

NoSQL, systèmes distribués et passage en production de projets Data

Thierry GAMEIRO MARTINS

Séances

1. Introduction et prise en main d'Onyxia
2. Le stockage des données en NoSQL
3. Les systèmes de traitement distribués

4. Le passage en production

5. Orchestration et pratique DevOps
6. Déploiement conteneurisé sous Kubernetes

Du POC à l'industrialisation

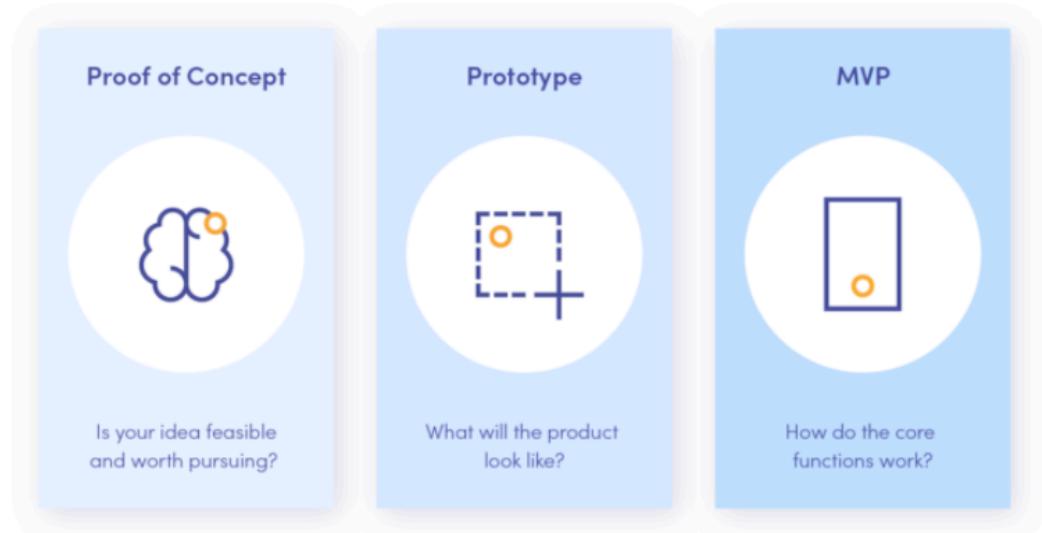
Proof of Concept

Qu'est ce c'est ?

- Prototype limité pour démontrer la faisabilité et la valeur ajoutée
- Évalué dans un environnement isolé et souvent simpliste
- Durée souvent courte

Pourquoi ?

- Réduire le risque et la compatibilité
- Permet de tester une nouvelle technologie / méthode
- Convaincre les parties prenantes



Le passage à l'industrialisation

La phase d'industrialisation dans le domaine de la data est souvent complexe et à fort risque d'échec :

- **Obstacles organisationnels** : gestion de projets, contraintes budgétaires, SSI, RGPD
- **Obstacles sur la valeur** : vulgarisation des résultats, valeurs ajoutées des algorithmes, acculturation « data »
- **Obstacles techniques** : orchestration, monitoring, sécurisation



invenis.co

<https://invenis.co/blog/pourquoi-85-des-projets-dat...> :

[Pourquoi 85% des projets data sont-ils voués à l'échec - Invenis](#)

29 oct. 2020 — Une étude Gartner estime que 85% des projets **data** sont un échec*. ... Mettre en place un projet **data**, développer des infrastructures Big **Data**, ...

Termes manquants : poc | Afficher les résultats avec : poc



Blog Business & Decision

<https://fr.blog.businessdecision.com/intelligence-artif...> :

[Intelligence artificielle : plus de projets, moins de PoC !](#)

13 déc. 2022 — ... taux d'échec de 85 %. Venturebeat va même encore plus loin en ... Data Scientist – Directeur **Data Science & IA** de Business & Decision ...



Ryax Technologies

<https://ryax.tech/realiser-bon-proof-of-concept-big-d...> :

[Réaliser un bon Proof of Concept en Big Data](#)

7 mai 2020 — Beaucoup d'observateurs narguent le taux d'échec des projets Big **Data**. On parle souvent de manière négative du faible pourcentage des ...

Les prérequis à la mise en production

Mise en place des environnements

Pourquoi des environnements multiples ?

- Séparation des phases de développement, de test et de production
- Réduire les risques liés à la propagation d'erreurs
- Faciliter le **debugging** et les tests



Ces environnements peuvent être nombreux et à la discréction du projet (pas tous obligatoires)

- **Développement**
 - Environnement dédié au développeur qui n'impacte pas les autres environnements.
 - Local (sur un poste) ou distant (via des outils collaboratifs comme Jupyter Lab, Dataiku, etc.)
 - L'accès aux données : données fictives, anonymisées ou scopées
- **Staging**
 - Permet aux développeurs de vérifier le fonctionnement d'une nouvelle brique
- **Qualification**
 - Faire tester aux utilisateurs les nouvelles fonctionnalités

- **Intégration**

- Vérifie que les déploiements sont compatibles avec l'infrastructure cible

- **Formation**

- Plateforme pour former les utilisateurs afin qu'il n'impacte pas la production

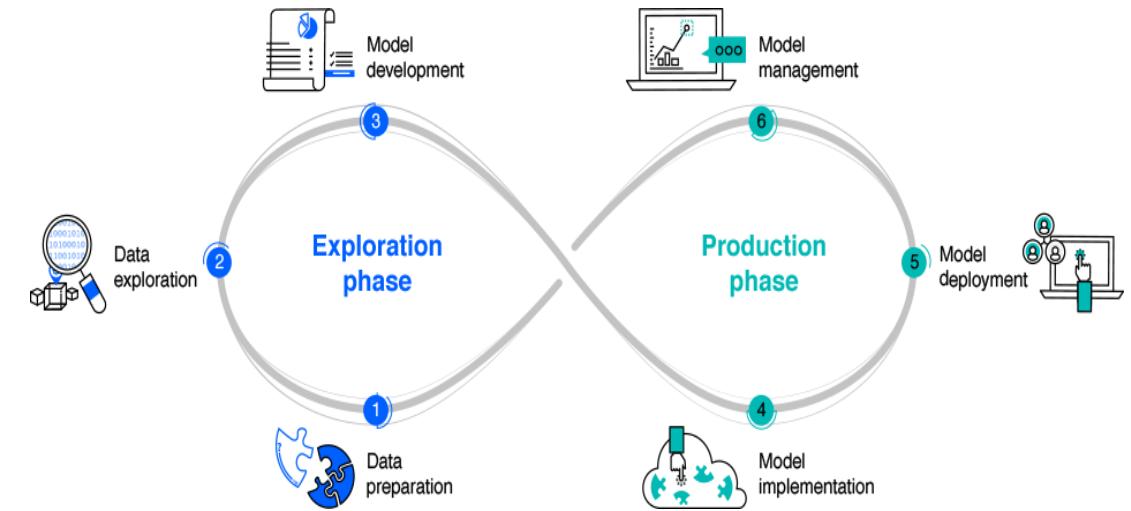
- **Pré-production**

- Environnement iso-production
 - Permet de reproduire les bugs
 - Peut être utilisé comme production de secours

Production

- Environnement où fait vivre l'application pour ses utilisateurs
- Il doit être opérationnel en permanence (disponible selon les SLA et sans bug)

