

Formation GitLab : Concepts et Architectures

El Hadji Gaye

Auteur El Hadji Gaye

Pour Formation

Date 16/09/2024

Objet Formation GitLab : Concepts et Architectures

I)	Qu'est-ce que le DevOps	3
II)	Quelques Outils DevOps pratiques	4
1.	Planification et collaboration.....	4
2.	Gestion du code (développement)	5
3.	Intégration Continue / Déploiement Continu (CI/CD)	6
4.	Gestion de l'infrastructure	7
5.	Surveillance et retour d'information	8
III)	La nécessité d'une architecture DevOps.....	9
IV)	Qu'est-ce que le CI/ CD ?	24
V)	Le langage YAML.....	26

I) Qu'est-ce que le DevOps

Devops est la concaténation des trois premières lettres du mot anglais development (développement) et de l'abréviation ops du mot anglais operations (exploitation).

C'est un terme qui a été inventé par le belge Patrick Debois en 2007.

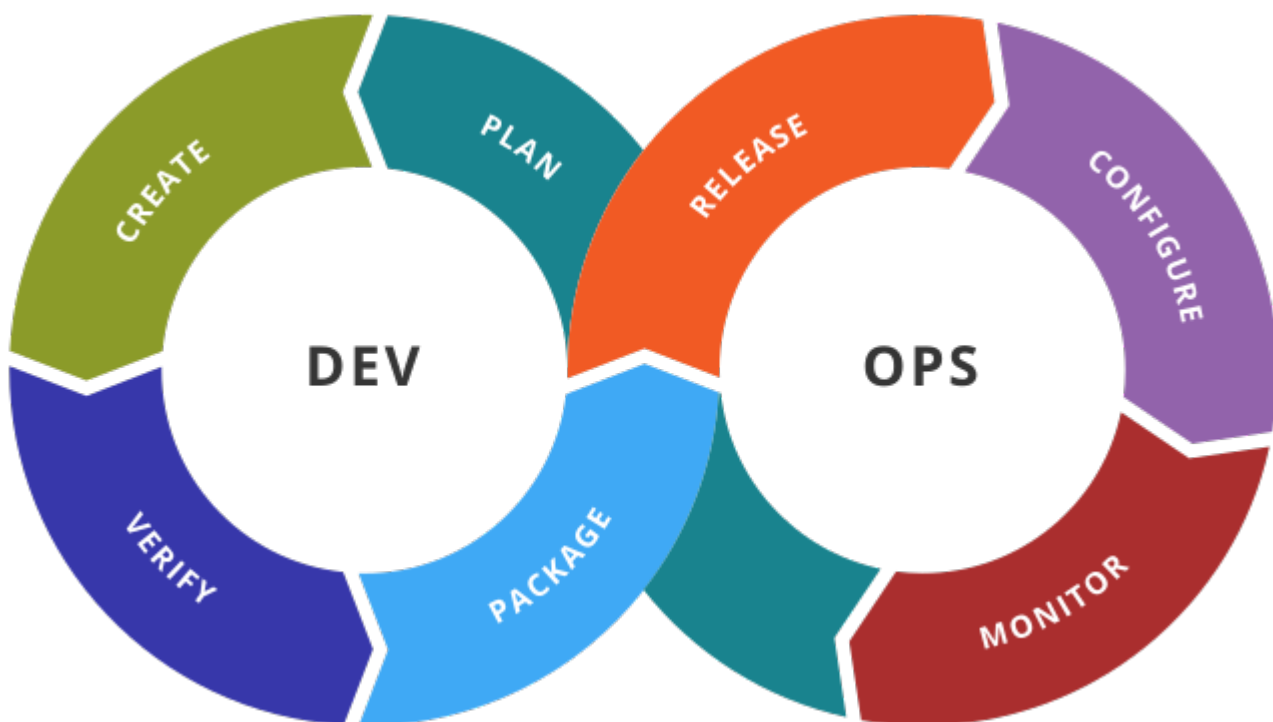
Le DevOps est un ensemble de pratiques qui visent à réduire le fossé entre le développement logiciel (Dev) et les opérations informatiques (Ops), d'où le terme.

L'idée est de favoriser une collaboration plus étroite et une meilleure communication entre ces deux entités qui, dans les modèles traditionnels, opèrent souvent de manière isolée.

Les principes clés du DevOps comprennent aujourd'hui :

- l'intégration continue (CI pour continuous integration en anglais - le code est régulièrement fusionné et testé),
- la livraison continue (CD pour continuous delivery - les mises à jour du logiciel sont régulièrement libérées pour la production),
- l'infrastructure en tant que code (la gestion et la provision des infrastructures informatiques via le code - IAC en anglais pour Infrastructures As Code),
- la surveillance et la journalisation (le suivi en temps réel de la performance et des erreurs du logiciel)
- la culture de la rétroaction (l'encouragement à l'amélioration constante via les retours d'information)

Le schéma classique est celui-ci :



II) Quelques Outils DevOps pratiques

1. Planification et collaboration

Ces outils permettent de créer des tâches et de gérer un projet. Les plus utilisés sont :

- Gitlab
- Github
- Jira

2. Gestion du code (développement)

Ces outils permettent d'effectuer un contrôle de version du code. Les plus utilisés sont :

- GitHub
- GitLab
- Bitbucket

3. Intégration Continue/Déploiement Continu (CI/CD)

Ces outils surveillent les commits dans votre dépôt par exemple Github. Lorsqu'un commit est effectué, ils lancent automatiquement un "pipeline" d'intégration continue qui peut compiler le code, exécuter des tests unitaires, des tests d'intégration, et d'autres types de tests pour s'assurer que les dernières modifications n'ont pas introduit de bugs.

Si tous les tests passent, ces outils peuvent être configurés pour déployer automatiquement les changements sur un environnement de production, de staging ou de test. Cela accélère le processus de livraison de nouvelles fonctionnalités et de corrections de bugs.

Des solutions très connues sont :

- Jenkins
- GitLab CI/CD
- Github actions
- AWS CodePipeline
- Azure DevOps
- CircleCI
- Travis CI

4. *Gestion de l'infrastructure*

Ces outils permettent de faire des choses très différentes mais concernent les serveurs et les clusters.

Nous pouvons citer quelques exemples :

- **Docker** : pour créer des images et ensuite les exécuter dans des conteneurs sur un cluster.
- **Docker Hub** (ou tout autre Container Registry - il y en a plusieurs dizaines) : plateforme de service cloud qui permet aux développeurs de stocker et d'utiliser des images d'applications conteneurisées.
- **Kubernetes** : plateforme qui automatise le déploiement, la mise à l'échelle et la gestion des applications conteneurisées, offrant un cadre pour orchestrer et coordonner des conteneurs au sein d'un environnement de cloud.
- **Terraform** : outil d'Infrastructure as Code (IaC) open source qui permet aux développeurs de définir et de fournir des infrastructures de centres de données en utilisant un langage de description déclaratif, facilitant ainsi la gestion et l'orchestration des ressources cloud.
- **Ansible** : outil d'automatisation open source qui permet la gestion de configuration, le déploiement d'applications et l'orchestration de tâches sur une variété de systèmes et de plateformes (en résumé permet de configurer et de gérer des serveurs plus simplement).

5. Surveillance et retour d'information

Ces outils permettent de surveiller (monitoring) des clusters ou plus généralement des applications exécutées sur des serveurs.

Voici une liste des outils les plus courants :

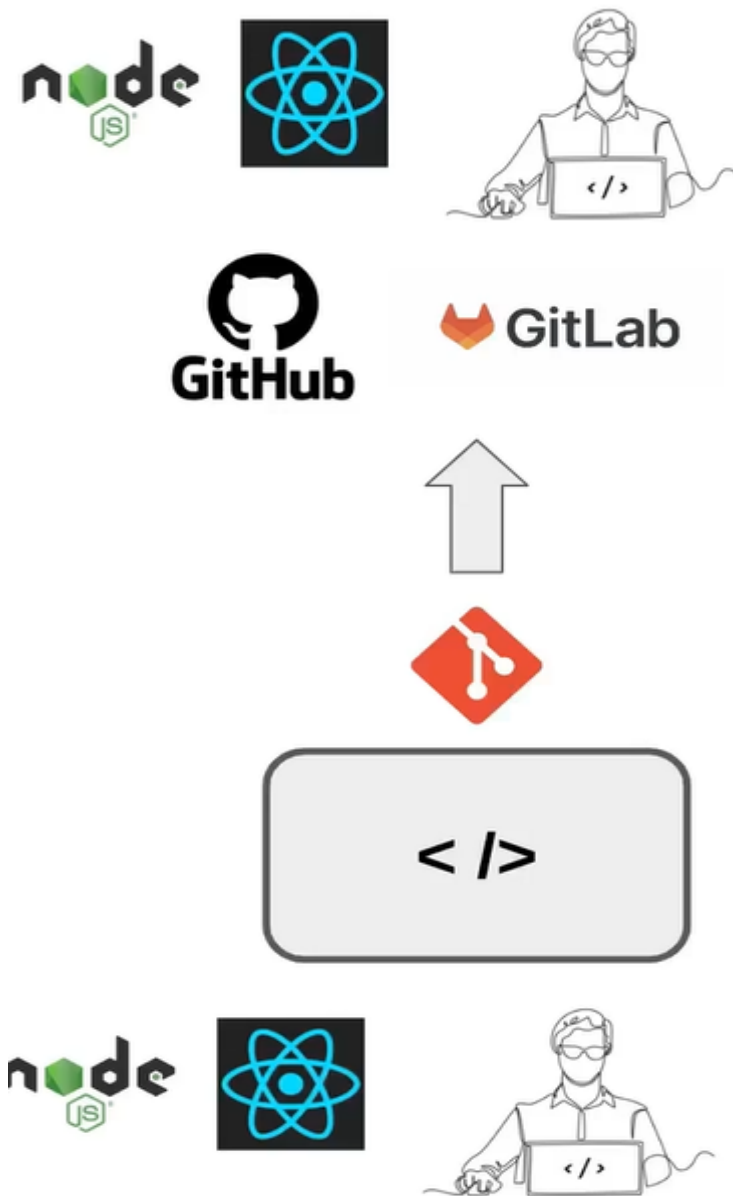
- **Prometheus** : système de surveillance et d'alerte qui collecte et stocke les métriques d'application et de système en temps réel, offrant des fonctionnalités de requête et d'alerte pour aider à la détection et à la résolution des problèmes.
- **Grafana** : plateforme pour la visualisation et l'analyse de données, permettant aux utilisateurs de créer des tableaux de bord interactifs et compréhensibles pour surveiller et analyser en temps réel les données provenant de diverses sources, le plus souvent de Prometheus.
- **ELK Stack (Elasticsearch, Logstash, Kibana)** : suite d'outils qui fournit des capacités de recherche, d'analyse, de journalisation et de visualisation de données, permettant aux utilisateurs de transformer leurs données en insights précieux.
- **Datadog / Nagios / New Relic / Sentry** : plateformes de surveillance et d'analyse des performances en temps réel pour les infrastructures cloud, les applications, les journaux et les métriques, facilitant la détection des problèmes et leur résolution.

