

UNIVERSITÉ NATIONALE DU VIETNAM, HANOÏ
INSTITUT FRANCOPHONE INTERNATIONAL

ANDRÉ Perrault

**Dockerisation du MaaS pour une architecture en
micro-services**

Docker hoá dịch vụ MaaS cho kiến trúc vi dịch vụ

MÉMOIRE DE FIN D'ÉTUDES DU MASTER INFORMATIQUE

HANOÏ - 2018

UNIVERSITÉ NATIONALE DU VIETNAM, HANOÏ
INSTITUT FRANCOPHONE INTERNATIONAL

ANDRÉ Perrault

**Dockerisation du MaaS pour une architecture en
micro-services**

Docker hoá dịch vụ MaaS cho kiến trúc vi dịch vụ

Spécialité : Systèmes Intelligents et Multimédia
Code : Programme pilote

MÉMOIRE DE FIN D'ÉTUDES DU MASTER INFORMATIQUE

Sous la direction de : Dr. Luiz-Otavio Goncalves BURITY

HANOÏ - 2018

Attestation sur l'honneur

J'atteste sur l'honneur que ce mémoire a été réalisé par moi-même et que les données et les résultats qui y sont présentés sont exacts et n'ont jamais été publiés ailleurs. La source des informations citées dans ce mémoire a bien été précisée.

LỜI CAM DOAN

Tôi cam đoan đây là công trình nghiên cứu của riêng tôi. Các số liệu, kết quả nêu trong Luận văn là trung thực và chưa từng được ai công bố trong bất công trình nào khác. Các thông tin trích dẫn trong Luận văn đã được chỉ rõ nguồn gốc.

Signature de l'étudiant



ANDRÉ Perrault

Table des matières

Remerciements	iv
Résumé	v
1 Présentation de la structure d'accueil	1
1.1 Département IPN	1
1.2 Missions de l'équipe RIV	2
2 Contexte d'étude et problématique	4
2.1 Contexte d'étude	4
2.2 Problématique	5
3 Introduction	5
3.1 Spécification des technologies	6
4 État de l'art	11
4.1 Outil d'orchestration	12
4.2 Différents outils qui composent une architecture d'orchestration	13
4.3 Tableau comparatif des orchestrateurs	13
4.4 Tableau comparatif des overlays	14
4.5 Tableau comparatif entre les interfaces utilisateur	15
4.6 Tableau comparatif des outils d'automatisation	17
5 Solution proposée	19
5.1 Architecture proposée	20
6 Implémentation	21
6.1 Environnement de travail	21
6.2 Étapes de l'implémentation	22
6.2.1 Installation d'OpenVPN et d'OpenSSL	23
6.2.2 Configuration d'OpenVPN	23
6.2.3 Installation des outils nécessaires	24
7 Résultats	30
8 Conclusion et perspectives	47

Table des figures

1	Architecture de Kubernetes	8
2	Tableau comparatif sur les orchestrateurs	14
3	Tableau comparatif sur les overlays	15
4	Tableau comparatif sur les interfaces utilisateur	17
5	Tableau comparatif sur les outils d'automatisation	18
6	Architecture de notre orchestration	20
7	Vue globale de la salle	21
8	Intel NUC	21
9	Tableau Caractéristique des machines Intel NUC	21
10	Livebox Orange	22
11	Représentation de la connexion entre les machines	22
12	Écran de visualisation	22
13	Dashboard indentifiant avec kubeconfig	30
14	Dashboard indentifiant avec token	30
15	Namespaces	31
16	Nodes	31
17	Services	32
18	Déploiements	32
19	Réplique	33
20	Pods	33
21	Rôles	34
22	Secrets	34
23	Master 0	34
24	Master 1	34
25	Master Capacity	35
26	Node1	35
27	Node1	35
28	Node 1 Capacity	36
29	Node 2	36
30	Node 2_1	36
31	Node 2 Capacity	37
32	Capacité du master et des nodes	37

33	Capacité des pods	38
34	Déploiement de l'application	38
35	Exposition du service de l'application	39
36	Mise à l'échelle de l'application	39
37	Représentation de l'application	40
38	Application identification	40
39	Application capacité 1	41
40	Représentation de la mise à l'échelle	42
41	Mise à l'échelle 1	42
42	Mise à l'échelle 2	43
43	Représentation de la mise à l'échelle 2	43
44	Autoscale Representation 3	43
45	Application Health 1	44
46	Node2 hors service	44
47	Pods du node2 hors service	45
48	Pourcentage du système hors service	45
49	Régénération des ressources hors service	45
50	Service de nouveau en marche	45
51	Représentations graphiques du node2 hors service	46
52	Attribution des pods sur le node fonctionnel (node1)	46

Remerciements

Ce travail de mémoire de Master recherche est le résultat de l'engagement de plusieurs personnes qui ont décidé de m'accompagner résolument dans cet exaltant parcours. Je remercie Dieu, qui m'a donné la force, le courage et la persévérande durant ces deux années. Je sais aussi cette occasion pour adresser mes profonds remerciements à toute l'équipe pédagogique, à l'équipe RIV d'Orange, et en particulier à mes encadrants, Mr. Luiz-Otavio Goncalves BURITY et Mr. Anthony LAMBERT qui m'ont beaucoup aidé lors de ce stage ; leurs conseils m'ont permis de cibler et de m'orienter tout au long de ce stage. J'ai eu l'occasion d'être associé à leur travail et d'acquérir de nouvelles connaissances et compétences. Celles-ci me seront fort précieuses pour la réalisation de mes projets à venir. Ainsi, le temps, l'attention, l'intérêt qu'ils ont bien voulu me témoigner n'ont pas été perdus, et ne seront pas perdus. Ils m'ont donné envie de persévérer dans ce métier pour lequel ils ont le plus grand respect. Je possède désormais une expérience du terrain qui me donne des pistes pour m'améliorer.

Un grand merci à ma mère Mme Ange Marie Datus ANDRÉ, à mon père Mr. Perpillan ANDRÉ et à mes frères et soeurs, pour leurs conseils ainsi que pour leur soutien inconditionnel, à la fois moral et économique. Un grand merci aussi à ma très chère amie Mme DO Hanh Thao. Je voudrais profiter de cet espace pour leur exprimer toute ma gratitude et ma reconnaissance.

Je ne pourrais pas terminer sans aussi remercier mes amis qui m'ont bien supporté durant ce long parcours.

Résumé

La gestion habile d'environnement informatique d'entreprise est très complexe et reste l'un des défis à relever pour la plupart des entreprises à l'ère du cloud computing, du big data et de l'internet des objets. Les conteneurs représentent la nouvelle révolution dans le monde de l'informatique avec le cloud. Ainsi pour le monitoring de tous ces sites, Orange a mis en place un matériel spécifique appelée "**Sonde**" en définissant les critères de sécurité, de stratégie de géolocalisation, pour être branchée sur un serveur en facilitant le monitoring de l'ensemble de son réseau en gérant au mieux les performances du réseau, de détecter les comportements malveillants sur le réseau, de déclencher automatiquement des actions facilitant leur migration. Mais en cas de dysfonctionnement ou d'une mauvaise configuration éventuelle, elle doit être remplacée, ainsi renvoyée au centre technique pour réparation, laissant le système sans monitoring. Une grande perte pour l'entreprise en termes de vulnérabilité, d'efficacité et de coût. L'objectif de ce stage était de concevoir et mettre en place une architecture d'orchestration pour les solutions de Monitoring as a Service (MaaS), une application de monitoring virtualisée, fonctionnant en tant que service grâce à notre architecture, qui est mise à disposition via le réseau sur le serveur du site choisi. La solution d'orchestration du MaaS nous a permis de remplacer le système de sondes (matériels fabriqués et configurés par Orange) par une application virtualisée. Ainsi nous réduisons considérablement les coûts en termes de fabrication du matériel, de transport et de main d'œuvre.

Mots clés : Orchestration, virtualisation, sonde, automatisation, monitoring, MaaS.

Abstract : The skillful management of the enterprise IT environment is very complex and remains one of the challenges for most companies in the age of cloud computing, big data and the Internet of Things. Containers represent the new revolution in the IT world with the cloud. For the monitoring of all its sites, Orange has set up a specific hardware called Probe to define security criteria, geolocation strategy, to be connected to a server by facilitating the monitoring of its entire network by managing network performance as effectively as possible, to detect malicious behaviour on the network, and to automatically trigger actions to facilitate their migration. But possibly in case of malfunction or misconfiguration, it must be replaced, thus returned to the technical center for repair, leaving the system unattended. A great loss for the company in terms of vulnerability, efficiency and cost. The objective of this internship was to design and implement an Orchestration architecture for the Monitoring as a Service (MaaS) solutions for a virtualized monitoring application, operating as a service, which is made available via the network on the server of the chosen site. The solution of MaaS orchestration allow us to replace the sensor system (hardware manufactured and configured by Orange) with a virtualized application. In this way we considerably reduce costs in terms of material manufacturing, transport and labour.

1 Présentation de la structure d'accueil

La Direction Orange Labs Networks and Infrastructures (OLN) définit et pilote la stratégie et la politique réseau du groupe. Elle organise et coordonne les travaux d'architecture de bout en bout couvrant les terminaux, les réseaux, les plates-formes de service et le système d'information. Elle assure la prise en compte de la stratégie du groupe, dans les standards (en lien avec des partenaires industriels ou académiques) et dans le choix des produits industriels fait par le groupe. Elle assure aussi la conception des solutions et des briques techniques réseaux pour les pays.

Au sein d'Orange Labs Networks and Infrastructures, la direction Wireline Networks and Infrastructures (WNI), définit et pilote la stratégie d'évolution des réseaux fixes, ainsi que la politique d'équipement du Groupe pour les domaines concernés, en fournissant aux entités des architectures optimisées, en soutenant les entités opérationnelles des filiales dans l'implémentation des évolutions du réseau et en pilotant les travaux de recherche et d'anticipation pour préparer le groupe aux évolutions tendancielles et en rupture sur le réseau.

1.1 Département IPN

L'équipe Routing IP VPN (RIV) est dans un département qui s'appelle IP Networks, Monitoring and Security (IPN). Le département IPN est dans la direction WNI, au sein de la direction OLN. Le département IPN est en charge de définir et piloter la stratégie d'évolution du domaine IP vers des réseaux de 5e génération convergents :

- en agençant les éléments matériels et logiciels.
- en contribuant aux travaux prenant en compte les aspects technico-économiques.
- en tirant profit de l'évolution des techniques de la virtualisation et d'automatisation pour les réseaux IP.
- en proposant des mécanismes innovant afin d'atteindre une expérience client inégalée.
- en contribuant aux projets pays sur le court et moyen terme.

IPN a pour missions de :

- Définir des solutions innovantes d'évolutions des réseaux IP et d'accompagner les pays dans l'introduction de ces évolutions.
- Définir la politique, la stratégie et la gestion technique des fournisseurs de solutions IP, matérielles et logicielles, pour le transport et les services avancés IP.

- Maîtriser la fraude et la sécurité des infrastructures et du transport des données (pare-feux, chiffrement, filtrage des attaques DDoS, contrôle des usages ...) de nos clients, en dépit des attaques réseaux.
- Assurer la transformation du domaine IP au travers de la softwarisation et de l'automatisation des fonctions et des réseaux IP (interface-nord du matériel et solutions logicielles IPSDN).
- Assurer une expérience client optimale, en travaillant les angles du monitoring, de la QoS, de la fraude, du trouble-shooting et de l'ingénierie de trafic.
- Évaluer et valider les équipements IP et les contrôleurs SDN, ainsi que leurs nouvelles déclinaisons, pour répondre aux besoins futurs des pays.

1.2 Missions de l'équipe RIV

L'équipe RIV, a pour missions principales, d'être le spécialiste du **plan de contrôle et des protocoles de routage des réseaux IP/MPLS**, de son **monitoring**, des mécanismes de **sécurité** associés et de préparer les évolutions de son **automatisation**. L'équipe apporte un **support aux pays** du groupe Orange sur quatre domaines techniques.

* **Le plan de contrôle des réseaux IP/MPLS**, en fournissant une expertise sur les évolutions des protocoles de routage et des fonctions du plan de contrôles sur les routeurs IP.

- Réaliser des travaux de recherche et d'anticipation autour du routage du plan de contrôle des réseaux IP/MPLS, à travers d'une présence continue dans les structures de normalisation comme IETF et l'ISOC.
- Être responsable de l'ingénierie, de la spécification et de la qualification des nouvelles fonctions du routage IP/MPLS et VPN/MPLS et de leur virtualisation et de leur activation, ainsi que de leur déclinaison aux différents éléments réseaux retenus par le groupe.
- Maintenir un niveau de compétences fonctionnelles (architecture) et organiques (équipements réseaux) permettant de renforcer les recommandations fournies par l'équipe.

* **Le monitoring**, en concevant et en mettant en place des solutions de Monitoring as a Service (Maas), à travers le développement de chaînes de monitoring du plan de contrôle des réseaux IP/MPLS.

- Concevoir et valider les mécanismes permettant la détection de comportements malveillants ainsi que le déclenchement automatique d'actions permettant leur mitigation.
- Fonctionner en mode « Devops » pour la conception et la mise en place de solutions de Monitoring as a Service (Maas) permettant d'assurer :
 - le déploiement, de manière agile et efficace, à distance, d'applications de monitoring au sein d'environnements virtualisés.
 - la collecte, l'analyse et le stockage des données issues des applications de monitoring déployées.
 - la visualisation de ces données, la remontée d'alarmes et la génération automatisée de dashboards.
- Utiliser et promouvoir des outils sur étagère comme le « Route-Explorer », pour renforcer notre capacité d'auditer les protocoles de routage IP et de fournir les recommandations d'optimisation pertinentes.

* **L'automatisation des réseaux IP**, en contribuant à la réflexion et en participant à l'expérimentation d'outils d'automatisation de la configuration des réseaux IP, et de sa déclinaison sur le WAN.

- Accompagner l'évolution des protocoles entre l'interface-nord des routeurs et l'Infrastructure.
- Monter des « Proof of Concept » (POC) pour valider et explorer ces nouvelles approches.
- Réaliser l'évaluation et la qualification des outils basés sur des protocoles normalisés comme Netconf, BGP FlowSpec,... ainsi que sur du data modeling à travers des langages comme YANG,...

* **La sécurité**, en pilotant le Fixed-Security-Center (FSC) et en développant les activités d'Anticipation et de Delivery couvrant la totalité de nos trois domaines techniques (plan de contrôle, monitoring et automatisation).

- Contribuer à la réflexion globale OLN sur la sécurité des réseaux et des infrastructure fixes. NTG, structure admin (réf ITS, MSC, FSC...) plan de renforcement de la sécurité, action ponctuelle.
- Développer les activités du domaine de la sécurité en couvrant nos domaines techniques. **Exemple** : sécurisation BGP, IS-IS, l'Internet, DDOS, les AFR...
- Assurer la qualification et la recommandation des solutions vers les filiales : MANRS, (question sur les dossiers stratégiques).

- Décliner les recommandations fonctionnelles en recommandations d'ingénierie et cela en couvrant les principaux constructeurs supportés par les Skill Centers.

* **Le support pays**, en soutenant les pays du groupe Orange, pour améliorer la qualité des réseaux en matière de robustesse, de résilience et de sécurité.

1. Contribuer à l'amélioration de la qualité des réseaux IP/MPLS des pays du groupe Orange en fournissant une expertise avancée en architecture et en ingénierie pour assurer une meilleure robustesse, la résilience (convergence) et la sécurité.
2. Mener des audits et formuler des recommandations auprès des pays durant la phase « Think », accompagner la mise en œuvre du « BUILD » et analyser « post-mortem » des problèmes opérationnels.

2 Contexte d'étude et problématique

2.1 Contexte d'étude

Pour le monitoring de tous ses sites, Orange a mis en place un matériel spécifique, constitué d'une petite boîte, appelée « Sonde ». Une sonde est configurée et envoyée à un des sites choisis de l'entreprise selon des critères de sécurité, de stratégie ou de géolocalisation, afin d'être branchée sur un serveur pour contribuer au monitoring de l'ensemble du réseau. Le monitoring permet de mieux gérer les performances du réseau, de détecter les comportements malveillants sur le réseau ainsi que de déclencher automatiquement des actions facilitant leur migration. La sonde est une machine à part entière, ayant son propre SE et nécessitant d'être branchée. En cas d'éventuel problème sur une sonde, comme un dysfonctionnement, une mauvaise configuration ou encore une panne, elle doit être rapidement remplacée, donc être renvoyée au plus vite au centre technique pour vérification ou réparation. Ce qui peut prendre des jours, laissant le système sans monitoring. Ce qui peut parfois s'avérer dangereux pour le système et surtout pour l'entreprise en termes de vulnérabilité, d'efficacité et de coût.

L'équipe RIV, dont je fais partie, a eu dans le cadre de l'évolution de ce système de sondes, l'objectif de concevoir et de mettre en place des solutions de **Monitoring as a Service (MaaS)**. Un MaaS est une application de monitoring virtualisée, fonctionnant en tant que service, qui est mise à disposition via le réseau sur le serveur du site choisi.

Cette solution de MaaS a atteint son but en remplaçant le système de sondes (matériels fabriqués et configurés par Orange) par une application virtualisée.

Ce qui réduit considérablement les coûts en termes de fabrication du matériel, de transport et de main d'œuvre, car la sonde n'existe plus.

2.2 Problématique

La problématique est la mise en place de l'architecture, qui permettra de gérer toutes les machines sur des réseaux hétérogènes des sites distants de l'entreprise, et qui devra être capable de faire le déploiement à distance et la gestion des applications de monitoring. Afin de déployer ces applications de manière agile, efficace et sécurisée, et afin d'assurer leur forte disponibilité au sein d'environnements virtualisés sur tous les sites distants de l'entreprise, les trois tâches suivantes seront entreprises :

Tâche 1 : Créer un réseau privé et sécurisé entre les différentes machines des sites distants, situées sur des réseaux hétérogènes.

Tâche 2 : Mettre en place une architecture d'orchestration, qui soit open source, sécurisé et utilisable en entreprise, visant à répondre à tous les critères précédents.

