

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

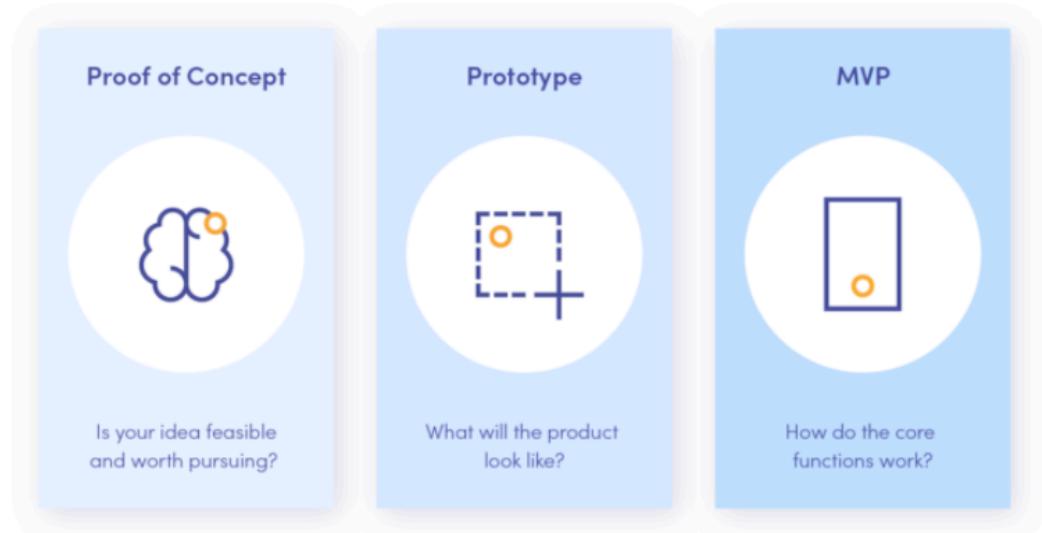
Proof of Concept

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

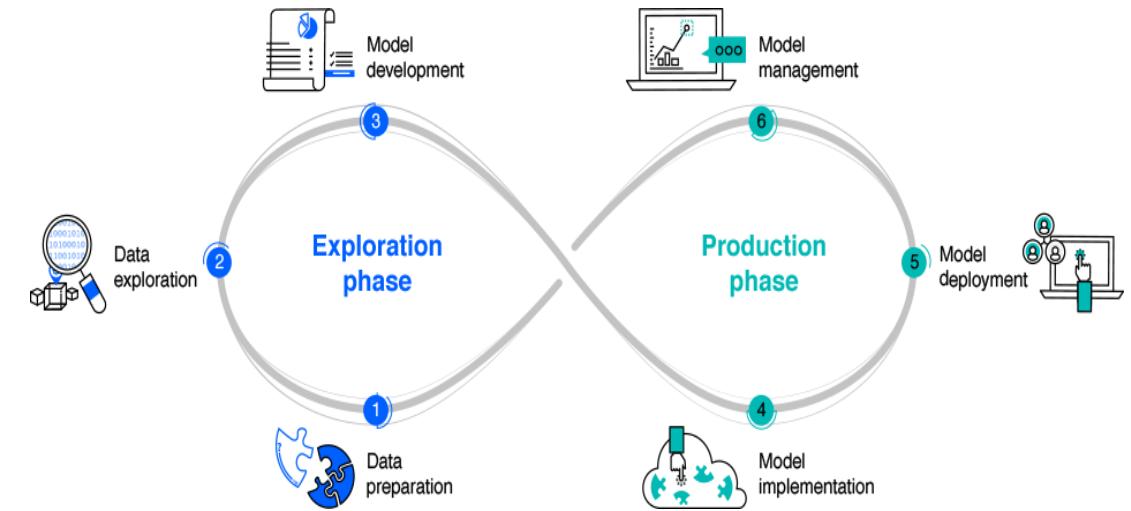
- **Pré-production**

- 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

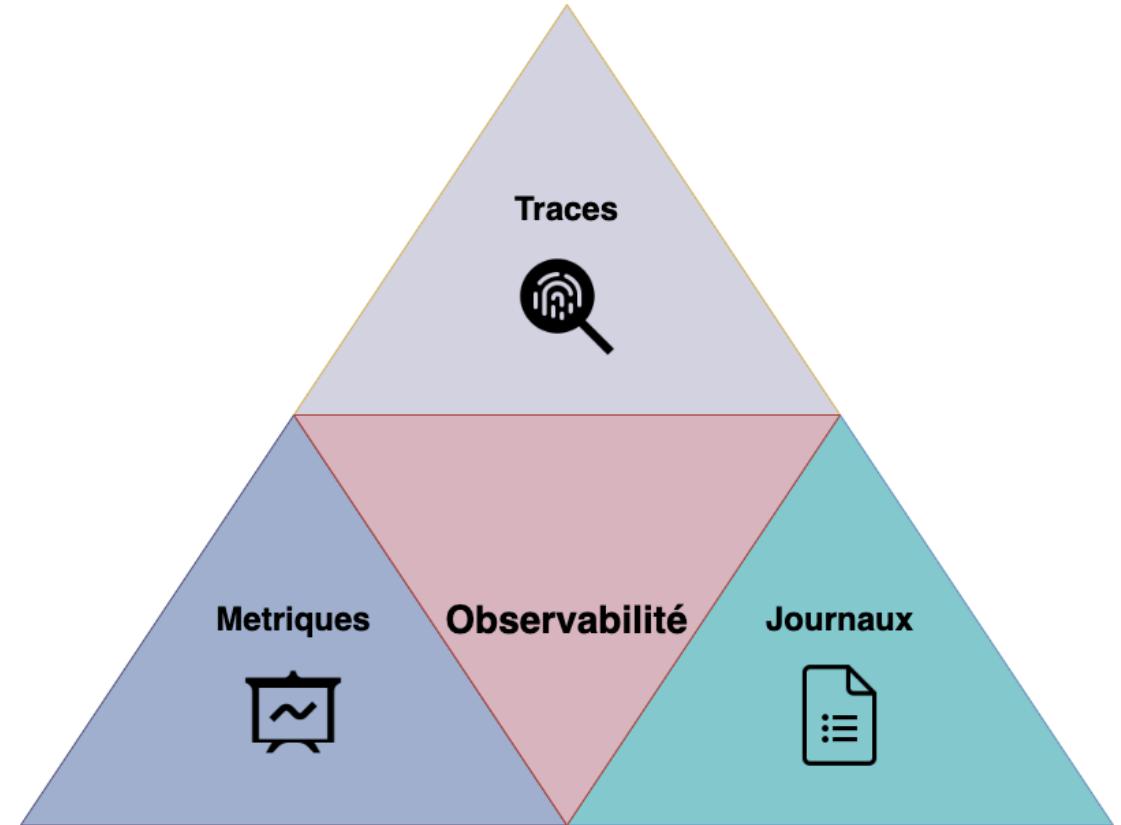