III) La nécessité d'une architecture DevOps

Partons d'une situation simple d'un développeur qui a envie de développer une application Java, .NET, JavaScript ect...



Supposons que le développeur décide de développer une application frontend en React et backend en NodeJS.



Pour gerer les nombreuses versions de l'application nous allons stocker le code dans un repository GitHub ou GitLab.

Nous allons par la suite exposer notre application chez OVH par exemple :



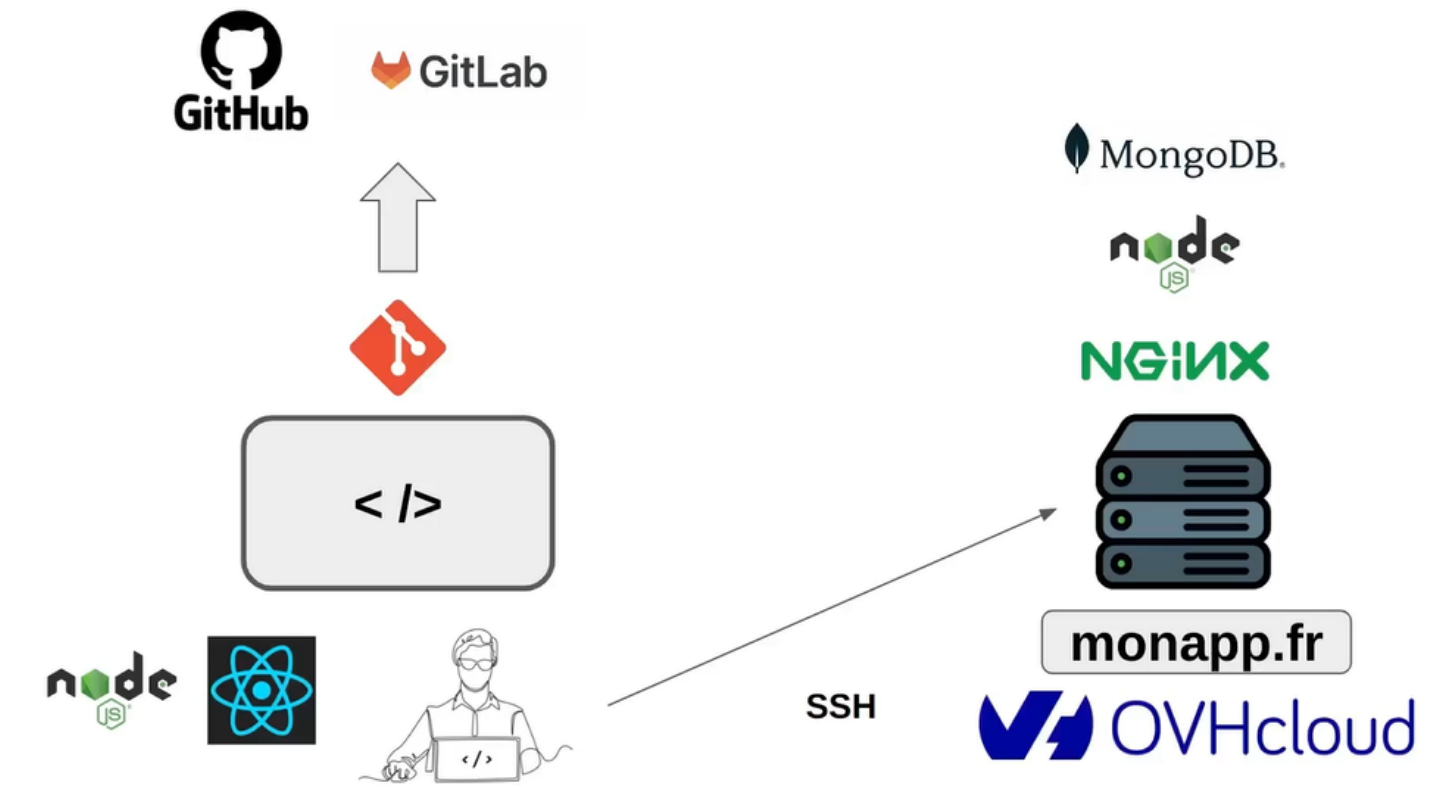
Ensuite vous achetez un nom de domaine par exemple **monapp.fr** :



monapp.fr



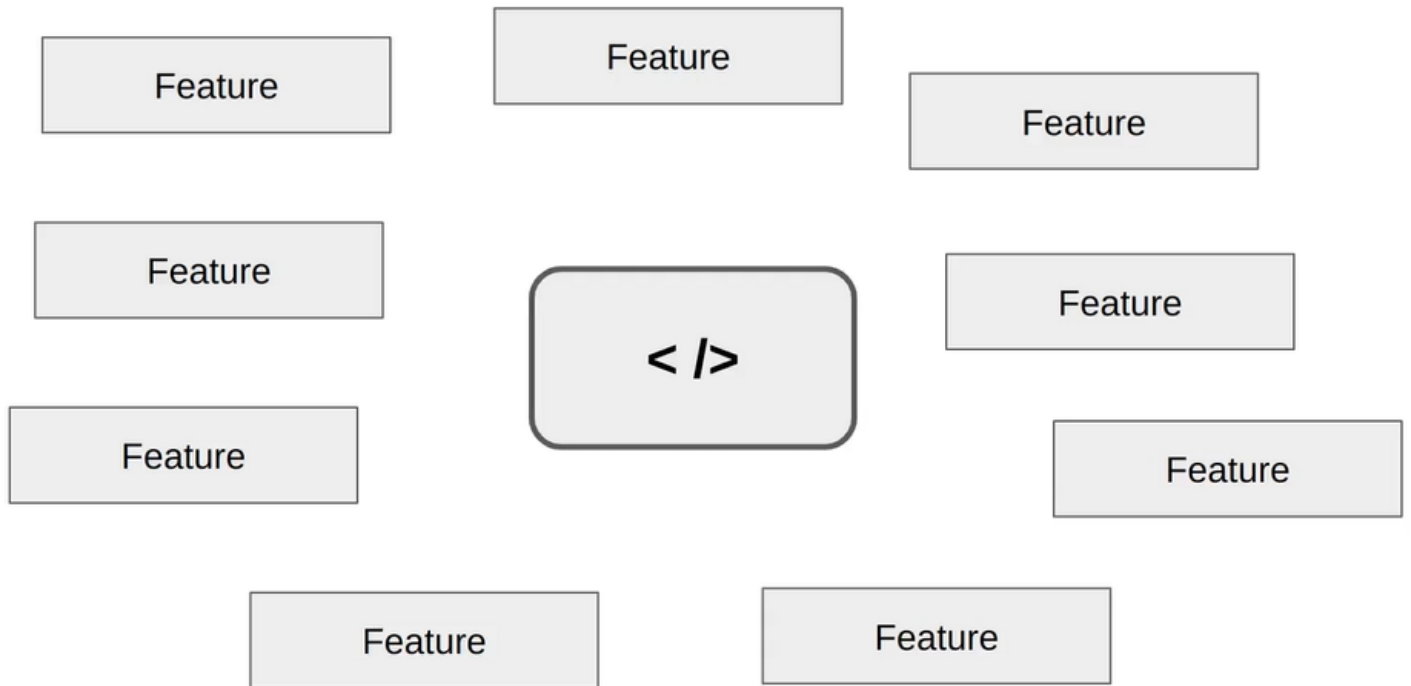
Nous allons ainsi pouvoir récupérer notre code dans le serveur « **monapp** » puis lancer l'application avec toutes ces dépendances (MongoDB, Node JS et NGINX).



Nous avons donc deux environnements distinct : un environnement de développement et un environnement de production.



