Projet DevOps

Tom Avenel

2023 / 2024

Table des matières

Contexte de l'atelier

Dans le but d'accélérer le *time-to-market* d'une application et d'augmenter la qualité de celle-ci, il est décidé de réaliser un déploiement continu de l'application par l'utilisation de conteneurs et de CI/CD, en utilisant un pattern d'Infrastructure-as-Code.

Présentation du projet

Contexte

Le projet à utiliser est une application multi-composants (typiquement, une application Web) - on pourra réutiliser une application déjà créée par le groupe (à préférer) ou l'application donnée en exemple. Ce projet devra être productivisé dans un cadre Devops (voir contraintes ci-dessous).

Le formateur jouera l'ensemble des rôles externes à l'équipe projet (et donc principalement celui de client pour l'application finale).

Contraintes

Certaines contraintes ont déjà été identifiées sur le projet. Ces contraintes sont susceptibles d'évoluer en fonction des retours des utilisateurs, et de nouvelles contraintes pourront être ajoutées par le client si besoin.

- **Déploiement au plus tôt :** le projet est très prometteur et de nombreux concurrents se sont déjà positionnés sur le marché. Il est donc absolument primordial de disposer d'une première version en production de toute urgence afin de créer un marché captif au plut tôt. Cette version sera certainement très limitée dans un premier temps, et améliorée au fur et à mesure des mises à jour.
- **Auto-scaling** : il est difficile de prévoir le modèle d'adoption des applications proposées, cependant le service marketing prévoit déjà une montée en charge importante des utilisateurs dès les premières itérations. Il est donc nécessaire que l'infrastructure supporte un service de scaling **dès la 2e itération**

- **Zero-downtime** : ce projet est critique pour le business du client, celui-ci insiste sur le fait qu'il n'est pas envisageable d'avoir une interruption de service lors de la mise à jour des composants applicatifs ou lors de changements dans l'infrastructure.

Partie Devops

Le projet sera réalisé en suivant les préconisations des pratiques DevOps.

L'ensemble des points ci-dessous sont des réflexions à avancer en parallèle pour permettre une compatibilité entre chaque point technique du projet.

Environnements à déployer

- Afin de pouvoir déployer des conteneurs, il faudra un environnement de production robuste : on déploira donc un cluster Kubernetes. Étant donné les ressources limitées sur ce projet, on pourra se limiter à une machine virtuelle tournant sur la machine d'un des apprenants afin de démarrer un noeud Kubernetes unique (par exemple, un déploiement Minikube).
- Il n'est bien sûr pas envisageable de déployer directement une application non testée en production. On veillera donc à créer un 2e environnement qui servira de test et/ou staging. Les contraintes étant moindres sur cet environnement, on pourra au choix déployer un autre cluster Kubernetes (limité) ou se contenter d'un orchestrateur `docker-compose`.

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.

Infrastructure as Code (IaC)

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 partagés.

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 ausssi être scripté (en général, fichiers `Yaml`) !

Lien

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

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.

Zero-downtime L'application devra être mise à jour en respectant des contraintes de zero downtime : les clients doivent pouvoir accéder en permanence à l'application. On réfléchira donc à l'adaptation du modèle de déploiement continu, et on pourra utiliser un outil comme `flagger` pour éviter une interruption de service.

On ne demande cependant pas de mettre en place un HA proxy (pas de "vrai" service de haute disponibilité).

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.

Lien

- 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/.

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.

Résultat attendu

- Un document décrivant les différents éléments à mettre en place et résumant l'ensemble des principes Devops suivis dans le projet et comment ceux-ci seront implémentés. Ce document sera remis en cours de réalisation
- Le ou les dépôts de code source utilisés, que ce soit pour la gestion du code source des applications métier, ou pour la gestion des configurations des infrastructures.
- Le rapport et les détails d'implémentation seront présentés lors d'une soutenance de projet (dernier cours).

Legal