Tâche 3 : Faire fonctionner l'architecture d'orchestration, qui permettra aux applications de monitoring dockerisées de fonctionner sur le même principe des micro-services, en les déployant sur les différentes machines, toutes reliées entre elles et coordonnées via un serveur central.

3 Introduction

En informatique, les micro-services sont un style d'architecture logicielle à partir duquel un ensemble complexe d'applications est décomposé en plusieurs processus indépendants et faiblement couplés, souvent spécialisés dans une seule tâche.

Avec l'évolution des micro-services et des conteneurs, ces dernières années, la manière dont nous concevons, développons et exécutons des logiciels, a considérablement évolué. Les nouvelles applications modernes sont optimisées pour l'évolutivité, l'élasticité, la rapidité des changements et contre la défaillance. Entraînées par de nouveaux principes, ces architectures modernes nécessitent un ensemble différent de modèles et de pratiques à appliquer.

Les grandes sociétés de l'Internet telles que Google[1], Twitter, Amazon, Netflix, eBay ... ont déjà déployés des applications dans des conteneurs depuis plusieurs années. Bien que le format de conteneur Docker constitue une solution élégante pour créer des applications en tant que micro-services, il ne comprend cependant pas tous les éléments nécessaires, au déploiement et à la gestion de nombreux conteneurs devant fonctionner ensemble et évoluer au fur et à mesure que la demande augmente.

Au moment où, chacune de ces entreprises commençait à créer ses propres outils de déve-

loppelement, de déploiement, de gestion et d'extension des applications conteneurisées, elles se sont rendu compte que le travail était trop important, pour que les plus grandes entreprises puissent s'y attaquer seules. Certaines entités (personnes, entreprises ...) construisant leurs propres outils de gestion de conteneurs propriétaires et de nombreux projets open source démarrant pour résoudre des problèmes de différentes manières, menaçaient sérieusement de fragmenter le mouvement naissant de conteneurs.

Google a publié Kubernetes en tant que projet open source dans le but de standardiser la gestion des applications conteneurisées. Kubernetes fait office de superposition au-dessus du centre de données d'une entreprise. Et d'autres projets open source pourraient résoudre d'autres aspects de la gestion des conteneurs (comme la création de systèmes d'exploitation rationalisés ou d'outils d'administration graphique).

En juillet 2015[4], Kubernetes a franchi la barre de 1.0, indiquant que le projet était prêt à être utilisé en production. Avec Kubernetes 1.0, Google a également formé la Cloud Native Computing Foundation (CNCF). Crée en tant que projet collaboratif en association avec la Linux Foundation¹, CNCF a pris un bon départ. Parmi ceux qui rejoignent la CNCF, on trouve des sociétés comme RedHat, eBay, AT&T, Cisco, IBM, Intel, Twitter, Mesosphere, VMware, etc.

L'objectif, ici, est de vous présenter Kubernetes et ses fonctionnalités et de décrire comment il peut améliorer l'expérience du déploiement des conteneurs au format Docker. Embarqué sur un projet impliquant Kubernetes, cela vous donne la possibilité d'utiliser les mêmes outils pour le déploiement des conteneurs, choisis actuellement par les plus grandes et les plus performantes entreprises.

Kubernetes est une plate-forme d'orchestration de conteneurs. L'origine de Kubernetes se situe quelque part dans les centres de données de Google, à partir de la plate-forme interne de Google, Borg. Google utilise Borg depuis de nombreuses années pour orchestrer des applications exécutées dans un format de conteneur personnalisé. En 2014, Google a décidé d'ouvrir Borg en tant que «Kubernetes» (en grec pour «helmsman» ou «pilote») et en 2015, Kubernetes a été le premier projet donné à la Cloud Native Computing Foundation (CNCF).

3.1 Spécification des technologies

- Kubernetes[3] est un gestionnaire de clusters pour les conteneurs Linux. Alors que Kubernetes peut prendre en charge plusieurs types de conteneurs tels que Rocket,

1. <http://collabprojects.linuxfoundation.org>

runC ou containerd, s'ils sont au préalablement déclarés, nous discuterons de Kubernetes uniquement dans le contexte des conteneurs de type Docker.

Docker est une plate-forme de virtualisation de conteneur open source permettant de créer, de regrouper et d'exécuter des applications distribuées dans des conteneurs, qui constituent des instantanés légers du système d'exploitation sous-jacent. Une image Docker, spécifique à l'application, encapsule tous les logiciels requis, y compris les dépendances d'une application, et permet de créer des conteneurs Docker pour exécuter des applications dans les conteneurs. Les conteneurs Docker isolés les uns des autres, disposent de leur propre réseau et système de fichiers et fournissent un conteneur en tant que service (CaaS). Docker est similaire aux machines virtuelles basées sur des plateformes de virtualisation telles qu'**Oracle VirtualBox** et **VMWare**, en ce sens il s'agit d'une virtualisation sur le système d'exploitation sous-jacent. En revanche une machine virtuelle utilise un système d'exploitation complet et s'exécute séparément sur le système d'exploitation hôte.

Les **conteneurs Docker** ont introduit un nouveau niveau de modularité et de fluidité pour les applications, avec la possibilité de regrouper les applications, y compris les dépendances, et de transférer et d'exécuter les applications dans différents environnements. Mais avec l'utilisation des conteneurs Docker en production et leur mise en pratique, des problèmes sont apparus :

- Planification et répartition impossibles, car le conteneur ne peut pas être exécuté sur différents nodes.
- Mise à l'échelle impossible, car impossibilité d'augmenter / diminuer le nombre de conteneurs en cours d'exécution pour une application.
- L'isolement des conteneurs ne permet pas la communication entre eux.

Kubernetes a été conçu pour surmonter tous ces problèmes et pour offrir d'autres aspects pratiques à la gestion des grappes de conteneurs.

Kubernetes fournit une orchestration dynamique des clusters de conteneurs en temps réel.

1. Composantes de Kubernetes :

- **Node** : c'est une machine physique ou virtuelle exécutant Kubernetes et sur laquelle des modules peuvent être programmés. Le node peut être master (noeud maître), ou worker (noeud travailleur).
- **Node master** : c'est le node sur lequel les modules de contrôle Kubernetes sont en cours d'exécution (notez que rien n'empêche un node master d'être aussi un node worker si les ressources le permettent).

- **Node worker** : c'est un node utilisé uniquement pour déployer et lancer les conteneurs d'applications.

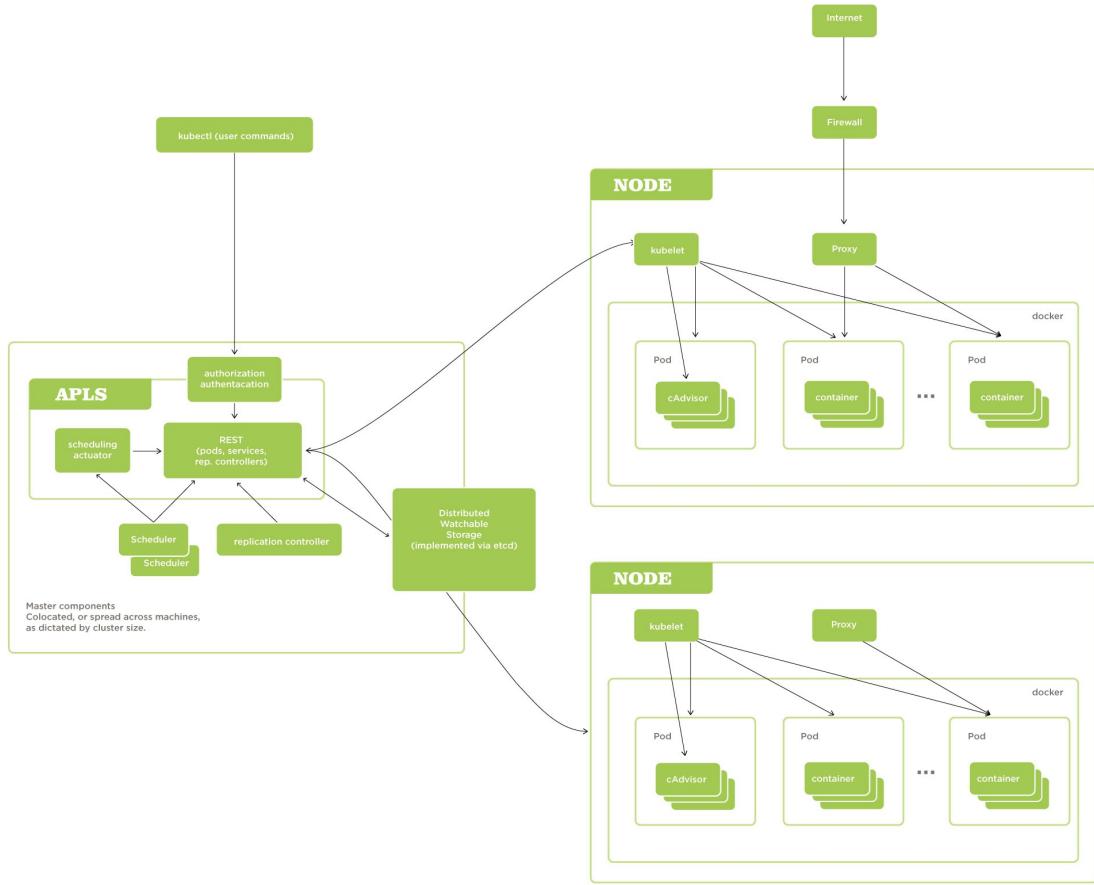


FIGURE 1 – Architecture de Kubernetes

- **Cluster** : c'est un ensemble de nodes et de ressources, telles que le stockage, pour exécuter des applications Kubernetes. Un cluster Kubernetes possède généralement un seul master et zéro ou plusieurs workers, tandis qu'un cluster **hautement disponible** se compose de plusieurs masters qui peuvent répartir la charge entre eux ou servir à assurer la tolérance de panne. A la fin, dans le cas d'un cluster hautement disponible, un seul master se comportera en master chef et les autres masters en workers.
- **Pod** : c'est un ensemble de conteneurs qui sont colocalisés et qui forment une unité atomique. Plusieurs applications peuvent être exécutées dans un pod et bien que les différents conteneurs d'un pod puissent être destinés à la même application, les différents conteneurs sont généralement destinés à des applications différentes. Un pod est une abstraction de niveau supérieur pour gérer un groupe de conteneurs avec des volumes partagés et un espace de noms (namespace). Toutes les applications (conteneurs) d'un pod partagent le même système

de fichiers et la même adresse IP avec les différents ports sur lesquels chaque application est exposée.

Les applications exécutées dans un pod peuvent accéder au «localhost». La planification et la réPLICATION sont effectuées au niveau du pod plutôt qu'au niveau du conteneur individuel. Par exemple, si un pod définit deux conteneurs pour deux applications différentes et que le niveau de réPLICATION est défini à 1, une seule réPLIQUE du pod se compose de deux conteneurs, un pour chacune des deux applications.

Les pods facilitent le partage des ressources et la communication, ce qui aurait été implémenté autrement à l'aide du paramètre (`-link`) dans des conteneurs Docker exécutés individuellement. Un pod composé de plusieurs conteneurs est généralement utilisé pour des applications étroitement couplées.

Cas d'exemple : si une application Nginx utilise la base de données MySQL, les deux applications peuvent interagir avec les conteneurs exécutant chacun dans le même pod sur kubernetes.

- **Service :** c'est une interface externe d'un ou de plusieurs pods fournissant le ou les points de terminaison (endpoints) par lesquels les applications représentées par le service peuvent être appelées. Un service est hébergé sur une seule adresse IP mais fournit zéro ou plusieurs points de terminaison (endpoints) en fonction des applications appelées et interfacées par le service. Les services sont connectés aux pods à l'aide de sélecteurs d'étiquettes (label selector). Les pods portent les étiquettes et un service, avec une expression de sélecteur identique à l'étiquette du pod, représentant ce pod pour un client interne. Un client externe ne sait pas ou n'a pas besoin de connaître les pods représentés par un service. Un client externe doit uniquement connaître le nom du service et le port auquel une application particulière est exposée. Le service achemine les demandes pour une application vers l'un des modules sélectionnés à l'aide d'un sélecteur d'étiquette (label selector). Ainsi, un service est une abstraction de haut niveau pour un ensemble d'applications pour acheminer une requête vers le service, sans se focaliser sur le détail de chaque pod. Un service peut également être utilisé pour l'équilibrage de charge (LoadBalancer).
- **Replication Controller :** gère le niveau de réPLICATION des Pods comme spécifié par le paramètre «replicas» dans une définition de Replication Controller ou en ligne de commande avec le paramètre "replicas". Un Replication Controller s'assure que le niveau configuré des réPLICAS Pod s'exécute. Si une réPLIQUE échoue ou est arrêtée délibérément, une nouvelle réPLIQUE est démarrée automatiquement. Un Replication Controller est utilisé pour redimensionner les modules dans un cluster.

N.B. Un replica est défini au niveau du pod, ce qui implique que si un pod

est constitué de deux conteneurs, la réplique se constituera d'un pod de deux conteneurs identique au premier.

- **Label** : c'est une paire clé-valeur identifiant une ressource telle qu'un pod, un service ou un contrôleur de réPLICATION : le plus souvent un pod. Les étiquettes (label) permettent d'identifier un groupe ou un sous-ensemble de ressources pour des tâches telles que leur affectation à un service. Les services utilisent des sélecteurs d'étiquettes pour sélectionner les modules qu'ils gèrent. Par exemple, si un pod est étiqueté «app = helloApp» et qu'un sélecteur de service est défini sur «app = helloApp», le pod est présenté par ce service.

Les sélecteurs de service sont basés sur des étiquettes et non sur le type d'application qu'ils gèrent. Par exemple, un service peut présenter un pod exécutant un conteneur d'application hello-world avec une étiquette spécifique. Un autre pod exécutant également un conteneur hello-world mais avec une étiquette différente de celle du sélecteur de service ne serait pas présenté par le service. Et un troisième pod exécutant une application qui n'est pas une application hello-world mais qui porte le même libellé que le sélecteur de service serait également présenté par le même service.

- **Selector** : c'est une expression clé-valeur (key-value) permettant d'identifier les ressources à l'aide d'étiquettes (label) correspondantes. Comme indiqué dans la sous-section précédente, une expression de sélecteur de service «app = helloApp» sélectionnera tous les pods avec l'étiquette «app = helloApp». Tandis qu'un service définit généralement un sélecteur pour sélectionner des modules, un service peut être défini pour ne pas inclure de sélecteur et être défini pour abstraire d'autres types d'arrière-plans. Deux types de sélecteurs sont pris en charge : basés sur l'égalité ou sur des ensembles. Un sélecteur pourrait être composé de plusieurs exigences, ce qui implique que plusieurs expressions (basées sur l'égalité ou sur des ensembles) séparées par ',' pourraient être spécifiées. Toutes les exigences doivent être satisfaites par une ressource correspondante, pour quelle soit sélectionnée. Une ressource telle qu'un pod peut avoir des étiquettes (label) supplémentaires, mais le sélecteur doit spécifier au moins un des label de la ressource pour qu'elle soit sélectionnée. Le sélecteur d'égalité, plus communément utilisé, supporte les opérateurs `=, !=, ==` (= équivalent de `==`).
- **Name** : sert à identifier une ressource mais est différent d'un label, qui est utilisé pour faire correspondre une ressource à un service.
- **Namespace** : Un namespace est un niveau au-dessus du nom pour délimiter un groupe de ressources pour un projet ou une équipe afin d'empêcher les collisions de noms. Les ressources des différents namespaces peuvent avoir le même nom, mais les ressources d'un même namespace ont des noms différents.

- **Volume** : Un volume est un répertoire dans le système de fichiers d'un conteneur. Un volume est utilisé pour stocker des données. Les volumes Kubernetes s'appuyant sur des volumes Docker, évoluent selon les volumes Docker.
- **Réseau overlay (superposé)** : La communication réseau entre les conteneurs est la partie la plus difficile. Parce que lorsque vous exécutez Docker, une adresse IP est attribuée dynamiquement à chaque conteneur ; les applications conteneuriées nécessitant de communiquer entre elles doivent connaître l'adresse IP et le numéro de port conjointement et réciproquement. Ce qui est permis par le réseau overlay. Si le réseau de communication du conteneur se trouve uniquement dans l'hôte, vous pouvez utiliser le paramètre `--link` dans Docker pour générer les variables d'environnement afin de découvrir les conteneurs voisins. Cependant, Kubernetes fonctionne généralement comme un cluster et un modèle d'ambassadeur, où un réseau superposé pourrait aider à connecter chaque node. Kubernetes utilise un réseau superposé pour gérer la communication entre les pods sur différents nodes.

Kubernetes en tant que gestionnaire de clusters offre les avantages suivants.

2. Avantages de Kubernetes

- Fonctionne selon une architecture de micro-services, en décomposant une application en composants plus petits, gérables et évolutifs, pouvant être utilisés par des groupes ayant des besoins différents.
- Permet de créer un cluster, offrant la tolérance aux pannes, appelée selfhealing. Si une réplique d'un pod (un conteneur ou un groupe de conteneurs) échoue (par exemple, en raison d'une défaillance d'un node), une autre réplique est lancée automatiquement.
- Fournit la mise à l'échelle horizontale la plus rapide et la moins coûteuse possible en répartissant les conteneurs Docker sur plusieurs hôtes, ce qui réduit le besoin de nouvelles ressources matérielles lors de la mise à l'échelle.
- Utilise efficacement et optimalement les ressources.
- Permet la séparation des préoccupations de l'équipe de développement de service de celles de l'équipe d'infrastructure de clusters.
- Permet d'augmenter la tolérance aux pannes et de minimiser les temps d'arrêt en lançant des conteneurs sur différentes machines.

