Cailletaud R., Coussot Charly, Schaeffer J., Mella G., Bourgès L., Chaffard Véronique. (2022). Kubernetes à l'OSUG : regards croisés administrateurs/utilisateurs. [s.l.] : [s.n.], 15 p. JRES 2021 : Journées Réseaux de l'Enseignement et de la Recherche, 14., Marseille (FRA), 2022/05/17-20.

Kubernetes à l'OSUG : regards croisés administrateurs/utilisateurs

Rémi Cailletaud, Charly Coussot, Jonathan Schaeffer, Guillaume Mella, Laurent Bourgès
Observatoire des Sciences de l'Univers de Grenoble
Domaine universitaire, bâtiment OSUG-D
122 rue de la piscine
38400 Saint-Martin-d'Hères

Véronique Chaffard
Institut des Géosciences de l'Environnement
Domaine universitaire
460 rue de la piscine
38400 Saint-Martin-d'Hères

Résumé

L'Observatoire des Sciences de l'Univers de Grenoble (OSUG) a mis en place, depuis 2019, une infrastructure Kubernetes pour offrir un service d'hébergement de conteneurs. Les Services Nationaux d'Observation de l'OSUG ont investi la plateforme pour déployer leurs services autour des données scientifiques.

Nous présentons ici un regard croisé sur « le virage Kubernetes » qui a été abordé de plusieurs manières par les développeurs d'application, les acteurs en charge de la production et du déploiement, et les administrateurs de la plateforme.

Pour ces acteurs, l'arrivée de la technologie Kubernetes a exigé un changement profond pour le déploiement, la supervision, l'administration et la maintenance. Cela a aussi poussé l'adoption de meilleures pratiques de développement et renforcé les liens entre administrateurs systèmes, développeurs et responsables d'exploitation.

Mots-clefs

Kubernetes, conteneurs, DevOps, GitOps

1 Introduction

En 2019, l'Observatoire des Sciences de l'Univers de Grenoble (OSUG) a fait le choix de Kubernetes comme solution d'orchestration de conteneurs [1]. L'objectif était de permettre aux Services Nationaux d'Observation de déployer et maintenir leurs services autour des données scientifiques de manière robuste et fluide.

Force est de constater que ce choix a été un succès : Kubernetes est devenu la référence indiscutable en matière d'orchestration de conteneurs ; à l'OSUG, les équipes de développement se sont emparées de la solution et ont investi la plateforme : une dizaine de services sont désormais déployés sur les plateformes Kubernetes de l'OSUG et les nouveaux développements sont pensés pour en tirer parti.

Ces quelques années de production nous donnent un certain recul sur la technologie et nous permettent de tirer les enseignements d'une migration vers les technologies d'orchestration de conteneurs. C'est aussi l'occasion d'un regard croisé sur « le virage Kubernetes » qui a été abordé de plusieurs manières par les développeurs d'applications, les acteurs en charge de la production et du déploiement, et les administrateurs de la plateforme.

2 Contexte

L'OSUG est une structure fédérative, qui regroupe 6 unités de recherche, 5 équipes de recherche associées et 2 unités de services 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'UAR (Unité d'Appui à la Recherche) 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 Un cluster mutualisé

Le service infrastructure de l'OSUG exploite un cluster VMWare vSphere composé de cinq ESXi. Il accueille environ 180 machines virtuelles (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. C'est en observant le fonctionnement de ces derniers que l'idée d'offrir une solution d'orchestration de conteneurs a fait son chemin : ce sont en grande majorité des développeurs, par forcément à l'aise avec les tâches d'administration système. L'objectif était donc de les décharger de ces tâches pour les laisser se focaliser autant que possible sur les activités de développement.

3.1 Le déploiement

Nous nous appuyons sur plusieurs outils pour déployer les clusters sur notre infrastructure de virtualisation :

- Packer[1], qui nous permet de construire des images de VM de manière automatique ;
- Salt-Cloud[2] qui permet de provisionner les machines virtuelles ;
- Salt[3], qui permet de gérer la configuration de ces dernières ;
- enfin, Rancher Kubernetes Engine[4] (RKE), pour déployer les clusters Kubernetes (K8s).

1 https://www.packer.io/

2 https://docs.saltproject.io/en/latest/topics/cloud/index.html

3 https://docs.saltproject.io/en/latest/

4 https://rancher.com/docs/rke/latest/en/

L’ensemble de cette pile nous permet de nous approcher d’une infrastructure programmable. Avec le recul cependant, l’utilisation de Salt-Cloud n’est pas le plus approprié et nous nous orientons vers Terraform pour l’avenir, qui propose une solution plus complète et stable : gestion de multiples fournisseurs, gestion robuste des états, intégration avec d’autres outils DevOps.

La couche Kubernetes est donc déployée à l’aide de RKE, qui propose une solution simple et légère basée sur des conteneurs. Elle a été développée par Rancher Labs, société qui a récemment été acquise par SUSE. Cette solution nous a procuré une grande satisfaction, nous permettant de passer à l’échelle et de monter en version aisément. C’est une solution libre, dotée d’une communauté et de développeurs réactifs.

