Team C

- AYARI Hadil
- GUIBLIN Nicolas
- MOVSESSIAN Chahan
- PARIS Floriane

Déploiement et Orchestration de Systèmes à Large Echelle

Rapport du TP fil rouge

Pour le 31 janvier 2024

Table des matières

TD1

Partie A : Première version

Nous avons commencé par chacun développer une version des fonctionnalités de base, afin de prendre en main Redis, Prometheus, et les autres technologies demandées.

Pamir les langages proposés nous avons décidé d'utiliser le langage Python pour la rapidité de la création de nouveaux APIs. Notre connaissance du langage Go était trop limitée pour un projet de ce type, et nous étions plus à l'aise en Python qu'en Java.

- Déploiements des bases de données :

Postgresql

auth:

username: postgres

password: xYOa2gvS8N

database: postgres

Redis

host: redis-master

pass: 9DoTmFKdQu

Partie B : Kube

A)

Quelles informations pouvez-vous déterminer avec la commande "kubectl cluster-info" ?

```
floriane@LAPTOP-A980019P:~/orchestration-at-scale-23-24-polymetrie-c$ kubectl cluster-info
Kubernetes control plane is running at https://xyetv9.c1.gra7.k8s.ovh.net
CoreDNS is running at https://xyetv9.c1.gra7.k8s.ovh.net/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
Metrics-server is running at https://xyetv9.c1.gra7.k8s.ovh.net/api/v1/namespaces/kube-system/services/https:metrics-server:/proxy

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

On peut déterminer l'adresse du serveur Kubernetes master auquel kubectl est actuellement configuré pour se connecter, l'adresse de l'API server et l'API http où sont exposés les métriques sur l'utilisation des ressources.

De combien de workers est constitué le cluster livré ? Quelles sont les ressources disponibles (CPU/RAM) ?

Le cluster livré est constitué de 3 workers, chacun disposant de 2 CPU et 6926344 Kib (~= 7,09 Go) de RAM.

