent accéder au contenu de l'objet en cours de création ou de mise à jour. Exemple : Le contrôleur d'admission NamespaceLifecycle impose qu'un nouveau namespace en cours de terminaison ne peut contenir de nouveaux objets, et veille à ce que les demandes dans un namespace non-existant soient rejetées.

Keycloak et OpenID Connect

Afin de fournir à Kubernetes une méthode d'authentification, nous avons choisi d'installer Keycloak, qui est un gestionnaire d'identités et d'accès open source.

```
helm install --name auth --namespace auth -f values.yaml stable/keycloak
```

Une fois installé, nous créons un realm Kubernetes et un client Kubernetes de type `openid-connect` afin de pouvoir authentifier toutes les requêtes Kubernetes en utilisant le module OpenID Connect, comme l'indique la figure VIII.3.

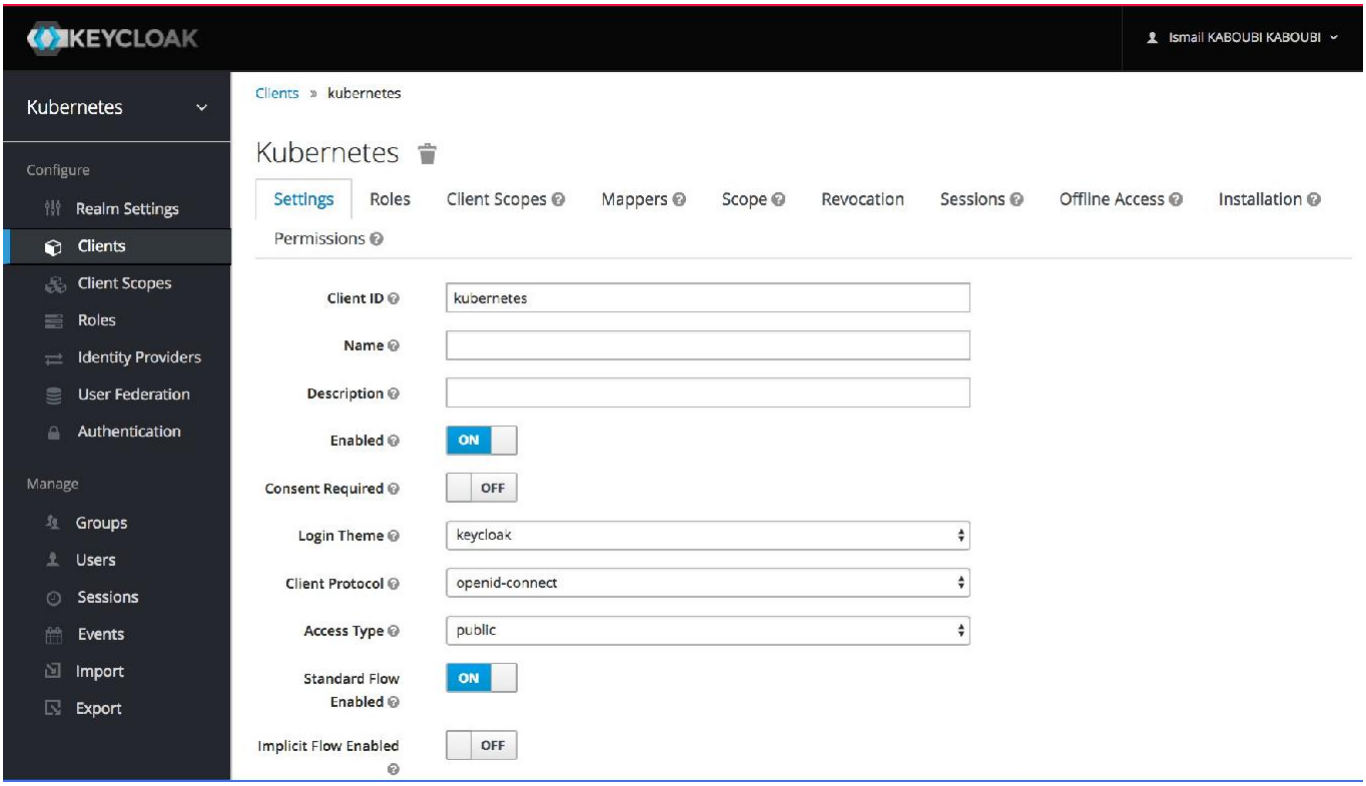

Figure VIII.3 : L'architecture du Lab Kubernetes

Il faut bien évidemment configurer l'API server pour déléguer la procédure d'authentification à Keycloak.

```
apiVersion: v1
kind: Pod
metadata:
labels:
  component: kube-apiserver tier:
  control-plane
name: kube-apiserver