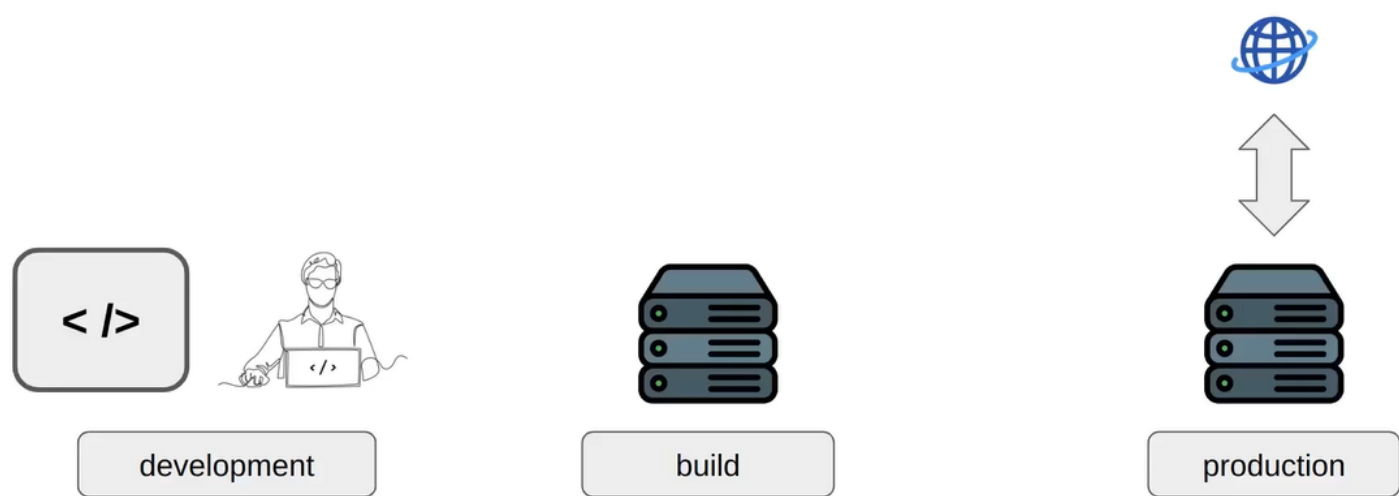
Imaginons maintenant que notre application devienne très populaire et qu'on ait développé pleinde features.



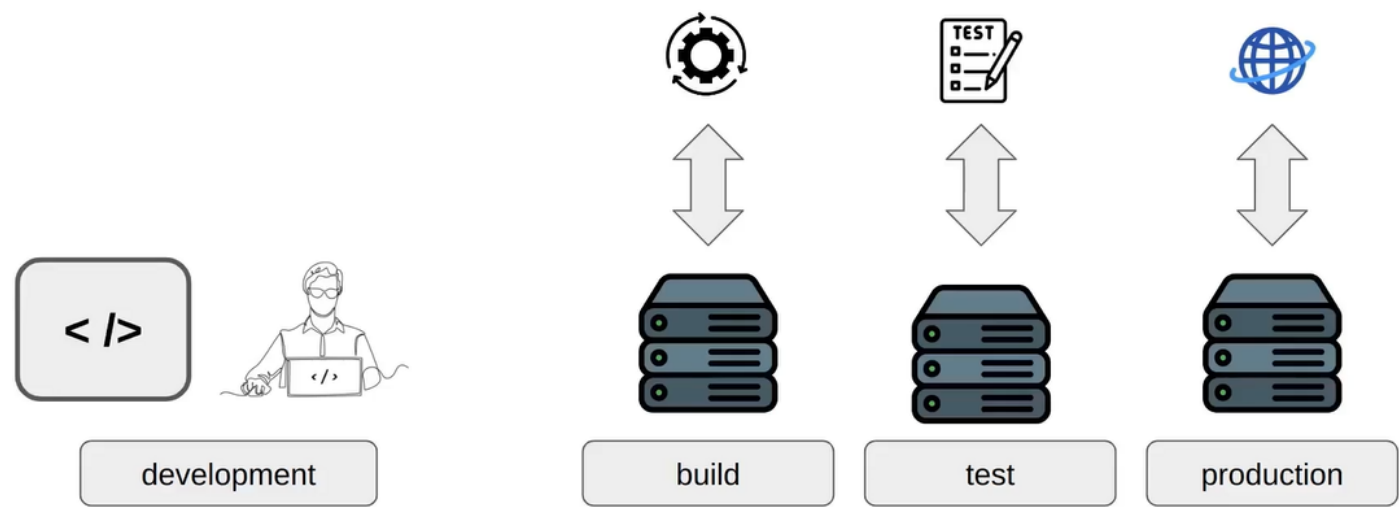
A chaque fois qu'un feature est développé et livré alors on accroît le risque d'apparition de régressions.



Pour eviter d’etre confronté à ce genre de probleme nous allons améliorer l’architecture applicative :
La première amelioration consiste à ajouter un serveur qui va gérer le build de l’application.



La deuxième amélioration consiste à ajouter un serveur qui va gérer les tests de l’application.

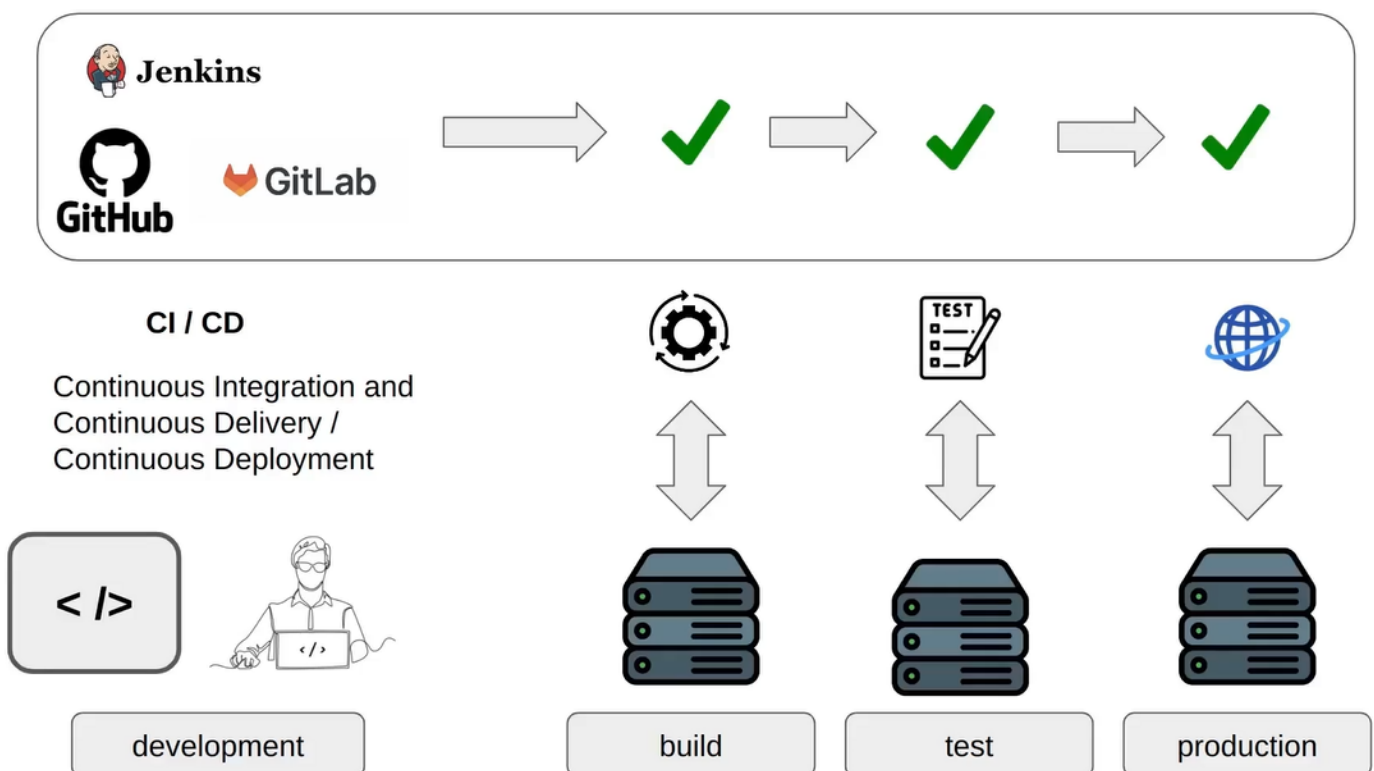


La troisième amélioration consiste à ajouter une chaîne de CI/CD qui va gérer l'intégration continue et le déploiement continu de l'application.

