Baltache Othmane **Mivelle Erwan** **Spessotto Johan** **Lopes Amaro Kevin** **Nicolet Valentin**	**Projet DevOps** **Tom Avenel** **2023 / 2024**	**30/10/2024**

https://github.com/emivelle/DevOps

Tâche	*Personne Attribuée*	*Outil Utilisé*
Isolation par conteneurs	*Valentin*	*Docker, Kubernetes*
Infrastructure as Code (IaC)	*Johan*	*Ansible, Terraform*
Intégration continue (CI)	*Erwan*	*GitHub Action*
Déploiement continu (CD)	*Erwan/Valentin*	*GitHub Action, Rolling update*
Observabilité	*Kevin*	*Grafana, Prometheus*
Logging centralisé	Othmane	*ELK Stack (Elasticsearch, Logstash, Kibana)*
Documentation	*Othmane*	*WORD*

Table des matières

1. Isolation par conteneurs

Il est demandé d'isoler chaque composant métier dans des conteneurs applicatifs. On utilisera donc des conteneurs Docker, que l'on pourra stocker dans le répertoire d'images par défaut (*Docker Hub*) ou dans une registry privée de l'équipe déployée dans le datacenter de test/staging.

Dans un véritable environnement de production, la sécurisation de ce répertoire d'images est une contrainte importante et pouvant induire des choix d'architecture spécifique. Pour le prototype, on pourra exceptionnellement omettre ces questions de sécurité pour simplifier le déploiement.

2. Infrastructure as Code

Afin de suivre la recommandations DevOps d'IaC, il est demandé de rendre reproductible tous les déploiements sous forme de fichiers de code et/ou de fichiers de configuration (Yaml, ...).

On utilisera Ansible pour automatiser l'installation et la configuration des environnements : Docker / k8s, stack applicative. On ne demande cependant pas de provisionner les OS des serveurs des différents environnements depuis Ansible (on démarrera donc le projet

avec un ou plusieurs OS déjà installés). On pourra également utiliser Terraform pour provisionner les ressources (optionnel).

L'ensemble du code nécessaire au déploiement et à la maintenance de l'infrastructure (scripts, fichiers de configuration, …) devront également être versionnés dans un ou plusieurs dépôt(s) de code partagé.

3. Intégration continue (CI)

Afin de garantir l'intégrité des images déployées, on mettra en place un pipeline de CI/CD contenant à la fois des étapes de contrôle de la qualité (tests automatiques, analyse statique de code, …) et la génération et publication des artéfacts de production.

Ce pipeline devra donc en priorité générer une image Docker depuis un commit du code source provenant du dépôt de code.

Pour l'exécution du pipeline, on pourra au choix :

- Déployer un orchestrateur d'intégration continue : Jenkins, Gitlab, …
- Utiliser un orchestrateur SaaS : pipelines Bitbucket, Gitlab cloud, Github

Attention à bien respecter les contraintes d'IaC : la configuration du CI/CD devra aussi être scripté (en général, fichiers Yaml) !

Pour utiliser Jenkins avec Docker en IaC, on pourra se référer à la documentation des pipelines : https://www.jenkins.io/doc/book/pipeline/docker/

4. Déploiement continu (CD)

Une fois les composants applicatifs générés, il faut pouvoir réaliser leur déploiement sur la plateforme.

En bonus, on pourra réaliser également un *déploiement continu de l'infrastructure* suivant un modèle *GitOps*, grâce à un outil comme fluxCD (ou ArgoCD, plus complet mais plus compliqué). Ces outils permettent de mettre à jour automatiquement l'infrastructure de production depuis le dépôt Git des fichiers de configuration de k8s !

En absence de déploiement continu de l'infrastructure, il est tout de même demandé une mise à jour automatique des composants métier en utilisant des scripts et/ou des exécutions Ansible / Terraform.