Notons que nous utilisons aussi Rancher 2, une interface web qui permet de donner un accès limité aux différents utilisateurs. Ils peuvent ainsi profiter d’une vue simplifiée et intuitive du cluster. Et surtout, c’est cette interface qui gère la couche d’authentification et de règles d’accès, qui serait en son absence très contraignant à mettre en œuvre et à maintenir.

3.2 Stockage et base de données

Un des points sensibles de la fourniture d’une infrastructure d’hébergement de conteneurs est la question du stockage et des bases de données. En effet, de par leur nature éphémère, les conteneurs n’ont pas vocation à stocker des données et il faut donc s’appuyer sur un mécanisme externe pour en assurer la persistance.

Nous avons pu expérimenter trois approches pour assurer la persistance des données.

vSAN (in-tree vSphere Cloud Provider [ref])

Nos clusters s’exécutant sur une plateforme vSphere, il nous paraissait naturel de profiter des capacités d’intégration de Kubernetes aux infrastructures sous-jacentes comme présenté dans la Figure 1. En effet, ce dernier est capable de tailler et attacher des volumes vSAN dynamiquement. Malheureusement, l’implémentation souffrait de quelques problèmes lors des mises à jour de la plateforme vSphere, des systèmes d’exploitation des VM ou de Kubernetes. De plus, le support communautaire sur cette intégration est minimal et VMware ne fait pas beaucoup d’efforts dans ce sens (d’autant que l’éditeur propose désormais une distribution Kubernetes maison…). Enfin, le développement de Kubernetes s’élargissant, le support des fournisseurs de stockage « *in-tree* » a été remplacé par une architecture plus modulaire [2]. Il existait bien un chemin pour migrer vers le nouveau fournisseur externe[5], mais l’expérience mitigée nous a amené à abandonner cette solution pour le moment.

5 https://github.com/kubernetes-sigs/vsphere-csi-driver/blob/master/docs/book/features/vsphere_csi_migration.md

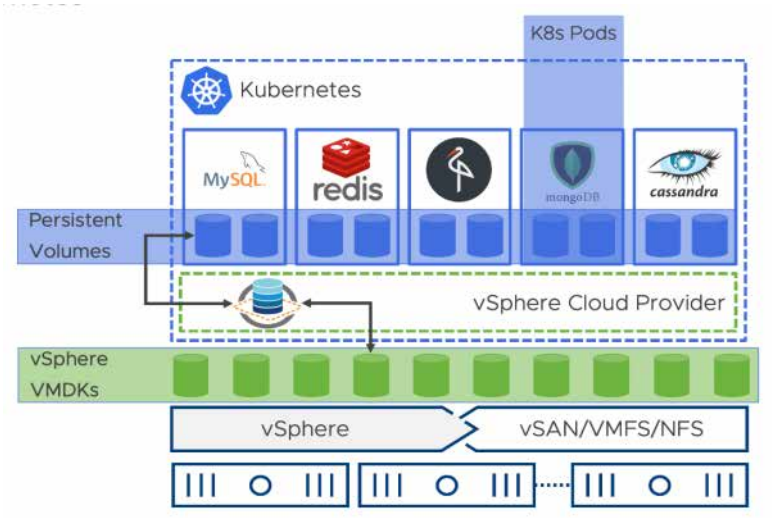

Figure 1: vSphere Cloud Provider

NetApp Trident

L'Université Grenoble Alpes (UGA) propose une solution de stockage mutualisé à l'échelle de l'Université intégrée, basée sur la technologie Data Ontap de NetApp [3]. Cette solution est performante et sécurisée et il nous paraissait naturel d'en tirer parti pour notre infrastructure, d'autant que NetApp propose une solution intégrée de gestion du stockage des conteneurs : Astra Trident[6].

Malheureusement, dans la pratique cette solution s'est heurtée à deux écueils. Le premier était le déploiement, qui nécessitait beaucoup d'opérations manuelles et allait à l'encontre de nos pratiques *Infrastructure As Code*. Le second était dû à l'aspect mutualisé de l'infrastructure SUMMER, qui ne nous permettait pas d'accéder à l'ensemble des fonctionnalités offertes par l'intégration Trident.

Sur ces deux points cependant, les choses semblent avoir évoluées récemment et nous n'excluons pas de faire de nouveaux tests afin de réaliser si cette intégration nous semble la plus pertinente.