4 État de l'art

Les environnements informatiques d'entreprise sont très complexes à gérer et constituent un défi majeur ; une grande partie des entreprises devant faire face à l'ère de l'IoT, du

cloud computing et le Big Data. Ainsi, les systèmes d'automatisation et d'orchestration avancés sont les solutions mises en oeuvre pour aider le personnel informatique à gérer des centres de données cloud à grande échelle. Les conteneurs sont une révolution dans le monde du cloud computing, ils sont plus légers que les machines virtuelles (VM) et peuvent réduire considérablement le temps de démarrage des instances et le traitement, ainsi que la surcharge de stockage par rapport aux machines virtuelles traditionnelles. Dans ce chapitre nous donnerons une description complète des approches d'orchestration avec des conteneurs, analyserons les efforts de recherche actuels sous forme de tableaux et étudierons les solutions existantes mises en oeuvre. L'orchestration peut être définie comme l'arrangement, la coordination et la gestion automatique des systèmes informatiques, des logiciels intermédiaires et des services. L'orchestration tire parti de plusieurs tâches automatisées afin d'exécuter automatiquement un flux de travail ou un processus plus important, pouvant impliquer plusieurs systèmes.

- L'objectif de l'orchestration est de rationaliser et d'optimiser les processus fréquents et reproductibles afin de garantir un déploiement plus rapide et plus précis des logiciels, car les entreprises savent que, plus le délai de commercialisation est court, plus les chances de succès sont grandes. Chaque fois qu'un processus est répétable et que ses tâches peuvent être automatisées, l'orchestration peut être utilisée pour optimiser le processus afin d'éliminer les redondances.

4.1 Outil d'orchestration

À mesure que la vitesse devient de plus en plus vitale, les organisations continuent à adopter de plus en plus d'outils pour se donner un avantage. En effet, au fur et à mesure que l'entreprise se développe, elle peut absorber de nouvelles entreprises avec leurs différents ensembles d'outils et de processus. Très rapidement, cela peut mener au phénomène connu sous le nom de «développement d'outils». La prolifération des chaînes d'outils apporte un éventail de risques et de défis en matière de gestion, notamment un manque de contrôle et de visibilité. Et, si vos outils ne sont pas correctement orchestrés, ils peuvent vous ralentir et nuire à la qualité de vos versions, contrairement à vos intentions. Les outils d'orchestration s'intègrent à vos systèmes existants, ce qui vous permet de gérer et d'aligner votre chaîne d'outils de manière efficace et transparente, le tout à partir d'une interface centralisée.

Les outils d'orchestration[6] connectent des silos disparates et isolés en un processus harmonieux entièrement automatisé. Ils offrent un contrôle et une visibilité, vous permettant de gérer un cycle de vie des versions, par ailleurs, de plus en plus complexe et de fournir une stratégie de livraison continue cohérente pour votre organisation. En outre, ils prennent en charge des initiatives de livraison continue avec des niveaux supplémentaires de fiabilité, d'audibilité et d'évolutivité, ajoutant à la fois rapidité et qualité à toutes vos versions. Ceci est réalisé en supprimant les processus manuels et leurs contrôles, élimi-

nant ainsi une grande partie de la gestion fastidieuse qui peut consommer des membres importants d'une équipe et afin de les libérer pour qu'ils soient à la fois plus productifs et innovants. En fin de compte, les outils d'orchestration apportent une plus grande collaboration à une organisation, permettant une facilité et une agilité.

4.2 Différents outils qui composent une architecture d'orchestration

Dans une architecture d'orchestration nous trouvons, un orchestrateur qui assure la gestion du système, un dashboard qui permet de visualiser tous les composants du système afin de mieux le gérer, un overlay qui offre la possibilité aux pods de se communiquer entre eux via un réseau virtuel, créé par l'overlay et aussi un outil d'automatisation pour automatiser toutes les tâches, les installations, les configurations et le déploiement.

4.3 Tableau comparatif des orchestrateurs

Nous avons défini un tableau de comparaison entre quelques orchestrateurs tirés d'une liste d'orchestrateur. Quatre orchestrateurs, dont **kubernetes**[22], parmis les plus répandus sur le marché, les plus puissant et les plus utilisés de nos jours dans les entreprise, sont présents dans le comparatif.

1. **HashiCorp Nomad** : c'est un fichier binaire unique qui programme des applications et des services sur Linux, Windows et Mac. C'est un planificateur open source qui utilise un fichier de travail déclaratif pour planifier des applications virtualisées, conteneurisées et autonomes.
2. **Mesosphere** : c'est une solution logicielle qui étend les capacités de gestion de grappes d'Apache Mesos à des composants supplémentaires pour fournir une nouvelle méthode novatrice pour la gestion des infrastructures de serveur. En combinant plusieurs composants avec Mesos, tels que **Marathon** et **Chronos**, Mesosphere permet de mettre facilement la scabilité des applications en éliminant un grand nombre des défis associés à la mise à l'échelle. Mesosphere fournit des fonctionnalités telles que la planification des applications, la mise à l'échelle, la tolérance aux pannes et l'auto réparation. Il fournit également la découverte du service d'application, l'unification des ports et l'élasticité des points d'extrémité.
3. **Docker Swarm[17]** : c'est un outil de clustering et de planification pour les conteneurs Docker. Avec Swarm, les administrateurs informatiques et les développeurs peuvent créer et gérer un cluster de noeuds Docker en tant que système virtuel unique. Le mode Swarm existe également en mode natif pour Docker Engine, la couche entre le système d'exploitation et les images de conteneur. Le mode Swarm intègre les capacités d'orchestration de Docker Swarm dans Docker Engine 1.12 et les versions plus récentes.

	Nomad	Mesos Marathon	Swarm	Kubernetes
Crée par	Hashicorp	Mesosphere, Microsoft	Docker	Linux Fondations, Google
Année	2015	2011	2013	2014
Type	Conteneur d'orchestration et planification	Conteneur d'orchestration et gestion d'infrastructure	Conteneur d'orchestration et planification	Conteneur d'orchestration et planification
Réseau virtuel	Calico, weave, flannel, etc.	Calico, weave, flannel, etc.	Calico, weave, flannel, etc.	Calico, weave, flannel, etc.
Critères importants				
Niveau de complexité d'installation et de configuration	Moyen	Moyen	Simple	Complexe
Contributeurs	251	259	165	1798
Support pour entreprise	Oui	Oui	Oui	Oui
Plusieurs Datacenter	Oui	Oui	Docker Cloud, Docker Machine Drivers, Docker for AWS/Azure	Oui
Support Bare Metal	Oui	Oui	Oui	Oui
Gestion des secrets	vault	Oui	Oui	Oui
Montage de volume	-	Oui	Oui	Oui
On-premise	-	Oui	Oui	Oui
Clusters	Jusqu'à 5000 nodes	Jusqu'à 10000 nodes	Jusqu'à 4700 nodes	Jusqu'à 5000 nodes
Pods	-	-	-	Jusqu'à 150000
Conteneurs	Jusqu'à 1000000	-	Jusqu'à 50000	Jusqu'à 300000

FIGURE 2 – Tableau comparatif sur les orchestrateurs

A l'aide de ce tableau comparatif, nous pouvons faire le choix de notre orchestrateur, tout en nous basant sur les critères importants que nous avons définis. Ainsi, nous constatons que les caractéristiques de **Kubernetes** répondent parfaitement à nos exigences.

4.4 Tableau comparatif des overlays

Nous avons dressé un tableau comparatif pour les overlay[17] à partir duquel nous allons sélectionner l'overlay le plus adapté pour notre architecture.

- **Weave** : prend en charge un réseau superposé pouvant couvrir différentes configurations de mise en réseau cloud, simplifiant ainsi l'exécution des charges de travail héritées de Kubernetes. Par exemple, Weave prend en charge la multidiffusion, même lorsque le réseau sous-jacent ne le fait pas. Weave peut configurer la mise en réseau VPC sous-jacente et ignorer la superposition lors de l'exécution sur AWS. Ce fournisseur forme un réseau maillé d'hôtes partitionnables et éventuellement cohérents ; ce qui signifie que la configuration est quasiment nulle, et qu'il n'a pas besoin de compter sur une sauvegarde externe. Weave prend en charge le cryptage et la stratégie réseau de Kubernetes, garantissant ainsi sa sécurité au niveau du réseau.
- **Flannel** : est un moyen simple et facile de configurer une structure de réseau de couche 2, conçue pour Kubernetes. Flannel fonctionne de manière simple, n'utilise aucune base de données externe mais utilise l'API Kubernetes ou VXLAN par défaut. Il peut être superposé avec le moteur de politique Calico (Projet Canal).

- **Calico[32]** : fournit un réseau simple et évolutif utilisant une approche de couche 3 pure. Il permet une mise en réseau, non encapsulée et prise en charge nativement par des environnements comme AWS, AZURE ou dans d'autres environnements avec contiguïté de couche 2 ainsi bien entre les nœuds que dans les déploiements d'un réseau comparable à celui d'une infrastructure BGP. Calico fournit également un mode IP-in-IP sans état qui peut être utilisé dans d'autres environnements, si nécessaire. Au-delà de la mise en réseau évolutive, Calico offre également une isolation des règles, ce qui vous permet de sécuriser et de gérer votre infrastructure de microservices/conteneurs à l'aide de stratégies d'entrée et de sortie avancées.
- **Canal** : est un fournisseur CNI qui vous offre le meilleur de Flannel et de Calico, fournissant un réseau VXLAN simple et facile à utiliser, tout en vous permettant de tirer parti de l'isolation des stratégies avec les stratégies Calico. Ce fournisseur est une solution pour tous ceux qui souhaitent être opérationnels tout en profitant des technologies familières qu'ils utilisent déjà.

	Docker Overlay Network	Flannel	Weave	Calico	Canal
Modèle réseau	VxLAN	VxLAN or UDP Channel	VxLAN or UDP Channel	Pure Layer-3 Solution	Flannel + Calico
Isolation d'application	CIDR Schéma	CIDR Schéma	CIDR Schéma	Profile Schéma	
Protocol Support	Tout	Tout	Tout	TCP, UDP, ICMP & ICMPv6	
Nom de service	Non	Non	Oui	Nom	
Exigences de stockage distribué	Oui	Oui	Non	Oui	
Chaîne de cryptage	Non	TLS	NaCI Library	Oui	
Support réseau partiellement connecté	Non	Non	Oui	Non	
vNIC séparé pour le conteneur	Non	Non	Oui	Non	
Prise en charge du chevauchement IP	Oui	Oui	Oui	Non	
Restriction de sous-réseau de conteneur	Oui, configurable après démarrage	Non	Oui, configurable après démarrage.	Non	
Approche	Overlay	Routing, overlay	Overlay	Routing	
Spécification	CNM	CNI, CNM	CNI, CNM	CNI, CNM	
Sauvegarde de données externe	Aucun	Aucun	Aucun	Etd	

FIGURE 3 – Tableau comparatif sur les overlays

A l'aide de ce tableau comparatif, nous pouvons faire le choix de notre overlay, tout en nous basant sur les critères importants que nous avons définis. Ainsi, nous constatons que les caractéristiques de **Calico** répondent parfaitement à nos exigences.

4.5 Tableau comparatif entre les interfaces utilisateur

1. **Panamax** : c'est un créateur d'applications conteneurisées avec un marché d'applications open source hébergé dans GitHub. Panamax fournit une interface conviviale

pour les utilisateurs de Docker, Fleet & CoreOS. Avec Panamax, vous pouvez facilement créer, partager et déployer n'importe quelle application conteneurisée, quelle que soit sa complexité.

2. **Kitematic** : c'est un projet open source conçu pour simplifier et rationaliser l'utilisation de Docker sur un PC Mac ou Windows. Kitematic automatise le processus d'installation et de configuration de Docker et fournit une interface utilisateur graphique intuitive pour exécuter les conteneurs Docker. Kitematic s'intègre à Docker Machine pour mettre en service une machine virtuelle VirtualBox et installer le moteur Docker localement sur votre machine.
3. **Shipyard** : c'est un outil Web basé sur Docker Swarm qui vous permet de gérer les ressources Docker, notamment les conteneurs, les images, les registres privés..., via un outil graphique convivial. Il peut être construit via une image Docker, donc pour "installer" Shipyard, vous utilisez Docker ou Vous pouvez vous y prendre à la dure, en l'installant manuellement.
4. **DockStation** : c'est une application centrée sur les développeurs pour la gestion de projets basés sur Docker. Au lieu de nombreuses commandes CLI, vous pouvez surveiller, configurer et gérer les services et les conteneurs en utilisant simplement une interface graphique.
5. **Kubernetes Dashboard** : c'est une interface utilisateur Web Kubernetes. Vous pouvez aussi utiliser le Dashboard pour déployer des applications conteneurisées sur un cluster Kubernetes, dépanner votre application conteneurisée et gérer le cluster avec ses ressources associées. Vous pouvez utiliser Dashboard pour obtenir une vue d'ensemble des applications exécutées sur votre cluster, ainsi que pour créer ou modifier des ressources Kubernetes individuelles (telles que Deployments, Jobs, DaemonSets, etc.). Par exemple, vous pouvez mettre à l'échelle un déploiement, lancer une mise à jour progressive, redémarrer un pod ou déployer de nouvelles applications à l'aide d'un assistant de déploiement. Kubernetes Dashboard fournit également des informations sur l'état des ressources de votre cluster et sur les erreurs éventuelles.
6. **Rancher[27]** : c'est un outil qui permet d'élargir l'interface graphique de Docker et qui est utile pour les clusters de production. Reflétant cet intérêt pour la production, Rancher est conçu pour fonctionner sur des machines Linux, donc pour le tester sur d'autres systèmes, vous devrez l'installer sur une machine virtuelle. Rancher offre les fonctionnalités natives de Kubernetes via kubectl, mais propose également son propre tableau de bord pour vous permettre de déployer Kubernetes facilement, en évitant une surcharge de travail et sans avoir à comprendre toute la complexité de Kubernetes.

	Panamax	Kitematic	Shipyard	DockStation	Portainer	Kubernetes Dashboard	Rancher
Gérer les conteneurs docker	Oui	Oui	Oui	Oui	Oui	Oui	Oui
Accéder à la console des conteneurs	Non	Oui	Oui	Oui	Oui	Oui	Oui
Gérer les images de docker	Oui	Oui	Oui	Oui	Oui	Oui	Oui
Tag et push les images docker	Oui	Non	Oui	Oui	Oui	Oui	Oui
Critères importants							
Gérer le réseau de docker	Non	Non	Non	Non	Oui	Oui	Oui
Gérer les volumes de docker	Non	Non	Non	Non	Oui	Oui	Oui
Regarder les événements de docker	Non	Non	Oui	Oui	Oui	Oui	Oui
Préconfigurer les modèles de conteneur	Oui	Non	Non	Non	Oui	Oui	Oui
Vue d'ensemble du cluster	Non	Non	Oui	Non	Oui	Oui	Oui
Niveau de complexité d'installation et/ou configuration	Simple	Simple	Simple	Simple	Simple	Moyen	Moyen
Type de conteneurs et/ou orchestrateurs	Docker	Docker	Docker	Docker	Docker / Swarm	Docker / Kubernetes	Tout
Gérer les Stacks	Non	Non	Non	Non	Oui	Oui	Oui
Gérer les secrets	Non	Non	Non	Non	Oui	Oui	Oui
Contributeurs	14	74	45	2	106	141	46
Gérer les services	Non	Non	Non	Non	Oui	Oui	Oui
Catalogue	Oui	Oui	Non	Oui	Oui	Non	Oui

FIGURE 4 – Tableau comparatif sur les interfaces utilisateur

A l'aide de ce tableau comparatif, nous pouvons faire le choix de notre interface utilisateur, tout en nous basant sur les critères importants que nous avons définis. Ainsi, nous constatons que les caractéristiques de **Kubernetes Dashboard** répondent parfaitement à nos exigences.

4.6 Tableau comparatif des outils d'automatisation

- **Puppet[8]** : c'est un outil de gestion de configuration de logiciel open source. Il fonctionne sur de nombreux systèmes de type Unix, ainsi que sur Microsoft Windows, et inclut son propre langage déclaratif pour décrire la configuration du système.
- **Chef[8]** : c'est une plate-forme d'automatisation d'infrastructure cloud et de gestion de systèmes open source. La société Opscode a créé l'outil de gestion de configuration Chef et elle a ensuite changé son nom pour Chef.

Chef transforme l'infrastructure en code pour automatiser le déploiement et la gestion des serveurs. Un professionnel DevOps installe le kit de développement Chef (Chef DK) sur un poste de travail pour définir des composants et interagir avec le serveur Chef.

Chef peut gérer divers types de nœuds, notamment des serveurs, des machines vir-

tuelles en nuage, des périphériques réseau et des conteneurs. Il gère Linux, Windows, mainframe et plusieurs autres systèmes. L'outil est conçu pour permettre aux développeurs et aux professionnels des opérations informatiques de travailler ensemble pour déployer des applications sur une infrastructure informatique.

- **Ansible[23]** : c'est une plate-forme d'automatisation open source. Il est très simple à configurer et puissant. Ansible peut vous aider dans la gestion de la configuration, le déploiement d'applications, l'automatisation des tâches. Il peut également s'agir d'une orchestration informatique, dans laquelle vous devez exécuter des tâches en séquence et créer une chaîne d'événements qui doit se produire sur plusieurs serveurs ou périphériques différents. Par exemple, si vous avez un groupe de serveurs Web derrière un équilibrEUR de charge, Ansible peut mettre à niveau les serveurs Web un par un et, lors de la mise à niveau, il peut supprimer le serveur Web actuel de l'équilibrEUR de charge et le désactiver dans votre système de surveillance. En bref, vous pouvez gérer des tâches complexes avec un outil facile à utiliser.
- **SaltStack** : également connu sous le nom de Salt, est un outil de gestion de la configuration et d'orchestration. Il utilise un référentiel central pour mettre en service de nouveaux serveurs et autres infrastructures informatiques, apporter des modifications aux serveurs existants et installer des logiciels dans des environnements informatiques, y compris des serveurs physiques et virtuels, ainsi que dans le cloud. SaltStack automatise les tâches d'administration de système et de déploiement de code répétées, éliminant ainsi les processus manuels de manière à réduire les erreurs qui surviennent lorsque les organisations informatiques configurent des systèmes.