```
guiblin@PC-Nicolas:~$ kubectl get nodes -o=custom-columns='NAME:.metadata.name,
CPU:.status.capacity.cpu,MEMORY:.status.capacity.memory'
NAME                                          CPU   MEMORY
nodepool-8e6c2ef1-5e06-4fee-9a-node-127299    2     6926344Ki
nodepool-8e6c2ef1-5e06-4fee-9a-node-b90cfc    2     6926344Ki
nodepool-8e6c2ef1-5e06-4fee-9a-node-f6593b    2     6926344Ki
```

Quels sont les namespaces déjà créés ? Quels sont les pods actifs et leur role ?

Default: Le namespace default est destiné à accueillir les ressources et les applications créées par les utilisateurs qui ne spécifient pas explicitement un namespace. C'est l'espace de travail principal pour le déploiement des applications et des services courants.

Pods actifs dans namespace default : Généralement les pods crées par l'utilisateur pour leurs applications et services

kube-system: Le namespace kube-system est réservé aux composants internes de Kubernetes. Il contient les éléments essentiels au fonctionnement du cluster, tels que les contrôleurs, les services DNS internes, et d'autres processus critiques. Les interactions avec ce namespace doivent être limitées et effectuées avec une compréhension approfondie de Kubernetes, étant donné que des modifications imprudentes peuvent affecter la stabilité et la performance du cluster.

Pods actifs dans namespace kube-system : Les pods actifs sont principalement dédiés au fonctionnement interne de Kubernetes (Kube-apiserver, Kube-scheduler, Kube-controller-manager, Etcd)

kube-public: Le namespace kube-public est automatiquement créé et est accessible à tous les utilisateurs, y compris ceux qui ne sont pas authentifiés. Ce namespace peut être utilisé pour des ressources qui doivent être visibles et accessibles à travers l'ensemble du cluster. Par exemple, il peut contenir des informations de configuration qui doivent être disponibles publiquement.

Pods actifs dans namespace kube-public : On peut trouver des pods qui servent des informations ou des services qui doivent être accessibles à l'ensemble du cluster. Cela pourrait inclure des serveurs de

configuration ou des services de découverte.

B)

Quelle est la fonction de la commande "kubectl proxy" ? Quelle est la différence avec "kubectl port-forward" ? Quels sont les cas d'usage de chacun ?

kubectl proxy est utile pour accéder aux API du cluster, explorer les ressources, et utiliser divers outils qui nécessitent un accès aux API du cluster.

kubectl port-forward est utile pour déboguer des applications, accéder à des bases de données ou d'autres services internes à un pod, sans avoir à exposer ces services directement au public.

Lors de l'installation de "Kubernetes Dashboard" plusieurs ressources ont été créées : quelles sont les relations entre-elles et leur intérêt dans le déploiement de l'application ?

- **Service Account**: Identité utilisée par le tableau de bord pour interagir avec le cluster.
- **Role et RoleBinding**: Définissent les autorisations accordées au compte de service.
- **Deployment**: Définit le déploiement du tableau de bord, spécifiant le nombre de répliques, l'image à utiliser, etc.
- **Service**: Expose le tableau de bord en tant que service interne au cluster.

C)

TD2

Partie A : Terraforming

	Terraform	**Ansible**	**Helm**
Cas d'utilisation	Gestion de l'infrastructure en tant que code, création et gestion de ressources sur divers fournisseurs de Cloud	Configuration d'infrastructures, déploiement d'applications, automatisation des tâches, gestion de la configuration	Gestion de packages d'applications Kubernetes, déploiement et mise à jour d'applications sur des clusters Kubernetes
Forces	Déclaratif, multi-cloud, supporte un large éventail de fournisseurs, gestion des dépendances entre ressources	Agentless, facilité d'utilisation, supporte de nombreux systèmes d'exploitation, modules disponibles pour une grande variété de tâches	Standard pour la gestion des applications Kubernetes, réutilisation des charts, gestion des versions
Faiblesses	Moins adapté à la configuration détaillée des logiciels sur les instances	Limitations sur la gestion des états intermédiaires, moins adapté pour les tâches de provisionnement d'infrastructures	Peut-être trop complexe pour des cas simples, nécessite une connaissance approfondie de Kubernetes
Compatibilité avec GitOps	Automatisation de la gestion de l'infrastructure à partir de fichiers de configuration stockés dans un dépôt Git	Stockage des playbooks dans un dépôt Git et en les exécution automatique	Gestions des déploiements d'applications Kubernetes

Le choix entre Terraform, Ansible et Helm dépend des besoins spécifiques du projet. Ils peuvent être intégrés dans un modèle GitOps, mais cela nécessite une planification et une configuration appropriées.

https://medium.com/@RajeevJ76/the-art-of-automation-tools-helm-operators-ansible-terraform-fb584cf5588d

https://sourceforge.net/software/compare/Ansible-vs-Helm-Kubernetes-vs-Terraform/

https://www.linkedin.com/advice/1/what-best-practices-common-pitfalls-helm-vs-ansible

Partie B : Déploiement de “Nginx Ingress Controller”

Installation : Install WordPress on Kubernetes [Installing WordPress on OVHcloud Managed Kubernetes]

- On a besoin du répertoire bitnami pour être disponible sur helm, donc on l’ajoute →

helm repo add bitnami https://charts.bitnami.com/bitnami

- Installation wordpress

helm install my-first-k8s-wordpress bitnami/wordpress --set allowOverrideNone=true

[Installing Nginx Ingress on OVHcloud Managed Kubernetes]

On déploie ingress-nginx dans un répertoire dédié avec les commandes suivantes :

helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx

helm repo update

helm -n ingress-nginx install ingress-nginx ingress-nginx/ingress-nginx --create-namespace

Une fois que l’ip externe est visible → argocd est prêt

ip : 162.19.109.236

Partie C : ArgoCD

Le principe d'ArgoCD est d'automatiser le déploiement d'une application. Il s'intègre dans la méthodologie GitOps. Les fichiers de déploiements sont sur Git et ce dernier constitue la seule source de vérité. Un Workflow GitHub effectue un test simple (construire et lancer l'application) si le test s'effectue sans erreur l'image construite est uploader vers Docker Hub. Les fichiers de déploiements récupèrent l'image depuis le hub pour construire l'application. ArgoCD écoute ce repository et utilisent ces fichiers .yaml pour redéployer l'app.

Pour installer argoCD dans un cluster Kubernetes (k8) :

1. Cloner le référentiel de l'opérateur argoCD.

2. Installer GoLand.

3. Exécuter les commandes suivantes :

    cd argocd-operator-*

    make deploy

    # Pour désinstaller

    make undeploy

Pour accéder à argoCD

Effectuer un port-forward avec la commande suivante :

kubectl port-forward -n nb-test-argocd service/example-argocd-server 8080:80

vers localhost:8080

ou utiliser k9s, trouver le serveur et appuyer sur SHIFT+F

mdp argo : team-c-argo

Nous avons ensuite mis en place, pour remplacer le port-forward, un ingress pour exposé de manière permanente le service de l'extérieur sur le host :

argo.orch-team-c.pns-projects.fr.eu.org