SLA : *Service Level Agreement*



L'observabilité

1. Logs

- Collecte des événements système et applicatifs

2. Métriques

- Analyse des performances : latence, utilisation CPU, mémoire

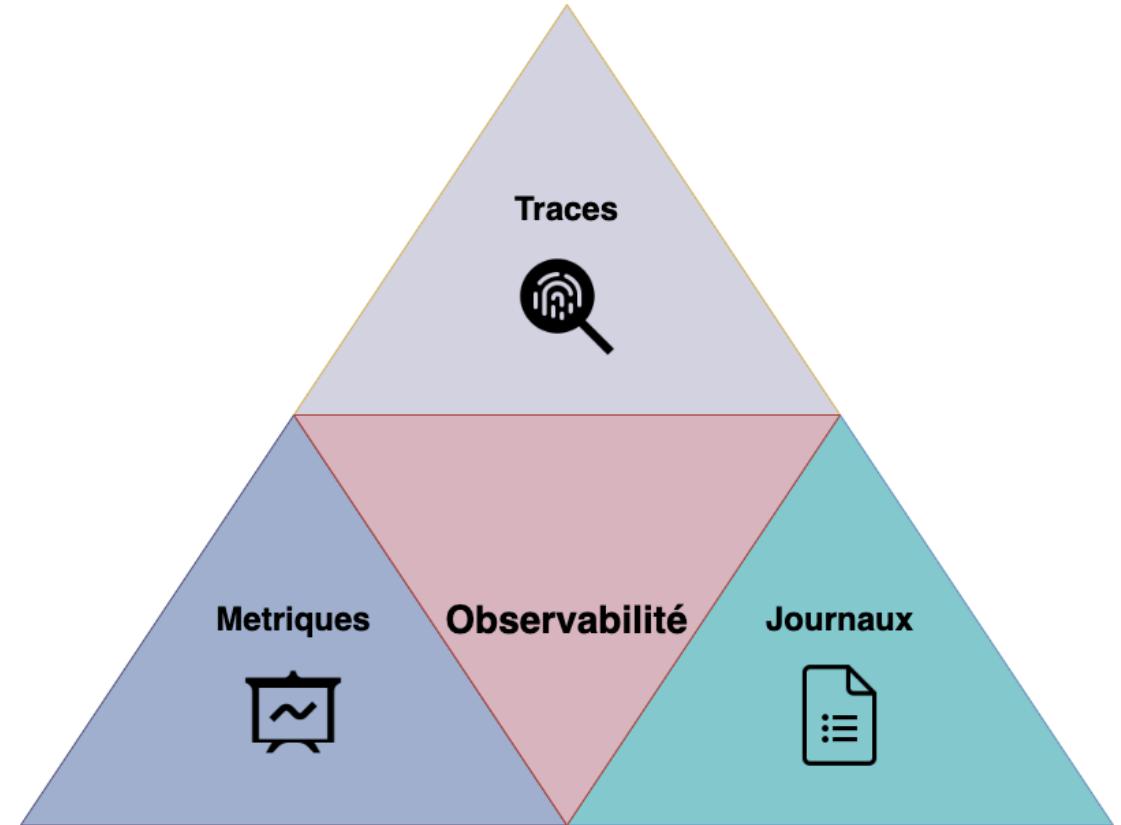
3. Traces

- Suivi des flux entre services

4. Alerting

- Envoi des états

Prometheus, Grafana, ELK, Loki



Le logging ou journalisation est l'enregistrement des événements, des opérations effectuées

Les types de logs

- Détection des erreurs et des pannes
- Suivi de l'activité
- Conformité aux réglementations

Les points d'attention

- Niveaux de journalisation (ERROR, INFO, WARN, DEBUG, etc.)
- Rotation des journaux (archivage, nettoyage selon une durée, etc.)
- Stockage sécurisé (chiffrement, conformité des données, sensibilité)



Le monitoring consiste à surveiller, mesurer le système et à suivre son évolution

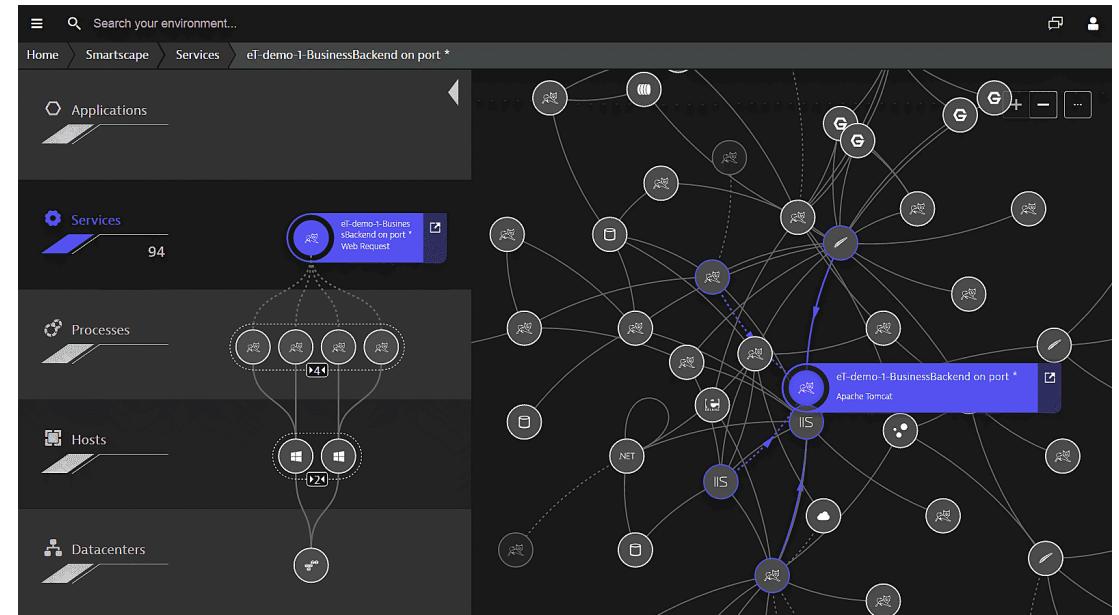
- Les performances (nombre de connexions actives, de requêtes, consommation de ressources, etc.)
- Sa disponibilité (fonctionnement du système, downtime, etc.)
- Son intégrité (résultats corrects)



Dashboard Grafana

Les traces permettent de suivre le parcours des interactions à travers un système

- Observer les interactions entre les différentes services
- Analyser les éventuels goulets d'étranglements
- Partie la plus avancée dans l'observabilité



Dépendances sur Dynatrace

L'alerting est le principe de créer des alerte en cas d'évènements particuliers dans un système (défaillances, bon déroulement d'un job)

- Solution à développer pour le cas d'usage (envoi sur des canaux de discussions, mails, etc.)
- Certains outils d'orchestrations embarquent des outils d'alerting
- Les solutions d'observabilités (Elastic, Splunk comportent des plugins d'alerting)
- Technologie qui envoit des alertes

La sécurisation

Plusieurs niveaux à sécuriser :

- L'infrastructure
- Les accès
- La donnée

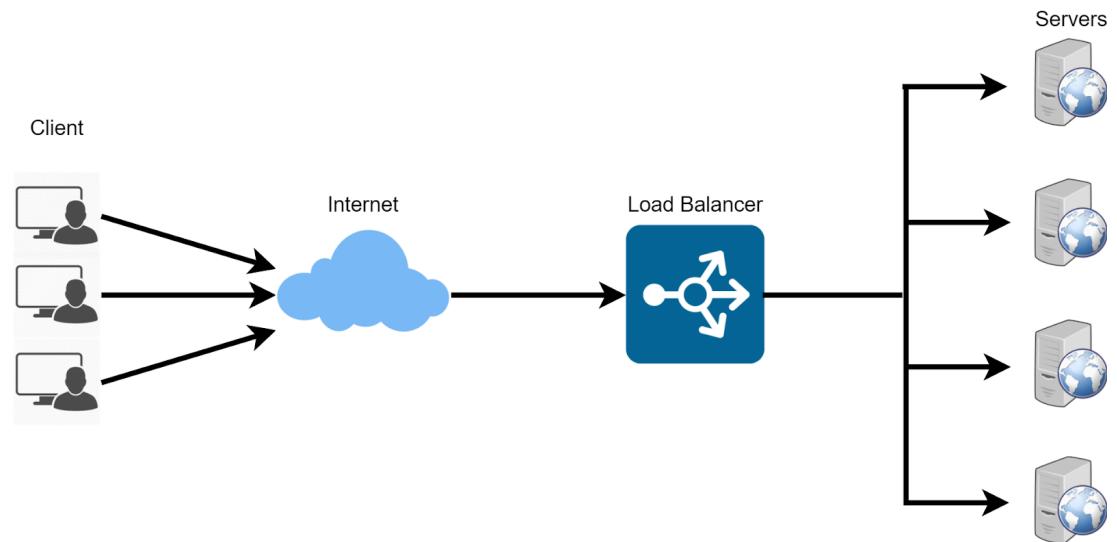
Pourquoi ?