NFS Provisionner

Nous nous sommes donc rabattus sur le provisionneur « du pauvre », qui permet d'allouer dynamiquement des volumes NFS (en fait de simples répertoires…). Cette solution s'appuie là aussi sur le stockage mutualisé offert par SUMMER mais présente l'avantage d'être très simple à mettre en œuvre et de pouvoir être déployée partout : un simple serveur NFS suffit…

Les bases de données

Dès le début, nous avions fait le choix de ne pas héberger les serveurs de bases de données à l'intérieur des clusters Kubernetes : même si de nombreuses solutions existaient, elles étaient toutes très jeunes et il nous semblait prématuré de se lancer dans cette voie. De plus, l'OSUG propose un service mutualisé d'hébergement de bases de données PostgreSQL. Dès lors, l'architecture standard d'une application développée à l'OSUG s'appuie sur des conteneurs s'exécutant sur les clusters

6 https://docs.netapp.com/us-en/trident/index.html

Kubernetes et des bases de données PostgreSQL. On voit donc l'effet structurant qu'a pu avoir ce mouvement sur nos architectures logicielles.

La leçon que nous avons tirée de ces aspects stockage est la nécessité de privilégier le stockage objet[7] au stockage fichier, quand cela est possible : il permet une grande évolutivité et une très grande souplesse à l'utilisation. Certains projets l'ont déjà adopté et une telle offre est en train de voir le jour à l'OSUG.

3.3 Des clusters « jetables »

Si nous avons vu précédemment que la distribution RKE nous a donné entière satisfaction, il est tout de même rarement arrivé qu'une montée en version soit la source de bugs. L'existence de trois clusters différents (test, pré-production et production) permet de détecter ces bugs sur des infrastructures non critiques mais les mises à jour de Kubernetes restent toujours un moment délicat.

De plus, les systèmes d'exploitation des nœuds et le *container runtime* nécessitent aussi des mises à jour régulières. Ces opérations peuvent aussi être source de problèmes et d'indisponibilités des applications.

Nous avons donc déplacé le fameux problème *Pet vs Cattle [4][5]* : si nos applications sont devenues « du bétail », nos clusters sont à présent les animaux de compagnie. Et ceci devient d'autant plus vrai quand le nombre d'applications hébergées par le cluster augmente !

L'idée est donc venue de franchir une nouvelle étape et de déployer des clusters Kubernetes « jetables » : lors des mises à jour (OS, version de Kubernetes, composants du cluster), c'est tout le cluster qui est remplacé, depuis les machines virtuelles jusqu'aux applications finales, et ce de manière automatique. La Figure 2 illustre l'architecture générale de cette infrastructure. Les détails techniques se trouvent dans l'annexe cattle-rke2.

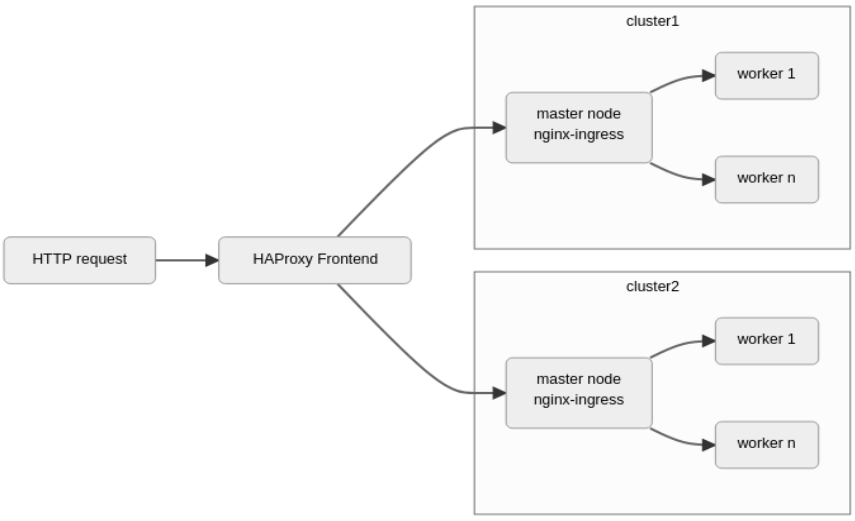

Figure 2: Clusters « jetables », architecture générale

Évidemment, toutes les applications ne supportent pas ce mode de déploiement : il faut des applications *stateless* (sans état) et *scalables* (capables d'avoir plusieurs instances de l'application

7 Stockage non hierarchisé au modèle de métadonnées et de sécurité très riches, généralement accessible via une interface REST.

s’exécutant en parallèle). Cela met en lumière la nécessité de développer les applications selon des standards qui faciliteront les déploiements sur ce type de plateforme et l’accompagnement nécessaire des équipes de développement à ces nouvelles pratiques. Les applications *statefull* restent hébergées sur les clusters gérés de manière plus classiques.

4 Un travail main dans la main

Tirer pleinement parti des plateformes de déploiement modernes tel Kubernetes demande quelques bonnes pratiques en matière de conception, de développement et de livraison d’applications. Un travail de sensibilisation et d’accompagnement des équipes de développement est donc nécessaire pour favoriser l’adoption d’une telle plateforme.

4.1 L’architecture logicielle

L’objectif n’est pas d’imposer un langage de programmation ou une architecture, mais plutôt de fixer des bonnes pratiques pour concevoir les logiciels (applications web). Cela implique en particulier l’utilisation de formats déclaratifs pour l’automatisation (fichiers YAML), une portabilité maximale, l’abstraction de l’OS sous-jacent, un code identique entre développement et production et la possibilité de passage à l’échelle. Parmi ces pratiques, certaines peuvent paraître évidentes, d’autres peuvent être plus compliquées à mettre en œuvre.