	Puppet	Chef	Ansible	Salt
Support	Puppet Labs	Opscode	Ansible Works	saltStack
Année	2005	2009	2012	2011
Références	Google, eBay, Twitter	Facebook, Ancestry, Splunk	Rackspace, Evernote	LinkedIn, HP Cloud
Interface de contrôle	Manifest (DSL)	Recipes (DSL: Ruby)	Playbook (YAML, JSON)	SLS (SoLt Sate/YAML)
Code	Open source	Open source	Open source	Open source
Cloud	Tout	Tout	Tout	Tout
Critères importants				
Sécurité		Chef Vault	Elevé avec SSH	
Prix (Node/année)	Std : \$88 / Prem : \$152	Std : \$72 / Prem : \$137	Up to 100 Nodes Std:\$10000/ Prem:\$14000	Contacter l'entreprise
Communication	http, SSH/SSL	STOMP, rest/SSL	SSH/JSON	ZeroMQ
Contrôleur d'exécution	Mcollective, challenging	Knife, challenging	Built-in, easy	Built-in
Contributeurs	496	557	3616	2105
Type	Gestion de configuration	Gestion de configuration	Gestion de configuration	Gestion de configuration
Langage	Puppet DSL, Ruby	Ruby	Python	Python
Architecture	Client/Server	Client/Server	Client seulement	Client/Server
Difficulté de configuration	Difficile	Difficile	Facile	Facile
Évolutivité	Elevé	Elevé	Elevé	Elevé
Licence	Apache license v2	Apache license v2	GNU Public license	Apache license v2
Plus utilisé	Grande entreprise avec un environnement hétérogène.	Adapter pour le développement.	Adapter facilement pour l'automatisation.	-

FIGURE 5 – Tableau comparatif sur les outils d'automatisation

A l'aide de ce tableau comparatif, nous pouvons faire le choix de notre outil d'automatisation, tout en nous basant sur les critères importants que nous avons définis. Ainsi, nous constatons que les caractéristiques de **Ansible** répondent parfaitement à nos exigences.

5 Solution proposée

A travers une étude comparative et un test des différentes solutions existantes ; nous avons pu sélectionner des critères définis sur ses technologies, qui permettront de faciliter la mise en oeuvre d'une architecture d'orchestration complète, disponible, fiable tout en étant performante et opérationnelle.

En premier lieu, pour mettre en réseau tous les machines nous utiliserons le logiciel **OpenVPN** qui permettra de créer un tunnel entre tous les machines, vu que celles-ci se trouvent sur de différents réseaux avec des adresses IP publiques. Il s'avère important de les mettre sur le même réseau afin qu'elles puissent se communiquer entre elles.

Pour la composition de notre architecture d'orchestration, nous avons retenu comme orchestrateur de système, **Kubernetes**.

Le choix de **kubernetes** répond à tous nos critères de sélection et offre beaucoup plus d'avantages que les autres orchestrateurs (**fig.2**). **Kubernetes Dashboard** sera utilisé comme interface utilisateur pour la visualisation des composants de Kubernetes et les futures applications qui seront déployés par Kubernetes. Comme overlay, **Calico** servira à créer un tunnel entre les pods des différents nodes pour qu'ils puissent se communiquer entre eux.

Afin de faciliter l'installation et la configuration des outils pour la mise en place de cette architecture sur tous les machines distants, nous avons sélectionné un outil d'automatisation, **Ansible**, qui automatisera les tâches, l'installation et la configuration.

En constatant la difficulté pour la visualisation des ressources machines, les pourcentages de mémoires qu'occupent Kubernetes, ses composants et les applications qui fonctionnent dessus, l'intégration d'un logiciel de monitoring dans notre architecture permettra de faciliter l'affichage de ses capacités.

Le choix de **Prometheus** comme logiciel de monitoring et **Grafana** comme interface utilisateur permettra de visualiser la performance des machines et aussi celle de Kubernetes sous forme de graphe.

5.1 Architecture proposée

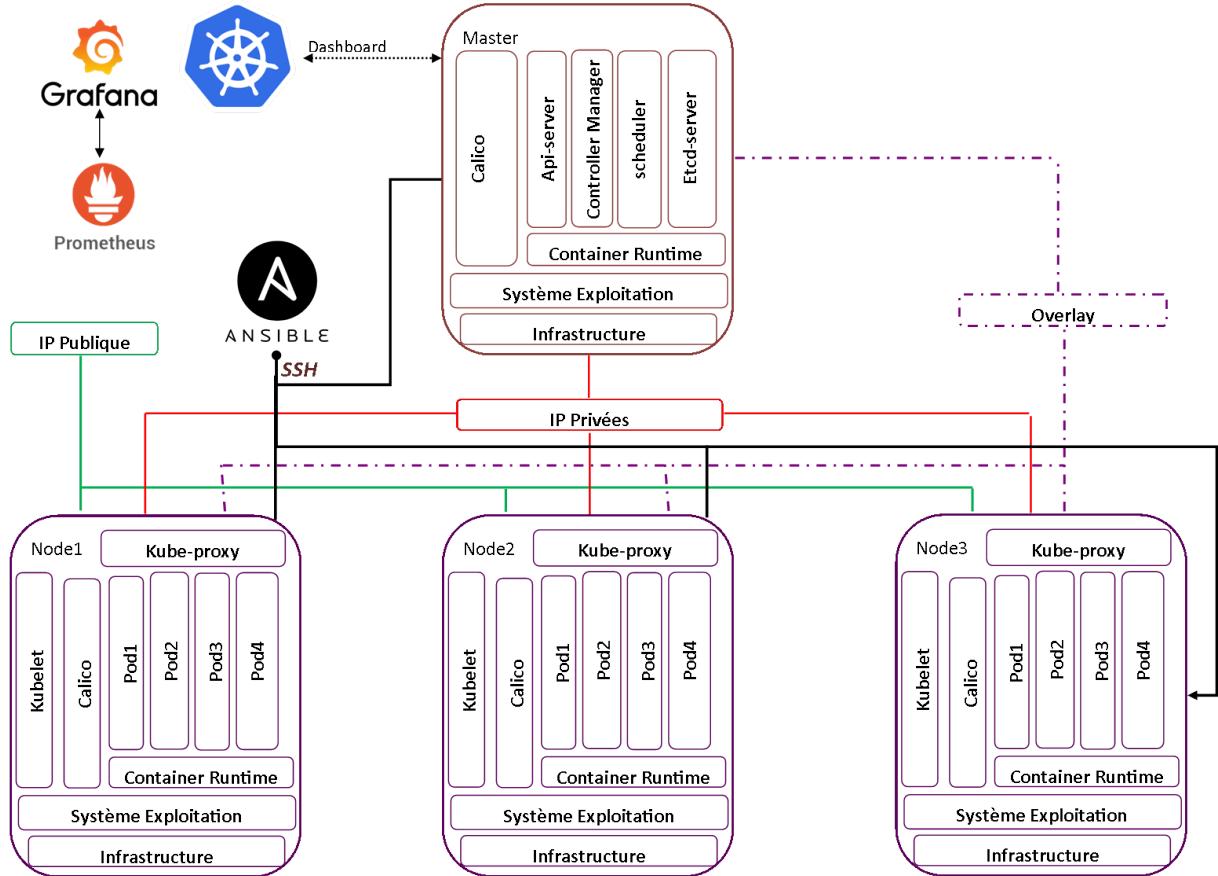


FIGURE 6 – Architecture de notre orchestration

L'architecture de notre orchestration comporte un master et trois nodes. Les lignes en vert représentent les adresses IP publiques qui sont attribuées à chaque machine par les Livebox. Les lignes en rouges représentent le réseau qui va être créé par OpenVPN afin d'attribuer à chaque machines une adresse IP privée. Les lignes en noir représentent la relation entre Ansible et les nodes via le master pour pouvoir faire le déploiement à distance sur chaque node. La dernière ligne en mauve et en pointillé créée par l'overlay Calico, sert pour la communication entre les pods des différents nodes. En haut à gauche, nous avons deux interfaces utilisateur (dashboard) sur l'image. La première en bleu est le dashboard de kubernetes, il permet de visualiser kubernetes et tous ses composants. La seconde, Grafana est relié à Prometheus afin d'y puiser dans celui-ci toutes les données métriques du système pour les afficher sous forme de graphe.

6 Implémentation

6.1 Environnement de travail

La mise en place de cette architecture a été faite dans une salle nommée « Salle innovation», où se trouve tous les matériels physiques nécessaires pour mettre en place ce système. Ci-dessous, nous trouvons les différents matériels qui ont été utilisés pour la composition physique de cette architecture et une vue globale de la salle (**fig.7**).



FIGURE 7 – Vue globale de la salle



FIGURE 8 – Intel NUC

Les petites boîtes que vous voyez sur la (**fig.8**) sont des mini machines appelées Intel NUC, fonctionnant comme toutes autres machines, à la seule différence qu'elles sont beaucoup plus légères, et par conséquent plus faciles à transporter et requérant moins d'énergie.

• Caractéristiques des machines (Intel NUC)

Le tableau ci-dessous fait un récapitulatif du système d'exploitation utilisé tout au long de notre expérience, des capacités des machines (mémoire et CPU) et de leur type de processeur.

Noms	Système d'exploitation	Mémoire RAM	Mémoire ROM	CPU	Génération
Master	Ubuntu 18.04.1	8GB	SATA 120GB	4	Intel Core i3 Processor inside
Node1	Ubuntu 18.04.1	8GB	SATA 120GB	4	Intel Core i3 Processor inside
Node2	Ubuntu 18.04.1	8GB	SATA 120GB	4	Intel Core i3 Processor inside

FIGURE 9 – Tableau Caractéristique des machines Intel NUC

L'ensemble des trois figures ci-dessous est une représentation globale de l'espace de travail, composé par les livebox d'Orange (**fig.10**) permettant d'assurer la connexion à l'Internet

à travers des adresses IP publiques différentes pour chaque machine (**fig.11**), et incluant un écran de visualisation (**fig.12**).



FIGURE 10 – *Livebox Orange*



FIGURE 11 – *Représentation de la connexion entre les machines*



FIGURE 12 – *Écran de visualisation*

6.2 Étapes de l'implémentation

Dans un premier temps, nous avons dû assembler les NUC (**fig.8**) parce que tous les pièces étaient détachées dans leurs boîtes. Les NUC ne possédaient pas de mémoire ROM et de mémoire RAM. Une fois l'assemblage terminé, nous associons chaque machine à un Livebox afin d'avoir une connexion à l'Internet et une adresse IP publique différente pour chacune des machines. L'architecture physique étant maintenant en place, nous passons à l'installation et à la configuration des machines.

Comme système d'exploitation, nous avons installé **Ubuntu 18.04.1 LTS bionic** sur chaque machine, puis fait la mise à jour du système. Pour installer Ubuntu² 18.04.1. Maintenant que tous les machines ont été installées, je peux commencer à mettre en place mon orchestration. Comme je l'ai mentionné au début, chaque machine est connectée à

2. <https://linuxconfig.org/how-to-install-ubuntu-18-04-bionic-beaver>

un Livebox qui lui permet d'avoir une connexion à l'Internet et de lui attribue une adresse IP publique qui la rend indépendante des autres machines et par conséquent incapable de communiquer avec elles. Pour le projet les machines doivent pouvoir se communiquer entre elles. De ce fait, il va falloir mettre les machines sur le même réseau afin qu'elles puissent se voir et communiquer entre elles. Pour assurer cette communication nous avons installé et configuré les deux logiciels libres, **OpenVPN** et **OpenSSL**. **OpenVPN** permet de créer un réseau privé virtuel (VPN) et **OpenSSL** permet l'authentification des machines lors de la connexion. **N.B :** Pour mettre en place cette architecture sur un système existant, il faut tenir compte du configuration préalable du système et de son réseau, afin de le configurer de manière efficace.

6.2.1 Installation d'OpenVPN et d'OpenSSL

OpenVPN s'installe au moyen de la commande **apt-get install -y openvpn**. La version que j'ai installée est OpenVPN 2.4.4 avec ses dépendances sur Ubuntu. OpenSSL est généralement installé par défaut sur les machines et nécessite pas d'être réinstaller. Mais, dans le cas où OpenSSL ne serait pas installé par défaut, il pourrait l'être avec cette commande : **apt-get install -y openssl**.

6.2.2 Configuration d'OpenVPN

La configuration de l'OpenVPN se fait sur le serveur VPN. L'installation d'OpenVPN crée un dossier dans **/usr/share/doc/openvpn/exemples/easy-rsa/** contenant tous les scripts permettant de générer facilement tous les certificats et les clés d'authentification nécessaires au fonctionnement d'OpenVPN. En revanche, ce dossier est manquant sur certaine distribution lors de l'installation d'OpenVPN, ce qui a été notre cas. Afin d'avoir ces dossiers de configuration, j'ai dû installer **easy-rsa** avec la commande suivante : **apt-get install -y easy-rsa**. Pour notre cas, après l'installation d'easy-rsa le dossier contenant les scripts se trouve dans **/usr/share/easy-rsa**. Avant d'attaquer les configurations, il faut créer un dossier **easy-rsa** qui va contenir les scripts et un sous dossier **keys** qui contiendra les différents certificats et clés générés dans le répertoire d'openvpn, et y copier les scripts originaux afin de centraliser les applications et les scripts. Pour créer le dossier et le sous dossier, exécuter la commande : **mkdir -p /etc/openvpn/easy-rsa/keys**. Une fois que c'est fait, je copie le contenu original d'easy-rsa dans le dossier que je viens de créer avec cette commande : **cp -r /usr/share/easy-rsa/* /etc/openvpn/easy-rsa/**. A partir du dossier **/etc/openvpn/easy-rsa/**, il faut dans un premier temps éditer le fichier **vars**, pour l'initialisation des différentes variables servant à la génération des certificats, en utilisant la commande : **nano /etc/openvpn/easy-rsa/vars**. Après l'affichage du fichier, il faut modifier cette partie en fournissant les informations personnelles de la sorte.

- Export KEY_DIR=\$D/keys

- Export KEY_COUNTRY=FRANCE
- Export KEY_PROVINCE=Île de France
- Export KEY_CITY=Paris
- Export KEY_ORG=Orange
- Export KEY_EMAIL=perrault.andre@cluster.com

Une fois le fichier modifié, on l'enregistre.

On exécute la commande³ : `../vars` ou `source ./vars` pour mettre à jour le fichier.

Pour générer le certificat d'autorité de certification et la clé correspondante (fichiers « ca.crt » et « ca.key »), à partir du dossier `/etc/openvpn/easy-rsa`, on exécute la commande : `./clean-all` suivi de `./build-ca`. Presser Entrée à toutes les questions.

Le certificat et la clé du serveur VPN sont générés par l'exécution du script se trouvant dans le dossier `/etc/openvpn/easy-rsa` : `./build-key-server nom_du_serveur`. Le certificat et la clé des clients sont générés par l'exécution du script se trouvant dans le dossier `/etc/openvpn/easy-rsa` : `./build-key nom_du_client`.

Pour mieux échanger les clés en toute sécurité, nous allons utiliser le protocole Diffie-Hellman qui est un protocole de cryptographie utilisé dans les échanges de clés. De ce fait, on exécute le script `build-dh`, qui se trouve dans le dossier `/etc/openvpn/easy-rsa` : `./build-dh`. Cette commande va créer un fichier nommé **dh1024.pem**. La création des certificats et des clés a été générée automatiquement dans le répertoire `/etc/openvpn/easy-rsa/keys`.

6.2.3 Installation des outils nécessaires

L'installation du serveur SSH nous permettra d'accéder à les machines via SSH. Pour installer le serveur ssh sur toutes les machines, y compris le serveur, on exécute la commande : `apt-get install -y ssh-server` et `apt-get install -y ssh-client`.

Générer une clé pour le serveur ssh et l'attribuer à tous ces clients, évitera la saisie de mot de passe à chaque connexion, et ce qui sera aussi utile lors du déploiement automatique. La commande `ssh-keygen -t rsa -b 4096` nous génère la clé ssh et la commande `ssh-copy-id nom_host@IPV4` nous permet d'attribuer la clé à chaque client. Une fois que les machines sont mises en réseau, nous pouvons maintenant installer et configurer les outils pour l'orchestration. Nous devons nous assurer que les machines ont les pré-requis installés pour le déploiement de l'orchestration. Nous allons faire l'installation et la configuration des pré-requis pour l'outil d'automatisation et aussi pour le déploiement de l'orchestration. Avant tout, nous devons mettre à jour le système d'exploitation, avec les commandes suivantes : `apt-get update` et `apt-get upgrade`. Une fois terminée, on

3. N.B. La commande mentionnée est bien : point – espace – point/vars.

peut installer **python 3.6**, ses dépendances et les outils pour gérer le réseau en exécutant ces lignes de commandes.

- apt-get install build-essential libpq-dev libssl-dev openssl libffi-dev zlib1g-dev
- apt-cache search python3.6
- add-apt-repository ppa :jonathonf/python-3.6
- apt-get update
- apt-get install -y python3.6
- apt-get install -y yum-utils
- apt-get install python3-pip python3-dev
- python3 -m pip install netaddr
- pip install --upgrade Jinja2

Une fois **python 3.6** installé sur notre serveur, la prochaine étape est l'installation et la configuration de l'outil d'automatisation Ansible. Pour l'installer, on exécute les commandes suivantes :

- apt-add-repository ppa :ansible/ansible
- apt-get update -y
- apt-get install ansible -y

Une fois **Ansible** installé, il faut le configurer en modifiant le fichier hosts, qui se trouve dans le dossier **/etc/ansible/hosts**.

- sudo nano /etc/ansible/hosts

Puis on ajoute les lignes suivantes dans le fichier.