- Se protéger contre les cyberattaques
- Maintenir la continuité
- Garantir la confidentialité des données
- Préserver sa réputation



L'objectif est de sécuriser les points d'entrées et de sorties du système

- **VPN** : tunnel sécurisé entre l'utilisateur et le réseau (confidentialité, connexion à distance)
- **LoadBalancer** : répartit la charges sur plusieurs serveurs (fiabilité)
- **Proxy** : filtre le trafic entrant ou sortant d'une application
- **Firewall** : permet de cloisonner le réseau
- **API Gateway** : gère le trafic, la limitation du débit, l'authentification



La gestion des accès est une composante importante. On distingue :

- **Authentification** : déterminer l'identité d'un utilisateur ou d'un service
- **Autorisation** : identifier les droits d'accès

Quelques bonnes pratique

- Le moindre privilège
- Utiliser des comptes de services pour des tâches d'automatisation
- Privilégier l'identification unique (SSO) par rapport aux comptes locaux
- Appliquer de la MFA



Authentication

Confirms users are who they say they are.



Authorization

Gives users permission to access a resource.

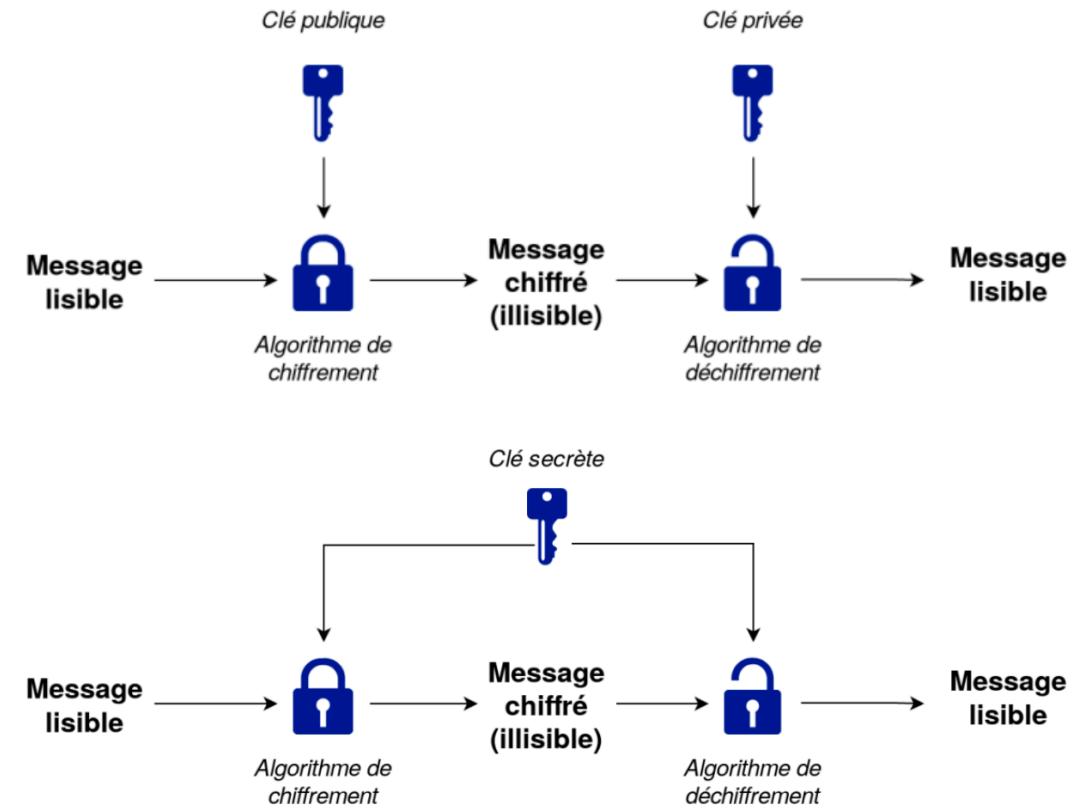
Chiffrer la donnée dans tous ses états

Au repos

- chiffrer les bases de données
- chiffrer les fichiers plats (csv, parquet, etc.)

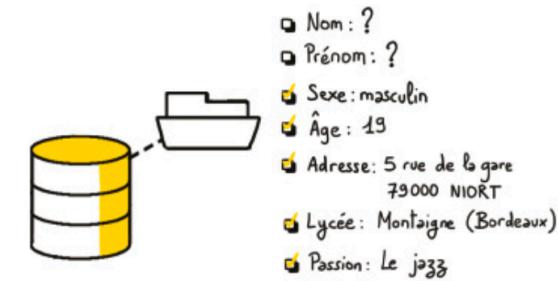
En transit

- En HTTPS pour le transit par le web
- Chiffrer les connexions aux base de données avec du SSL



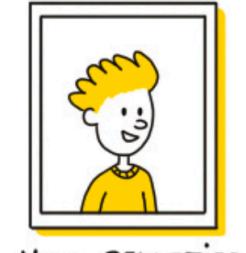
La réglementation

- RGPD : protection des données personnelles, droit d'accès et suppression pour les utilisateurs
- PCI-DSS : sécurisation des transactions bancaires



Implications techniques

- Collecter uniquement les données nécessaires
- Stocker les données dans des régions autorisées
- Mettre en place des mécanismes d'anonymisation ou de pseudonymisation



Marc PELLETIER



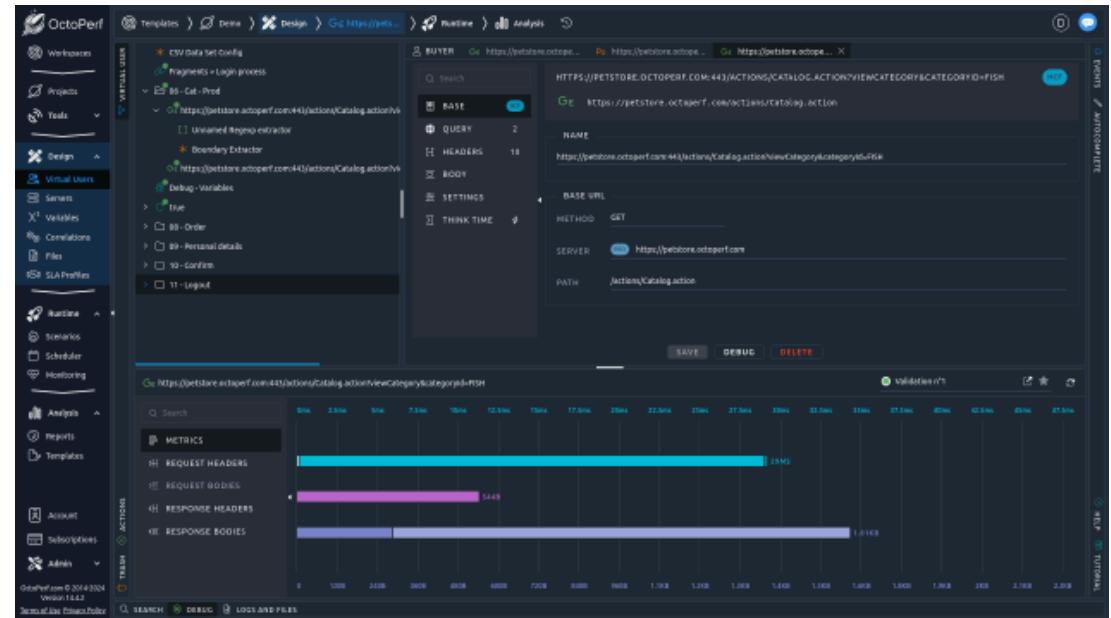
Je suis une base
de données personnelles

Tests de performances

- Identifier les goulots d'étranglement
- Garantir la scalabilité pour les volumes de données réels
- Évaluer la résilience face à des pics d'activité

Types de tests

- 1. Tests de charge :** mesurer les performances sous charge normale
- 2. Stress tests :** simuler des conditions extrêmes
- 3. Tests d'endurance :** évaluer la stabilité sur de longues durées



