Osez Kubernetes !

Rémi Cailletaud
Domaine universitaire, bâtiment OSUG-D
122 rue de la piscine
38 400 Saint Martin d'Hères

Résumé

Kubernetes a la réputation d'être un système complexe, qui n'aurait d'utilité que dans le cadre d'infrastructures conséquentes. On entend souvent qu'y exécuter des applications statefull est compliqué, voire inconscient.

Et pourtant… Kubernetes apporte certes des concepts nouveaux, mais il n'est rien d'autre qu'un orchestrateur de conteneurs tirant parti de fonctionnalités connues et éprouvées du noyau Linux.

En s'appuyant sur l'exemple du maquettage puis de la mise en production de clusters Kubernetes au sein de l'Observatoire des Sciences de l'Univers de Grenoble (OSUG), nous démontrerons que cet outil n'est ni opaque ni magique, qu'il est adapté à des organisations de toutes tailles, aussi bien pour des tâches d'administration système que dans le cadre de pratiques DevOps. Nous verrons comment il facilite le travail en équipe et la mise en place d'infrastructures programmables.

Nous justifierons le choix de Kubernetes, détaillerons ses concepts puis plongerons dans son fonctionnement interne et son API.

Nous présenterons ensuite les choix techniques que nous avons faits pour le provisioning et le déploiement des clusters, puis les choix d'infrastructure pour les clusters eux-mêmes : équilibrage de charge L2, reverse proxy, provisioning dynamique des volumes, métriques, monitoring et alertes.

Enfin nous aborderons les bonnes pratiques que les outils présentés permettent de mettre en place. La configuration déclarative des clusters nous permet de tendre vers une infrastructure programmable et de développer les méthodes GitOps. Nous présenterons succinctement ces méthodes et les outils utilisés pour les mettre en œuvre.

Mots-clefs

Kubernetes, orchestration, conteneurs, GitOps

1 Introduction

Quelques années seulement après leur apparition, les solutions basées sur des conteneurs sont devenues très à la mode pour le déploiement d'applications. Si elles permettent à l'évidence de réduire la friction entre les phases de développement et les phases de mise en production, elles soulèvent aussi certaines questions, du point de vue de la sécurité et de la gestion.

Si l'on souhaite partir dans cette voie, il n'est vite plus possible de gérer sa flotte de conteneurs manuellement. Il faut alors se pencher sur les solutions d'orchestration, dont les plus connues sont Docker Swarm et Kubernetes.

Cette problématique, nous avons dû y répondre à l'Observatoire des Sciences de l'Univers de Grenoble (OSUG), et nous avons choisi pour cela Kubernetes. Dans la suite, nous allons expliquer notre choix et sa mise en œuvre, et présenter les possibilités offertes par la mise en place d'un tel outil.

2 Contexte

L'OSUG est une structure fédérative, qui regroupe 6 unités de recherche, 5 équipes de recherche associées et 2 unités de services, sous multi-tutelles. L'OSUG œuvre dans tous les domaines des Sciences de l'Univers, de la planète Terre et de l'Environnement. Il est engagé dans 28 Services Nationaux d'Observation (SNO) en lien avec les recherches menées au sein de ses laboratoires.

Le service infrastructure de l'UMS OSUG fournit des services et un appui aux différents laboratoires de la structure et aux SNO qui développent des outils à destination des différentes communautés scientifiques. Nous œuvrons donc à la croisée des chemins des administrateurs système et des développeurs.