```
        port:
          number: 8080
```

Une fois ArgoCD déployé et accessible il suffit de lier le repo GitHub et de préciser les fichiers à synchroniser pour l'application en question. Nous avons une configuration équivalente pour Grafana et prometheus, ce qui permet de les redéployer rapidement en cas de changement.

TD3

Partie A : Kube-Prom-Stack

Prometheus Dashboard : http://prometheus.orch-team-c.pns-projects.fr.eu.org/

Quelle est l'intérêt d'utiliser Prometheus dans un contexte de déploiement containerisé et de passage à l'échelle ?

- Prometheus peut collecter des métriques à partir de diverses sources, ce qui le rend adapté aux environnements dynamiques et déployés à l'échelle, tels que les applications conteneurisées
- Prometheus dispose d'un stockage local efficace qui permet de stocker des données sur une période prolongée.
- Prometheus Query Language (PromQL) permet d'effectuer des requêtes flexibles pour extraire des informations spécifiques à partir des métriques collectées

Quelles sont les alternatives à cette technologie ?

- Grafana : interface graphique utilisée en conjonction avec Prometheus
- InfluxDB : base de données de séries temporelles pour stocker et interroger des métriques de performances
- Datadog : service de surveillance cloud
- Sysdig : outil de surveillance avec visibilité approfondie dans les conteneurs et applications

Quelle est la différence entre l'usage de Prometheus et le composant Kubernetes « natif » nommé « Kubernetes Metrics Server » (https://github.com/kubernetes-sigs/metrics-server) ?

- Prometheus est un système de surveillance complet alors que Kubernetes Metric Server se concentre sur la fourniture de métriques de base
- Prometheus composant externe, Kubernetes Metric Server intégré à Kubernetes
- Prometheus a un stockage local pour les métriques, Kubernetes Metric Server stocke les métriques en mémoire
- Prometheus adapté aux besoins de surveillance avancés, et approfondis, Kubernetes Metrics Server fournit rapidement des métriques de base

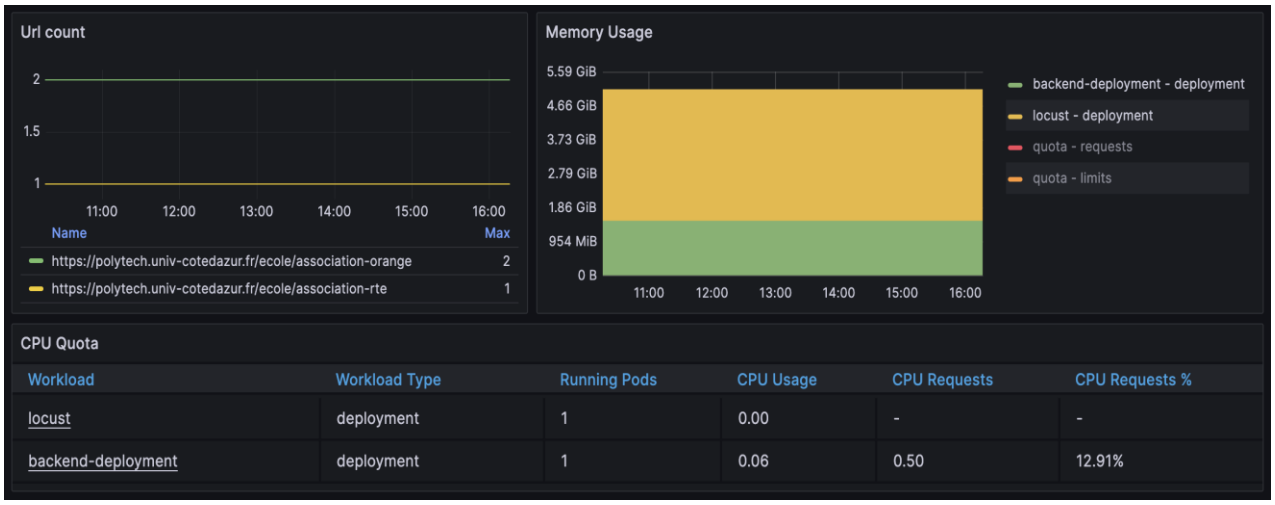
Partie B : Graphes

Grafana : http://grafana.orch-team-c.pns-projects.fr.eu.org/login

User: admin

Password: prom-operator

Premier Dashboard :

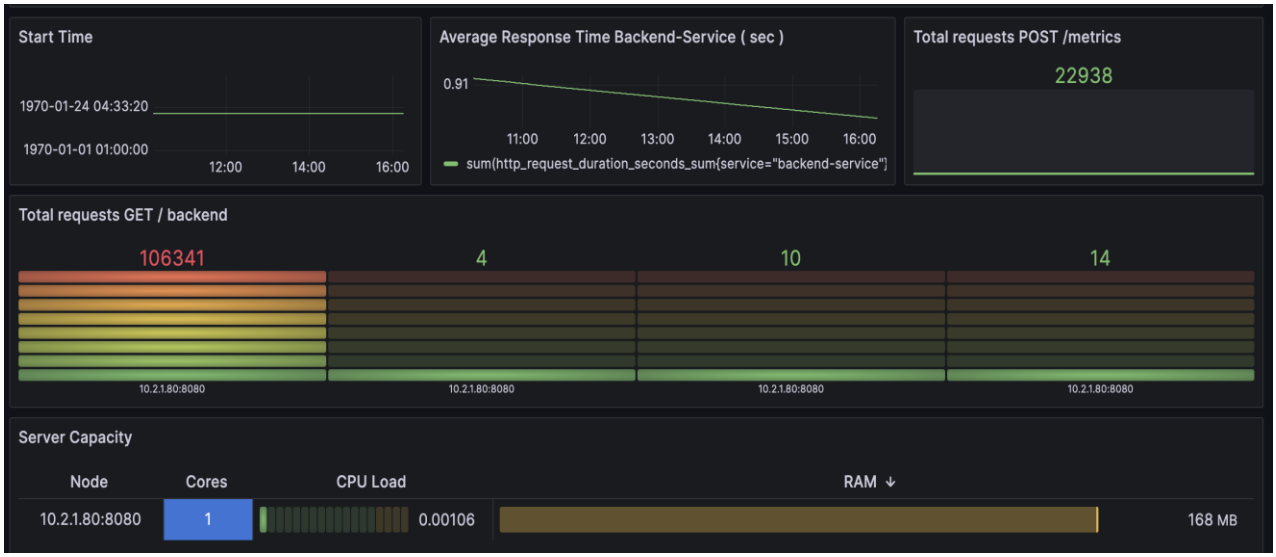


- **Url Count:** Il affiche le nombre de différentes URL surveillées par l'application, ce qui peut refléter la diversité des points de terminaison ou des services que l'application est en train de monitorer. (Métrique métier)

- **Memory Usage:** Il montre la consommation de mémoire des différentes charges de travail ou services au sein de l'application. Les couleurs différentes peuvent représenter la mémoire utilisée par différents déploiements ou limites et demandes de mémoire (quotas).