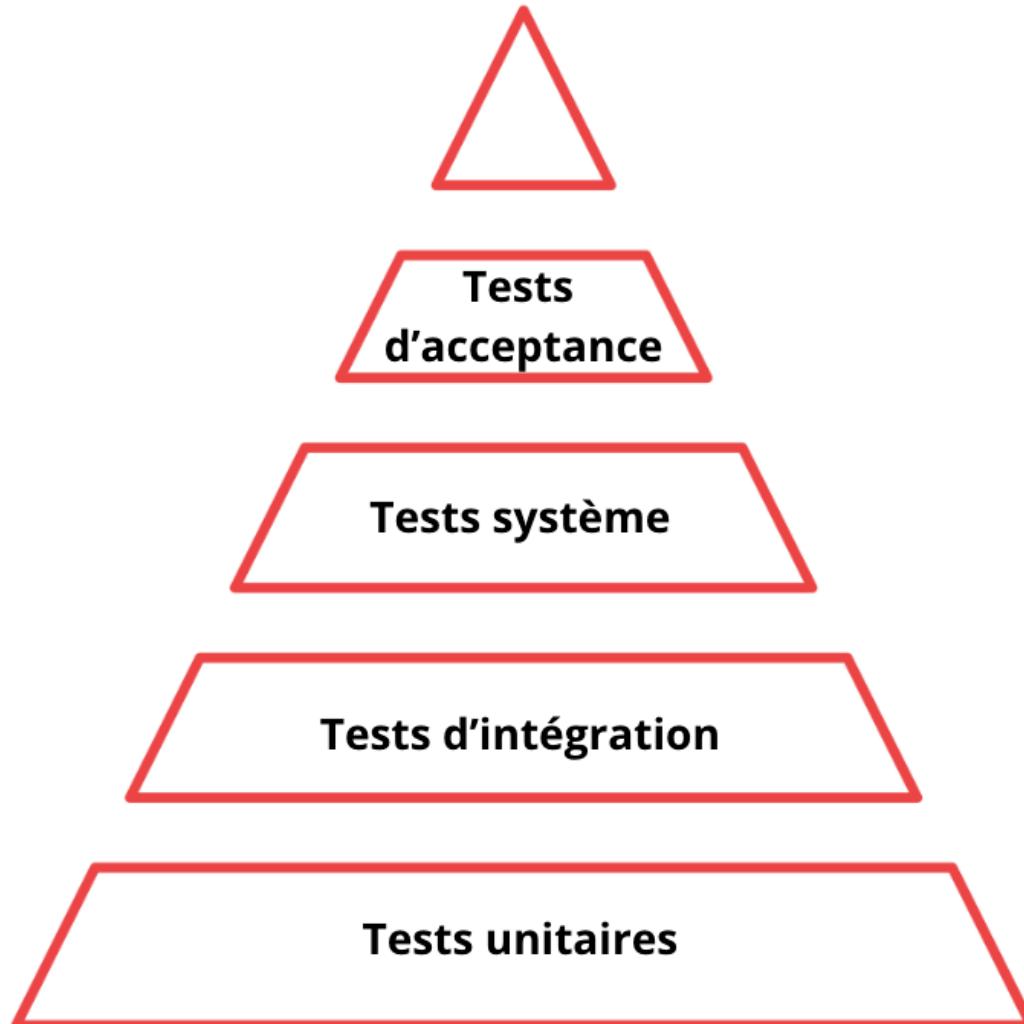
Tests fonctionnels et techniques

Tests fonctionnels

- Vérifier que chaque fonctionnalité respecte les besoins métier
- Appliquer des scénarios types

Tests techniques

- Vérification du contenu des réponses pour une API
- Vérification de la non-régression



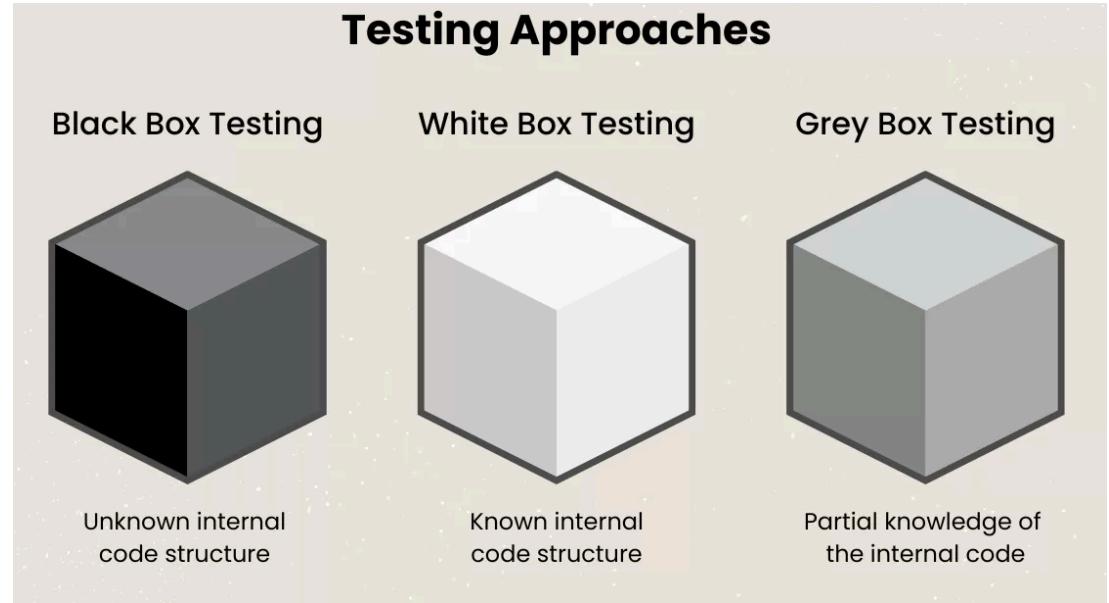
Audit de sécurité

Pourquoi auditer ?

- Identifier les vulnérabilités avant une mise en production
- Renforcer la confiance des parties prenantes

Comment ?

- Simuler des attaques pour évaluer les défenses (*pentest*)
- Analyse du code pour détecter les failles connues
- Audit des configurations système



La documentation

Objectifs de la documentation

- Faciliter la maintenance et l'évolutivité
- Permettre une transmission aux nouvelles équipes

Types de documentation

1. Technique

- Architecture du système
- Configuration des services

2. Fonctionnelle :

- Modes d'utilisation des outils
- Cas d'usage métier



Sphinx documentation

Orchestration et pratique DevOps

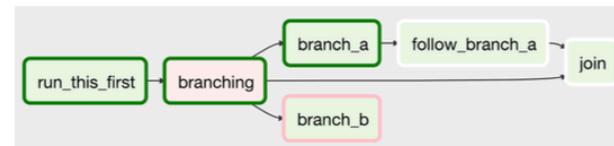
Présentation de Airflow



Airflow est un *framework open-source* pour l'ordonnancement de *workflows*

- **Dynamique** : Les pipelines sont configurés sous forme de code Python
- **Extensible** : contient des opérateurs permettant d'utiliser de nombreuses technologies
- **Flexible** : Les workflows (DAGs) peuvent être hautement paramétrables

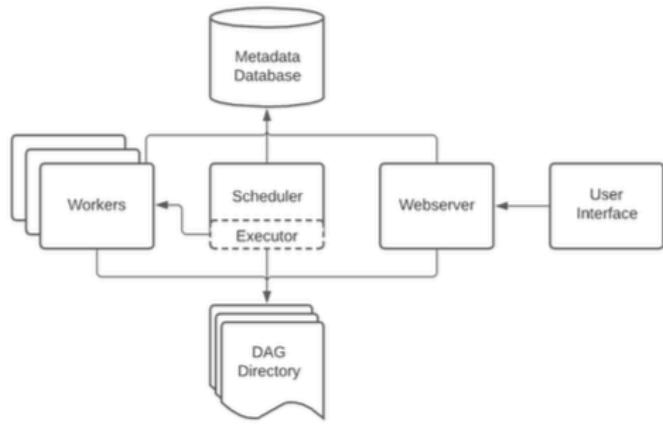
Un DAG (Directed Acyclic Graph) rassemble des tâches, organisées avec des dépendances



La page principale affiche la liste des DAGs

A screenshot of the Airflow web interface. At the top, there's a navigation bar with links for "DAGs", "Security", "Browse", "Admin", and "Docs". The main area is titled "DAGs" and shows a table of active DAGs. The columns include "Owner", "Runs", "Schedule", "Last Run", "Recent Tasks", "Actions", and "Links". There are 14 rows listed, each representing a different DAG with its name, owner (airflow), and last run details. For example, "example_bash_operator" was last run on 2020-10-26 at 21:08:11. The table has a light gray background with alternating row colors.

Les DAGs sont lancés par le scheduler



Le DAG est lu en continu, mais les tâches à l'intérieur ne sont effectivement jouées qu'à l'exécution

```
from datetime import datetime

from airflow import DAG
from airflow.decorators import task
from airflow.operators.bash import BashOperator

# A DAG represents a workflow, a collection of tasks
with DAG(dag_id="demo", start_date=datetime(2022, 1, 1), schedule="0 0 * * *") as dag:

    # Tasks are represented as operators
    hello = BashOperator(task_id="hello", bash_command="echo hello")

    @task()
    def airflow():
        print("airflow")

    # Set dependencies between tasks
    hello >> airflow()
```

Une tâche (*Task*) est une action effectuée dans un DAG. Elle peut être :

Une fonction python

Une fonction python (avec le décorateur *task*) qui est lancée pour effectuer une action.