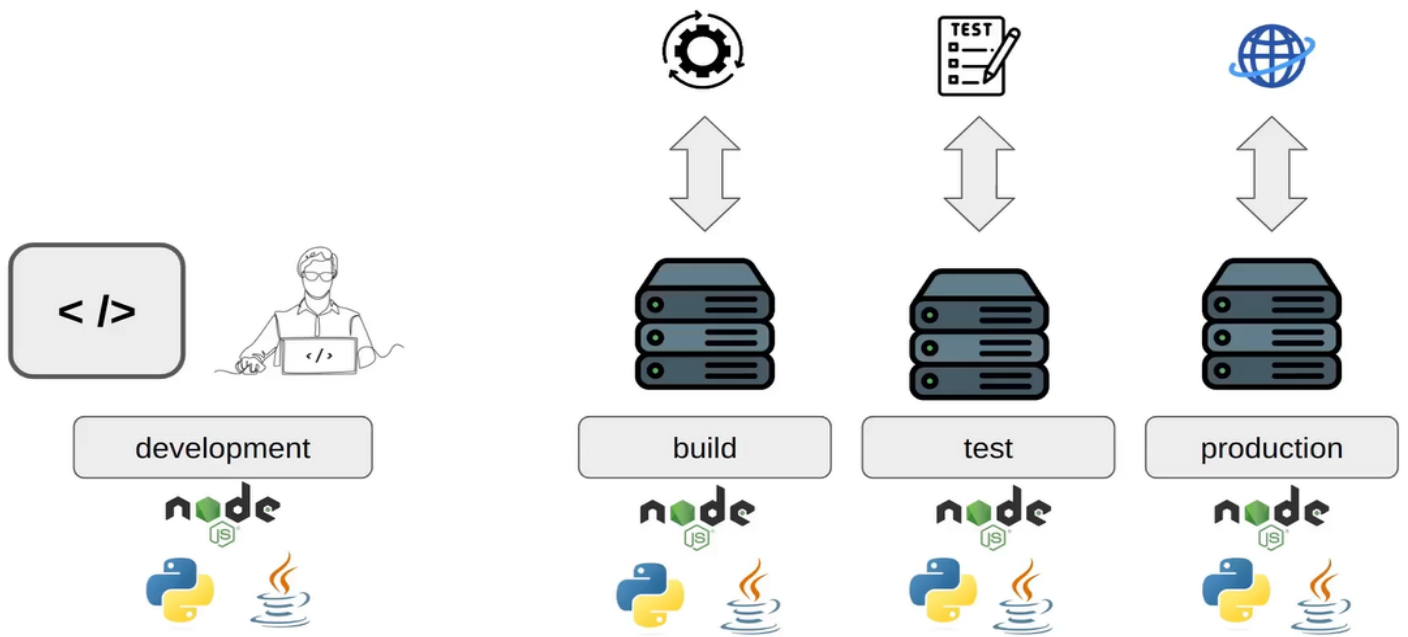
CI / CD

Continuous Integration and Continuous Delivery / Continuous Deployment

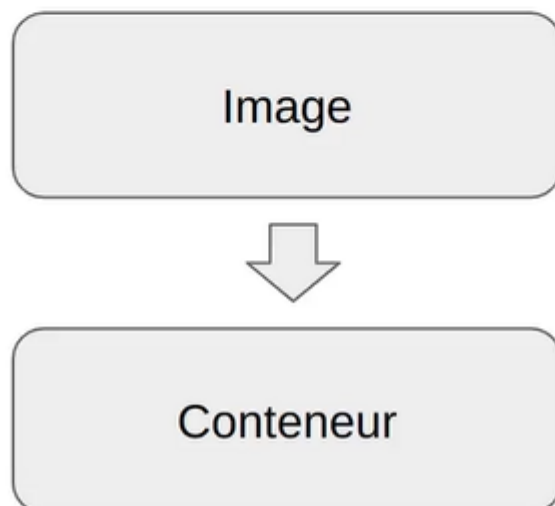
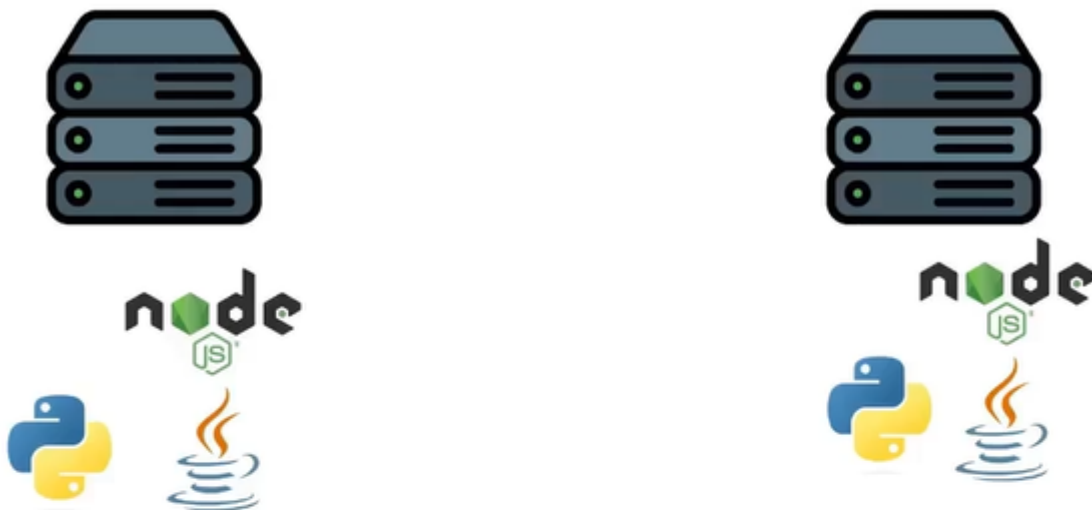
Cette chaîne de CI/CD va permettre de détecter les commit puis de lancer un Build de l'application puis le Test de l'application via le code du Build et enfin le déploiement de l'application sur le serveur de production. On pourra utiliser des pipelines pour réaliser toutes ces opérations.

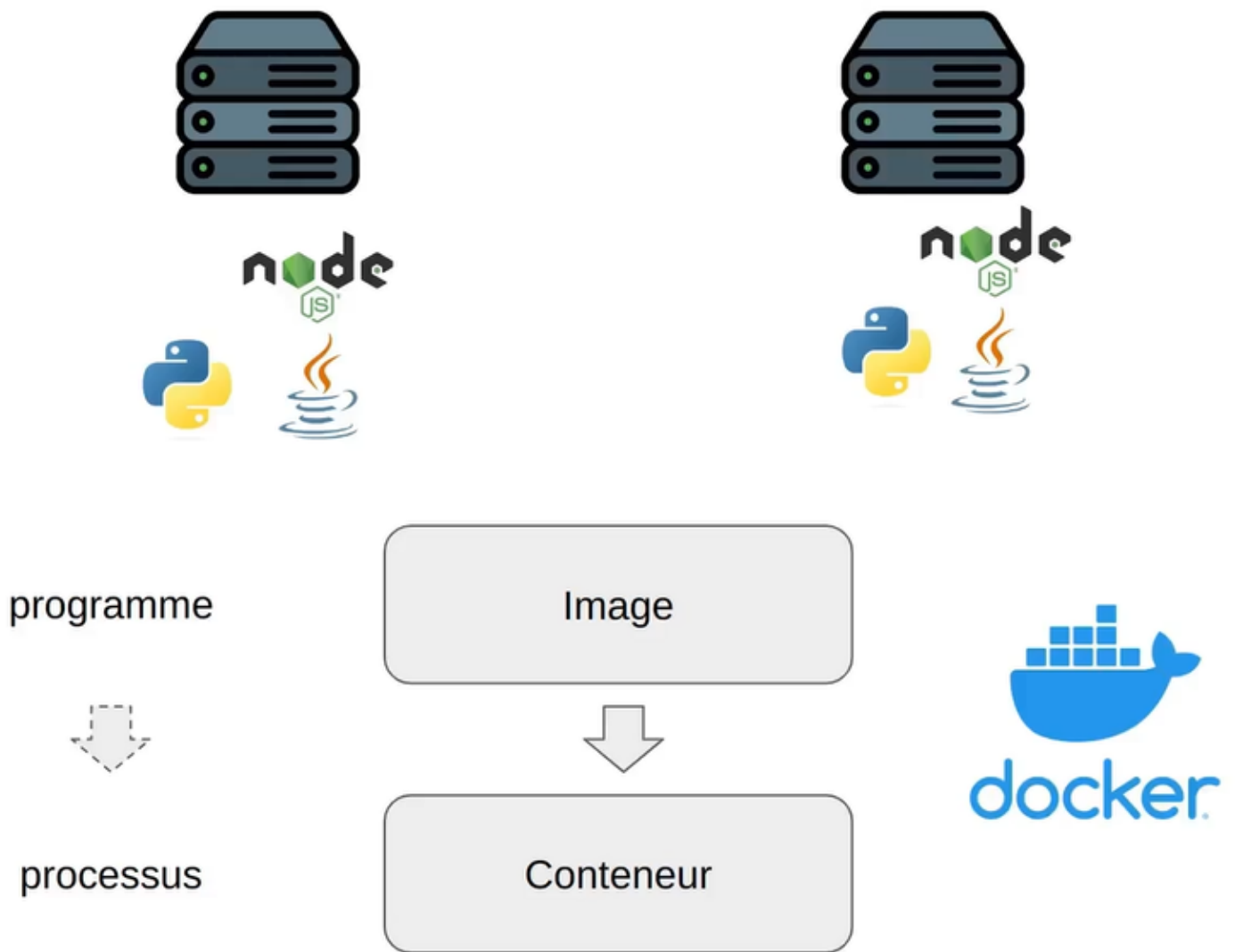


La prochaine problématique auquel on va être confronté est que comme nous avons plusieurs serveurs et que nous devons installer notre application dans chaque serveur en respectant les versions. Car une différence de versions de logiciels entre les serveurs pourrait conduire à des dysfonctionnements.



Nous allons donc utilisé des images et des container Docker.