namespace: kube-system
spec:
containers:
- command:
  - kube-apiserver
  - --authorization-mode=Node,RBAC
  - --advertise-address=172.17.42.90
  - ...
  - --oidc-ca-file=/etc/kubernetes/pki/smile.crt
  - --oidc-client-id=kubernetes
  - --oidc-issuer-url=https:/ keycloak.k8s.lab-si.eqx.intranet/auth/realms/kubernetes
  - --oidc-username-claim=email
  - --oidc-groups-claim=groups
  image: gcr.io/google_containers/kube-apiserver:v1.12.2 imagePullPolicy:
  IfNotPresent
  ...
```

Cette étape configure Kubernetes avec le module OpenID Connect. Dans cette configuration, toutes les requêtes initiées via `kubectl` seront rejetées par l'API server tant qu'un token d'accès n'est pas fourni.

```
$ kubectl get nodes
Unable to connect to the server: failed to refresh token: oauth2: cannot fetch
token: 400 Bad Request
Response: {"error":"invalid_grant","error_description":"Stale refresh token"}
```

kubelogin

kubelogin est un simple plugin pour l'authentification Kubernetes via OpenID Connect (OIDC). Il reçoit un token du fournisseur OIDC et l'écrit dans kubeconfig. La figure VIII.4 illustre ce flux.