3. Traces

- 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

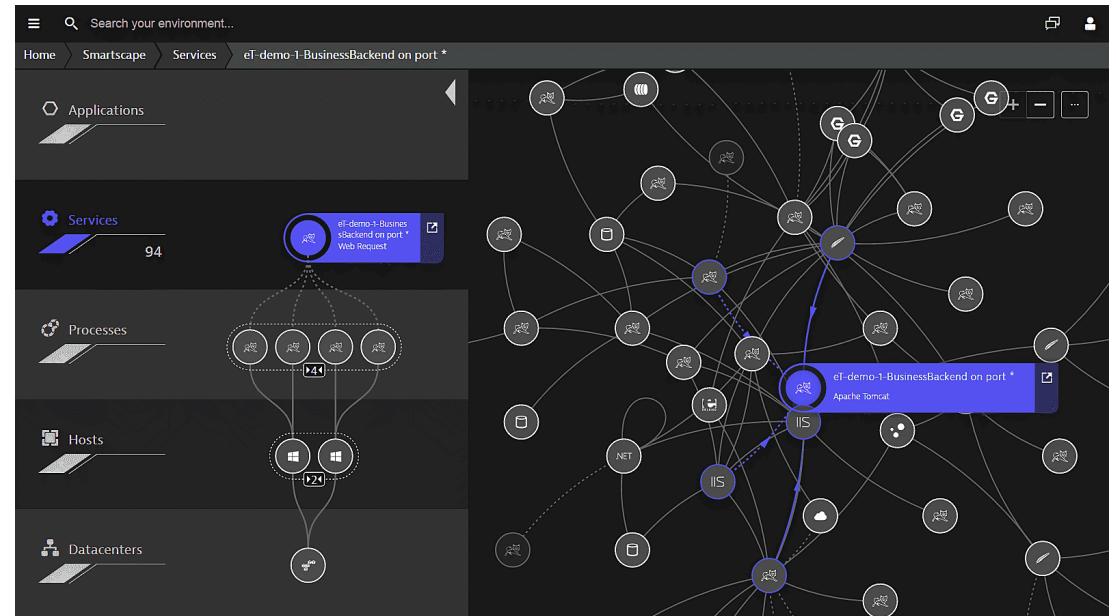
- Les performances (nombre de connexions actives, de requêtes, consommation de ressources, etc.)
- Sa disponibilité (fonctionnement du système, downtime, etc.)
- Son intégrité (résultats corrects)