5. Observabilité

Un système de monitoring devra être mis en place, tant au niveau métier (crash de l'application, …) qu'au niveau de l'infrastructure technique (état des services Docker ou k8s, …)

L'utilisation du couple Prometheus / Grafana est la solution de monitoring par excellence dans les environnements Cloud. Grafana est un outil puissant d'affichage de tableaux de contrôles. Prometheus est un outil de monitoring disposant de multiples sondes permettant de surveiller presque tout type d'application (et d'afficher des tableaux de contrôle minimalistes). Prometheus s'intègre très bien avec Grafana et les 2 outils sont presque toujours utilisés ensemble.

Pour récupérer des sondes Docker, on pourra utiliser si besoin cAdvisor qui permet d'envoyer à Prometheus les informations des conteneurs.

- Pour apprendre à configurer simplement Prometheus, Grafana et cAdvisor en utilisant docker compose, on pourra par exemple utiliser ce tutoriel.
- On pourra, au choix, créer son propre dashboard dans Grafana, ou utiliser un modèle existant, par exemple : https://grafana.com/grafana/dashboards/179-docker-prometheus-monitoring/.

6. Logging centralisé

Un environnement de type Cloud peut contenir de nombreux éléments, générant chacun des logs. Or il est important en cas de problème d'être capable de corréler les différents éléments de l'architecture, et les différents composants applicatifs.

Pour faciliter ce travail, on mettra en place un système de logging centralisé : stack ELK, …

Si ce travail est nécessaire en 1e approche, ce n'est souvent pas suffisant dans un cas réel, car il devient vite compliqué de suivre la requête d'un client depuis le reverse proxy à l'entrée du datacenter, jusqu'aux différents services applicatifs et techniques. En pratique, on utilise donc également des outils de tracing qui permettent de suivre précisément une requête à travers tous les composants déployés : zipkin, OpenTelemetry, … On ne demande pas de mettre en place ce service de tracing dans ce projet.

7. Tâches réalisées

7.1 Isolation par conteneurs

Mise en place de deux deployments (un pour mysql et un pour PhpMyAdmin) avec deux réplicas par conteneur. Le PhpMyAdmin est accessible et la communication entre les pods PhpMyAdmin et ceux de la BBD fonctionne. Le rolling update fonctionne avec un maxUnavailable à 50%.

7.2 Infrastructure as Code

Création de 3 fichiers yaml pour le déploiement de Docker et Minikube sur notre serveur cible. Le premier fichier yaml install_docker.yml va servir à l'installation de docker sur la machine et au démarrage de docker sur le serveur.

Le second fichier install_minikube.yml va servir à l'installation de minikube et kubectl. Ce fichier doit être exécuter après le premier afin que minikube puisse se démarrer avec le driver docker.

Le 3e fichier main_playbook.yml est juste une importation des 2 autres fichiers afin de lancer qu'un seul playbook pour l'installation.

Voici le lien des fichiers yaml:

7.3 Intégration continue (CI)

J'ai créé un Dockerfile pour définir la configuration de phpmyadmin, et un pipeline CI/CD sur GitHub qui s'exécute automatiquement lors d'un push sur la branche principale. Ce pipeline récupère le code, construit une image Docker à partir du Dockerfile, se connecte à Docker Hub avec des identifiants sécurisés, et publie l' image, rendant ainsi l'application prête pour le déploiement.

7.4 Déploiement continu (CD)

L'upload de l'image se fait sur DockerHub tout en intégrant une mise à jour continue (RollingUpdate) dans la configuration Kubernetes pour un déploiement fluide.

7.5 Observabilité

J'ai créé un total de sept fichiers YAML pour la mise en place de mon projet. Parmi ceux-ci, deux fichiers sont dédiés à la création du pod et du service de Grafana, tandis que trois autres fichiers concernent le pod, le service et la configuration de Prometheus. De plus, j'ai élaboré deux fichiers supplémentaires pour ajuster le rôle par défaut,

permettant ainsi à Prometheus de lister les pods et de récupérer les métriques des autres services en cours d'exécution.