Une méthodologie, présentée dans un premier temps par des développeurs de la plateforme Heroku[8], une plateforme de *PaaS* (Platform-as-a-Service) a fait surface et s’est imposée comme une référence des bonnes pratiques pour le développement d’applications web. Cette méthode comporte 12 règles à respecter d’où son nom de « *Twelve-Factor App methodology* »[6]. Nous ne la détaillerons par ici mais laisserons le lecteur s’y référer.

4.2 L’intégration et le déploiement continus

L’utilisation de plateformes telles que Kubernetes permet de mettre en place très facilement des processus d’automatisation de déploiements. Ceux-ci présentent l’avantage de limiter la friction entre les phases de développement et de mise en production, de limiter les erreurs humaines et de laisser une très grande liberté aux développeurs.

La Figure 3 illustre le processus de déploiement continu des applications. Un premier dépôt, contenant le code de l’application, est chargé de générer un artefact, dans notre cas un conteneur. Un second dépôt, contenant le code de mise en production de l’application, consomme et configure cet artefact afin de décrire le déploiement. Dans notre cas, il s’agit de manifestes Kubernetes.

8 https://www.heroku.com/

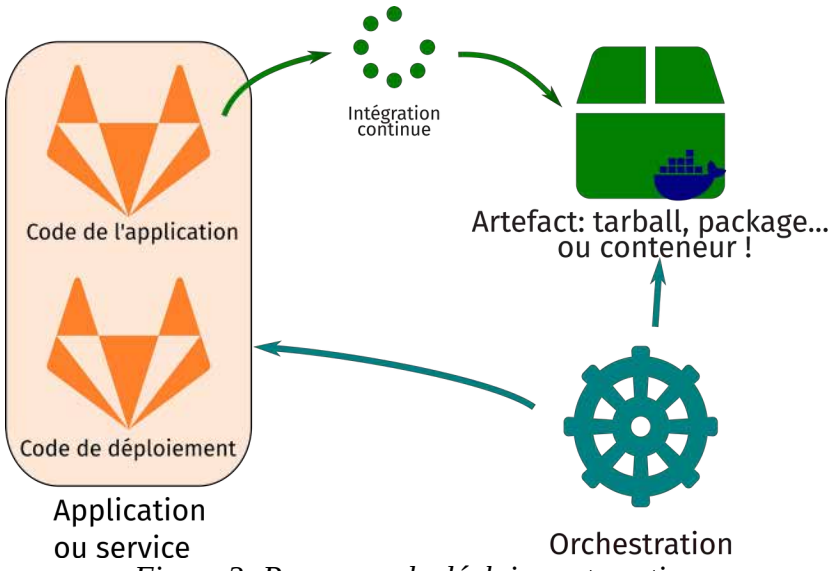

Figure 3: Processus de déploiement continu

Cette seconde phase est encore manuelle pour certains projets à l'OSUG, généralement les plus complexes, pour lesquels une totale automatisation est prématurée. Pour d'autres, elle est entièrement automatisée : la modification du code de déploiement déclenche une mise à jour automatique de l'application. Dans ce cas, la gestion de l'application repose donc entièrement sur l'utilisation de Git, on parle donc de *GitOps*, concept popularisé par Weaveworks[9].

4.3 Les conteneurs

Nous venons de voir que les applications et services sont déployés sous forme de conteneurs. Leur multiplication apporte de nouvelles questions, en particulier d'un point de vue de la sécurité. Nous avons donc effectué un travail, en collaboration avec un groupe de développeurs et d'administrateurs de la région grenobloise, afin de fixer des bonnes pratiques pour l'écriture de recettes d'images de conteneurs[10] dédiées à la mise en production avec Kubernetes. Nous laisserons là encore le lecteur curieux s'y référer.

L'adoption de cette plateforme à l'OSUG a été l'occasion d'un véritable dialogue entre les équipes de développement et les équipes d'exploitation. L'accompagnement des premiers est indispensable, pour l'écriture des recettes d'images de conteneurs et des manifestes Kubernetes. Ce travail important demande un peu de temps, mais entraîne par la suite une séparation des rôles plus claire et une grande souplesse pour les équipes de développement.

5 Développements *from scratch*

Commençons par aborder le cas le plus simple d'applications écrites avec le déploiement Kubernetes en ligne de mire pour tirer au mieux parti de ces nouvelles architectures. Nous aborderons d'abord le cas d'applications simples qui nous ont permis d'appréhender les concepts et méthodes, avant de nous pencher sur le développement d'une application complexe[11], le système d'information Theia/OZCAR[7].