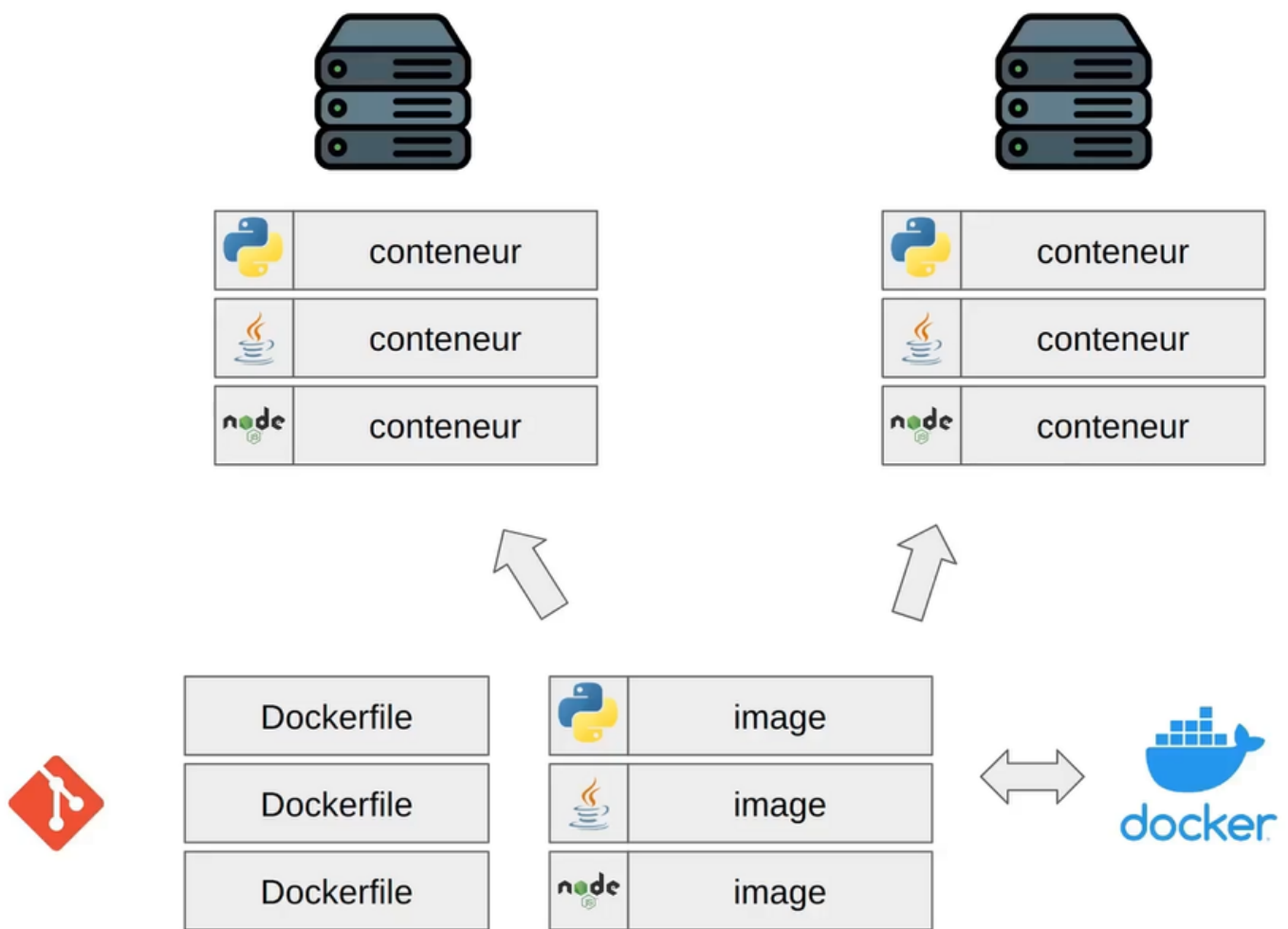




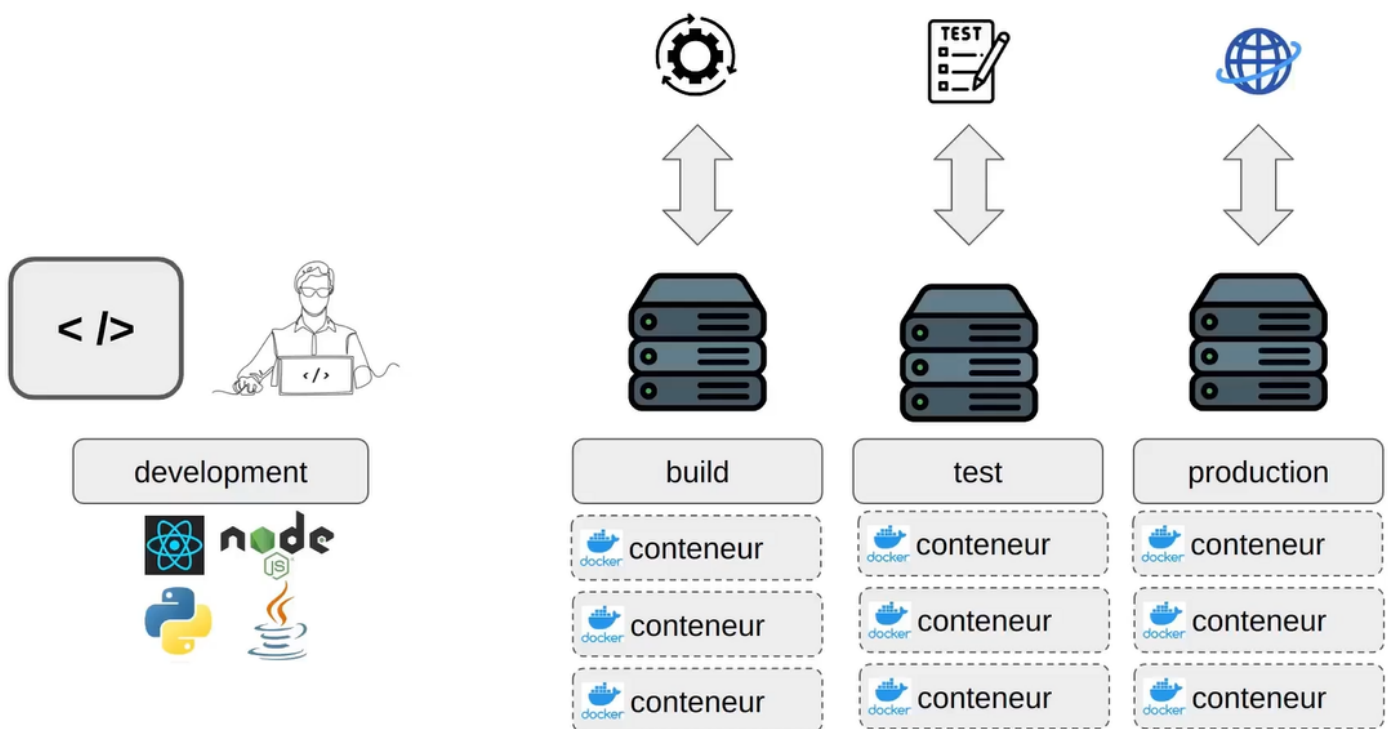
Le conteneur étant une exécution de ou des images.



Quelque soit la technologie on pourras utilisé un Dockerfile qui sera un fichier descriptif de docker qui nous permettra de définir nos images.



Ainsi chaque environnement pourra récupérer son Dockerfile et construire son application.
L'architecture améliorée de notre application devient :



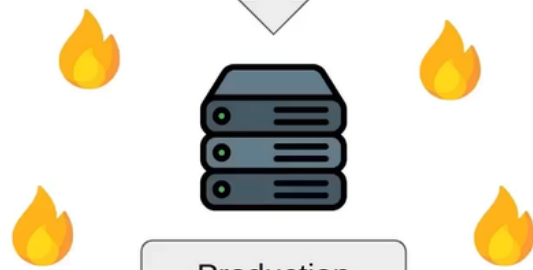
Certes l'architecture est améliorée mais qu'en ai il si on a énormément de charge en production ?