[nomServers afin de délimiter les groupes de hosts]

Nom_machine ansible_host=nom_host@IPV4_host ip=IPV4_host ansible_connection=ssh
ansible_ssh_user=nom_user ansible_ssh_pass=password ansible_become_pass=password

Enregistrer et fermer le fichier après l'ajout, en notant que ces lignes peuvent être ajoutées autant de fois nécessaires qu'il y a de machines disponibles sur le réseau. Pour tester si la configuration ansible marche parfaitement et arriver à détecter les machines sur le réseau, on exécute la commande : **ansible -m ping nom_servers**. Cette commande vous permettra de vérifier toutes les machines qui sont disponibles sur le réseau et qui sont dans le groupement **nom_servers**. Nous pouvons aussi vérifier chaque machine séparément, en

remplaçant **nom_servers** par le nom de la machine. Après avoir installé et configuré ces outils, notre serveur est finalement prêt à recevoir le déploiement de notre orchestrateur.

L'installation de Kubernetes peut se faire de différentes manières, qui sont : **minikube**, **kops**, **kubeadm**, **kubespray**. Kubernetes peut être installé sur des machines virtuelles et sur du bare métal.

Machine Virtuelle (VM)

Kubernetes peut être installé sur des machines virtuelles créées via Vagrant, VMware vSphere, KVM, etc. Différents outils sont disponibles pour automatiser l'installation, comme Ansible ou kubeadm.

Bare Metal

Kubernetes peut être installé sur du bare metal, au-dessus de différents systèmes d'exploitation, tels que RHEL, CoreOS, CentOS, Fedora, Ubuntu, etc. La plupart des outils d'installation de machines virtuelles peuvent également être utilisés. Il existe quelques options d'installation de localhost disponibles pour déployer des clusters Kubernetes à un ou plusieurs nœuds sur notre station de travail :

Minikube est la méthode préférée et recommandée pour créer une configuration Kubernetes tout-en-un, qui est uniquement utilisé en local.

Kubeadm est sur l'écosystème de Kubernetes. C'est un moyen sécurisé et recommandé pour amorcer le cluster Kubernetes. Il dispose d'un ensemble de blocs de construction pour configurer le cluster, mais il est facilement extensible pour ajouter plus de fonctionnalités. Veuillez noter que kubeadm ne prend pas en charge le provisionnement des machines, et est très difficile à configurer.

KubeSpray anciennement connu sous le nom de Kargo, nous permet d'installer des clusters Kubernetes hautement disponibles sur AWS, GCE, Azure, OpenStack ou bare metal. KubeSpray est basé sur Ansible et est disponible sur la plupart des distributions Linux. C'est un projet d'incubateur Kubernetes. Il vous faut des connaissances de base sur kubernetes et Ansible pour mettre en place un cluster avec kubespray.

Kops nous permet de créer, de détruire, de mettre à niveau et maintenir des clusters Kubernetes de haute qualité de production à partir de la ligne de commande. Il peut également approvisionner les machines. Actuellement, AWS est officiellement pris en charge. La prise en charge de GCE et de VMware vSphere est en phase alpha et d'autres plates-formes sont prévues pour le futur. Si les solutions et les outils existants ne correspondent pas à vos besoins, vous pouvez toujours installer Kubernetes à partir de zéro. Il est intéres-

sant de vérifier le projet Kubernetes **The Hard Way GitHub de Kelsey Hightower**, qui partage les étapes manuelles impliquées dans le démarrage d'un cluster Kubernetes.

Dans notre cas, nous allons utiliser Kubespray, vu qu'on veut faire le déploiement sur du bare metal et sur plusieurs machines et qu'on veut avoir un cluster hautement disponible. Avec Ansible, on va diminuer considérablement les tâches manuelles. De ce fait, Kubespray est le plus adapté pour notre architecture.

Avant d'installer Kubernetes, nous allons cloner dans un premier temps les fichiers de configuration de Kubernetes sur le Github officiel, puis configurer tous les fichiers de Kubernetes pour les adapter à nos besoins. Pour cloner le Github, on exécute la commande : git clone⁴.

Une fois terminé, on rentre dans le répertoire, on fait une copie du dossier principal dans le cas où nous supprimerons ou ferons des erreurs dans notre configuration. De sorte que nous puissions retourner à l'état initial en cas d'erreur. Maintenant, nous allons configurer le fichier principal pour notre déploiement, qui se trouve dans le dossier kubespray qu'on vient de cloner : kubespray/incubator/nom_fichier_copier/hosts.ini

On configure ce fichier de la manière.

```
[all]
master ansible_host=master@10.8.0.1 ip=10.8.0.1 ansible_user=master ansible_ssh_pass=*
ansible_become_pass=*

node1 ansible_host=node1@10.8.0.6 ip=10.8.0.6 ansible_user=node1 ansible_ssh_pass=*
ansible_become_pass=*

node2 ansible_host=node2@10.8.0.10 ip=10.8.0.10 ansible_user=node2 ansible_ssh_pass=*
ansible_become_pass=*

[kube-master]
master

[kube-node]
node1
node2

[etcd]
master
```

4. <https://github.com/kubernetes-incubator/kubespray>

```
[k8s-cluster :children]
  kube-node
  kube-master
```

Nous avons les noms entre crochet qui sert à regrouper les machines, [all] auxquelles Ansible doit se connecter sur le réseau, en utilisant le mot de la première colonne comme un alias pour tous les informations qui le suivent.

Ex : Master est un alias qui réfère aux informations suivantes :

ansible_host : fait référence au domaine ou au nom qui permet de loger via ssh sur la machine.
ansible_user : le nom de l'utilisateur de la machine.
ip : l'adresse IPV4 de la machine.
ansible_ssh_pass : le mot de passe de la machine.
ansible_become_pass : le mot de passe root de la machine.

Pour les autres mots entre crochet :

kube-master : regroupe les machines qu'on souhaite mettre en tant que master.
kube-node : regroupe les machines qu'on souhaite mettre en tant que node.
etcd : regroupe les machines sur lesquelles on souhaite exécuter etcd dessus.
k8s-cluster :children : regroupe les différentes entités du cluster.

N.B. On peut ajouter autant de machines qu'on veut dans notre cluster à l'aide de ce fichier.

Une fois terminée, on enregistre et on passe à la configuration des fichiers de Kubernetes et afin de spécifier les versions des outils souhaitées, le système à utiliser, la capacité minimale du cluster, le type d'overlay souhaité, la version de docker qu'il faut utiliser. Les fichiers de configuration de Kubernetes étant trop nombreux, nous les avons mis en annexe sur notre Github :⁵. Sur ce Github, vous trouverez les fichiers de configuration avec les explications.

Avant le lancement du script pour le déploiement, nous allons installer quelques outils et quelques dépendances d'Ansible.

- sudo pip install -r requirements.txt

5. <https://github.com/pdjy/Kubernetes>

- pip install ansible-modules-hashivault

- pip install --upgrade cryptography

Notre configuration est finalement prête, on peut procéder au déploiement de l'orchestration. Pour lancer le script, nous devons nous positionner dans le répertoire kubespray et on exécute la commande suivante :

```
ansible-playbook -T 300 -i inventory/local/hosts.ini cluster.yml -b -v --private-key= ./ssh/id_rsa
```

ansible-playbook : l'outil pour exécuter les playbooks ansible, qui sont un système de déploiement et de configuration de multi-nodes.

--private-key : utiliser ce fichier pour authentifier la connexion.

-T : remplacer le délai de connexion en secondes (par défaut = 10).

-b : exécuter les opérations à venir, n'implique pas une invite de mot de passe.

-i : spécifier le chemin d'hôte de l'inventaire (par défaut =/etc/ansible/hosts) ou une liste d'hôtes séparés par des virgules.

-v : mode verbeux (-vv pour plus, -vvvv pour activer le débogage de la connexion).

L'installation et la configuration des machines peut prendre environ 20 à 40 minutes, dépendant du nombre de machines, de leur vitesse et de la vitesse de la connexion. C'est une estimation en moyenne pour un cluster de petite taille (3 à 10 machines), mais pour un cluster moyen ou de grande taille cent milles (100000 machines), ça peut aller jusqu'à 1 à 12 heures.

Notre orchestrateur étant en marche, nous allons maintenant installer et configurer les métriques sur le master qui nous affichera les pourcentages d'utilisation de tout notre système (CPU, ROM, RAM, réseau) et aussi les pourcentages que les composants de Kubernetes occupent sur le système. Tout ça nous permettra de mieux gérer notre cluster, d'émettre des alertes en cas de saturation de l'un des composants du système, en cas de coupure du réseau, ou si un node n'est plus fonctionnel.

Pour mettre en place ces métriques, nous allons dans un premier temps déployer notre métrique avec la commande.

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/kops/master/addons/metrics-server/v1.8.x.yaml
```

Après le déploiement, nous allons modifier les paramètres de la métrique afin qu'il prenne en compte tous les composants de notre système. Nous allons accéder au fichier du déploiement à l'aide de la commande : kubectl namespace kube-system edit deploy metrics-server Dans la section spec/template/spec/command, nous allons remplacer :

- --source=kubernetes ou même si la ligne est plus longue, on la remplace par :
- --source=kubernetes.summary_api :https://kubernetes.default ?kubeletHttps=true&kubeletPort=1025

On enregistre, puis on ferme le fichier. Cette ligne permettra à la métrique de communiquer avec les nodes du cluster via l'API et Kubelet en leurs disant d'accepter la requête HTTPS qui n'est pas sécurisée.

La métrique que nous venons d'installer nous permet de voir les pourcentages de la machine ou des composants du cluster en mode console. Afin de mieux visualiser les performances de notre cluster, nous allons installer et configurer Prometheus et Grafana.

Une fois terminé avec l'implémentation, nous allons présenter graphiquement à l'aide de l'interface utilisateur de Kubernetes, notre cluster et les différents parties qui le composent. Nous prenons aussi le soin de tester notre système avec une application (Gestion de commande d'un restaurant) que nous avons développée en Java, afin de montrer l'importance de ce système pour le bon fonctionnement des applications.

7 Résultats

Pour lancer l'interface utilisateur de Kubernetes (kubernetes Dashboard), nous utilisons le lien ci-dessous, qui est l'adresse IP du master suivi de l'API :

<https://10.8.0.1:6443/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/>

Nous pouvons constater que Kubernetes nous offre deux possibilités d'identification, soit à l'aide d'un fichier kubeconfig (**fig.13**), soit à l'aide d'un token (**fig.14**).

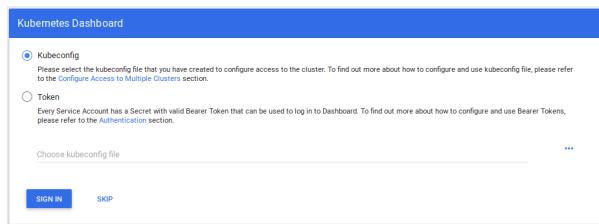


FIGURE 13 – Dashboard identifiant avec kubeconfig

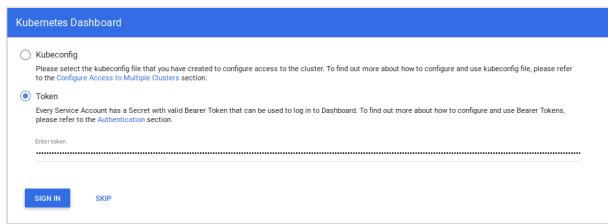


FIGURE 14 – Dashboard identifiant avec token

L'identification à l'aide de kubeconfig, sur la figure 12, se fait à partir d'un fichier YAML qui peut être conçu en fonction du niveau d'accès qu'on veut attribuer à un groupe ou à l'entité qui va utiliser l'interface afin de mieux contrôler le système et éviter toute modification.

L'identification à l'aide du token, sur la figure 13, se fait à partir d'une clé générée par

Kubernetes lors de la création du dashboard. Cette clé peut être récupérée à l'aide de ces commandes :