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

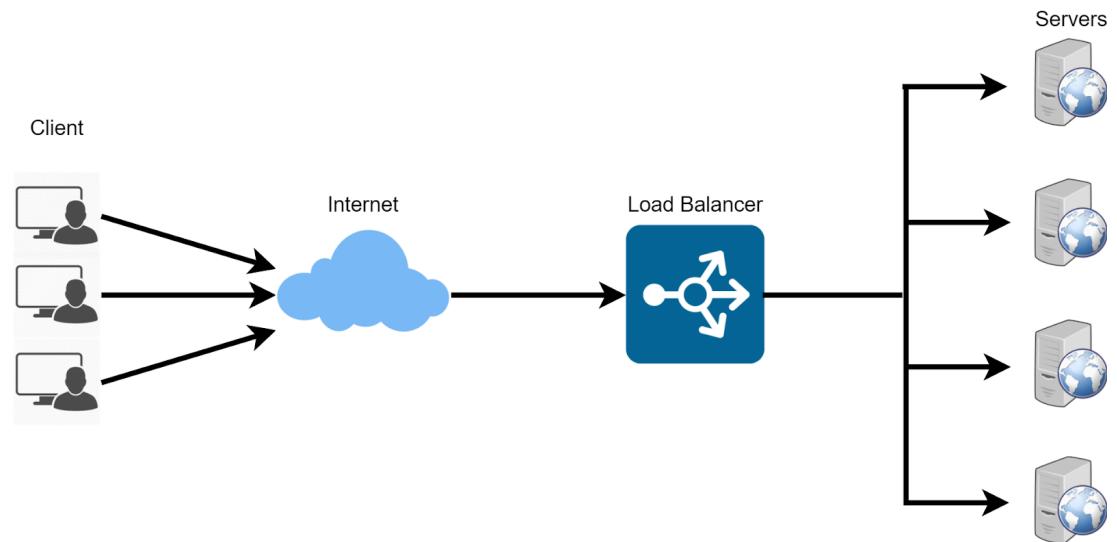
Pourquoi ?

- 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.