3 Périmètre technique

Le service infrastructure de l'OSUG exploite un cluster vSphere composé de trois ESX. Il accueille environ 180 VM, qui hébergent des services communs à destination de l'ensemble de la communauté, d'autres à destination de laboratoires en particulier et enfin ceux des SNO. Il n'y a actuellement pas de pratique commune concernant les systèmes et la configuration des VM, qui ne sont pas toutes gérées par un outil de gestion de configuration. Il en résulte une infrastructure hétérogène, difficile à administrer, nécessitant de nombreuses actions manuelles pas toujours correctement documentées et leur lot d'effets de bord. De plus, nous avions un nombre croissant de demandes d'hébergement de conteneurs et la mise à disposition de VM Docker mutualisées ne nous convenait pas, autant du point de vue de la sécurité que de celui de la gestion. Nous nous sommes donc mis en quête d'un orchestrateur de conteneurs…

4 Le choix de Kubernetes

4.1 Les différentes solutions d'orchestration

Lors de la phase de veille, nous avons évalué plusieurs solutions d'orchestration avant de porter notre choix sur Kubernetes. Avant toute chose, il nous a paru significatif que deux des solutions parmi les plus répandues abandonnent leur orchestrateur pour se tourner vers Kubernetes : CoreOs a abandonné Fleet début 2017[1], et Rancher se détourne progressivement de Cattle depuis depuis mi-2018 et la version 2[2].

Hashicorp Nomad est un orchestrateur léger. Mais il paie sa légèreté au prix de son manque de fonctionnalités : il s'agit d'un simple ordonnanceur, qui ne propose par exemple pas de fonctionnalités de *service discovery*, de répartion de charge, de gestion de configuration ou encore de gestion des volumes persistants. De plus la communauté n'est pas très importante, et l'effet de réseau peu présent.

Apache Mesos (associé généralement à l'orchestrateur Marathon) est une solution éprouvée et très complète qui supporte un passage à l'échelle conséquent principalement via ses modèles de fédération, mais elle est difficile à mettre en œuvre, puisqu'il faut déployer d'abord le framework Mesos puis un orchestrateur, et ne dispose pas d'outils « tout-en -un ». De plus, elle répond plutôt à des besoins très spécifiques : besoins de scalabilité à très grande échelle (plusieurs dizaines de milliers de nœuds), traitement de flux d'événements massifs (plusieurs millions par seconde).

Le seul véritable concurrent dans notre contexte était Docker Swarm. Il propose le même genre de fonctionnalités que Kubernetes, est plutôt plus simple à déployer puisqu'il n'a besoin que du démon Docker, mais bénéficie d'une tolérance aux fautes moins importante.

Notre choix s'est finalement porté sur Kubernetes pour plusieurs raisons techniques mais aussi stratégiques :

- Là ou Docker Swarm est un service monolithique, Kubernetes est pensé de manière très modulaire, chaque composant remplissant un rôle précis, dans la philosophie « *KISS* » (Keep It Simple Stupid) ;
- Kubernetes a la faculté de s'intégrer dans l'architecture sous-jacente (*Openstack*, *vmWare*) et d'en tirer parti ;
- Si Kubernetes est un projet initié par Google, son pilotage est maintenant confié à la Cloud Native Computing Foundation (CNCF), qui est une partie de la Linux Foundation. Docker Swarm est un projet de la société Docker, Inc. et son pilotage est moins transparent ;
- Enfin, Kubernetes bénéficie actuellement d'une adoption impressionnante : outre les solutions citées précédemment, tous les fournisseurs de cloud proposent une offre Kubernetes (Google bien entendu, mais aussi Amazon, Microsoft, IBM, ou encore OVH récemment). L'écosystème est très dynamique, et l'effet de réseau considérable.

1. https://coreos.com/blog/migrating-from-fleet-to-kubernetes.html
2. https://rancher.com/announcing-rancher-2-0/

4.2 Présentation générale

4.2.1 Historique

Kubernetes est le fruit de la réécriture en Go de Google Borg, un outil Java utilisé en interne chez Google, qui permettait la gestion de cluster orientée conteneur. La version 1.0 voit le jour en 2015. Le développement a été piloté par Google jusqu'en août 2018, date à laquelle le projet a été transféré au CNCF, qui est une partie de la Linux Foundation.