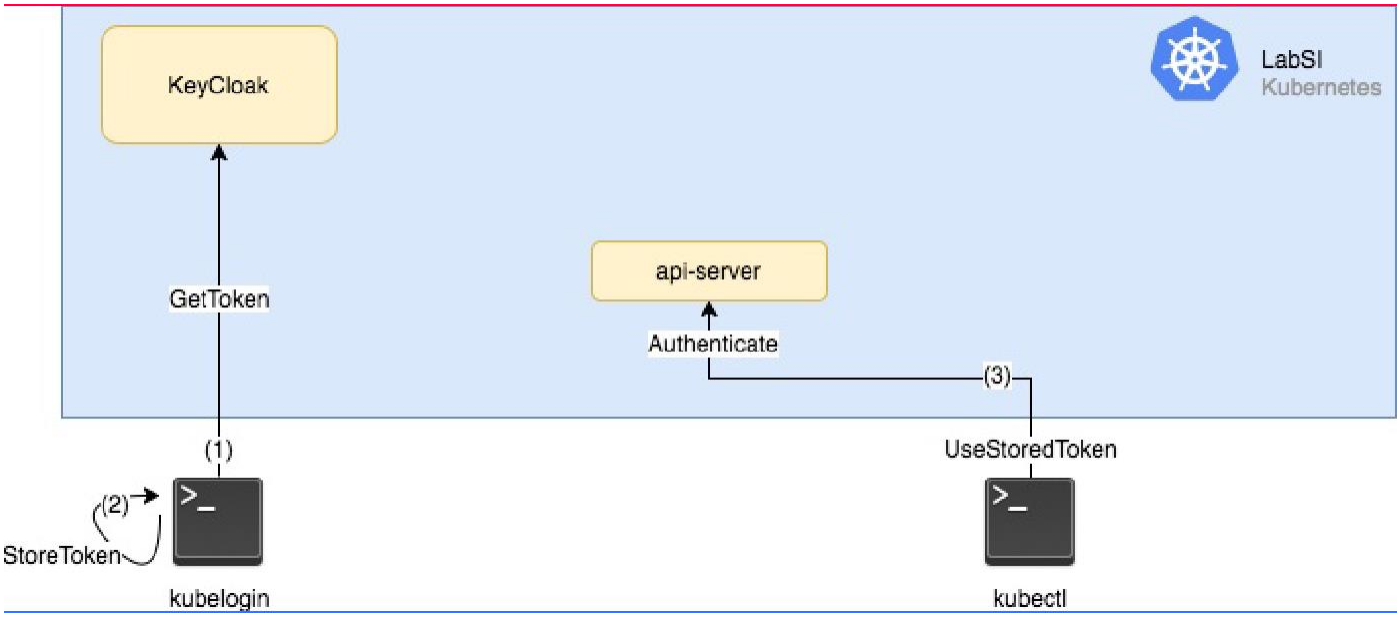

Figure VIII.4 : Authentification OIDC

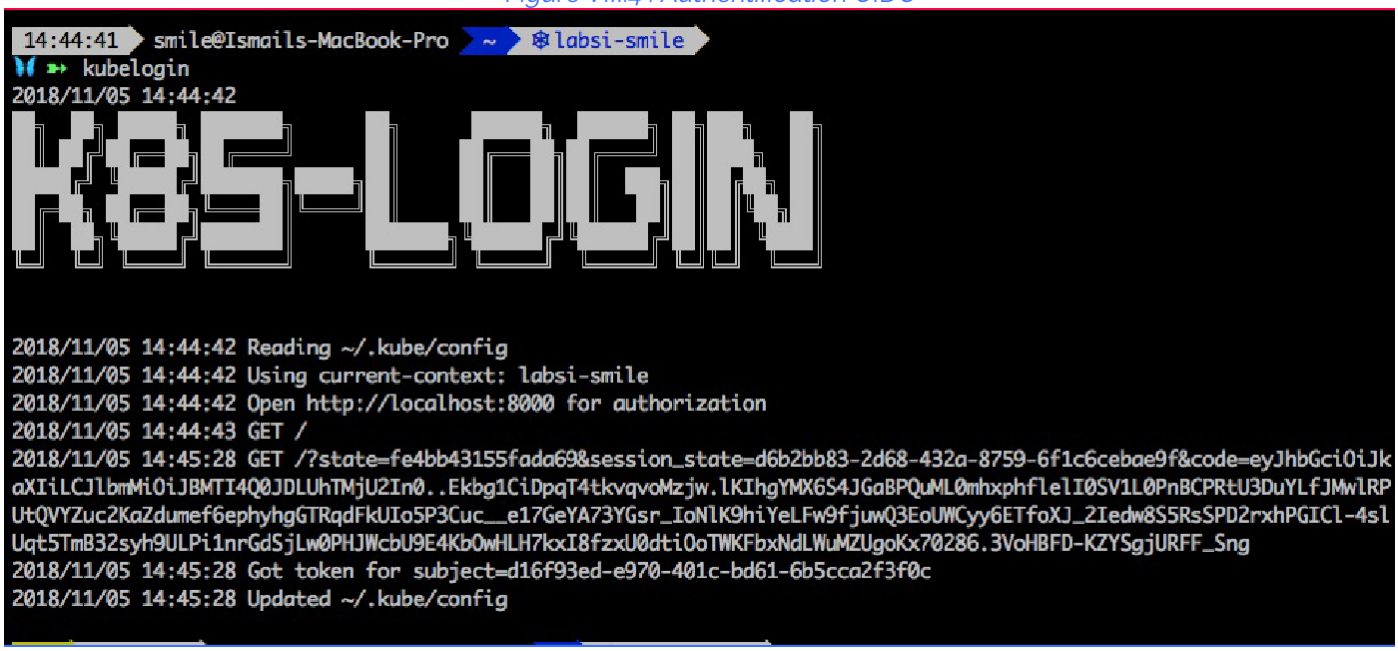

Figure VIII.5 : Kubelogin

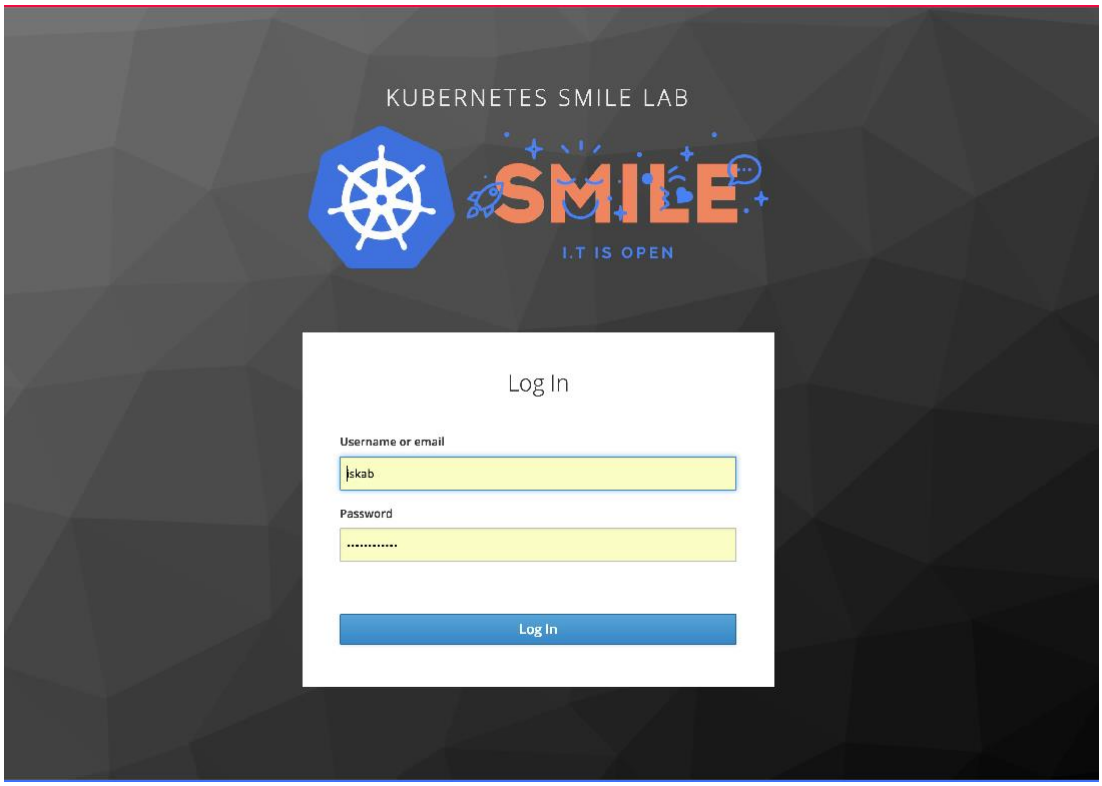

Figure VIII.6 : Authentification Keycloak

Figure VIII.7 : Resultat d'Authentification Keycloak

IX. Conclusion

Les conteneurs, dont la popularité ne cesse de croître, présentent des avantages importants dans la gestion de la montée en charge des applications cloud-native et de la transformation en microservices.

Plusieurs problématiques liées à la gestion des conteneurs ont été résolues par les solutions d'orchestration telles que Kubernetes.

Dans ce livre blanc, nous avons adressé plusieurs sujets tels que les modéles d'installation, les concepts de base, ainsi que quelques ressources complémentaires pour le déploiement des applications.

Et pour finir un petit clin d'œil aux travaux réalisés dans le cadre des labs Kubernetes chez Smile.

Fin du document.