9 https://www.weave.works/technologies/gitops/

10 https://kubernetes-alpes.gricad-pages.univ-grenoble-alpes.fr/dockerfile-for-prod/

11 https://in-situ.theia-land.fr/

5.1 Applications simples

Nous avions rapidement identifié quelques applications en développement qui étaient de bonnes candidates à un déploiement sur Kubernetes. Il s'agit d'applications développées avec les *frameworks* Symfony (php) ou Django (python), par des développeurs seuls et sans connaissances système suffisantes. Il faut donc dans ce cas épauler les développeurs pour la conception des conteneurs et l'écriture des manifestes Kubernetes. Mais une fois que cela est fait et que le processus *GitOps* est mis en place, les développeurs ont la main sur le déploiement de leurs applications, et ce de manière extrêmement simple et sans accès au cluster.

Si ces méthodes sont très appréciées, il faut tout de même insister sur la nécessité d'un suivi de sécurité sur les images de conteneurs de ces projets : la mise en place d'un scanner de vulnérabilités comme Trivy[12] est très conseillée.

5.2 Le système d'information Theia|OZCAR

La mise à disposition de la solution Kubernetes par l'OSUG a concordé avec les débuts des développements du système d'information Theia/OZCAR, dont l'objectif est de rendre visibles, accessibles, interopérables et réutilisables (*FAIR*[13]) l'ensemble des données in-situ d'observation des surfaces continentales collectées par les organismes de recherche français et leurs partenaires en France et à l'étranger. Son équipe technique a décidé de s'appuyer sur cette solution prometteuse et les développements ont été réalisés en appliquant directement une logique de conteneurisation de services afin de se rapprocher d'une application cloud-native.

La conteneurisation est facilitée par des workflows *DevOps* ; chaque brique de l'application est compilée, testée, empaquetée, conteneurisée via un processus d'intégration continue. Les images sont stockées sur le registre d'image fourni par le service GitLab de l'UGA et sont prêtes à être utilisées par l'orchestrateur.

L'objectif d'un déploiement sur Kubernetes impose deux règles de développement qui n'auraient pas forcément été suivies pour une application *standalone.* Ces règles sont évidemment définies dans la méthodologie *Twelve-Factor* :

- les services déployés doivent être *stateless* afin de profiter des mécanismes de scalabilité et de tolérance aux pannes. La définition de ces services applicatifs permet alors de tirer avantage des approches de développement API-first tout en assurant leur implémentation *stateless* ;
- la configuration doit être strictement séparée du code et fournie par l'environnement de déploiement pour permettre l'utilisation d'applicatifs strictement similaires sur les environnements de développement et de production ~~et~~ afin de profiter des mécanismes de gestion des secrets Kubernetes.

Dans le cadre du déploiement du SI Theia/OZCAR sur l'orchestrateur Kubernetes de l'OSUG, deux perspectives d'évolution sont rapidement envisageables. La poursuite du workflow *DevOps* avec du

12 https://www.aquasec.com/products/trivy/

13 https://fr.wikipedia.org/wiki/Fair_data

déploiement continu *GitOps* et la mise en place de stockage objet pour augmenter la scalabilité des volumes et s'affranchir des limitations du stockage bloc.

6 Migration d'applications

Dans cette partie, nous allons aborder le cas particulier de la migration d'applications existantes vers l'écosystème Kubernetes. Commençons par énoncer deux évidences : les applications qui avaient déjà fait le pas vers la conteneurisation (docker) et celles qui bénéficient d'une architecture stateless/microservices sont avantagées. Parmi les chantiers de migration, ceux du JMMC[14] et de Résif-DC[15] [8] sont révélateurs et riches d'enseignements.

6.1 Le Centre Jean-Marie Mariotti (JMMC)

Depuis 20 ans, le JMMC développe, maintient et met à disposition de la communauté internationale d'astronomes des outils et bases de données pour l'observation par interférométrie optique. Ceux-ci sont disponibles sous forme de webservices qui nécessitent une infrastructure d'hébergement adéquate. Le choix de JMMC a été de s'appuyer dès que possible sur les services offerts par l'Observatoire ou l'UGA (stockage, virtualisation, supervision, forges…) pour se concentrer sur l'aspect scientifique de sa mission. Les services étaient virtualisés depuis 2009 (Proxmox puis VMware) puis en 2017, les premiers services conteneurisés ont vu le jour, en s'appuyant sur docker-compose, des recettes Ansible et des volumes NFS pour la persistance des données.

Le JMMC avait donc déjà fait un premier pas vers l'abstraction du système d'exploitation et c'est très naturellement qu'un premier service *stateless* simple a pu être déployé dès la mise à disposition de la plateforme Kubernetes. Les services *statefull* se sont ensuite appuyés sur des volumes NFS (data et logs) et des bases de données du serveur mutualisé PostgreSQL, en veillant à la bonne gestion de la réentrance au niveau applicatif et système de fichiers (fichiers verrous et écritures atomiques dans les dossiers partagés).