4.2.2 Objectifs

« *We must treat the datacenter itself as one massive warehouse-scale computer*. » [1]

Cette citation, tirée d'un article de 2009 (réédité depuis) résume bien l'objectif de Kubernetes. Si le papier, écrit par des ingénieurs de Google, semble aborder des problématiques qui sont bien éloignées de notre quotidien, nous pouvons en tirer quelques enseignements qui peuvent nous permettre de gérer plus facilement des systèmes de plus en plus gros et complexes. En particulier, nous avons besoin d'un système qui abstrait non seulement les couches matérielles, mais aussi les couches système.

Les principaux objectifs qui ont mené à la conception de Kubernetes sont :

- une abstraction du matériel et des systèmes d'exploitation ;
- un couplage faible des composants afin de faciliter leur remplacement ;
- un surcoût minimal, on parle de 5 % par nœud, 1 % pour l'ensemble du cluster ;
- un fonctionnement identique sur serveurs physiques et sur des machines virtuelles.

Si l'installation est tout à fait possible sur des serveurs physiques en particulier s'il y a des besoins spécifiques (performance, périphériques, …), le terrain de jeu favori de Kubernetes est le cloud, qu'il soit public (Amazon, Google,…) ou privé (Vsphere, Openstack,…). En effet, sa capacité à communiquer avec les systèmes sous-jacents lui permet de s'intégrer avec élégance, et de tirer parti de ces derniers.

4.2.3 Principes

Kubernetes est bâti autour de trois grands principes.

- **Une *API* (*Application Programming Interface*) purement déclarative** : les objets de l'*API* déclarent un état désiré, jamais les étapes pour atteindre cet état. Cette approche permet d'éviter les effets de bords. C'est aux différents composants du cluster de le faire converger vers l'état désiré. On ignore tout des couches hardware et système.
- **Des capacités d'auto réparation**: les composants du cluster s'occupant de le faire converger dans l'état désiré, les services défaillants seront automatiquement relancés. On a donc nativement une surveillance au niveau applicatif, ce qui est un gros avantage à la fois pour l'utilisateur du service et pour son administrateur.

- **L'immutabilité** : les composants en fonctionnement ne peuvent pas être modifiés, on les remplace. Dans la littérature abondante sur le sujet, on retrouve souvent l'analogie « *Animaux de compagnie contre bétail* » [2][3]. Lorsqu'un composant a un problème, on le **supprime** et on le remplace. Cette approche présente de nombreux avantages, en particulier pour les tests, les mises à jours, et les retours en arrière.

4.2.4 L'architecture

La Figure 1 présente l'architecture simplifiée de Kubernetes. Elle est basée sur la séparation du *Control Plane*, ou plan de contrôle, qui est le cerveau du cluster, des *Nodes* où les charges de travail sont déployées. Le plan de contrôle conserve un enregistrement de tous les objets du système et exécute des boucles de contrôle continues pour gérer l'état de ces objets. À tout moment, les boucles de contrôle du système réagissent aux modifications du cluster et permettent de le faire converger vers l'état désiré.