- **CPU Quota:** Il présente l'utilisation du CPU par les différents workloads.

Deuxième Dashboard :



- **Start Time:** Il indique l'heure de démarrage de chaque instance

- **Average Response Time Backend-Service:** Il fournit des informations sur le temps de réponse moyen du backend, ce qui est crucial pour évaluer la performance et l'expérience utilisateur.

- **Total Requests GET /backend et POST /metrics:** Il affiche le volume de requêtes GET vers un service backend et le volume de requêtes POST pour envoyer des métriques, ce qui peut aider à surveiller la charge sur l'application et son utilisation. (Total requests POST est une métrique métier dans le Dashboard)

- **Server Capacity:** Il montre la capacité du serveur en termes de nombre de cœurs CPU, de charge CPU et de RAM disponible. Cela indique les ressources matérielles attribuées à un node spécifique.

TD4

Configuration du HPA

Le HPA a été configuré pour surveiller les métriques de performance clés de Polymétrie, notamment l'utilisation du CPU et de la mémoire, ainsi que le taux de requêtes par seconde. En fonction de ces métriques, le HPA ajustait le nombre de réplicas de pods pour répondre efficacement à la charge de travail sans surprovisionner les ressources. Actuellement, le HPA est configuré pour créer au maximum 5 réplicas.

Benchmarking

Deux outils de benchmarking ont été employés : **K6** et **Locust**. Ces outils ont été choisis pour leurs capacités distinctes à simuler des charges de travail réalistes sur Polymétrie, permettant ainsi de comparer leurs performances dans des scénarios variés.

Locust : Uniquement des requêtes GET. Locust a offert une flexibilité supérieure dans la définition des comportements des utilisateurs simulés.

K6 : Uniquement des requêtes POST. Il a été particulièrement utile pour tester la robustesse et la stabilité de Polymétrie face à des pics de trafic soudains.

Observations

- Les valeurs observées ont montré une corrélation directe entre l'augmentation de la charge de travail et l'activation du HPA pour augmenter le nombre de réplicas de pods, démontrant ainsi l'efficacité du mécanisme de mise à l'échelle automatique.

- Les métriques utilisées pour évaluer la performance incluaient le temps de réponse moyen, le taux d'erreur des requêtes, et l'utilisation des ressources par pod.

- Concernant les limites de notre application, nous avons soumis celle-ci à un volume important de requêtes, parfois de manière simultanée, en utilisant les deux outils Locust et K6. Bien que des échecs aient été observés lors de surcharges significatives, nous n'avons pas identifié de "limite". Notre hypothèse est que les capacités de nos machines définissent un plafond à la performance observable. Nous n'avons pas réussi à déterminer une limite absolue de traitement des requêtes par notre application.