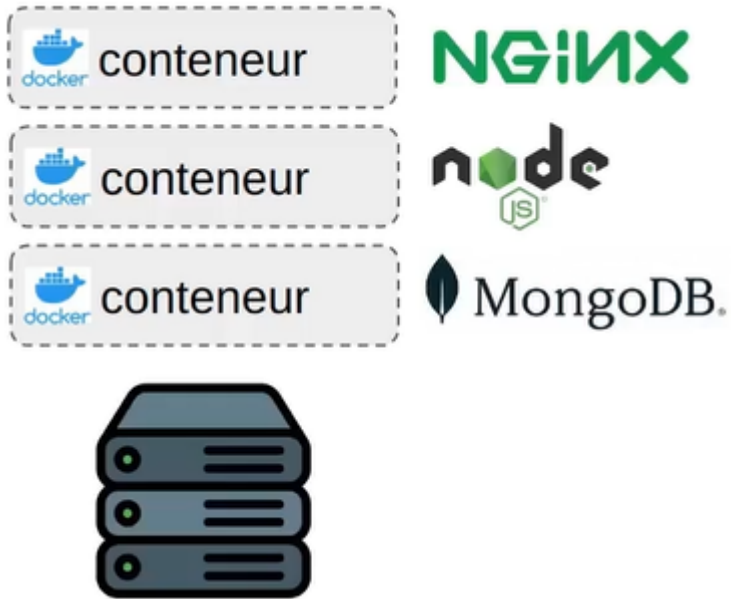
Production



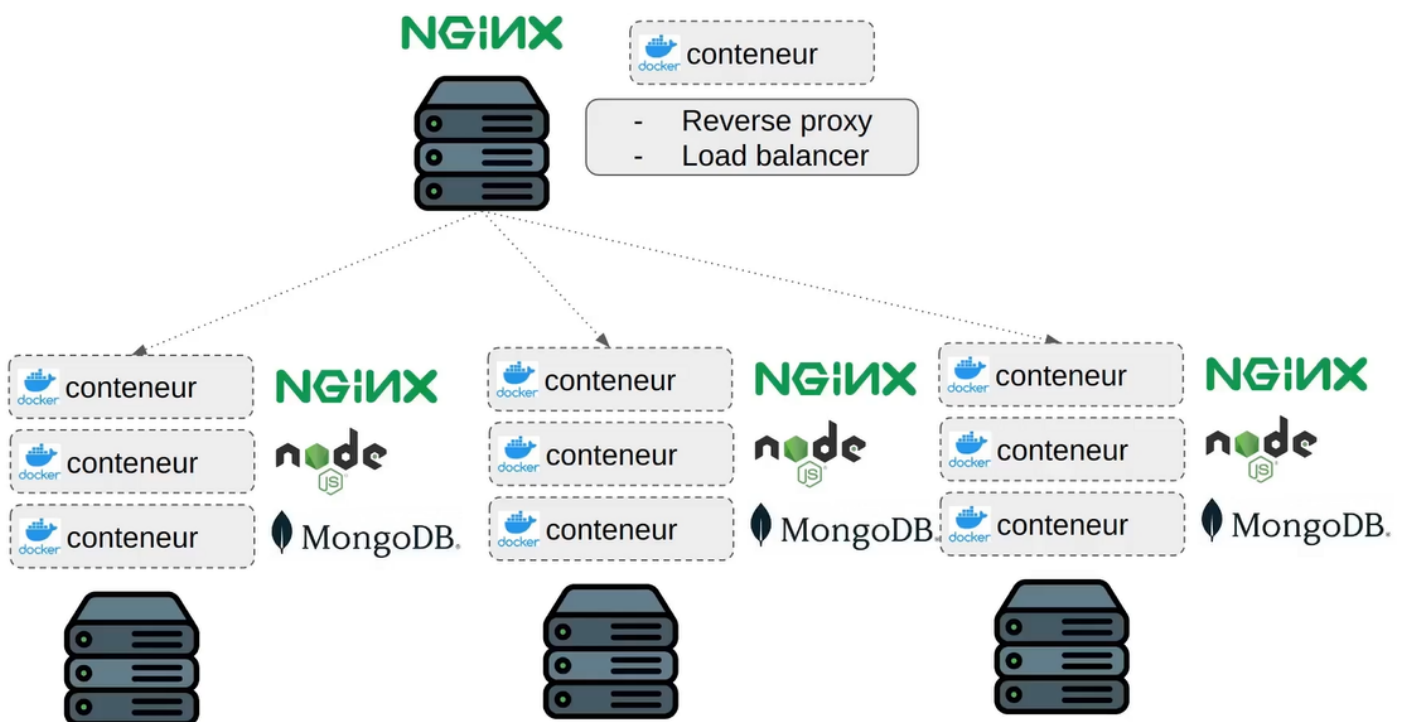
Production



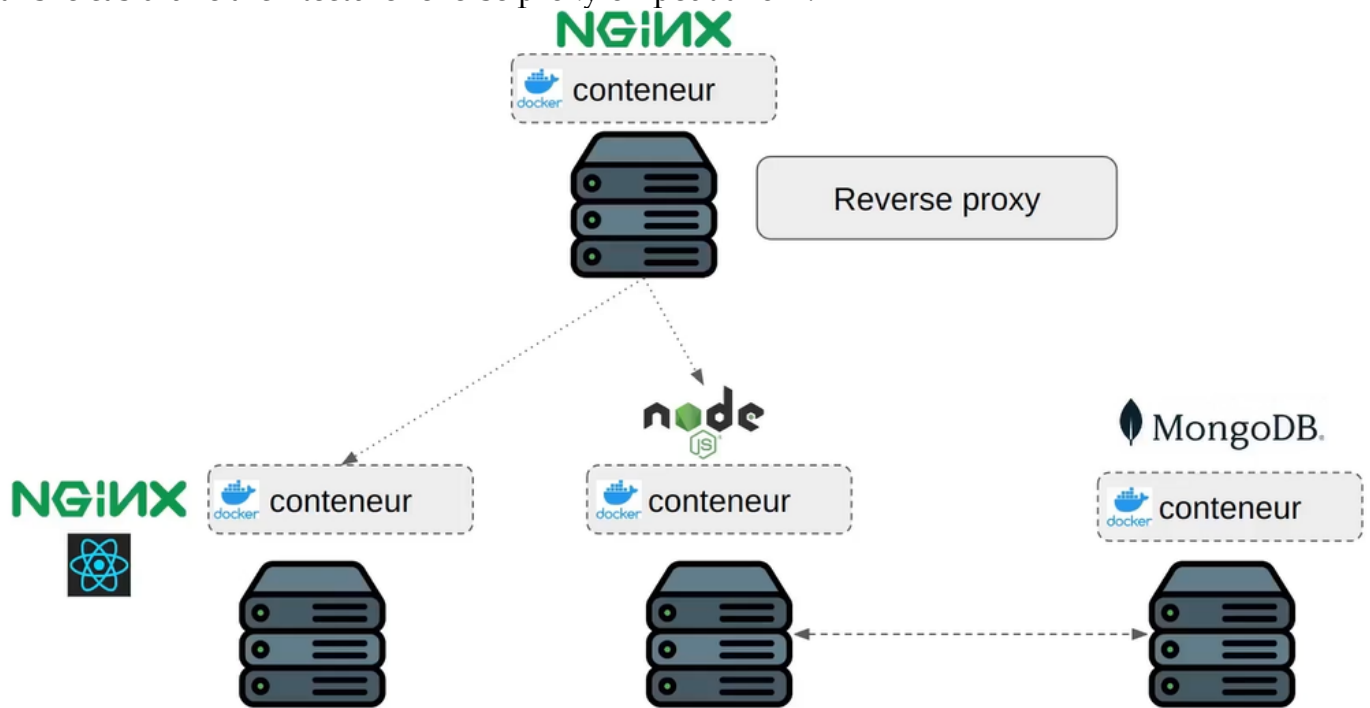
Pour rappel notre architecture de production ressemble à :



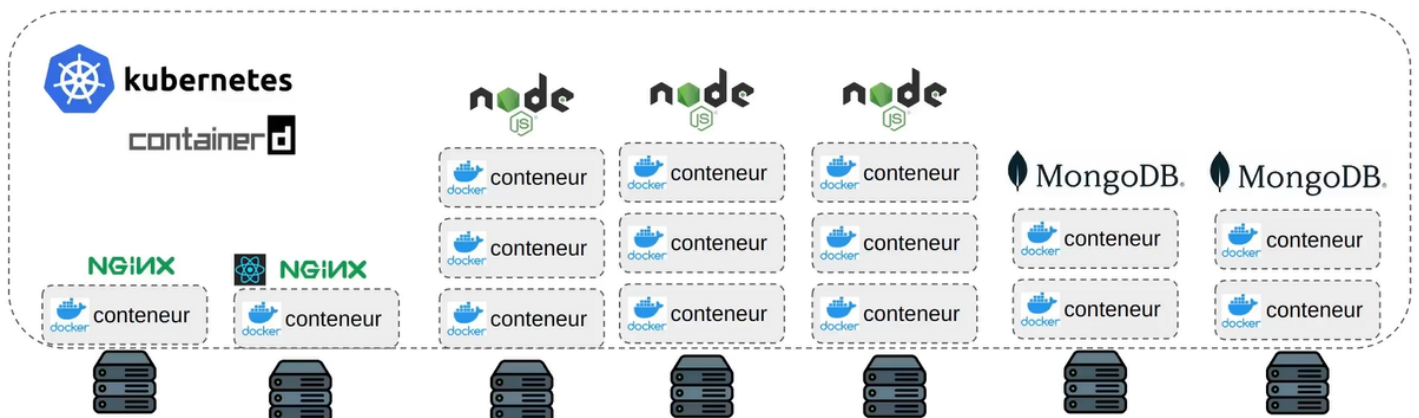
Nous allons dupliquer le serveur de production en utilisant une architecture **Load Balancer** pour ajouter deux autres instances.



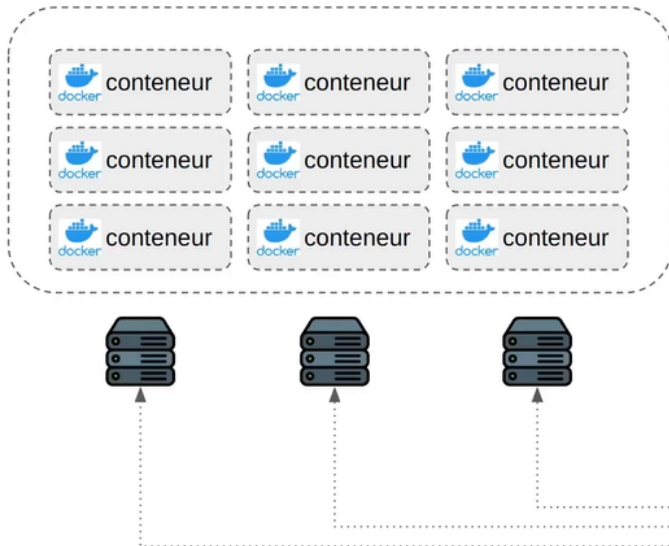
Dans le cas d'une architecture reverse proxy on peut avoir :



On commence à avoir beaucoup de conteneur qu'il va falloir gerer et il nous sera impossible de gerer tous ces conteneurs avec Docker. Nous allons gerer ces nombreux conteneurs avec Kubernetes.