```
kubectl get secret | grep kubernetes-dashboard-sa
kubectl describe secret kubernetes-dashboard-sa-token-....
```

Elle nous donne un accès total en tant qu'administrateur du système. Nous allons l'utiliser pour gérer, manipuler et tester notre système.

Une fois identifié, nous pouvons visualiser tous les entités de notre système et voir si réellement celui-ci fonctionne parfaitement de la manière dont on le souhaitait.

Namespaces				
Name	Labels	Status	Age	
monitoring	name:monitoring	Active	a day	
kube-public	-	Active	a day	
default	-	Active	a day	
kube-system	-	Active	a day	

FIGURE 15 – Namespaces

Sur la (**fig.15**) nous avons les différents namespaces de notre système qui sont activés et qui fonctionnent normalement, comme le montre la section "status".

Sur cette figure nous avons les différents nodes qui composent notre système.

Nodes							
Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	Memory requests (bytes)	Memory limits (bytes)	Age
node1	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/hostname:node1 node-role.kubernetes.io/node: true	True	0.648 (16.20%)	1.523 (38.07%)	618.588 Mi (7.86%)	3.084 Gi (40.15%)	a day
node2	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/hostname:node2 node-role.kubernetes.io/node: true	True	0.705 (17.63%)	1.4 (35.00%)	492.588 Mi (6.26%)	3.508 Gi (45.66%)	a day
master	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/hostname:master node-role.kubernetes.io/master: tr.	True	1.04 (26.00%)	2.5 (62.50%)	895.773 Mi (11.39%)	5.836 Gi (75.97%)	a day

FIGURE 16 – Nodes

Le master représente le serveur et les 2 nodes (node1 et node2) représentent les travailleurs. Nous constatons que les trois sont connectés et fonctionnent correctement, comme le montre la colonne "ready" qui affiche une valeur "True". Nous pouvons aussi contrôler le nombre de "CPU et mémoire" que requiert le système, y compris leur limite, pour chaque nodes et master dans le cluster, et leur date de création.

Les figures suivantes nous décrivent les services et les déploiements de notre système

qui ont été créés pour les applications et pour les différents outils qui composent notre architecture. Nous pouvons voir tous les informations relatives à ces ressources, telles que les namespaces, les labels, les pods, la date de création et les images qui les génèrent.

Services						
Name	Namespace	Labels	Cluster IP	Internal endpoints	External endpoints	Age
kubelet	kube-system	k8s-app: kubelet	None	kubelet.kube-system:10250/TCP	-	a day
kube-prometheus-ex-porter-kube-state	monitoring	app: exporter-kube-state chart: exporter-kube-state-0.2.6 component: kube-state heritage: Tiller release: kube-prometheus	10.233.60.85	kube-prometheus-exporter+kube-state.monitoring:80	-	a day
kube-prometheus-ex-porter-kube-controller-manager	kube-system	app: exporter-kube-controller-manager chart: exporter-kube-controller-manager-0.1.10 component: kube-controller-manager heritage: Tiller release: kube-prometheus	None	kube-prometheus-exporter+kube-controller-manager:80	-	a day
kube-prometheus-ex-porter-kube-dns	kube-system	app: exporter-kube-dns chart: exporter-kube-dns-0.1.7 component: kube-dns heritage: Tiller release: kube-prometheus	None	kube-prometheus-exporter+kube-dns.kube-system:1 kube-prometheus-exporter+kube-dns.kube-system:1	-	a day
kube-prometheus-ex-porter-kube-etcd	kube-system	app: exporter-kube-etcd chart: exporter-kube-etcd-0.1.15 component: kube-etcd heritage: Tiller release: kube-prometheus	None	kube-prometheus-exporter+kube-etcd.kube-system:80	-	a day
kube-prometheus-ex-porter-kube-scheduler	kube-system	app: exporter-kube-scheduler chart: exporter-kube-scheduler-0.1.9 component: kube-scheduler heritage: Tiller release: kube-prometheus	None	kube-prometheus-exporter+kube-scheduler.kube-system:80	-	a day
kube-prometheus-grafana	monitoring	app: kube-prometheus-grafana chart: grafana-0.37 heritage: Tiller release: kube-prometheus	10.233.21.137	kube-prometheus-grafana.monitoring:80/TCP	-	a day
kube-prometheus-exporter-node	monitoring	app: exporter-node chart: exporter-node-0.4.6 component: node-exporter heritage: Tiller release: kube-prometheus	10.233.59.219	kube-prometheus-exporter-node.monitoring:9100/TCP	-	a day
prometheus-operated	monitoring	operator: prometheus: true	None	prometheus-operated.monitoring:9090/TCP	-	a day
kube-prometheus	monitoring	app: prometheus chart: prometheus-0.51 heritage: Tiller image: gcr.io/google_containers/pause:3.1 imagePullPolicy: Always imagePullSecrets: [] imageType: Docker labels: { } livenessProbe: { } name: kube-prometheus nodeSelector: { } resources: { } replicas: 1 restartPolicy: Always selector: { } serviceAccountName: kube-prometheus terminationGracePeriodSeconds: 30 toleration: { } version: v1.2.0	10.233.4.92	kube-prometheus.us.monitoring:9090/TCP	-	a day

FIGURE 17 – Services

Deployments						
Name	Namespace	Labels	Pods	Age	Images	
kube-prometheus-exporter-kube-state	monitoring	app: exporter-kube-state chart: exporter-kube-state-0.2.6 component: kube-state heritage: Tiller release: kube-prometheus version: v1.2.0	1 / 1	a day	gcr.io/google_containers/kube-state-metrics:v2 gcr.io/google_containers/addon-resizer:v1.7	
kube-prometheus-grafana	monitoring	app: kube-prometheus-grafana chart: grafana-0.37 heritage: Tiller release: kube-prometheus	1 / 1	a day	grafana/grafana:5.0.0 quay.io/coreos/grafana-watcher:v0.0.8	
prometheus-operator	monitoring	app: prometheus-operator chart: prometheus-operator-0.0.29 heritage: Tiller operator: prometheus release: prometheus-operator	1 / 1	a day	quay.io/coreos/prometheus-operator:v0.20.0	
tiller-deploy	kube-system	app: helm name: tiller	1 / 1	a day	gcr.io/kubernetes-helm/tiller:v2.10.0	
metrics-server	kube-system	k8s-app: metrics-server	1 / 1	a day	gcr.io/google_containers/metrics-server-amd64	
kubernetes-dashboard	kube-system	k8s-app: kubernetes-dashboard	1 / 1	a day	gcr.io/google_containers/kubernetes-dashboard	
kubedns-autoscaler	kube-system	addOnManager: kubernetes.io/mode: Recreate k8s-app: kubedns-autoscaler kubernetes.io/cluster/service: true	1 / 1	a day	gcr.io/google_containers/cluster-proportional-autoscaler:v1.0.0	
kube-dns	kube-system	addOnManager: kubernetes.io/mode: Recreate k8s-app: kube-dns kubernetes.io/cluster/service: true	2 / 2	a day	gcr.io/google_containers/k8s-dns-kube-dns-amd64:v1.14.0 gcr.io/google_containers/k8s-dns-dnsMasq-amd64:v1.14.0 gcr.io/google_containers/k8s-dns-sidecar-amd64:v1.14.0	
calico-kube-controllers	kube-system	k8s-app: calico-kube-controllers kubernetes.io/cluster/service: true	1 / 1	a day	quay.io/calico/kube-controllers:v3.1.3	

FIGURE 18 – Déploiements

Les figures suivantes nous décrivent les répliques et les pods qui ont été créés par le déploiement. Nous pouvons voir tous les informations relatives à ces ressources, telles que les namespaces, les labels, les pods, la date de créations et leur statut et le nombre de répliques qui a été créées.

Replica Sets						
Name	Namespace	Labels	Pods	Age	Images	
kube-prometheus-exporter-kube-state-66b8849c9b	monitoring	app: kube-prometheus-exporter-kube-state component: kube-state pod-template-hash: 2264405756 release: kube-prometheus version: v1.2.0	1 / 1	a day	gcr.io/google_containers/kube-state-metrics:v1.2.0 gcr.io/google_containers/addon-resizer:1.7	⋮ ⋮
kube-prometheus-exporter-kube-state-658f46bd0d	monitoring	app: kube-prometheus-exporter-kube-state component: kube-state pod-template-hash: 2149026488 release: kube-prometheus version: v1.2.0	0 / 0	a day	gcr.io/google_containers/kube-state-metrics:v1.2.0 gcr.io/google_containers/addon-resizer:1.7	⋮ ⋮
kube-prometheus-grafana-f869c754	monitoring	app: kube-prometheus-grafana pod-template-hash: 94257310 release: kube-prometheus	1 / 1	a day	grafana/grafana:5.0.0 quay.io/coreos/grafana-watcher:v0.0.8	⋮ ⋮
prometheus-operator-858c485	monitoring	app: prometheus-operator operator: prometheus pod-template-hash: 4147041 release: prometheus-operator	1 / 1	a day	quay.io/coreos/prometheus-operator:v0.20.0	⋮ ⋮
tiller-deploy-56c4cf647b	kube-system	app: helm name: tiller pod-template-hash: 1270792036	1 / 1	a day	gcr.io/kubernetes-helm/tiller:v2.10.0	⋮ ⋮
tiller-deploy-64c9d747bd	kube-system	app: helm name: tiller pod-template-hash: 2075830368	0 / 0	a day	gcr.io/kubernetes-helm/tiller:v2.10.0	⋮ ⋮
metrics-server-694bcfcf5c	kube-system	k8s-app: metrics-server pod-template-hash: 2506767917	1 / 1	a day	gcr.io/google_containers/metrics-server-amd64:v0.2.1	⋮ ⋮
metrics-server-57c94d58b7	kube-system	k8s-app: metrics-server pod-template-hash: 1375082463	0 / 0	a day	gcr.io/google_containers/metrics-server-amd64:v0.2.1	⋮ ⋮
kubernetes-dashboard-786c896b97	kube-system	k8s-app: kubernetes-dashboard pod-template-hash: 3427452653	1 / 1	a day	gcr.io/google_containers/kubernetes-dashboard-amd64	⋮ ⋮
kubedns-autoscaler-76f87d889d	kube-system	k8s-app: kubedns-autoscaler pod-template-hash: 3294384458	1 / 1	a day	gcr.io/google_containers/cluster-proportional-autosca	⋮ ⋮

FIGURE 19 – Réplique

Pods						
Name	Namespace	Node	Status	Restarts	Age	
kube-prometheus-exporter-kube-state-66b8849c9b-jr5vt	monitoring	node1	Running	0	a day	⋮ ⋮
prometheus-kube-prometheus-0	monitoring	node1	Running	1	a day	⋮ ⋮
alertmanager-kube-prometheus-0	monitoring	node1	Running	0	a day	⋮ ⋮
kube-prometheus-grafana-f869c754-jcq84	monitoring	node1	Running	0	a day	⋮ ⋮
kube-prometheus-exporter-node-v89dx	monitoring	node2	Running	0	a day	⋮ ⋮
kube-prometheus-exporter-node-727wt	monitoring	node1	Running	0	a day	⋮ ⋮
kube-prometheus-exporter-node-lsrg	monitoring	master	Running	0	a day	⋮ ⋮
prometheus-operator-858c485-4wxbc	monitoring	node1	Running	0	a day	⋮ ⋮
tiller-deploy-56c4cf647b-c6hsz	kube-system	node1	Running	0	a day	⋮ ⋮
metrics-server-694bcfcf5c-8s5q8	kube-system	node2	Running	0	a day	⋮ ⋮

FIGURE 20 – Pods

Les figures suivantes nous décrivent les rôles et les secrets qui ont été créés pour mieux sécuriser notre système et ses ressources. Nous pouvons voir tous les informations relatives à ces ressources, telles que le rôle attribué à chaque entité composant le cluster, le namespace la date de création.

Roles				
Name	Role Type	Namespace	Age	
kube-prometheus-exporter-kube-state	Role	monitoring	a day	
psp-kube-prometheus	Cluster Role	All Namespaces	a day	
kube-prometheus-exporter-kube-state	Cluster Role	All Namespaces	a day	
kube-prometheus	Cluster Role	All Namespaces	a day	
psp-kube-prometheus-exporter-kube-state	Cluster Role	All Namespaces	a day	
psp-kube-prometheus-grafana	Cluster Role	All Namespaces	a day	
psp-kube-prometheus-exporter-node	Cluster Role	All Namespaces	a day	
psp-kube-prometheus-alertmanager	Cluster Role	All Namespaces	a day	
psp-prometheus-operator	Cluster Role	All Namespaces	a day	
prometheus-operator	Cluster Role	All Namespaces	a day	

FIGURE 21 – Rôles

Secrets				
Name	Namespace	Type	Age	
kube-prometheus-token-7zxls	monitoring	kubernetes.io/service-account-token	a day	⋮
prometheus-kube-prometheus	monitoring	Opaque	a day	⋮
kube-prometheus-grafana-token-chvds	monitoring	kubernetes.io/service-account-token	a day	⋮
kube-prometheus-exporter-kube-state-token-mfvnn	monitoring	kubernetes.io/service-account-token	a day	⋮
kube-prometheus-grafana	monitoring	Opaque	a day	⋮
kube-prometheus-exporter-node-token-96h56	monitoring	kubernetes.io/service-account-token	a day	⋮
alertmanager-kube-prometheus	monitoring	Opaque	a day	⋮
default-token-zm2lq	monitoring	kubernetes.io/service-account-token	a day	⋮
prometheus-operator-token-m9cqy	monitoring	kubernetes.io/service-account-token	a day	⋮
tiller-token-n2lqq	kube-system	kubernetes.io/service-account-token	a day	⋮

FIGURE 22 – Secrets

Les figures suivantes décrivent notre master, le système installé dessus, ses ressources, sa capacité. Nous avons deux types de représentations, sur les (fig.23), (fig.24), nous avons une représentation des capacités du master fait par Kubernetes Dashboard, et sur la figure (fig.25), nous avons une représentation des capacités du master fait par Grafana.

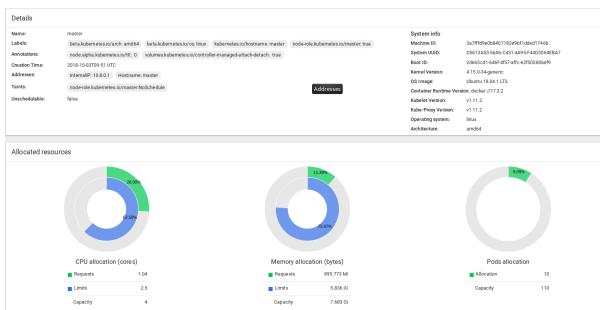


FIGURE 23 – Master 0

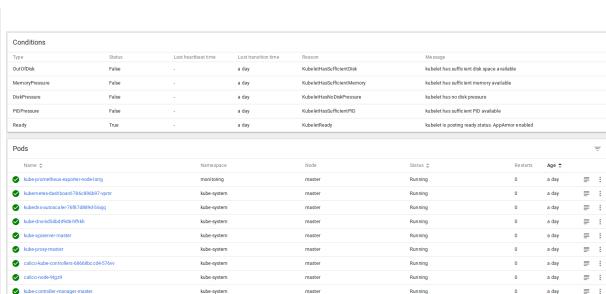


FIGURE 24 – Master 1

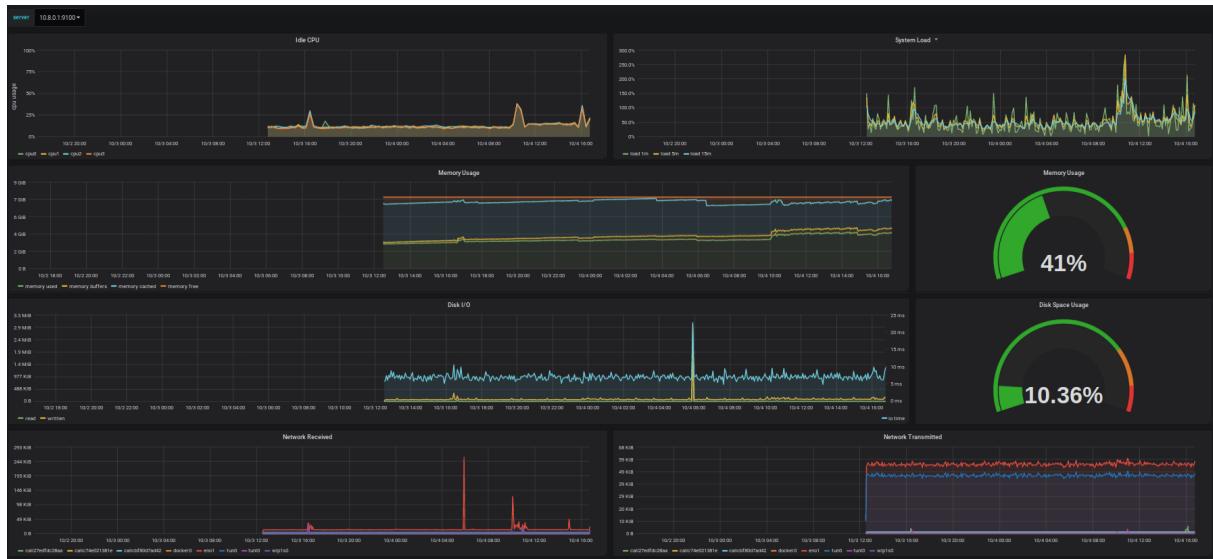


FIGURE 25 – Master Capacity

Les figures suivantes décrivent notre node1, le système installé dessus, ses ressources, sa capacité. Nous avons deux types de représentations, sur les (fig.26), (fig.27), nous avons une représentation des capacités du node1 fait par Kubernetes Dashboard, et sur la figure (fig.28), nous avons une représentation des capacités du node1 fait par Grafana.

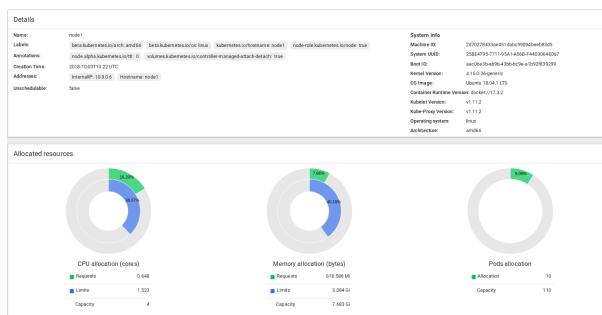


FIGURE 26 – Node1

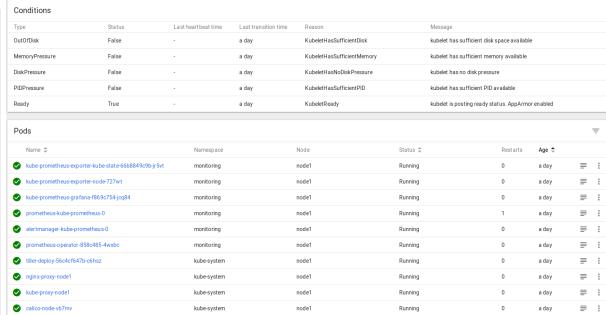


FIGURE 27 – Node1

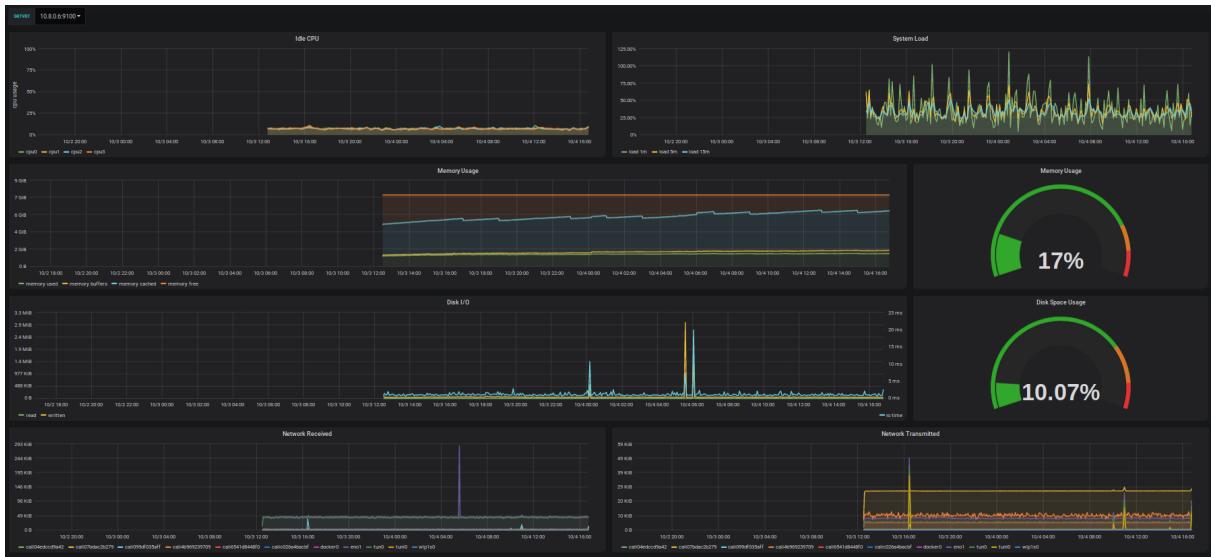


FIGURE 28 – *Node 1 Capacity*

Les figures suivantes décrivent notre node2, le système installé dessus, ses ressources, sa capacité. Nous avons deux types de représentations, sur les (fig.29), (fig.30), nous avons une représentation des capacités du node2 fait par Kubernetes Dashboard, et sur la figure (fig.31), nous avons une représentation des capacités du node2 fait par Grafana.

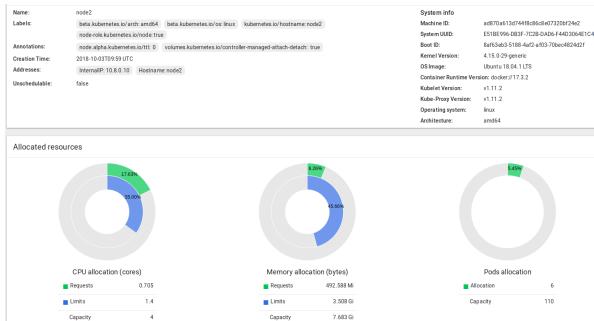


FIGURE 29 – *Node 2*

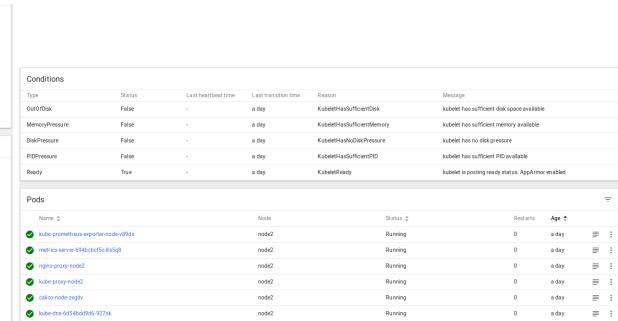


FIGURE 30 – *Node 2_1*



FIGURE 31 – Node 2 Capacity

Nous pouvons aussi vérifier les capacités de notre système (nodes et pods) en mode console.

```

root@master: /home/master
File Edit View Search Terminal Tabs Help
root@master: /home/master/kub... x root@master: /home/master x
root@master:/home/master# kubectl top node
NAME      CPU(cores)   CPU%     MEMORY(bytes)   MEMORY%
master    493m         12%      5311Mi          72%
node1    229m         5%       2024Mi          26%
node2    124m         3%       3205Mi          42%
root@master:/home/master#

```

FIGURE 32 – Capacité du master et des nodes

Sur la figure ci-dessus à l'aide de la commande **kubectl top node** nous pouvons vérifier en pourcentage le nombre de "CPU et mémoire" qu'occupe chaque node dans notre système.