7.6 Logging centralisé

Installation de la Stack ELK sur Minikube

Étape	Action	Description
1	**Démarrer Minikube**	Initialiser un cluster Kubernetes local.
2	**Créer ConfigMap pour Elasticsearch**	Configurer Elasticsearch via un ConfigMap.
3	**Déployer Elasticsearch**	Créer un déploiement pour Elasticsearch.
4	**Créer Service pour Elasticsearch**	Exposer Elasticsearch via un service.
5	**Déployer Logstash**	Créer un déploiement pour Logstash.
6	**Créer Service pour Logstash**	Exposer Logstash via un service.
7	**Déployer Kibana**	Créer un déploiement pour Kibana.
8	**Créer Service pour Kibana**	Exposer Kibana via un service.
9	**Déployer Filebeat**	Créer un déploiement pour Filebeat.
10	**Créer Service pour Filebeat**	Exposer Filebeat via un service (si nécessaire).
11	**Appliquer les Configurations**	Appliquer tous les fichiers de configuration dans Kubernetes.
12	**Accéder à Kibana**	Obtenir l'URL de Kibana via Minikube.

Conclusion :

Problèmes de Pods : Certains pods ne démarraient pas ou se retrouvaient dans un état CrashLoopBackOff.

Erreurs de Connexion : difficultés à connecter Kibana à Elasticsearch en raison de la sécurité X-Pack qui n'était pas activée.

Configuration de Filebeat : problèmes pour que Filebeat envoie les logs correctement à Elasticsearch.

Complexité Générale : L'écosystème ELK s'est avéré trop complexe à gérer, ce qui vous a amené à envisager une alternative.

8. Tâches à finir

8.1 Isolation par conteneurs

Autoscale et volumes persistant pour la BDD

8.2 Infrastructure as Code

Je dois finaliser un dernier fichier YAML qui permettra de récupérer automatiquement la dernière image de notre instance phpMyAdmin disponible sur Docker Hub. Ce fichier déploiera ensuite cette image avec Minikube et créera une tâche cron pour répéter ces opérations toutes les 10 minutes.

8.3 Intégration continue (CI)

Je prévois de m'assurer du bon fonctionnement de l'ensemble de la CI/CD lors de la prochaine séance

8.4 Déploiement continu (CD)

Je prévois de m'assurer du bon fonctionnement de l'ensemble de la CI/CD lors de la prochaine séance et si cela est possible d'un outil plus puissant pour l'application des modifications.

8.5 Observabilité

Il ne me reste plus qu'à mettre en place le tableau de bord Grafana et à vérifier la bonne récupération des métriques sur les autres pods

8.6 Logging centralisé

Après avoir rencontré ces difficultés avec la stack ELK, vous avez décidé de changer pour une solution plus simple, en optant pour Grafana et Loki. Voici les étapes que vous allez suivre pour effectuer cette migration.

Étape	Action	Description
1	**Démarrer Minikube**	Initialiser un cluster Kubernetes local.
2	**Créer ConfigMap pour Loki**	Configurer Loki via un ConfigMap.
3	**Déployer Loki**	Créer un déploiement pour Loki.
4	**Créer Service pour Loki**	Exposer Loki via un service.
5	**Déployer Grafana**	Créer un déploiement pour Grafana.
6	**Créer Service pour Grafana**	Exposer Grafana via un service.
7	**Configurer Grafana pour utiliser Loki**	Ajouter Loki comme source de données dans Grafana.
8	**Déployer un agent de collecte (par exemple, Promtail)**	Déployer un agent pour collecter les logs et les envoyer à Loki.
9	**Créer Service pour l'agent de collecte**	Exposer l'agent de collecte via un service (si nécessaire).
10	**Accéder à Grafana**	Obtenir l'URL de Grafana via Minikube.