Le besoin de disponibilité maximale des services opérés par JMMC est satisfait par les fonctionnalités natives de Kubernetes : stratégies de montée en version (*RollingUpdate*), bascule vers les nouveaux déploiements uniquement lorsqu'ils sont opérationnels (*startupProbe*), gestion des ressources pour éviter les fuites ou consommation excessive de mémoire, supervision et redémarrage automatique des services en panne (*livenessProbe*), et pour les services *statefull*, l'unicité garantie avec fail-over. L'utilisation de *Cronjob* a quant à elle permis de planifier des opérations de synchronisation de données à travers le déclenchement d'actions planifiées (protégées contre la réentrance au sein du cluster Kubernetes).

L'interface en ligne de commande *kubectl* est très riche et pratique pour se connecter directement au conteneur d'un service, consulter les logs et connaître l'état d'un service. Notons cependant qu'un temps d'adaptation aux concepts Kubernetes est nécessaire afin de ne pas se perdre dans la multitude des services et déploiements. L'utilisation de labels permet de répondre à cette problématique.

14 https://www.jmmc.fr / https://cat.opidor.fr/index.php/JMMC

15 https://seismology.resif.fr

L'utilisation de *Kustomize*[16], qui offre un mécanisme de template permet de déployer très facilement plusieurs versions simultanées (beta/prod ou blue/green) des applications.

Le service ObsPortal[17] est un bon exemple de service web adoptant l'approche « *Blue/Green deployement* »[18] pour exposer 2 instances du service (prod/pré-prod). Il est possible de basculer à chaud entre instances (permutation des *Ingress*) pour effectuer la mise à jour de la pré-production : la migration et la synchronisation complète des données peut prendre plusieurs heures. Nous laisserons les plus curieux se référer au manifeste complet du service[19]. Notons que pour faciliter l'identification des instances et les versions des applications, schéma et base de données associés, une page « *health*[20] » remonte l'ensemble de ces informations ainsi que l'état de la connexion à la base et des accès disques en lecture/écriture. Cette même page est utilisée pour la vérification du bon fonctionnement depuis Internet. Nous pouvons nous féliciter de la stabilité du service depuis sa publication en février 2020.

L'enthousiasme général ne doit pas masquer les difficultés rencontrées :

- La mémoire allouée aux applications Java doit être configurée au niveau applicatif et aussi dans le manifeste Kubernetes, ce qui peut entraîner des incohérences et rendre le réglage fastidieux ;
- certains services *restfull* s'appuient sur le framework ExistDB[21] et celui-ci s'inscrit mal dans le paradigme par manque du support d'accès concurrents (stockage NFS). Malgré tout, Kubernetes offre une large panoplie de stratégies de déploiement qui permet de contourner le problème.

La plupart des services opérés par JMMC sont désormais hébergés sur les clusters Kubernetes mutualisés de l'OSUG. Le trafic y est routé via un *reverse-proxy* (HAProxy), rendant l'architecture modulaire, facilement administrable et robuste. Ce frontal nous a permis par exemple d'appliquer à l'ensemble des services une mitigation de la faille Log4J[22] en décembre 2021 et de collecter des logs d'utilisation.

Dans l'avenir, le JMMC entend tirer parti des possibilités de passage à l'échelle pour optimiser le temps et les ressources nécessaires à certains services de traitement à la demande (analyse de données, calcul…).

6.2 L'entrepôt de données Résif-DC

Le centre de gestion de données sismologiques national Résif-DC exploite une douzaine de services en production pour la distribution des données, des métadonnées et de produits dérivés. Ils sont

16 https://kustomize.io/

17 http://obs.jmmc.fr/

18 https://martinfowler.com/bliki/BlueGreenDeployment.html

19 https://gricad-gitlab.univ-grenoble-alpes.fr/OSUG/JMMC/jmmc-obsportal-kubernetes/

20 http://obs.jmmc.fr/health

21 http://exist-db.org/

22 https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-44228

accessibles sous forme de webservices[23] et étaient déployés de manière traditionnelle sur des serveurs virtualisés.

Presque tous ces webservices étaient de bons candidats pour être déployés sous forme de conteneurs dans un cluster Kubernetes : ils sont *stateless* par conception puisque ils tirent leurs informations d'une base de données externe (PostgreSQL en général, et MongoDB dans un cas) et de volumes NFS, hébergés par la plateforme de stockage mutualisé de l'UGA.

Faisant face simultanément à une hausse de l'accès à la donnée et à la criticité des forces de travail disponibles pour l'administration des systèmes, les bonnes propriétés de Kubernetes (scalabilité, auto-réparation) étaient des atouts attractifs.

En raison de politiques de production particulières,Résif-DC a créé son propre cluster Kubernetes de production sur l'infrastructure virtuelle de l'UGA (WINTER) et utilise un cluster de développement à l'OSUG.