```

root@master:/home/master# kubectl top pod --all-namespaces
NAMESPACE      NAME                               CPU(cores)   MEMORY(bytes)
kube-system    calico-kube-controllers-68668bccd4-57vvv   0m          9Mi
kube-system    calico-node-9tgz9                  26m         23Mi
kube-system    calico-node-vb7mv                 30m         22Mi
kube-system    calico-node-zxgdv                 26m         22Mi
kube-system    kube-apiserver-master            78m        554Mi
kube-system    kube-controller-manager-master     63m        79Mi
kube-system    kube-dns-6d54bdd9d6-927nk          4m          33Mi
kube-system    kube-dns-6d54bdd9d6-hfhkh          2m          32Mi
kube-system    kube-proxy-master                5m          24Mi
kube-system    kube-proxy-node1                 5m          23Mi
kube-system    kube-proxy-node2                 5m          23Mi
kube-system    kube-scheduler-master           18m        34Mi
kube-system    kubedns-autoscaler-76f87d889d-56xjq  0m          8Mi
kube-system    kubernetes-dashboard-786c896b97-vprnr  0m         17Mi
kube-system    metrics-server-694bcacf5c-8s5q8       1m         16Mi
kube-system    nginx-proxy-node1                0m          4Mi
kube-system    nginx-proxy-node2                0m          4Mi
kube-system    tiller-deploy-56c4cf647b-c6hsz       0m         17Mi
monitoring     alertmanager-kube-prometheus-0      2m         11Mi
monitoring     kube-prometheus-exporter-kube-state-66b8849c9b-jr5vt  5m         25Mi
monitoring     kube-prometheus-exporter-node-727wt    5m          9Mi
monitoring     kube-prometheus-exporter-node-lsrrg     6m          9Mi
monitoring     kube-prometheus-exporter-node-v89dx    5m          9Mi
monitoring     kube-prometheus-grafana-f869c754-jcq84  2m         27Mi
monitoring     prometheus-kube-prometheus-0          58m        392Mi
monitoring     prometheus-operator-858c485-4wxbc       5m         19Mi
root@master:/home/master#

```

FIGURE 33 – Capacité des pods

Sur la figure ci-dessus à l'aide de la commande **kubectl top pod --all-namespaces** nous pouvons vérifier le nombre de "CPU et mémoire" qu'occupe chaque pod dans notre système, tout en indiquant l'appartenance de chaque pod aux namespaces.

Pour tester notre système nous avons utilisé une application que nous avons implémentée en Java et qui est connectée à une base de données MySQL.

Pour pouvoir le déployer dans Kubernetes, il est impératif d'empaqueter notre application premièrement dans Docker, puis le mettre sur notre compte Dockerhub. Une fois terminé, nous pouvons déployer notre application dans Kubernetes.

En annexe sur notre Github⁶, nous avons mis le dockerfile et un readme qui explique en détail les démarches à suivre pour faire un déploiement similaire.

```

root@master:/home/master# kubectl run perrault-restaurant --image=proa/perrault_resto:v1 --requests=cpu=10m --port=8080
deployment.apps/perrault-restaurant created
root@master:/home/master#

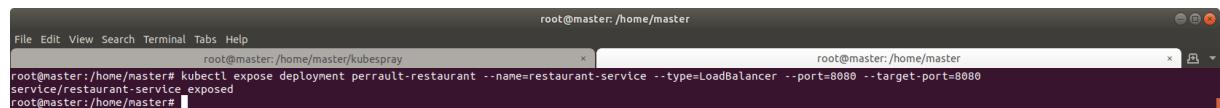
```

FIGURE 34 – Déploiement de l'application

6. <https://github.com/pdjy/Kubernetes>

La commande sur la figure ci-dessus permet de télécharger notre image sur le Dockerhub à l'aide du paramètre **-image** et de le renommer en **perrault-restaurant**. Le paramètre **-port=8080** est le port sur lequel notre application écoute, et **-request=cpu** nous permet de fixer le nombre de CPU que notre application doit utiliser qui est 10 millièmes de CPU.

Nous avons allons créer un service, afin d'avoir une interface externe de notre application sur une seule adresse IP quelque soit le node où se trouve notre application.

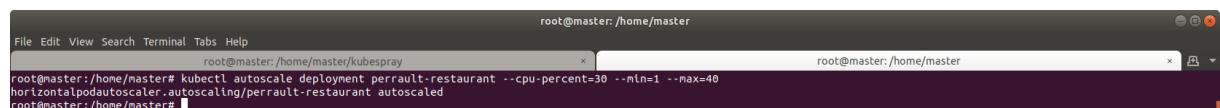


```
root@master:/home/master# kubectl expose deployment perrault-restaurant --name=restaurant-service --type=LoadBalancer --port=8080 --target-port=8080
service/restaurant-service exposed
root@master:/home/master#
```

FIGURE 35 – Exposition du service de l'application

La commande sur la figure ci-dessus permet d'exposer le déploiement de notre application pour qu'on puisse l'atteindre de l'extérieur tout en attribuant un nom à ce service **-name=restaurant-service**. Le paramètre **-port=8080** est le port sur lequel notre service écoute et **-target-port** est le port que ce service utilisera pour se connecter à l'application dans le pod. En utilisant le paramètre **-type=LoadBalancer** notre service sera exposé par un équilibrEUR de charge provisionné de manière dynamique.

Afin d'avoir une application fortement disponible même en cas de surcharge, nous allons mettre en place une mise à l'échelle automatique de l'application.



```
root@master:/home/master# kubectl autoscale deployment perrault-restaurant --cpu-percent=30 --min=1 --max=40
horizontalpodautoscaler.autoscaling/perrault-restaurant autoscaled
root@master:/home/master#
```

FIGURE 36 – Mise à l'échelle de l'application

La commande sur la figure ci-dessus nous permet de mettre en place la mise à l'échelle automatique de l'application en spécifiant le pourcentage de CPU à utiliser **-cpu-percent=30** des 10 millièmes de CPU demander, le nombre maximum **-max=40** et le nombre minimum **-min=1** de répliques qui doivent être créées si le pourcentage de CPU défini a été dépassé.

Sur la figure ci-dessous, nous avons notre application qui est en marche, d'où la couleur verte pour dire que le déploiement, les pods et les replicas sont activés.

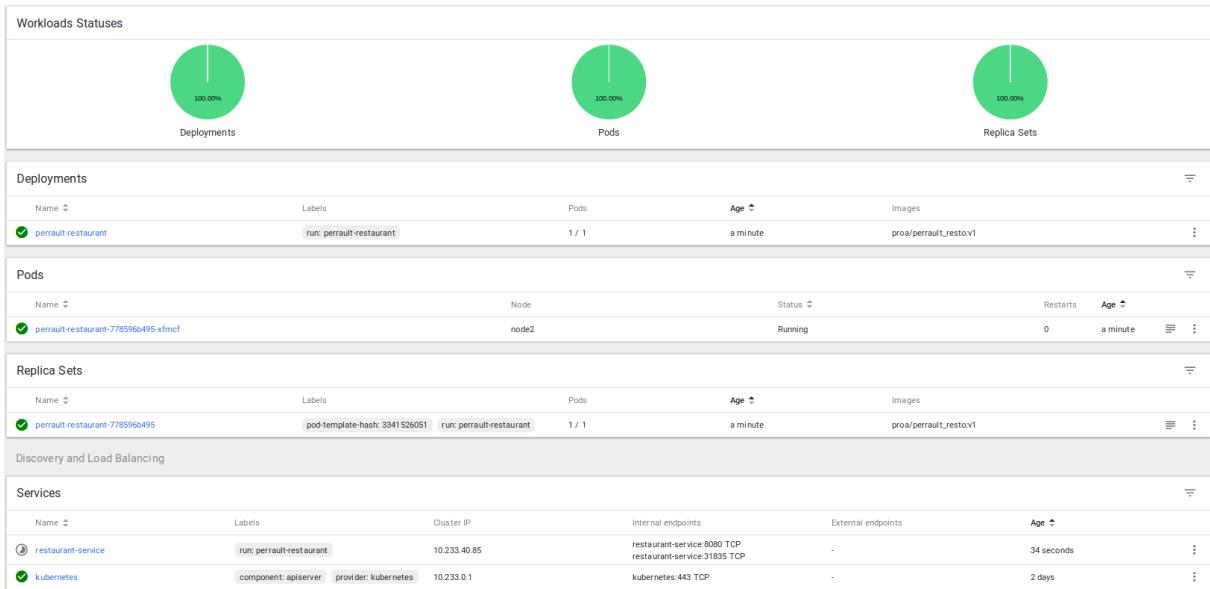


FIGURE 37 – Représentation de l’application

L’adresse IP du service qu’on a créé pour notre application, nous permet de lancer l’application quelque soit le node où se trouve cette application.

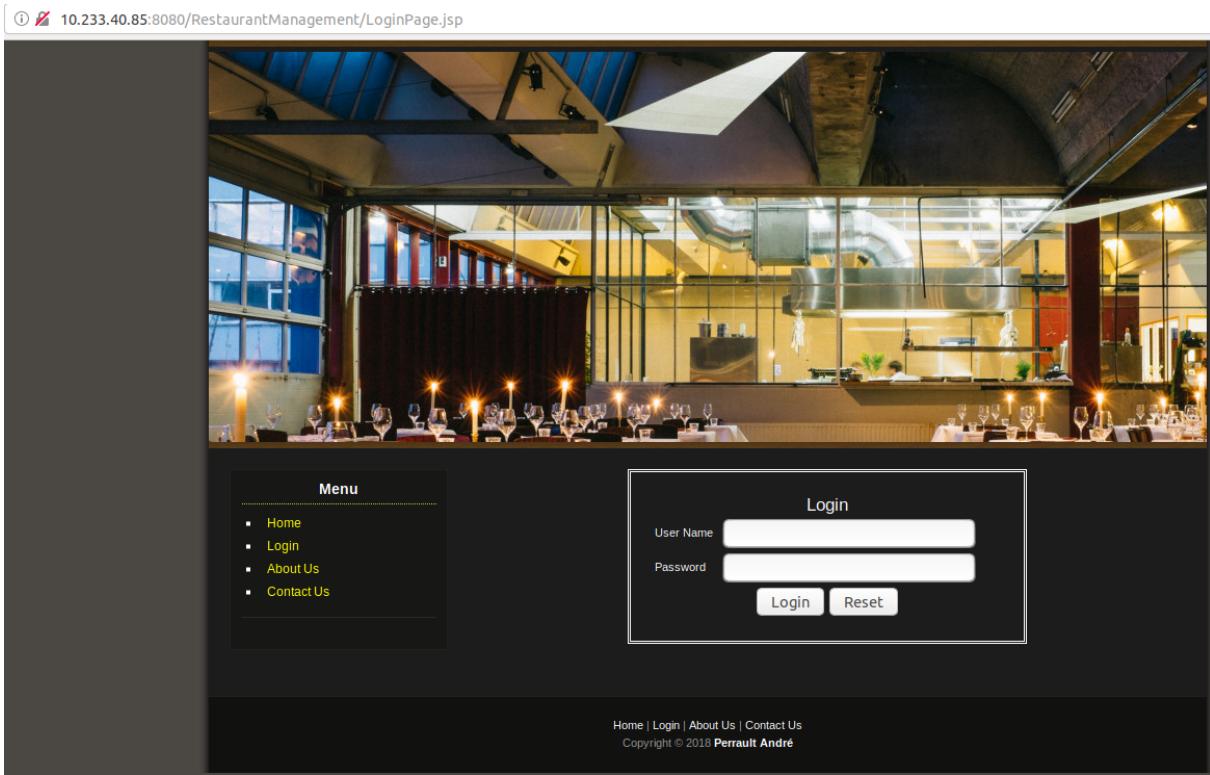


FIGURE 38 – Application identification

En surchargeant l’application, notre système détecte la montée en charge de l’application,

il la compare avec le pourcentage de CPU que nous avons défini et si le pourcentage actuel surpassé le pourcentage initial, il crée des répliques. En visualisant en mode console (**fig.39**), nous voyons le nombre de répliquee créées lors de la montée en charge, le nombre de pods minimum et maximum, le pourcentage initial à droite et le pourcentage actuel à gauche.