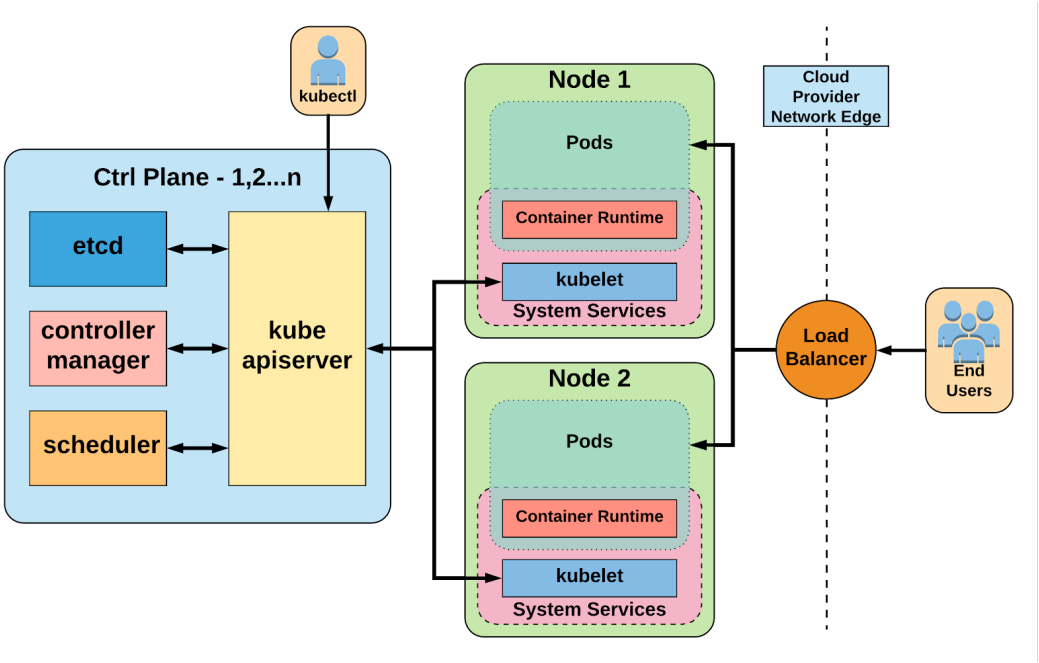

Figure 1 : Architecture simplifiée

La Figure 2 détaille les différents composants du plan de contrôle. Les plus importants sont le serveur d'API, le composant de stockage *etcd*, le gestionnaire de contrôle et l'ordonnanceur.

- *kube apiserver* est l'élément central du système, qui sert l'API REST de Kubernetes au format JSON via HTTP. Il gère l'interface externe du cluster : c'est au serveur d'*API* que l'on s'adresse pour configurer le cluster ; il gère aussi l'interface interne, puisque l'ensemble des composants ne communiquent qu'avec lui, et ignorent tout des autres composants. Cette architecture permet d'obtenir un couplage faible entre les composants.

- *etcd* est un composant de stockage clé/valeur distribué hautement disponible, développé initialement par CoreOS. Il utilise l'algorithme de consensus Raft [4], qui a la particularité de converger très rapidement. Il permet de stocker de manière fiable les données de configuration du cluster, représentant l'état du cluster à n'importe quel instant.
- Le *controller manager* est la boucle de contrôle dans lequel s'exécutent les différents contrôleurs de Kubernetes. Leur rôle est de communiquer avec le serveur d'*API* pour créer, mettre à jour et effacer les ressources qu'ils gèrent (*pods*, *service*, *endpoints*, etc.).
- Le *scheduler* (ordonnanceur) est le composant permettant de choisir le nœud sur lequel doivent tourner les différentes charges de travail en fonction de leurs besoins et de la disponibilité des ressources.
- Le *cloud controller manager* permet l'intégration à l'infrastructure sous-jacente. Il peut par exemple mettre en place de l'équilibrage de charge réseau ou tailler des volumes.

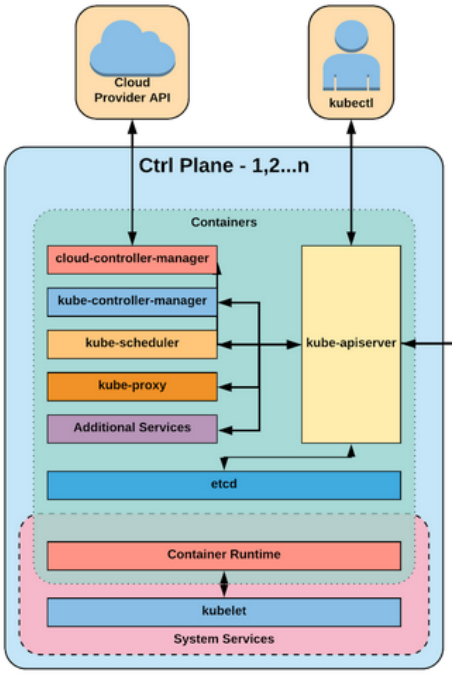

Figure 2 : Plan de contrôle

Les nœuds (*worker nodes)* sont les machines sur lesquelles les charges de travail sont exécutées. Kubernetes supporte jusqu'à 5000 nœuds. La Figure 3 présente les différents composants d'un nœud Kubernetes.