La migration a nécessité un effort très important afin de :

- livrer des containers Docker fonctionnels : certains webservices nécessitant de nombreux éléments de configuration, de dépendances ont nécessité un travail important de conception, en restant dans l'optique des bonnes pratiques pour la création de containers, en particulier la construction multistage qui permet l'utilisation de conteneurs intermédiaires.[24] ;
- concevoir des flux de travail nouveaux ;
- changer les habitudes d'administration et de déploiement : on ne se connecte plus en SSH sur des serveurs pour explorer le bon fonctionnement du service ;
- adapter les outils de supervision : on n'observe plus le bon fonctionnement de process, le remplissage de journaux depuis un serveur. En revanche, la supervision se fait d'un point de vue de l'interface du service, car on s'appuie sur les propriétés d'auto-réparabilité de Kubernetes pour se garantir que les process sont toujours opérationnels.

Les principaux enseignements que ces migrations d'applications ont apportés sont :

- les documentations de déploiement traditionnelles sont une base indispensable pour la création de containers, mais un service complexe nécessite beaucoup de travail pour être finalisé en container ;
- la démarche de conception de conteneur est une excellente approche pour valider une documentation technique. Un *Dockerfile* et un manifeste Kubernetes sont un très bon complément à ces documentations, mais ne les remplace pas toujours ;
- la reproductibilité de nos déploiements apporte un certain soulagement pour l'administration des systèmes ;

23 https://ws.resif.fr/

24 https://docs.docker.com/develop/develop-images/multistage-build/

- après la migration effective de la production sur le cluster Kubernetes, il faut prévoir une période d'adaptation. En effet, la gestion des ressources du cluster Kubernetes demande de s'approprier des concepts et des méthodes de travail différentes ;
- si le nombre de machines virtuelles utilisées pour l'exploitation n'a pas radicalement diminué, elles se sont largement uniformisées, passant du statut d'animal de compagnie (*pet*) à celui de bétail (*cattle*) et simplifiant donc grandement les opérations d'exploitation.

Les efforts déployés dans ce projet de migration ont été conséquents et le retour sur investissement se fait sur le moyen terme. Il faut noter que le rôle de l'OSUG comme soutien et accompagnateur a été déterminant.

7 Bilan et perspectives

L'adoption de Kubernetes à l'OSUG a été une aventure humaine et technique. Elle n'a été possible que grâce à un dialogue entre les différents métiers, qui a eu un double effet : une montée mutuelle en compétence, mais aussi une séparation plus nette du rôle de chacun. D'un point de vue technique, elle a permis l'adoption de nouvelles méthodes de travail pertinentes et efficaces. Les nouveaux développements sont désormais tous pensés avec le déploiement conteneurisé en ligne de mire, ce qui est un indicateur fort du succès de ce chantier. De plus, on note un effet structurant au niveau de l'infrastructure et des architectures logicielles, avec la mutualisation des services de base de données et de stockage.

Dans le futur, la généralisation des méthodes *GitOps* et le façonnage des applications doit permettre d'élargir l'utilisation des clusters « jetables », ce qui permettra à long terme d'envisager des déploiements sur plusieurs sites. Pour cela, un travail sur les bases de données, le stockage objet, la collecte de logs et de métriques est nécessaire.

Annexe

cattle-rke2

Le principe du déploiement des clusters « jetables » à l'OSUG est illustré dans la figure ci-dessous. Il est orchestré par Terraform[25].