TD5

1 - L'acronyme **ELK** signifie : **E**lacticsearch, **L**ogstash, **K**ibana.

Elasticsearch est un moteur de recherche et une base de données de documents distribuée. Dans le contexte d'ELK, Elasticsearch est utilisé pour stocker les données de journaux et permet une recherche rapide et flexible.

Logstash est un outil pour la saisie, le traitement et la sortie des données logs. Sa fonction est d'analyser, filtrer et découper les logs pour les transformer en documents formatés à destination d'Elasticsearch.

Kibana est un outil open source de visualisation et d'exploration de données d'Elastic (sous la licence Apache version 2), conçu pour les gros volumes de données en temps réel et en streaming. Il offre des fonctionnalités de tableau de bord, de création de graphiques, de requêtes interactives et de création de visualisations pour aider à comprendre et à analyser les données de journaux de manière plus conviviale.

2 – A la place de **ELK**, l'article nous parle de **ECK** : **E**lastic **C**loud on **K**ubernetes.

Dans l'article, l'alternative à Logstash est **Filebeat**. Filebeat est équipé de modules pour les sources de données d'observabilité et de sécurité qui simplifient la collecte, l'analyse et la visualisation des formats de logs les plus courants.

Filebeat est recommandé pour collecter des logs sur des machines distantes car il nécessite moins de ressources qu'une instance de Logstash ; c'est donc le meilleur choix pour récupérer une donnée déjà traitée et la transmettre directement à Elasticsearch. Pour manipuler des logs en supprimant/ajoutant des champs ou en enrichissant nos données à l'aide de filtres, Logstash aurait été une meilleure idée.

3 - Elasticsearch utilise **mmap** (memory-mapped files) pour gérer l'accès aux fichiers d'index. Le paramètre vm.max_map_count contrôle le nombre maximum de zones de mappage virtuelles pouvant être créées par un processus.

4 - La clé **elasticsearchRef.name** fait référence au nom de l'instance Elasticsearch à laquelle Kibana est censé se connecter. En s'assurant que ces noms correspondent, Kibana peut établir la connexion appropriée à Elasticsearch et commencer à interagir avec lui. Cela simplifie également la configuration et l'intégration, en s'assurant que Kibana sait où trouver Elasticsearch dans l'environnement.

5 - La variable d'environnement **ELASTICSEARCH_HOST** dans la configuration de Filebeat est utilisée pour spécifier l'emplacement de l'instance Elasticsearch à laquelle Filebeat doit envoyer les données de journaux collectées. Cette variable d'environnement permet de définir l'hôte (adresse IP ou nom du serveur) où Elasticsearch est en cours d'exécution.