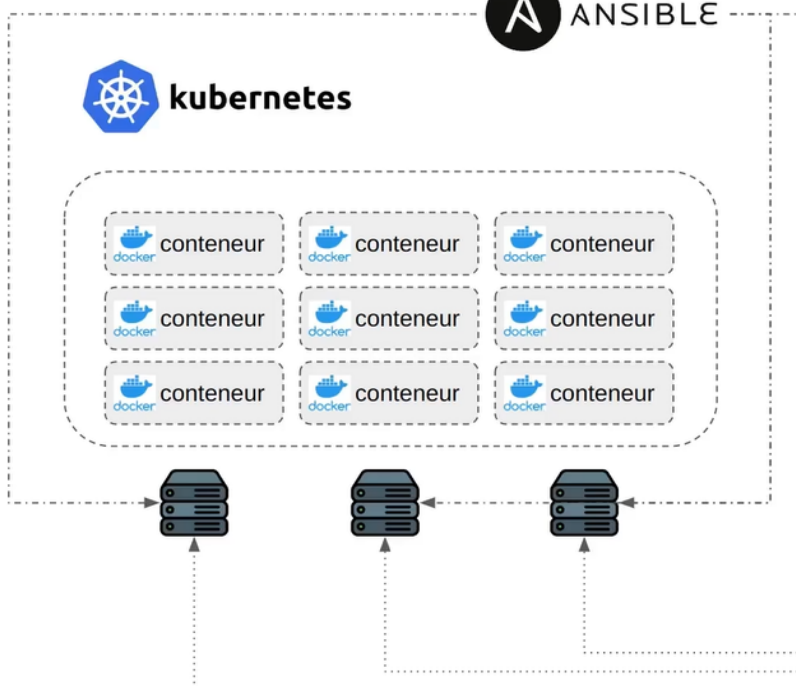
Infrastructure as Code (IaC)



Terraform



Pour faciliter la gestion de l'ensemble des serveurs nous allons utiliser Terraform. Cet outil va nous permettre de d'ouvrir et de fermer des serveurs facilement.
Nous allons décrire notre architecture serveur par l'intermédiaire de fichier manifest.
En plus d'ouvrir et de fermer des serveurs nous allons aussi avoir besoins de configurer ces serveur (Firewall, CRON ect...) pour cela nous allons utilisé ANSIBLE.

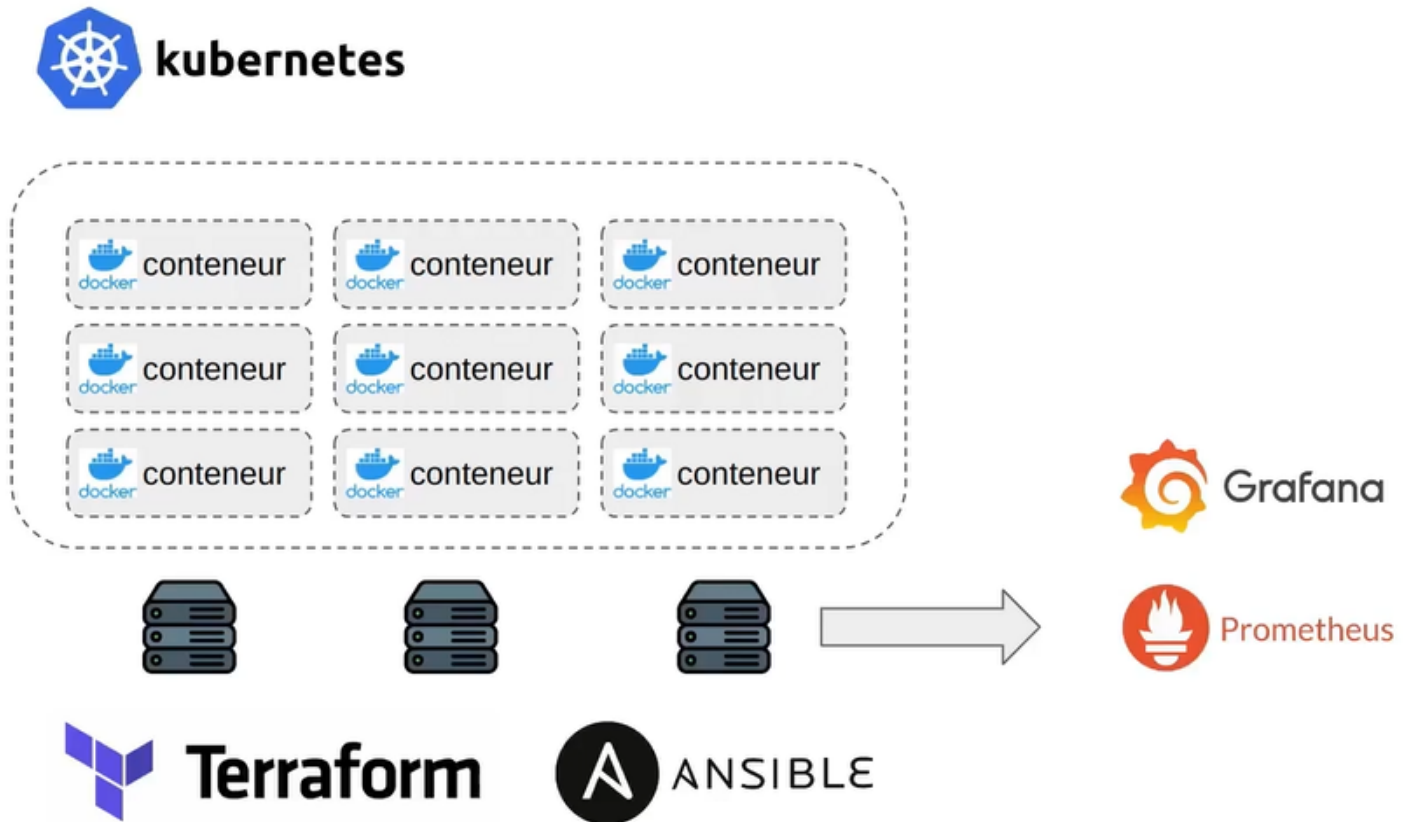


Terraform

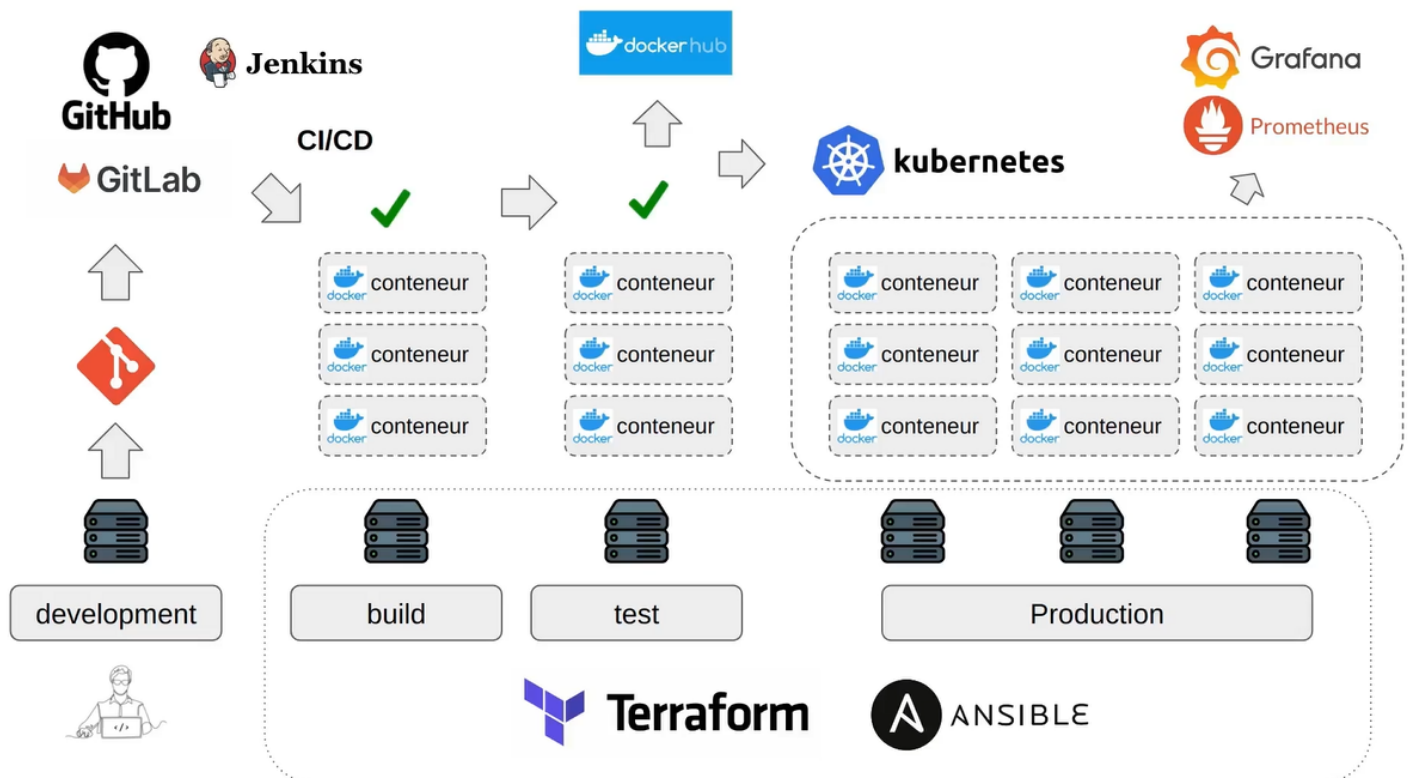
Infrastructure as Code (IaC)