- *Kubelet* est le composant responsable de s'assurer que les charges de travail (*pods)* sont dans l'état désiré. Il les démarre, les stoppe et les surveille selon les ordres du plan de contrôle. En cas de défaillance, il redéploie les *pods*.

- *Kube-proxy* est responsable des règles de *forwarding* et de l'équilibrage de charge réseau. Il route les requêtes vers les conteneurs. Il existe plusieurs implémentations à l'aide soit des *userspaces* Linux, soit d'*iptables*, soit d'i*pvs.*
- Le *container runtime* doit être compatible avec l'API Container Runtime Interface (CRI) [5]. C'est le composant qui fait effectivement tourner les charges de travail. Si containerd, le moteur de Docker est évidemment le plus utilisé, Kubernetes supporte aussi *rkt* (le moteur développé par CoreOS), CRI-O, Kata (virtualisation ultralégère), ou encore Virtlet (VM classiques).

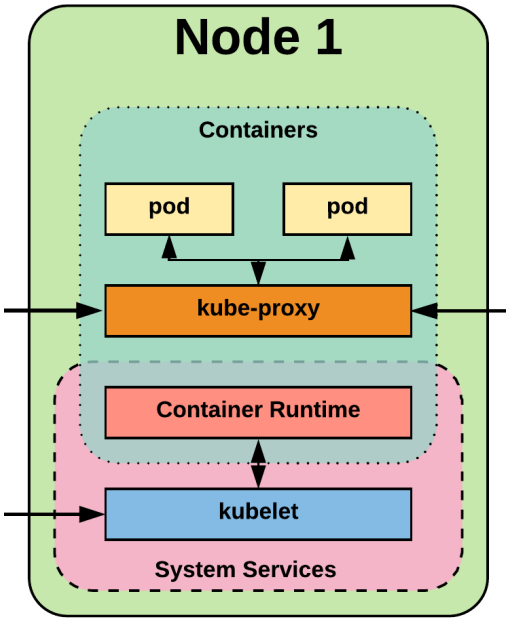

Figure 3 : Nœud Kubernetes

4.2.5 L'API

Comme nous l'avons vu un peu plus tôt, l'API REST est la clé de voûte de Kubernetes. Les administrateurs et utilisateurs du cluster ainsi que tous les composants ne communiquent qu'à travers elle, et tout dans Kubernetes est représenté comme un objet dans l'API, qui peut être décrit aux formats JSON ou YAML.

Sans rentrer dans les détails, nous allons présenter quelques objets de base.

- Les *namespaces* sont des environnements qui permettent simplement de partager le cluster en compartiments logiques.
- Les *pods* sont les charges de travail atomiques. Ils sont constitués d'un ou plusieurs conteneurs qui partagent les volumes, le réseau et font partie du même contexte. Il faut bien comprendre qu'ils sont éphémères, et que rien ne garantit qu'ils auront toujours le même nom ou la même IP.
- Les *services* sont le moyen d'exposer les pods. Ils offrent une méthode d'accès soit par IP, soit avec un DNS interne. La figure 4 représente leur fonctionnement qui peut s'apparenter à de l'équilibrage de charge réseau interne.