Cette dernière peut utiliser le contexte Airflow (informations de connexion, variables, du DAG en cours, etc.)

Un operator

Un opérateur est un modèle pour une tâche prédéfinie

- SimpleHttpOperator
- MySqlOperator
- PostgresOperator
- MsSqlOperator
- OracleOperator
- JdbcOperator
- DockerOperator
- HiveOperator
- S3FileTransformOperator
- PrestoToMySqlOperator
- SlackAPIOperator

Un sensor

Un type d'opérateurs qui attendent que quelque chose se produise (un fichier, un évènement)

Executors

Il s'agit de la manière dont les *tasks* sont lancés par Airflow :

- Local Executor
- Sequential Executor
- Celery Executor
- Kubernetes Executor

XComs

Permet de transférer des données entre les tâches grâce à un *backend*. Soit :

- Metadata database (par défaut)
- Custom backend

Utilise un mécanisme de *push* ou de *pull*.

Attention : ne convient pas pour de grands volumes de données pour le backend par défaut

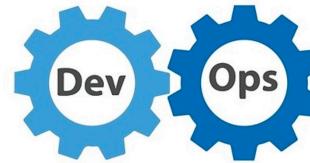
Best practices

- Idempotence des tasks
- Atomicité de chaque task
- *templated fields* et variables dans les tasks
- Attention aux communications d'informations entre les tasks
- Paramétriser les DAGs (retries, timeouts, etc.)

Les pratiques DevOps

Il faut embaucher des « Ops » qui pensent comme des Devs » et des « Devs qui pensent comme des Ops »

John Allspaw & Paul Hammond



Les principes

- Des équipes intégrées
- Un outillage automatisé (test unitaire, test d'intégration, déploiement continu, etc.)
- Extension de l'agilité (éviter le cycle en V)

Les bonnes pratiques

- Infra as Code (IaaC)
- Pas d'action manuelle
- Suivi de version
- Architecture microservice

Infrastructure as Code

- Création de fichier contenant les caractéristiques souhaités
- Pas besoin de gérer manuellement les serveurs

Ansible, Terraform

Module Terraform



```
provider "aws" {  
    region = "us-east-1"  
    assume_role {  
        role_arn = "arn:aws:iam::123456789012:role/admin-role"  
    }  
}  
  
module "vantage-integration" {  
    source  = "vantage-sh/vantage-integration/aws"  
}
```

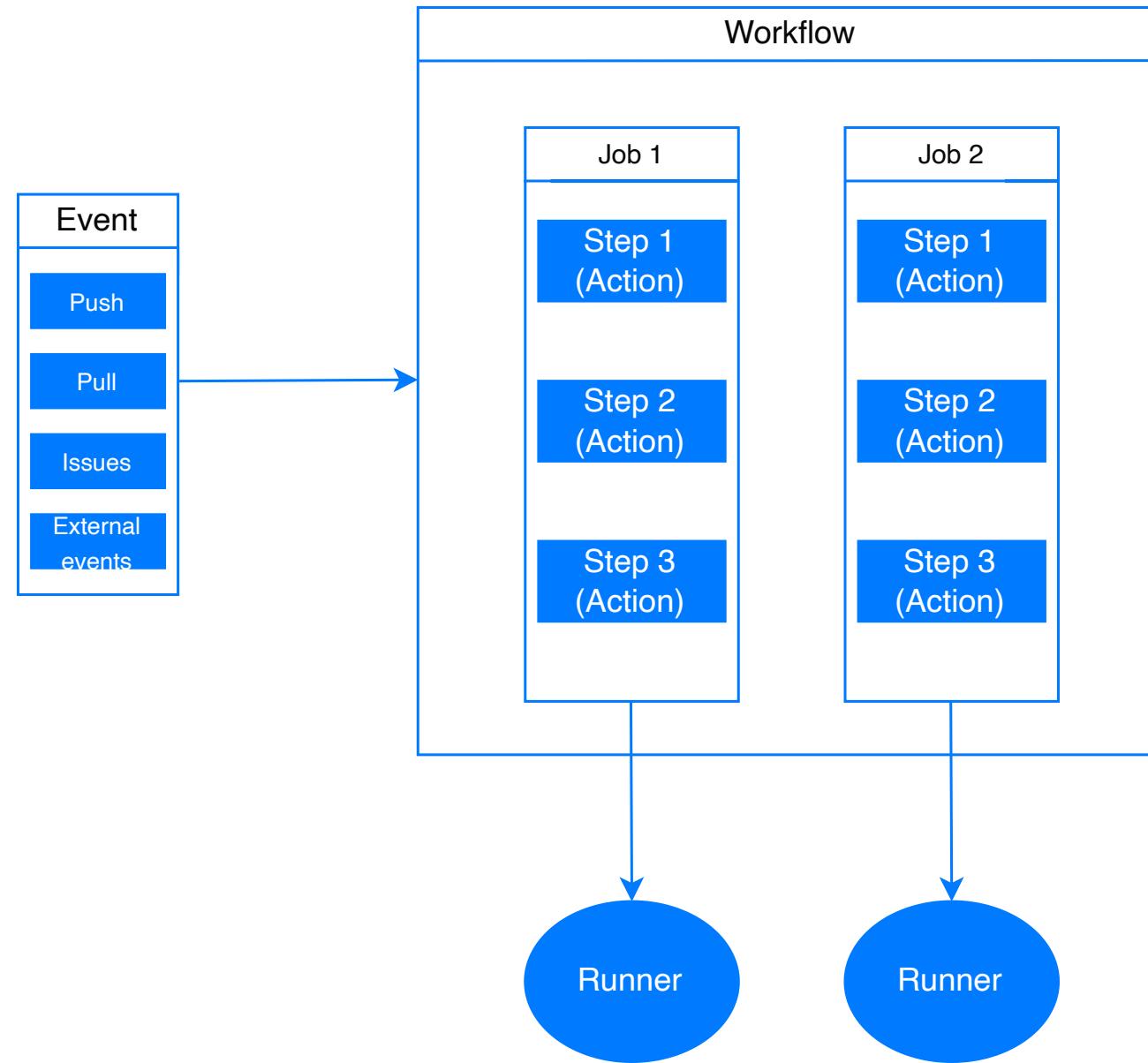
Continuous intégration (CI) et Continuous delivery (CD)

- CI : Chaque commit déclenche un processus test, build and release
 - Github Action
 - GitLab Workflow
- CD : les nouvelles releases sont automatiquement déployées
 - ArgoCD