```
root@master: /home/master
File Edit View Search Terminal Tabs Help
root@master: /home/master/kubespray x root@master: /home/master x
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 20%/30%  1        40        1          1m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 300%/30% 1        40        1          2m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 300%/30% 1        40        4          2m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 20%/30%  1        40        4          3m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 17%/30%  1        40        4          5m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 17%/30%  1        40        4          6m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 32%/30%  1        40        4          8m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 15%/30%  1        40        2          9m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 180%/30% 1        40        2          9m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 20%/30%  1        40        2          12m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 20%/30%  1        40        2          12m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 20%/30%  1        40        2          14m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 70%/30%  1        40        4          16m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 70%/30%  1        40        4          16m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 20%/30%  1        40        4          17m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 20%/30%  1        40        4          18m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 17%/30%  1        40        4          19m
root@master:/home/master# kubectl get hpa
NAME      REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
perrault-restaurant Deployment/perrault-restaurant 10%/30%  1        40        2          25m
```

FIGURE 39 – Application capacité 1

Nous pouvons aussi le visualiser avec Kubernetes Dashboard (**fig.40**), et sur la (**fig.41**),

nous voyons une baisse automatique des répliques de notre application quand le pourcentage actuel du CPU utilisé est inférieur à l'initial.

Horizontal Pod Autoscalers					
Name	Target CPU Utilization	Current CPU Utilization	Min Replicas	Max Replicas	Age
perrault-restaurant	30%	120%	1	40	46 minutes

FIGURE 40 – Représentation de la mise à l'échelle

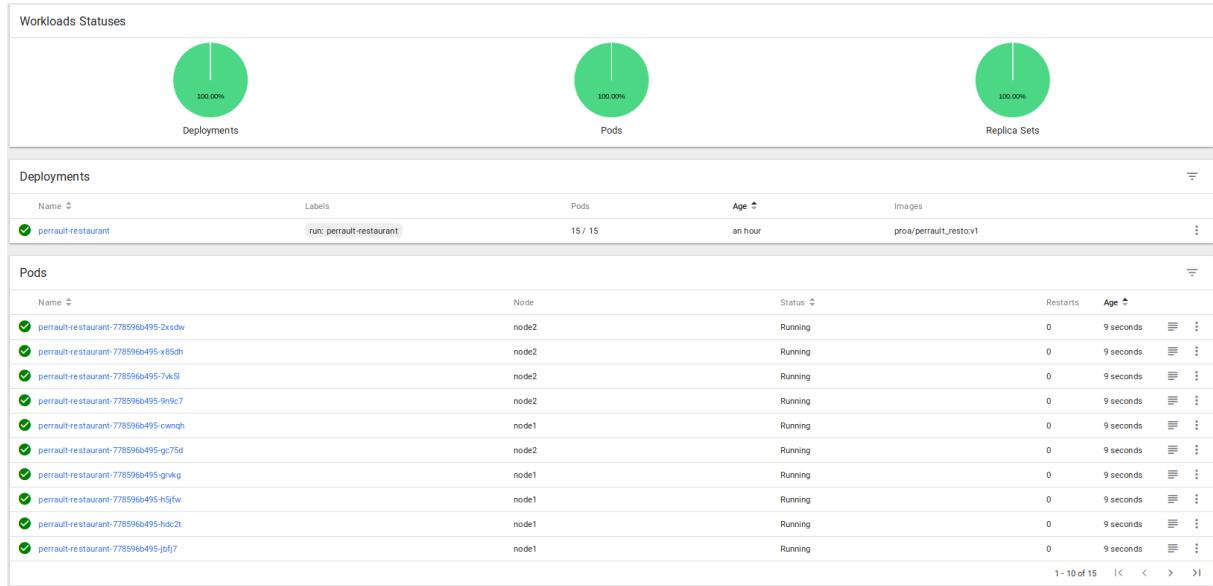


FIGURE 41 – Mise à l'échelle 1

Sur la (fig.42), nous constatons à l'aide de pourcentage, la baisse des réplique, et les ressources (déploiements, pods, et répliques) qui sont pas totalement prêtes.

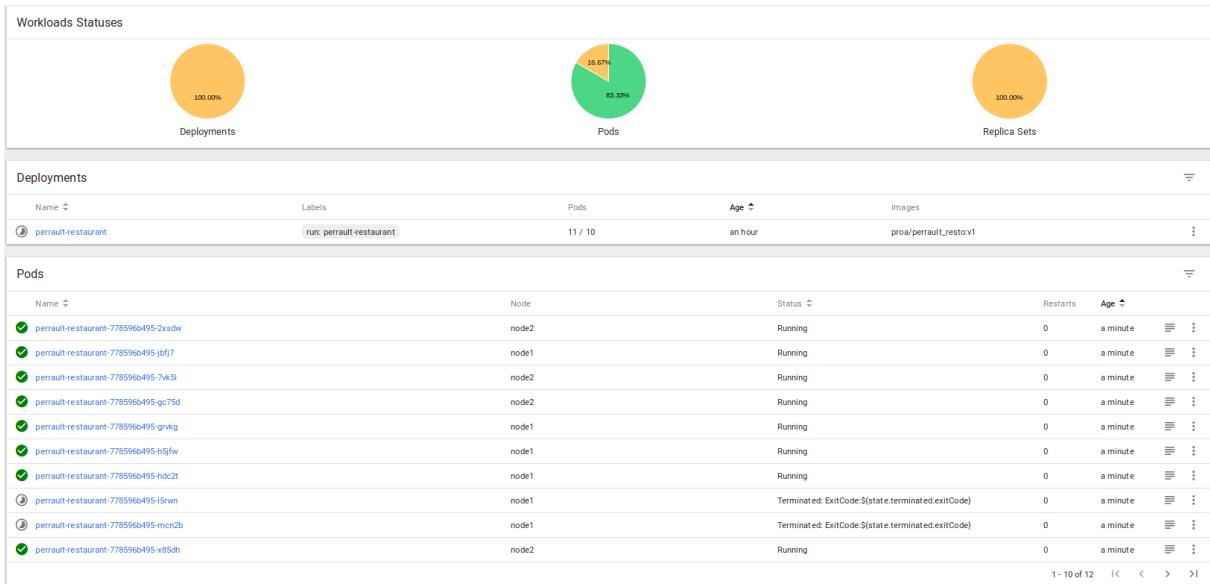


FIGURE 42 – Mise à l'échelle 2

Sur la (**fig.43**), le nombre de répliques est à quatre, en raison d'une baisse au niveau du pourcentage de CPU utilisé qui redevient inférieur au pourcentage initial. Sur la (**fig.44**), nous constatons que l'état notre application retourne à la normal, vu que le pourcentage de CPU utilisé est très basse, ce que signifie que application n'est pas trop chargée.

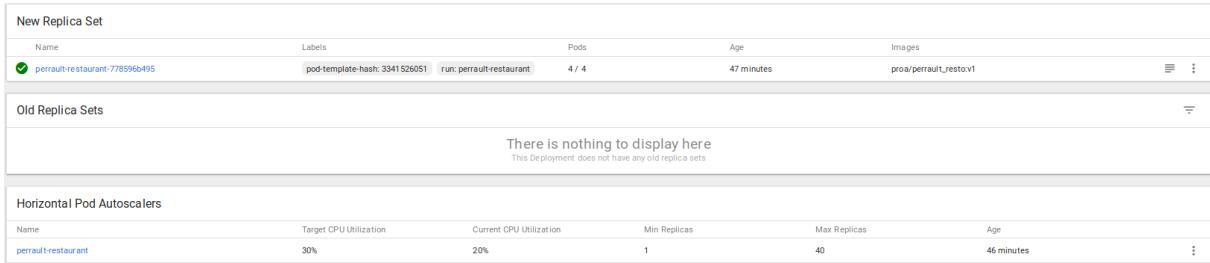


FIGURE 43 – Représentation de la mise à l'échelle 2

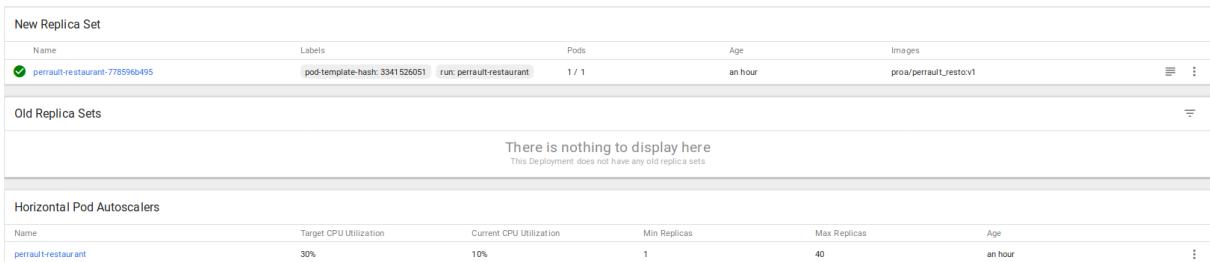


FIGURE 44 – Autoscale Representation 3

Grafana nous offre la possibilité pour visualiser à l'aide de graphes quand l'application

est surchargée et aussi quand elle ne l'est plus, tout en spécifiant le nombre de réplique désirée et le nombre actuelle, (**fig.45**).

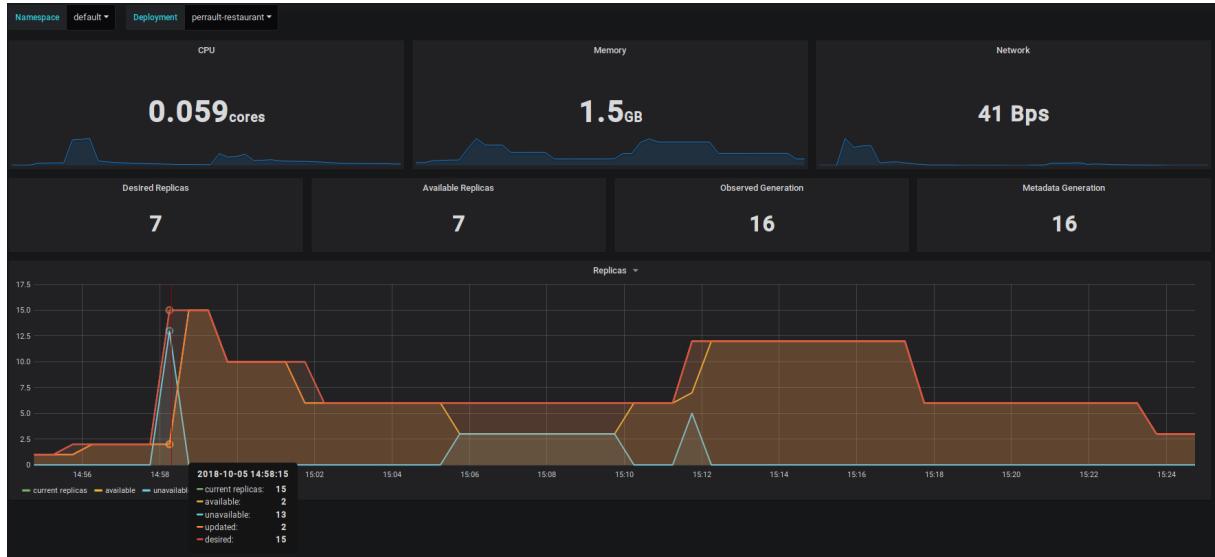


FIGURE 45 – Application Health 1

Pour mieux présenter les avantages de notre système, nous allons débrancher la connexion internet du node2. Une fois débranchée, le système prend environ deux-trois minutes afin de détecter que l'un des nodes n'est plus en service. (voir figure 46)

Nodes							
Name	Labels	Ready	CPU requests (cores)	CPU limits (cores)	Memory requests (bytes)	Memory limits (bytes)	Age
node1	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/hostname: node1 node-role.kubernetes.io/node: true	True	0.673 (16.83%)	1.518 (37.95%)	408.588 Mi (5.19%)	3.075 Gi (40.02%)	2 days
node2	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/hostname: node2 node-role.kubernetes.io/node: true beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/hostname: master node-role.kubernetes.io/master: true	Unknown	0.963 (24.08%)	1.708 (42.70%)	1.044 Gi (13.58%)	4.168 Gi (54.26%)	2 days
master	beta.kubernetes.io/arch: amd64 beta.kubernetes.io/os: linux kubernetes.io/hostname: master node-role.kubernetes.io/master: true	True	1.04 (26.00%)	2.5 (6.250%)	895.773 Mi (11.39%)	5.836 Gi (75.97%)	2 days

FIGURE 46 – Node2 hors service

Une fois que le node2 a été détecté par le système, toutes ses ressources deviennent aussi indisponibles. Nous pouvons constater l'indisponibilité des pods du node2 sur la figure ci-dessous.

Pods						
Name	Namespace	Node	Status	Restarts	Age	⋮
⌚ perrault-restaurant-778596b495-2xsdw	default	node2	Running	0	10 minutes	☰
⌚ perrault-restaurant-778596b495-gc75d	default	node2	Running	0	10 minutes	☰
⌚ perrault-restaurant-778596b495-grvkq	default	node1	Running	0	10 minutes	☰
⌚ perrault-restaurant-778596b495-h5jfw	default	node1	Running	0	10 minutes	☰
⌚ perrault-restaurant-778596b495-skksv	default	node1	Running	0	12 minutes	☰
⌚ prometheus-operator-858c485-qb966	monitoring	node1	Running	0	15 minutes	☰
⌚ kube-prometheus-exporter-kube-state-66b8849c9b-ww89z	monitoring	node1	Running	0	15 minutes	☰
⌚ tiller-deploy-56c4cf647b-pgrv5	kube-system	node1	Running	0	16 minutes	☰
⌚ kube-prometheus-grafana-f869c754-sgh49	monitoring	node1	Running	0	16 minutes	☰
⌚ prometheus-kube-prometheus-0	monitoring	node1	Running	1	34 minutes	☰

FIGURE 47 – Pods du node2 hors service

Ce qui rend hors service toutes les instances de notre application qui roulaient sur le node2.

Non seulement les pods sont hors services, mais les déploiements, les répliques créées par les déploiements, et les états qui leur ont été définis le sont aussi.

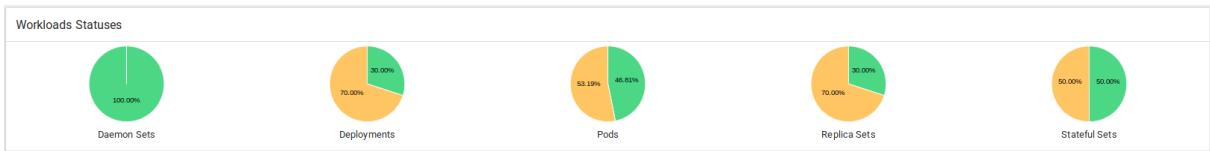


FIGURE 48 – Pourcentage du système hors service

Nous constatons sur les deux figures ci-dessous la régénération automatique des pods par le système. Cette régénération automatique est possible grâce aux spécifications de l'état du déploiement de l'application, quand le système voit que l'état actuel n'est pas égal à l'état souhaité, il produit d'autre pods jusqu'à ce que les deux états soient les mêmes.

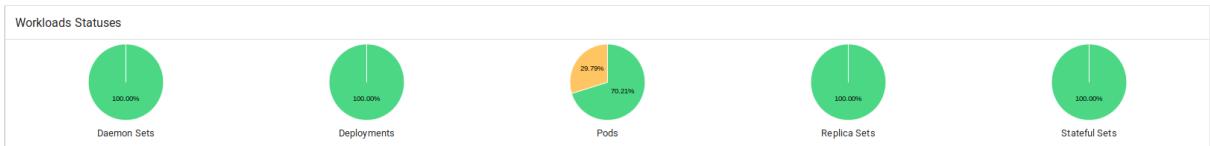


FIGURE 49 – Régénération des ressources hors service



FIGURE 50 – Service de nouveau en marche

Grafana nous permet de visualiser à l'aide des graphes quand notre système était en

marche et quans il ne l'est plus. Sur la (fig.51), nous constatons des graphes très élevés au niveau du réseau, du disque, du CPU, et une fois notre node a été mis hors service, nous voyons une chute totale des données que ce node transférait.

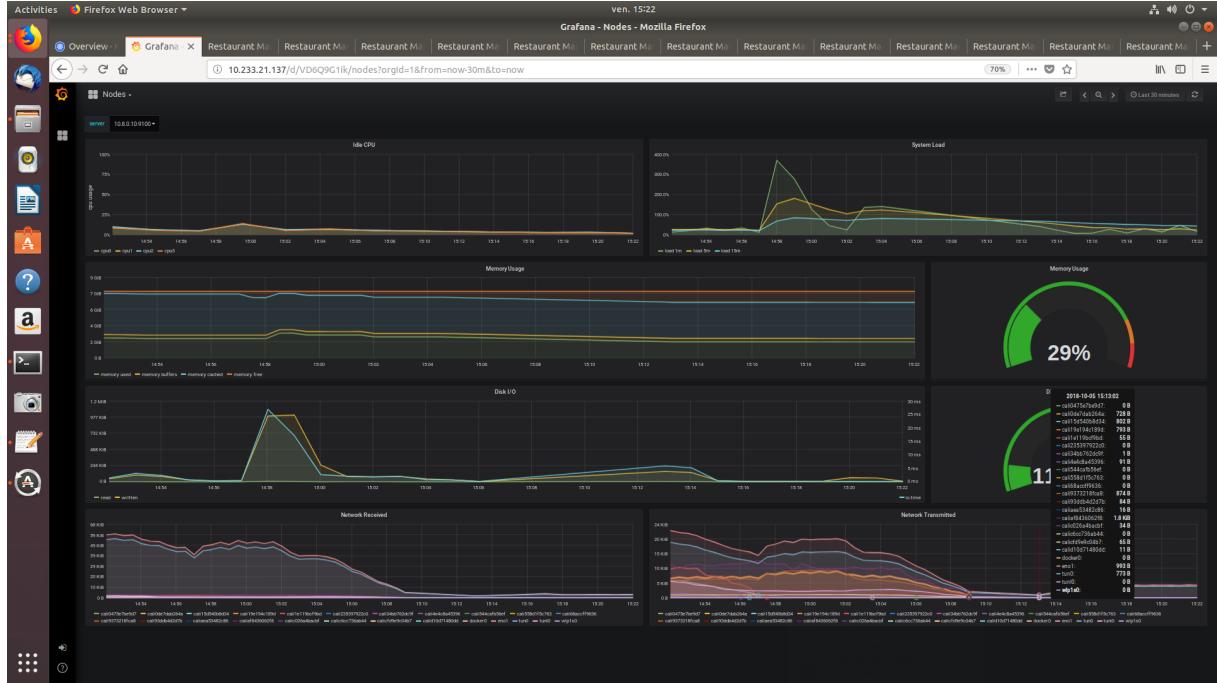


FIGURE 51 – Représentations graphiques du node2 hors service

Et enfin, sur la (fig.52), nous voyons que toutes les répliques de notre application qui tournaient sur le node2 ont été supprimées et ont été ré-attribuées au node1.

Pods						
Name	Namespace	Node	Status	Restarts	Age	
kube-proxy-node2	kube-system	node2	Running	0	a minute	⋮
nginx-proxy-node2	kube-system	node2	Running	0	a minute	⋮
berrault-restaurant-778596b495-nSjzp	default	node1	Running	0	2 minutes	⋮
berrault-restaurant-778596b495-cgjhn	default	node1	Running	0	2 minutes	⋮
berrault-restaurant-778596b495-fq5v4	default	node1	Running	0	2 minutes	⋮
berrault-restaurant-778596b495-4vstb	default	node1	Running	0	2 minutes	⋮
berrault-restaurant-778596b495-nrh7s	default	node1	Running	0	2 minutes	⋮
berrault-restaurant-778596b495-qzqv	default	node1	Running	0	2 minutes	⋮
berrault-restaurant-778596b495-tb6af	default	node1	Running	0	4 minutes	⋮
prometheus-operator-85bc485-zsnhn	monitoring	node1	Running	0	4 minutes	⋮

FIGURE 52 – Attribution des pods sur le node fonctionnel (node1)

8 Conclusion et perspectives

Dans les études réalisées sur l'analyse du sujet, nous avons conceptualisé le thème en donnant des esquisses détaillées, et dans la deuxième partie nous avons présenté les paradigmes de l'orchestration, le monitoring ainsi que la Dockerisation à travers une étude détaillée de l'existant. Par la suite, nous avons exposé quelques méthodes traitant de l'orchestration. Pour la dernière étape, nous avons présenté notre approche avec l'architecture et les résultats obtenus qui, d'un point de vue technique, permettront de résoudre le problème de départ qui était à traiter dans ce document. Ce travail n'est pas exhaustif, mais il est assez complet et peut être utilisé pour comprendre la Dockerisation du Monitoring as a Service (MaaS) et les technologies utilisées pour une reproduction dans un cas d'entreprise, voire académique. Cette reproduction peut être fait sur du bar métal, et pourrait nécessiter d'une autre approche dans d'autre environnement, notamment le cloud.

Comme perspective nous comptons évoluer notre architecture d'orchestration, en la reliant avec le cloud hybride. Nous aurons à ce moment une architecture hautement disponible, munie de plusieurs clusters. Dans le cas où, notre serveur central tombe en panne, nous aurons d'autres serveurs sur le cloud pour prendre le relais. Pourquoi le cloud hybride ? C'est parce qu'on a notre cluster sur du bare métal et pour l'évoluer en gardant le bare métal, il nous faudra un cloud hybride, car il nous offre les deux possibilités, barre métal et cloud.

Bibliographie

- [1] HUNTER II, Thomas. Advanced Microservices : A Hands-on Approach to Microservice Infrastructure and Tooling. Apress, 2017.
- [2] KUTNER, Joe. Deploying with JRuby 9k : deliver scalable web apps using the JVM. Pragmatic Bookshelf, 2016.
- [3] VOHRA, Deepak. Kubernetes microservices with Docker. Apress, 2016.
- [4] Kubernetes Patterns Patterns, Principles, and Practices for Designing Cloud Native Applications.2017
- [5] HOLLA, Shrikrishna. Orchestrating docker. Packt Publishing Ltd, 2015.
- [6] FARCIK, Viktor. The DevOps 2.0 Toolkit. Packt Publishing Ltd, 2016.
- [7] GIRIDHAR, Chetan. Automate it !-Recipes to upskill your business. Packt Publishing Ltd, 2017.
- [8] DUFFY, Michael. DevOps Automation Cookbook. Packt Publishing Ltd, 2015.
- [9] SONI, Mitesh. DevOps Bootcamp. Packt Publishing Ltd, 2017.
- [10] SAITO, Hideto, LEE, Hui-Chuan Chloe, et WU, Cheng-Yang. DevOps with Kubernetes : Accelerating software delivery with container orchestrators. Packt Publishing Ltd, 2017.
- [11] DAVIS, Jennifer et DANIELS, Ryn. Effective DevOps : building a culture of collaboration, affinity, and tooling at scale. " O'Reilly Media, Inc.", 2016.
- [12] MCALLISTER, Jonathan. Implementing DevOps with Ansible 2. Packt Publishing Ltd, 2017.
- [13] NAIK, Ganesh Sanjiv. Learning Linux Shell Scripting. Packt Publishing Ltd, 2015.
- [14] IGNACIO, Roger. Mesos in action. Manning Publications Co., 2016.
- [15] SHARMA, Sourabh, RAJESH, R. V., et GONZALEZ, David. Microservices : Building scalable software. Packt Publishing Ltd, 2017.

- [16] FARCIC, Viktor. The DevOps 2.0 Toolkit. Packt Publishing Ltd, 2016.
- [17] FARCIC, Viktor. The DevOps 2.1 Toolkit : Docker Swarm. Packt Publishing Ltd, 2017.
- [18] <https://docs.openvpn.net/getting-started/openvpn-access-server-installations-optimal>
- [19] <https://tecadmin.net/install-openvpn-server-ubuntu/>
- [20] <https://www.oreilly.com/ideas/swarm-v-fleet-v-kubernetes-v-mesos>
- [21] <https://www.docker.com/>
- [22] <https://kubernetes.io/>
- [23] <https://www.ansible.com/>
- [24] <https://prometheus.io/>
- [25] http://docs.grafana.org/guides/getting_started/
- [26] <https://dzone.com/articles/a-reference-architecture-for-deploying-wso2-middle>
- [27] <https://crondev.com/docker-orchestration-kubernetes-rancher/>
- [28] <http://nxgcloud.com/index.php/2018/05/12/docker-overlay-network/>
- [29] <https://cloudnativelabs.github.io/post/2017-04-18-kubernetes-networking/>
- [30] <https://journaldunadminlinux.fr/tutoriel-deployez-facilement-un-cluster-kubernetes/>
- [31] <https://codefresh.io/kubernetes-tutorial/kubernetes-cheat-sheet/>
- [32] <https://www.projectcalico.org/>