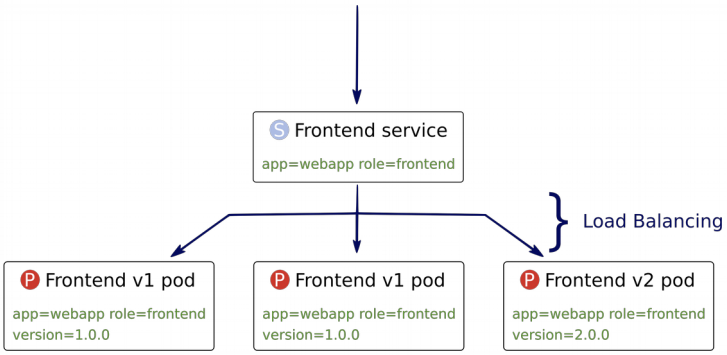

Figure 4 : Accès au pods via un service

- Les *deployments* permettent de gérer un ensemble de pods. Ils fournissent la possibilité de mise à jour et de retour en arrière fluides.
- Les *statefullsets* permettent de gérer des applications *statefull*. Leur réseau, nom d'hôte et stockage sont persistants.
- Les *Jobs* et *CronJobs* permettent d'exécuter des charges de travail temporaires, utiles dans le cas de calcul ou de tâches d'administration.
- Les *Volumes*, *PersistentVolumes*, et *PersistentVolumeClaims* permettent de gérer le stockage persistant.
- Les *ConfigMap* et les *Secrets* permettent la gestion des configurations des charges de travail.
- Les *Ingresses* permettent la configuration du reverse proxy.

Ci-dessous un exemple simple de fichiers permettant la création d'un service redis. L'utilisation de la commande *kubectl* permet de créer les objets dans l'API.

```
# deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
spec:
  replicas: 1
  selector:
    matchLabels:
      app: redis
  template:
    metadata:
      labels:
        app: redis
    spec:
      containers:
      - image: redis
        name: redis
```

```
# service.yml
apiVersion: v1
kind: Service
metadata:
  name: redis
spec:
  ports:
    - port: 6379
      targetPort: 6379
  selector:
    app: redis
```

4.3 La mise en œuvre

4.3.1 Le déploiement

L'écosystème Kubernetes étant très dynamique, il existe une très grande variété de solutions pour déployer un cluster. *kubeadm* est la solution de référence, mais les déploiements sont fastidieux, comportant beaucoup d'étapes manuelles.

Certains éditeurs proposent des solutions empaquetées, facile à installer. Parmi eux figurent VMware (Pivotal Container Service et VMware Cloud PKS) ou Red Hat (OpenShift), mais celles-ci ne sont pas forcément libres, et ne fournissent pas une expérience « vanilla » de Kubernetes. Or nous voulions limiter notre dépendance à un éditeur et cherchions une solution réellement libre et portable.

Rancher Kubernetes Engine (RKE), développée par Rancher Labs, est une solution simple et légère. Les composants du cluster sont conteneurisés, les déploiements sont reproductibles etles nœuds n'ont besoin que de Docker. L'installation est possible aussi bien sur machines physiques que virtuelles, le simple fichier YAML ci-dessous décrit l'installation d'un cluster comportant un nœud pour le plan de contrôle et etcd, et un *worker node*.

```
nodes:
    - address: 1.2.3.4
      user: ubuntu
      role:
        - controlplane
        - etcd
    - address: 1.2.3.5
      user: ubuntu
      role:
        - workers
```

Déployer un cluster avec RKE ne prend que quelques minutes, et y ajouter des nœuds et d'une facilité déconcertante. Par ailleurs, la configuration déclarative nous permet d'historiser notre infrastructure, comme nous le verrons plus tard.

Notons que pour le déploiement et la configuration des nœuds eux-mêmes, nous avons choisi d'utiliser Packer, Salt-Cloud et Salt. Ceci nous permet d'utiliser des outils de gestion de version. pour l'ensemble de notre pile, et de tendre vers une infrastructure programmable : l'ensemble de la configuration de notre infrastructure est gérée à l'aide de fichier de définition et ne nécessite aucune action manuelle.