Github Action

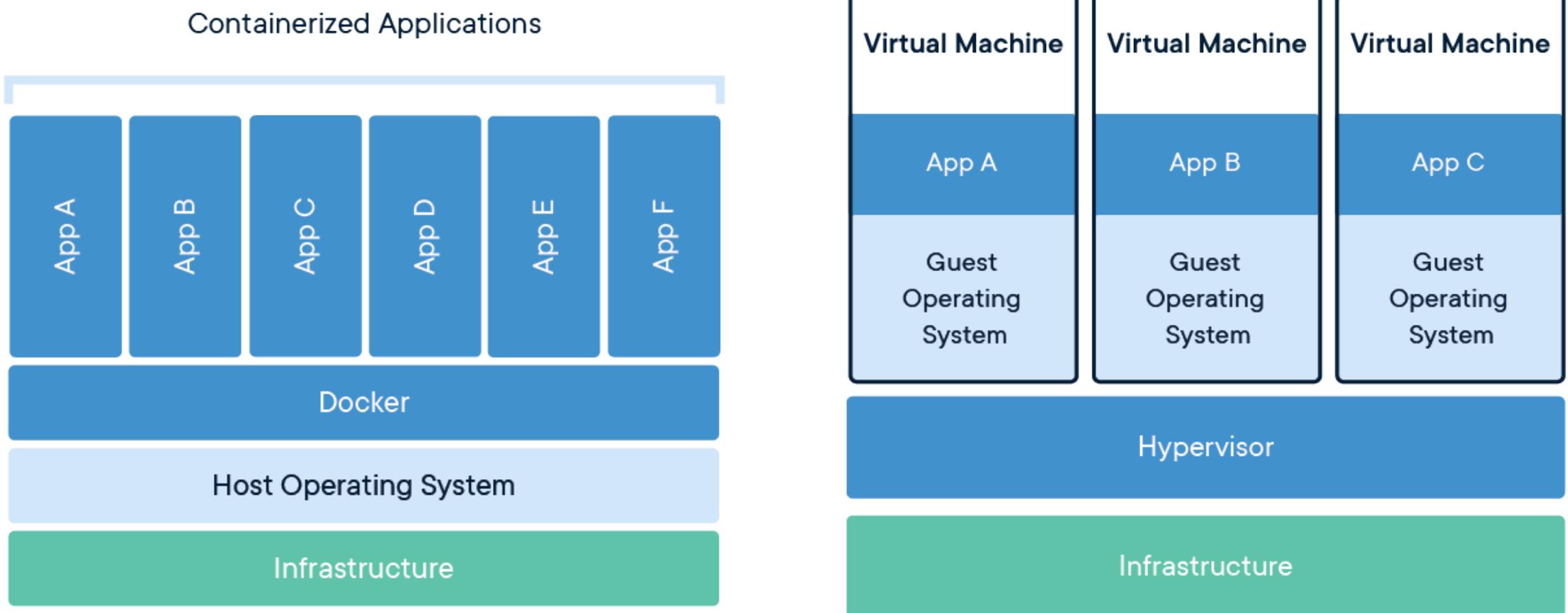
```
jobs:  
  build:  
    runs-on: ubuntu-latest  
    steps:  
      -  
        name: Login to Docker Hub  
        uses: docker/login-action@v3  
        with:  
          username: ${{ vars.DOCKERHUB_USERNAME }}  
          password: ${{ secrets.DOCKERHUB_TOKEN }}  
      -  
        name: Set up Docker Buildx  
        uses: docker/setup-buildx-action@v3  
      -  
        name: Build and push  
        uses: docker/build-push-action@v6  
        with:  
          push: true  
          tags: ${{ vars.DOCKERHUB_USERNAME }}/clockbox:latest
```





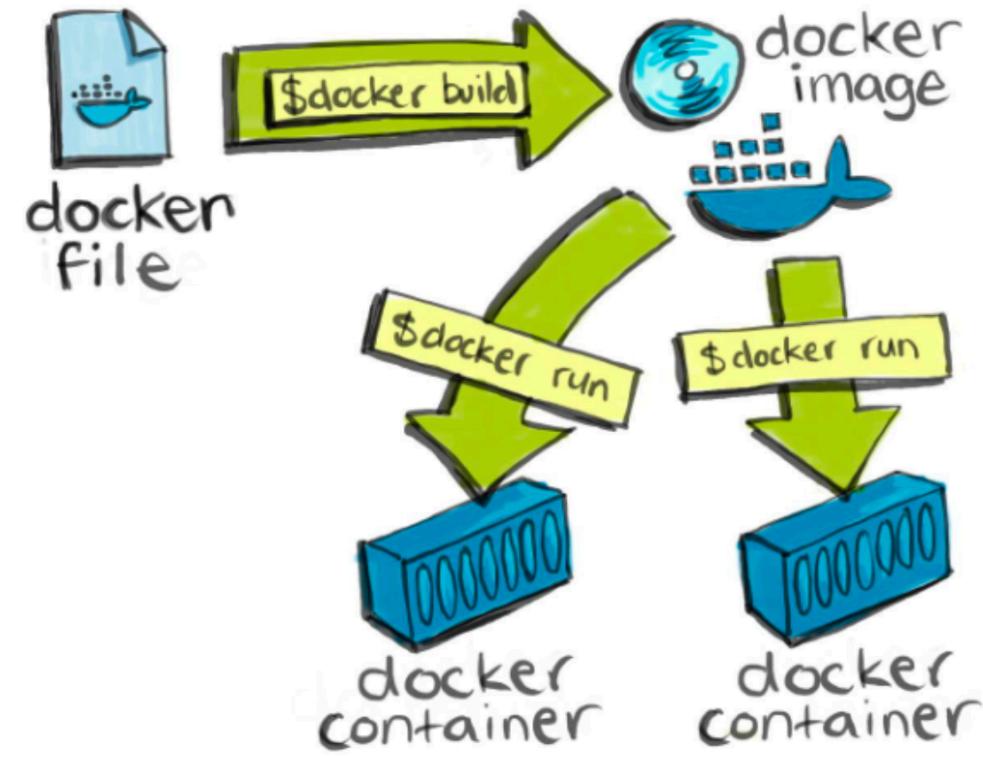
Déploiement conteneurisé sous Kubernetes

Conteneurs vs Machine virtuelles



Présentation de Docker

- **Portable** : il peut-être détruit et reconstruit très rapidement
- **Isolé** : tout comme une machine virtuelle, il permet de créer un environnement dédié, permettant d'éviter les aléas (éviter conflits de versions, seulement le strict nécessaire)
- **Léger** : il ne demande pas autant de ressource qu'une machine virtuelle
- **Standard** : quelque soit l'application, un conteneur se lance de la même manière



Ecrire un Dockerfile

- FROM <image> : image de base
- WORKDIR <path> : dossier courant
- COPY <host-path> <image-path> : copier un fichier dans le conteneur
- RUN <command> : lance une commande
- ENV <name> <value> : ajoute une variable d'environnement
- EXPOSE <port-number> : expose un port
- USER <user-or-uid> : change d'utilisateur
- CMD [<command>, <arg1>] commande par défaut au démarrage

```
FROM python:3.12
WORKDIR /usr/local/app

# Install the application dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy in the source code
COPY src ./src
EXPOSE 5000

# Setup an app user so the container doesn't run as the root user
RUN useradd app
USER app

CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8080"]
```

Construire une image

Pour construire une image avec un tag dans le dossier courant (`.`)

Docker lit par défaut le fichier

`Dockerfile`

```
docker build -t my-username/my-image .
```

Lister les images

```
docker image ls
```

Renommer une image

```
docker image tag my-username/my-image  
another-username/another-image:v1
```

Publier une image

```
docker push my-username/my-image
```

Stockage des images

The screenshot shows the Docker Hub interface for managing repositories.

Create repository

- Namespace:** ajeetraina777
- Repository Name:** docker-quickstart
- Short description:** A short description to identify your repository. If the repository is public, this description is used to index your content on Docker Hub and in search engines, and is visible to users in search results.
- Visibility:**
 - Public** (radio button selected)
 - Private** (radio button unselected)

Appears in Docker Hub search results
- Pushing images:** You can push a new image to this repository using the CLI:
docker tag local-image:tagname new-repo:tagname
docker push new-repo:tagname
- Make sure to replace `tagname` with your desired image repository tag.**

Cancel **Create**

Tags

Sort by: Newest

Filter Tags:

TAG	Digest	OS/ARCH	Last pull	Compressed Size
1.0	ec38dc5f898b	linux/arm64/v8	---	40.34 MB

Actions: docker pull ajeetraina777/docker-quickstart:1.0 **Copy**

Gérer son conteneur

Publier un port du conteneur vers le système

```
docker run -d -p 8080:80 nginx
```

Ajouter une variable d'environnement au lancement

```
docker run -e foo=bar postgres
```

Lancer un conteneur détaché en stockant de manière persistente les données

```
docker volume create log-data  
docker run -d -v log-data:/logs docker/welcome-to-docker
```

Suivre ses conteneurs

```
docker ps
```

Supprimer un conteneur

```
docker rm my-conteneur
```

Lancer un terminal bash dans un conteneur

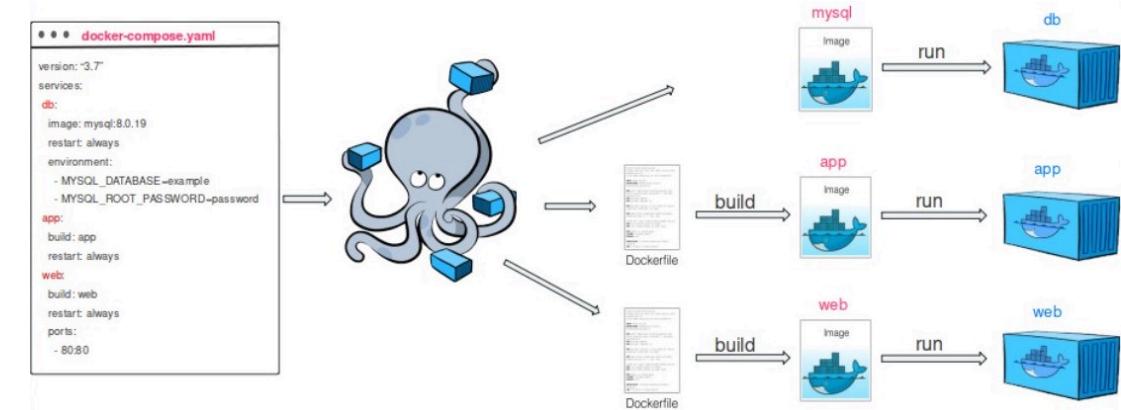
```
docker exec -it my-conteneur bash
```