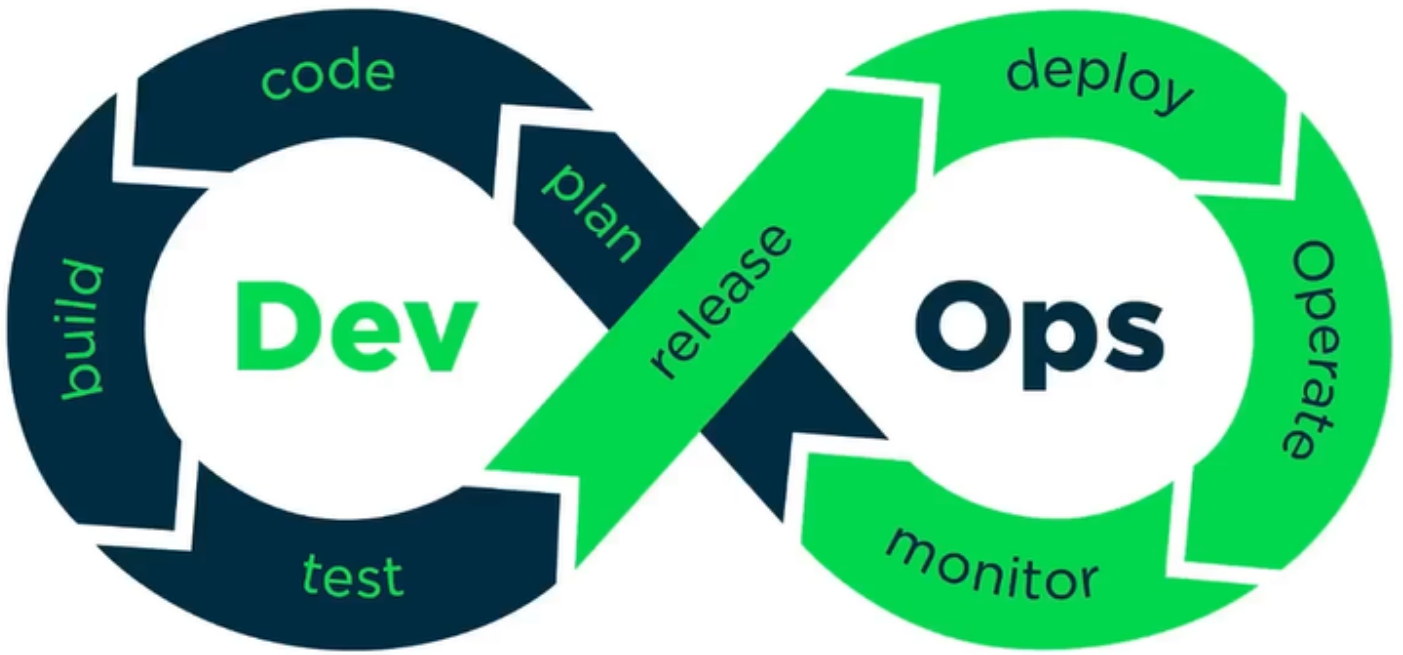


Nous aurons aussi besoin de gérer les logs avec des outils comme **Grafana** et **Prometheus**.



Au final on obtient l'architecture :





IV) Qu'est-ce que le CI/CD ?

CI (Continuous Integration - Intégration Continue)

L'Intégration Continue (CI), est une pratique de développement logiciel dans laquelle les développeurs intègrent régulièrement leur code dans un répertoire Git. Chaque fois qu'un développeur pousse du code, une série d'automatisations est déclenchée pour compiler, tester et vérifier le code nouvellement ajouté. L'objectif principal de la CI est de détecter les problèmes d'intégration tôt et de garantir que le code nouvellement ajouté ne casse pas l'application existante. Les étapes typiques incluent la compilation du code (build), l'exécution de tests unitaires et d'autres tests automatisés, la vérification des normes de code, la génération de rapports de test et la fourniture de commentaires rapide aux développeurs sur l'état de leur code. Dans un contexte de conteneurisation, comme avec Kubernetes, c'est également pendant l'intégration continue que sont build les images.

Les outils CI couramment utilisés incluent Jenkins, Travis CI, CircleCI, GitLab CI/CD, Github Actions, et bien d'autres.

CI



Continuous Integration



lint

build

Tests intégrations

CD (Continuous Deployment - Déploiement Continu)

Le CD, ou Déploiement Continu, est une extension de la CI qui vise à automatiser le déploiement des changements de code nouvellement validés vers les environnements de production ou de pré-production.

Contrairement à la CI qui se concentre principalement sur les builds, les tests et les vérifications, le CD s'intéresse à l'automatisation de la mise en production des changements.

Avec le CD, une fois que le code a passé avec succès les étapes de la CI et qu'il a été validé, il est automatiquement déployé dans un environnement cible, généralement à partir d'un pipeline automatisé. Le CD permet de minimiser le temps nécessaire pour mettre en production de nouvelles fonctionnalités ou des correctifs, tout en réduisant les risques liés aux déploiements manuels.

Il existe deux approches principales en matière de CD :

1. Continuous Deployment (Déploiement Continu) : dans cette approche, chaque changement de code validé est automatiquement déployé dans l'environnement de production sans intervention humaine. Cela nécessite une confiance élevée dans l'automatisation et une solide suite de tests automatisés.
2. Continuous Delivery (Livraison Continue) : ici, les changements de code validés sont automatiquement déployés dans un environnement de pré-production ou de staging. La décision de déployer dans l'environnement de production est prise manuellement, généralement par un responsable technique, après avoir évalué les risques.

CD



Continuous Delivery/Deployment



Déployer en production

Déployer en environnement
de pré-production

V) Le langage YAML

Le format YAML

YAML qui signifie "YAML Ain't Markup Language", est un format de données lisible par l'homme, largement utilisé pour la configuration des applications, la définition des déploiements d'infrastructure en tant que code, et d'autres tâches où les données sont à la fois lues par les machines et les humains.

Ce langage est extrêmement utilisé en DevOps : Docker, CI / CD, Ansible, Kubernetes etc utilisent tous le YAML.

Il est nécessaire de bien maîtriser le YAML avant de démarrer la formation.

La syntaxe YAML

La syntaxe YAML est conçue pour être simple et lisible par l'humain. Elle est moins verbeuse que le XML et plus descriptive que le CSV.

Elle est plus facile à lire que le JSON.

La syntaxe emprunte des éléments de langages de haut niveau, tels que l'utilisation du dièse (#) pour les commentaires et le tilde (~) pour représenter une valeur nulle.

Les concepts clés sont :

- Indentation : l'indentation (habituellement avec des espaces, non des tabulations) est utilisée pour indiquer les niveaux de hiérarchie des données.
- Listes : collections ordonnées d'éléments, et sont représentées en YAML en commençant chaque élément de la liste par un tiret (-).
- Dictionnaires : collection de paires de clés et de valeurs, et peut être représenté en YAML en utilisant des deux-points (:) pour séparer chaque clé de sa valeur.
- Les documents YAML peuvent être séparés dans un même fichier par trois traits d'union (--).

Commentaires

Les commentaires commencent par le signe dièse (#) et s'étendent jusqu'à la fin de la ligne.

Ceci est un commentaire

Valeurs Null

Vous pouvez représenter une valeur null avec le tilde (~).

clé: ~ # Ceci signifie que la valeur est nulle

Tableaux (listes)

Les tableaux sont définis avec des tirets (-) suivi d'une espace.

```
fruits:  
- Pomme  
- Orange  
- Banane
```

Dictionnaires (clé-valeur)

Les paires clé-valeur sont séparées par des deux-points (:), suivi d'une espace.

```
personne:  
  nom: Jean  
  âge: 30
```

Texte Multiligne

YAML permet d'écrire du texte sur plusieurs lignes en utilisant le pipe (|) pour conserver les retours à la ligne, ou le chevron (>) pour les convertir en espaces.

```
description: |  
  Ceci est un texte  
  qui conserve les retours à la ligne.
```

```
description: >  
  Ceci est un texte  
  qui remplace les retours à la ligne par des espaces.
```

Références et alias

Vous pouvez réutiliser des nœuds en les marquant avec une esperluette (&) et en y faisant référence avec un astérisque (*).

```
étudiants:  
- &jean  
  nom: Jean  
  âge: 20  
- nom: Marie  
  âge: 22  
- *jean # Ceci est une référence à l'étudiant "Jean"
```

Types de Données

YAML identifie automatiquement les types de données (comme les entiers, les flottants et les booléens) mais vous pouvez également les marquer explicitement.

```
un_entier: !!int 42
un_booléen: !!bool true
```

Indentation

L'indentation est utilisée pour représenter la structure imbriquée, et seuls les espaces sont autorisés pour l'indentation (les tabulations ne sont pas autorisées).

Documents multiples

Vous pouvez avoir plusieurs documents YAML dans un seul fichier, séparés par ---.

```
---
document1: valeur1
---
document2: valeur2
```