4.3.2 L'interface Rancher 2

Rancher Labs propose aussi une interface web de gestion du cluster. Si nous avons décidé de ne pas l'utiliser pour le déploiement et l'administration des clusters, nous nous appuyons dessus pour donner un accès limité aux différents utilisateurs. Ainsi, ils peuvent profiter d'une vue simplifiée et intuitive du cluster. Mais surtout, c'est cette interface qui gère la couche d'authentification et de règles d'accès, qui serait en son absence très contraignantes à mettre en œuvre et à maintenir.

4.3.3 Le choix des composants dans le cluster

Nous avons à nouveau un large éventail de choix pour les composants déployés dans le cluster. Nous allons en aborder certains, très rapidement puisque chacun des composants présentés ici mériterait un article à part entière.

- Le rôle du plugin réseau est de faire communiquer les pods entre eux. Ils peuvent travailler en niveau 2 ou en niveau 3. Pour notre part, nous avons choisi Canal, qui est la solution par défaut de RKE. Il travaille niveau 3 (tunnels VXLAN), et présente l'avantage de gérer les politiques de sécurité réseau à l'intérieur du cluster.
- Dans notre cas, l'équilibreur de charge L2 permet plutôt d'assurer la haute disponibilité, puisque nous n'avons pas d'équipement en face qui parle BGP. Nous utilisons MetalLB, que l'on peut comparer à une solution type KeepAlived, avec une IP virtuelle.
- L'*ingress controller* est en fait un *reverse proxy*, qui surveille les objets Kubernetes de type *Ingress*, et configure à la volée des règles d'accès de type Nom DNS ↔ Service Kubernetes. Nous utilisons Traefik, qui comporte l'avantage de présenter un tableau de bord des règles en place.
- Pour le stockage persistant, nous utilisons vsphere-cloud-provider, et nfs-client-provider. Le premier taille des volumes blocs dans le vSAN du cluster VMware, le second crée des répertoires sur un serveur NFS.
- Nous utilisons Prometheus et Grafana pour le monitoring. Prometheus a un mécanisme de découverte qui permet de remonter tout un ensemble de métriques de manière très simple.

5 Infrastructures programmables et méthodes GitOps

Comme nous l'avons vu auparavant, le choix des outils permet d'historiser l'ensemble de la configuration, depuis la génération des images des VM jusqu'au déploiement des clusters. Nous utilisons pour cela Gitlab, ce qui présente plusieurs avantages :

- Le travail en équipe est beaucoup plus fluide : l'accès à la configuration est possible en un point central pour tous les membres de l'équipe, et les demandes de modification sont possibles à travers des pull requests.
- Cela constitue un plan de reprise sur accident idéal. En effet, il suffit de quelques dizaines de minutes pour remonter l'ensemble des composants.

Outre l'aspect infrastructure, nous avons commencé à développer des méthodes équivalentes avec les développeurs de l'OSUG. Le principe de la méthode GitOps est de ne jamais effectuer de modification « manuelle » des déploiements. Celles-ci sont effectuées automatiquement, depuis un dépôt git. Ainsi, le dépôt devient une « source de vérité », qui indique explicitement quelle version du code est en train de tourner. Le passage d'un code d'une phase de test à une phase de production pourrait par exemple être déclenché par une simple commande *git tag*.

Nous n'en sommes qu'au balbutiement de ces pratiques que nous facilite Kubernetes. Nous explorons les outils comme ArgoCD qui permettent de les mettre en place, mais il reste du chemin à parcourir.

6 Bilan

Kubernetes peut paraître intimidant au premier abord. Pour les ASR, il s'agit d'un changement radical de paradigme. En particulier, l'aspect immutable est difficile à appréhender. D'autre part, la maîtrise des technologies de conteneurs est indispensable avant de se lancer dans une telle aventure. Par ailleurs, le dynamisme de l'écosystème est parfois déroutant. Il ne faut pas se précipiter sur les nouveautés, savoir prendre le temps de tester, et laisser les différentes solutions faire leurs preuves.