Suppression des images inutilisées

```
docker system prune
```

Gérer plusieurs conteneurs à la fois

- Outil pour définir la configuration de plusieurs conteneurs à la fois
- Utilise la syntaxe yaml avec avec un fichier `docker-compose.yaml`
- Gestion poussée de la gestion des conteneurs (lancer, arrêter de multiples conteneurs)
- Permet de gérer le build et le run des conteneurs



```
$ docker compose up

Creating network "composetest_default" with the default driver
Creating composetest_web_1 ...
Creating composetest_redis_1 ...
Creating composetest_web_1
Creating composetest_redis_1 ... done
Attaching to composetest_web_1, composetest_redis_1
web_1    | * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
redis_1  | 1:C 17 Aug 22:11:10.480 # 0000000000 Redis is starting oC
redis_1  | 1:C 17 Aug 22:11:10.480 # Redis version=4.0.1, bits=64, comm
redis_1  | 1:C 17 Aug 22:11:10.480 # Warning: no config file specified,
web_1    | * Restarting with stat
redis_1  | 1:M 17 Aug 22:11:10.483 * Running mode=standalone, port=6379
redis_1  | 1:M 17 Aug 22:11:10.483 # WARNING: The TCP backlog setting c
web_1    | * Debugger is active!
redis_1  | 1:M 17 Aug 22:11:10.483 # Server initialized
redis_1  | 1:M 17 Aug 22:11:10.483 # WARNING you have Transparent Huge
web_1    | * Debugger PIN: 330-787-903
redis_1  | 1:M 17 Aug 22:11:10.483 * Ready to accept connections
```

Kubernetes : l'orchestrateur de conteneur

Kubernetes est un outil d'orchestration de conteneurs, open-source, permettant d'automatiser le déploiement d'applications et leurs mise à l'échelle



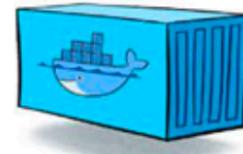
Open source

Initialement Borg chez Google, il a été donné à la CNCF (2015) en open source. Ce modèle permet à l'outil d'être au plus proche des besoins d'utilisateurs, à moindre coût, flexible et fiable



Automatisation

Élimine une bonne partie des processus manuels associés au déploiement et la mise à jour des applications, et offre une couche d'abstraction au prérequis techniques (stockage, réseau, mise à l'échelle, etc.)



L'orchestration de conteneurs

Ceci permet de gérer de la même manière de nombreuses applications et, dans un soucis d'efficience, de mutualiser au maximum les ressources informatiques

Fonctionnement de Kubernetes

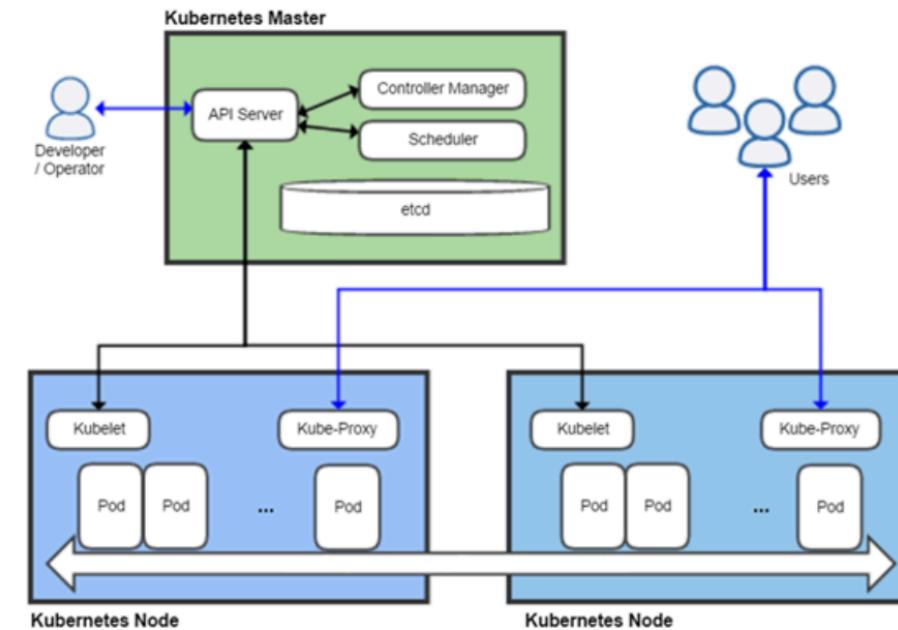
Node : machines physiques ou virtuelles formant le cluster, où sont installés les outils de Kubernetes

Master

- vérifie l'état souhaité et applique les changements (controller manager)
- sauvegarde l'état et la configuration du cluster (base de données etcd)
- dispatche les ressources aux nœuds (scheduler)

Worker

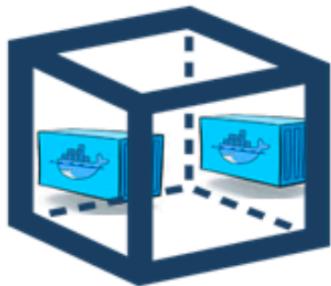
- lance les conteneurs applicatifs (dans des pods). appliquer les instructions de l'API du master via le kubelet
- le proxy s'occupe de l'acheminement réseau
- le stockage des données si besoin



Les objets dans Kubernetes

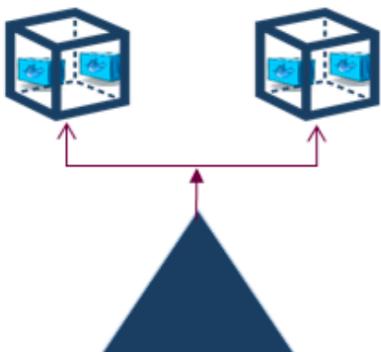
Pod

Le pod est l'unité la plus petite de Kubernetes, il représente généralement un (ou plusieurs) conteneur faisant référence à une même instance d'une application



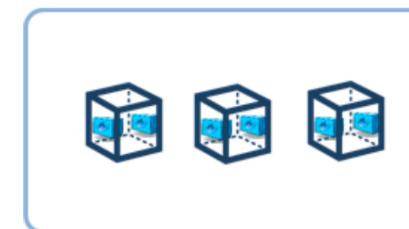
Service

Le service est une terminaison réseau permettant de rediriger chaque requête vers un ou un ensemble de pods (généralement la même instance dupliquée)



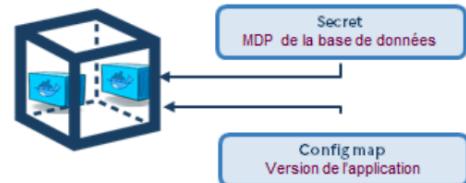
Déploiement

Le déploiement gère un ensemble de pods afin de vérifier leurs existences (le cas échéant leur redémarrage) et leurs mise à jours avec un versioning



ConfigMap et Secret

Il est parfois nécessaire de configurer les Pods avec des données (des mots de passe, des paramètres de déploiements, etc.) Les secrets et configmap sont des fichiers stockant ces valeurs pour provisionner les Pods



Volume

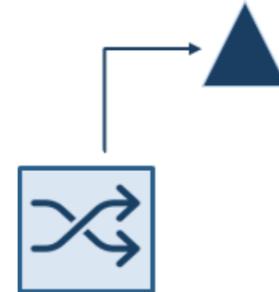
Les volumes sont des éléments de stockage dans le cluster, rattachable aux Pods, mais indépendant de son cycle de vie (il préserve les données même si le pod disparaît)



Ingress

L'ingress est un moyen d'exposer des applications à l'extérieur du cluster.