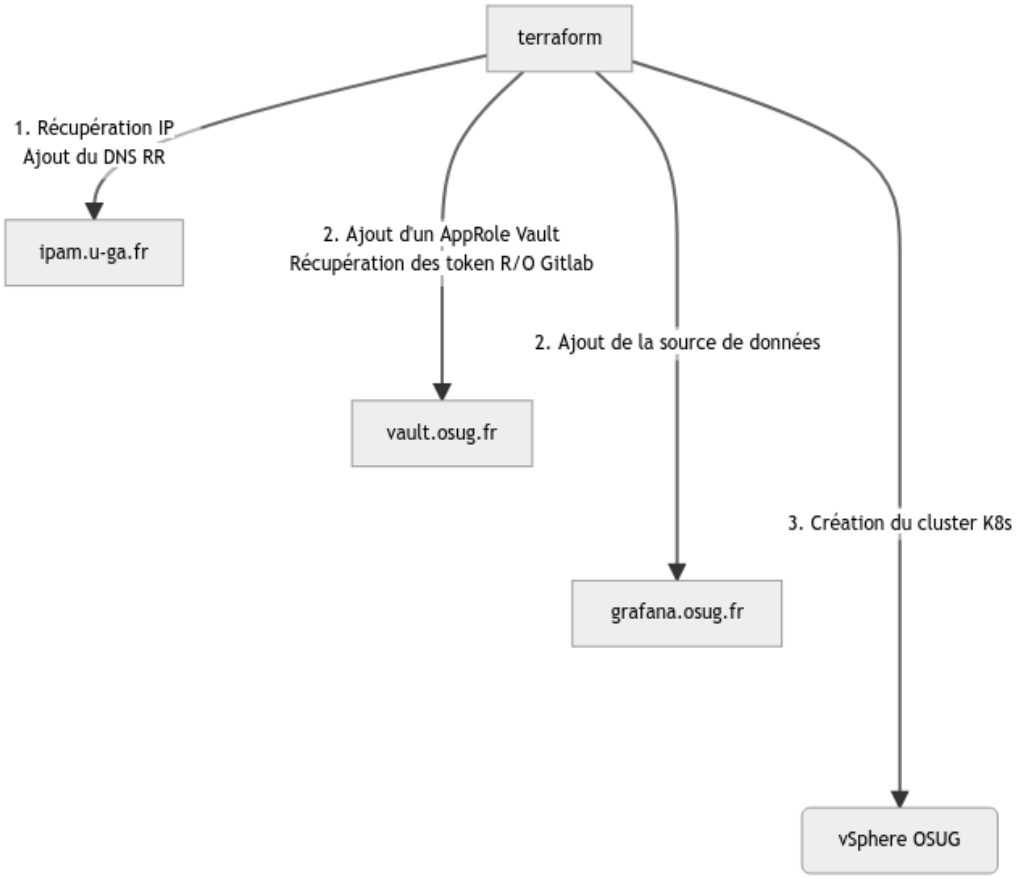

Figure 4: Déploiement des clusters cattle-rke2

La Figure 5 décrit l'architecture générale des clusters :

- Les pods nginx-ingress et Prometheus[26] tournent sur le nœud maître.
- ArgoCD[27] est utilisé pour déployer les applications. Une image légèrement modifiée est utilisée pour déchiffrer les secrets via Vault[28].
- Les applications à déployer sont déclarées dans un dépôt dédié.

25 https://www.terraform.io/

26 https://prometheus.io/

27 https://argo-cd.readthedocs.io/en/stable/

28 https://www.vaultproject.io/

- Un composant maison indique au frontal HAProxy que le cluster est prêt à recevoir du trafic.

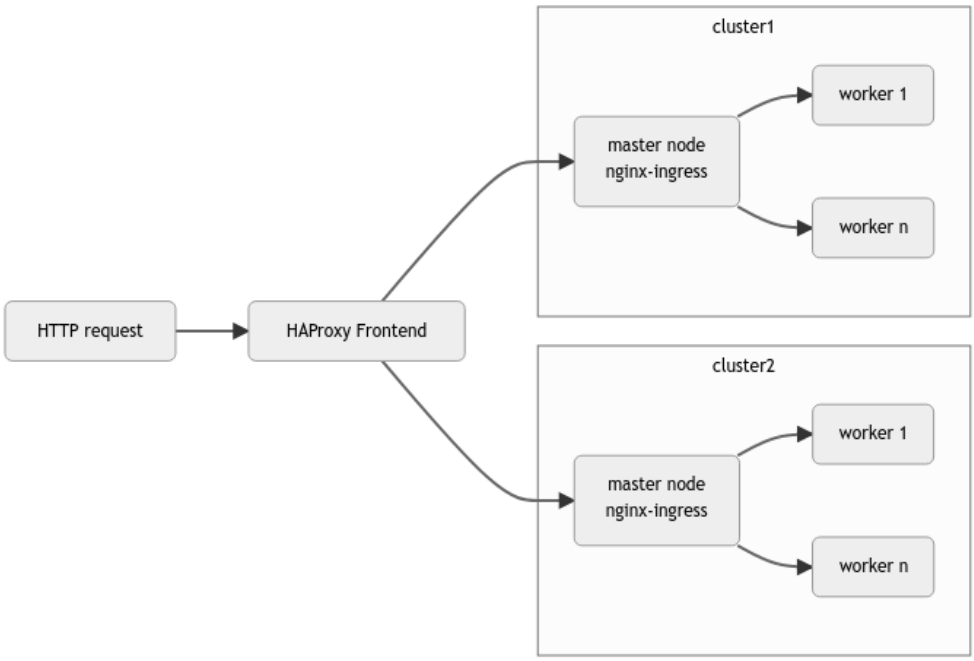

Figure 5: Architecture des cluster cattle-rke2

Bibliographie

[1] Rémi Cailletaud. Osez Kubernetes ! JRES2019, décembre 2019, Dijon, France. hal-02388005

[2] David Zhu. Kubernetes In-Tree to CSI Volume Migration Moves to Beta. Kubernetes Blog, décembre 2019.

[3] Defize Anthony, Drago-Rajon Fabien, Estevez Manuel, Rapacchi Dimitri, Sanchez Guenael. SUMMER, 4 ans de stockage mutualisé et réparti. JRES2017, Nantes, novembre 2013.

[4] Bill Baker. Scaling SQL Server. 24 Hours of PASS, novembre 2012.

[5] Randy Bias. The Cloud Revolution. Philippines Cloud Summit, Makati, septembre 2014.

[6] Adam Wiggins. The Twelve Factor App.

[7] Isabelle Braud et al. Building the information system of the French Critical Zone Observatories network: Theia/OZCAR-IS. Hydrological Sciences Journal, pp.1-19. Taylor & Francis, 2020.

[8] Catherine Péquegnat et al. RÉSIF-SI: A Distributed Information System for French Seismological Data. Seismological Research Letters, 92 (3), pp.1832–1853, 2021.