Il faut aussi insister sur le fait que Kubernetes ne doit pas être considérée comme une plateforme « clé en main », mais plutôt comme un ensemble de briques de base qui vont vous permettre de construire votre plateforme. Il existe bien souvent plusieurs chemins pour arriver au même résultat.

Néanmoins, une fois ces avertissements pris en compte, Kubernetes s'avère un système d'une fiabilité et d'une robustesse surprenantes pour un système jeune et dynamique. Son intégration aux infrastructures sous-jacentes est un point très positif pour centraliser la configuration de l'infrastructure.

Durant les mois de tests et de maquettage, nous n'avons rencontré que très peu de problèmes, et la communauté est très réactive. De plus, la transparence du pilotage inspire confiance et permet de se tenir au courant au plus tôt des évolutions futures.

Enfin, les pratiques qui découlent de l'adoption de Kubernetes portent très rapidement leurs fruits, tant pour le métier ASR que pour celui de développeur.

7 Perspectives

Jusqu'ici nous n'utilisons Kubernetes que pour des services internes à faible niveau de criticité. Nous avons trois clusters, qui nous permettent principalement de tester les montées en version. Nous avons désormais une maîtrise et une confiance suffisantes pour envisager une montée en charge progressive. Les premiers services extérieurs doivent être mis en production durant le mois de novembre.

Cela nécessite un travail en cours avec les développeurs de l'OSUG : s'il est tout à fait possible de faire tourner un code sans modifications, il faut au minimum passer par une livraison sous forme de conteneurs. Cependant, on tire tous les bénéfices d'un passage à Kubernetes en adaptant le code aux nouveaux paradigmes qu'il apporte. L'effort nécessaire est alors plus conséquent, mais le bénéfice pour les développeurs est tel qu'ils sont généralement prêts à fournir ce travail. Il ne faut alors pas négliger notre rôle d'accompagnement et de conseil.

Au cours des phases de tests et maquettage, nous avons eu l'occasion de nous appuyer sur Nova (la plateforme Openstack de la COMUE grenobloise). En combinant les capacités d'orchestration d'Openstack pour déployer les machine virtuelles (avec Heat ou Terraform par exemple) et la souplesse de RKE, le déploiement des clusters est facile et rapide. L'intégration avec Cinder , le service de stockage bloc d'Openstack permet d'exploiter les différents stockages attachés à la plateforme. Avec cette solution, nous tirons parti d'une pile entièrement libre et très facilement programmable. Ce type de solution est vouée à prendre de plus en plus d'importance dans nos métiers, découplant ainsi un peu plus l'aspect infrastructure de l'aspect applicatif : nous avons une couche dont le rôle est de fournir les ressources (CPU, RAM, disques) et une couche dont le rôle est de les exploiter, en s'affranchissant des questions purement système.

Bibliographie

[1] Luiz André Barroso, Jimmy Clidaras, Urs Hölzle. 2009, 2013 ; The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines. Synthesis Lectures on Computer Architecture ; http://web.eecs.umich.edu/~mosharaf/Readings/DC-Computer.pdf

[2] Bill Baker. 2012 ; Scaling SQL Server 2012. 24 Hours of PASS ; http://www.pass.org/eventdownload.aspx?suid=1902

[3] Randy Bias. 2014; The Cloud Revolution. Philippines Cloud Summit; https://fr.slideshare.net/randybias/the-cloud-revolution-cyber-press-forum-philippines

[4] Diego Ongaro, John Ousterhout. 2014 ; In search of anunderstandable consensus algorithm. Proc ATC'14, USENIX Annual Technical Conference. https://raft.github.io/raft.pdf

[5] Yu-Ju Hong. 2016; Introducing Container Runtime Interface (CRI) in Kubernetes. Five Days of Kubernetes 1.5; https://kubernetes.io/blog/2016/12/container-runtime-interface-cri-in-kubernetes/

[6] Alexis Richardson. 2017 ; GitOps - Operations by Pull Request ; https://www.weave.works/blog/gitops-operations-by-pull-request