Lorsque cette ressource est créée, l'ingress controller va vérifier la ressource et rediriger le flux vers un service Kubernetes



Légende :



: Pod

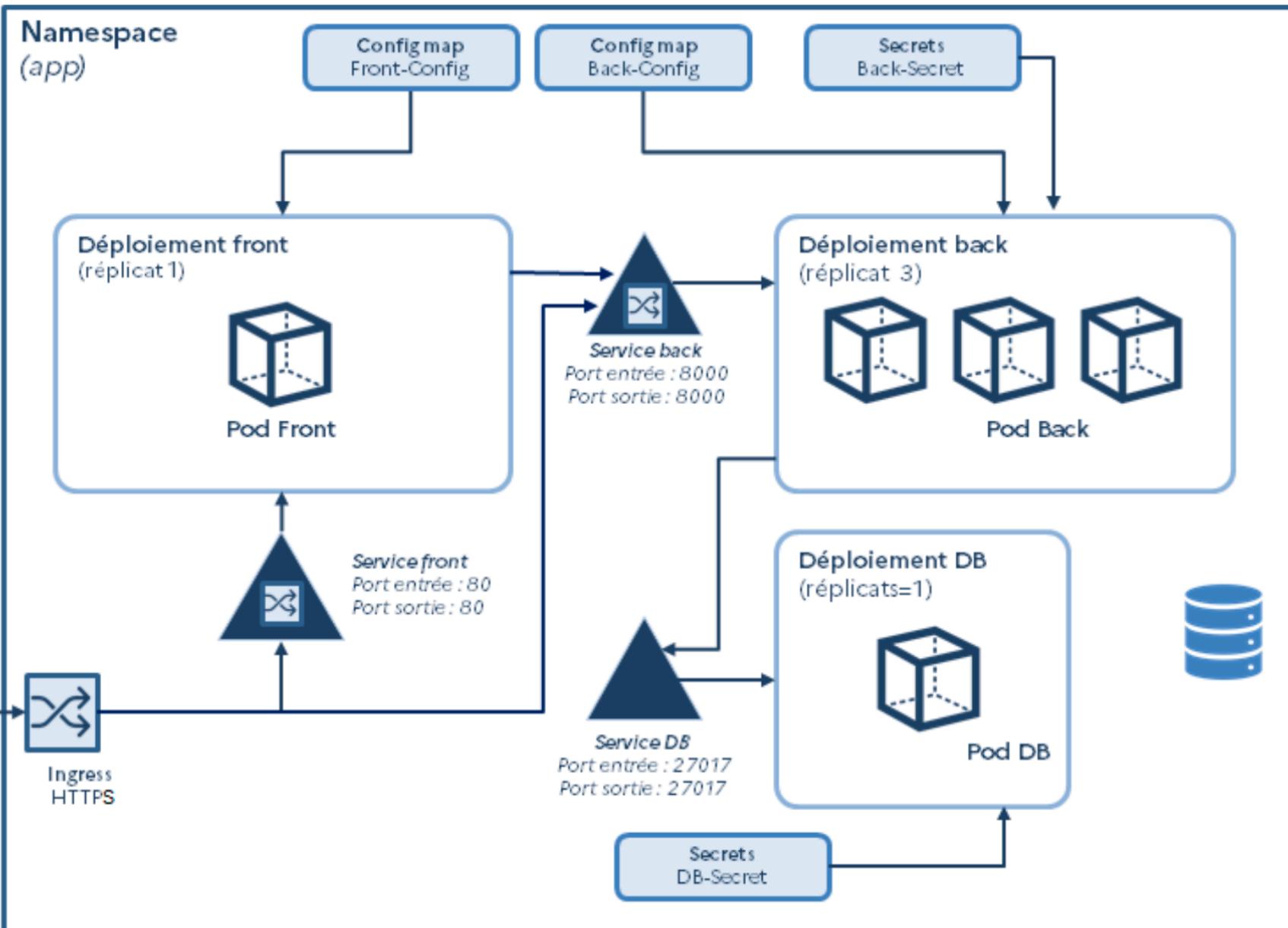


: Service



Internet

Accès URL



Créer un Pod

1. Le fichier descriptif au format .yaml permet de créer un Pod en lançant un conteneur nginx (version 1.14.2) et l'exposer sur le port 80

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
    - name: nginx
      image: nginx:1.14.2
    ports:
      - containerPort: 80
```

2. On demande à Kubernetes d'appliquer la configuration souhaitée

```
kubectl apply -f pod.yaml
```

et ensuite, on peut vérifier si notre Pod fonctionne :

```
kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
jupyter-pyspark-335145-0	1/1	Running	4 (33d ago)	33d
jupyter-python-989169-0	1/1	Running	0	33d
nginx	1/1	Running	0	2m15s
pyspark-shell-14d1a2853aaa299e-exec-35	1/1	Running	0	32d
pyspark-shell-40fb9b8552bd38ad-exec-8	1/1	Running	0	13d
pyspark-shell-7e4b068596dfb873-exec-2	1/1	Running	0	13d
pyspark-shell-d062a885a058c2ab-exec-3	1/1	Running	0	13d

Utiliser la CLI

Créer un Pod nginx avec le port 80 exposé

```
kubectl run nginx --image nginx --port 80
```

Créer un service à partir du pod `nginx`

```
kubectl expose pod/nginx
```

Créer un ingress nommé `web` pour exposer l'application

```
kubectl create ingress  
--class onyxia  
--rule test.example.fr/*=nginx:80 web
```

Afficher la liste des pods

```
kubectl get pods
```

Afficher la configuration de la ressource web de type ingress

```
kubectl get ingress web -o yaml
```

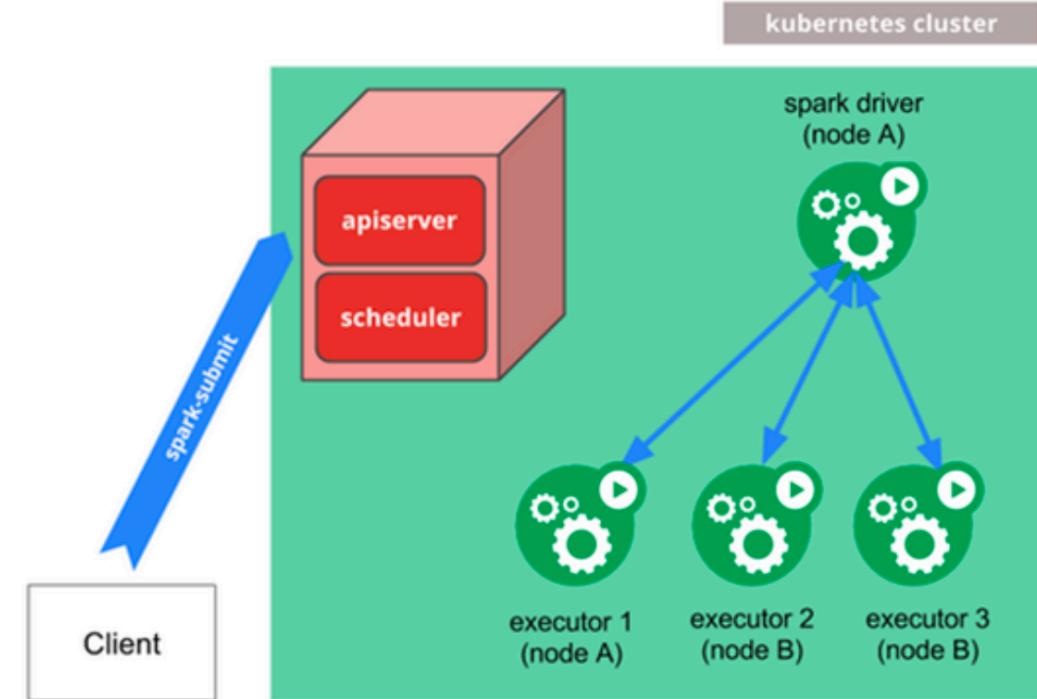
Supprimer une ressource

```
kubectl delete pod nginx
```

Cas d'usage : Spark sous Kubernetes

Apache Spark peut-être utilisé dans un cluster Kubernetes afin d'exécuter des traitement de données parallèles :

- Spark crée un driver Pod
- Le driver crée ensuite un Pod workers (executors) qui sont chargés d'exécuter le code
- Les Pods sont ensuite supprimés par le driver Pod



*Note : le driver reste toujours présent
(dans un état Completed) afin de
conserver les logs du job*