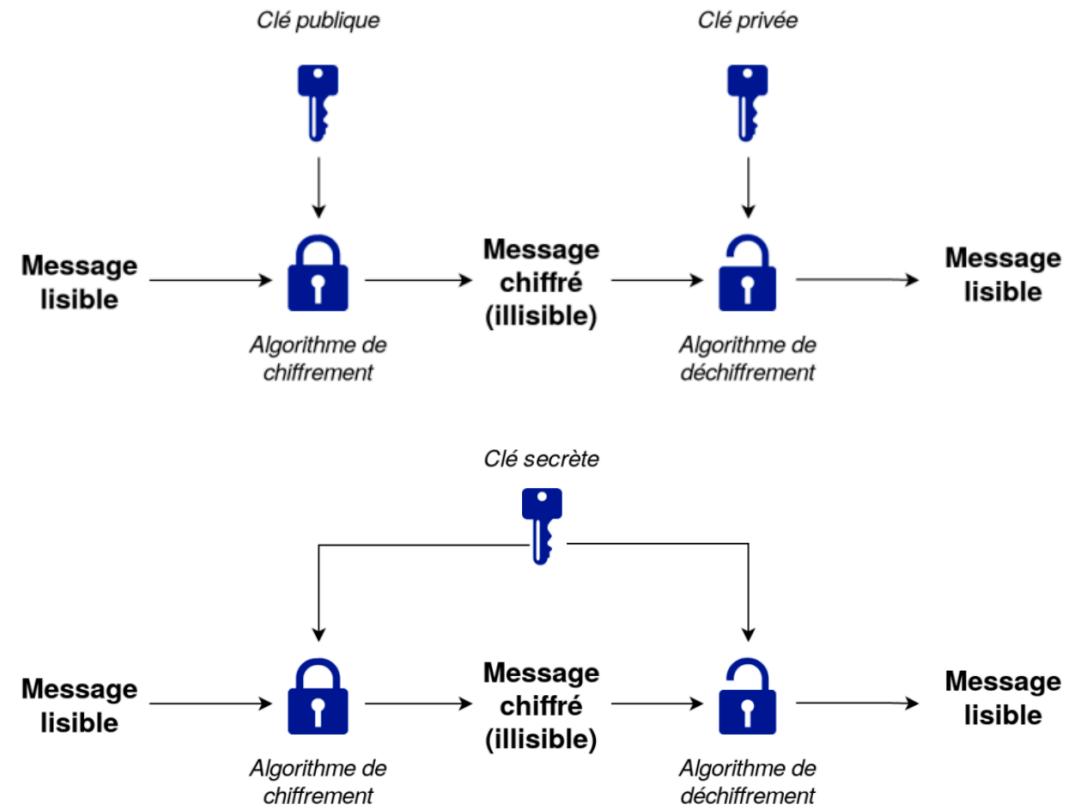
Chiffrer la donnée dans tous ses états

Au repos

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

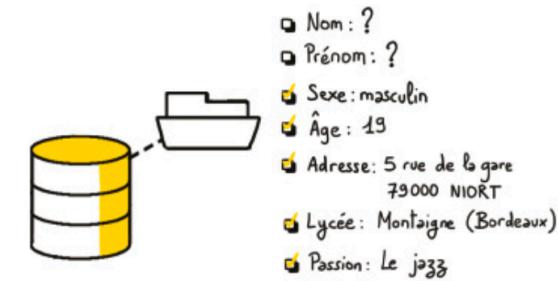
En transit

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



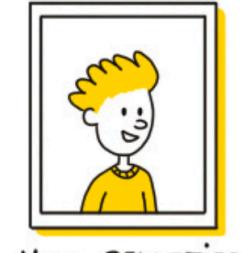
La réglementation

- 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



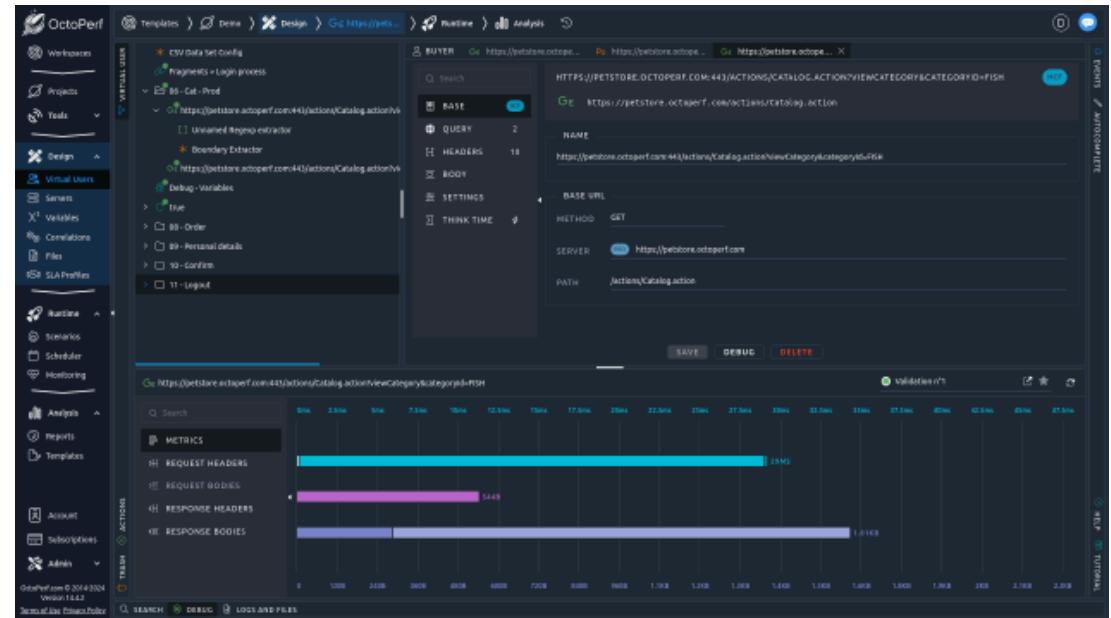
Je suis une base
de données personnelles

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



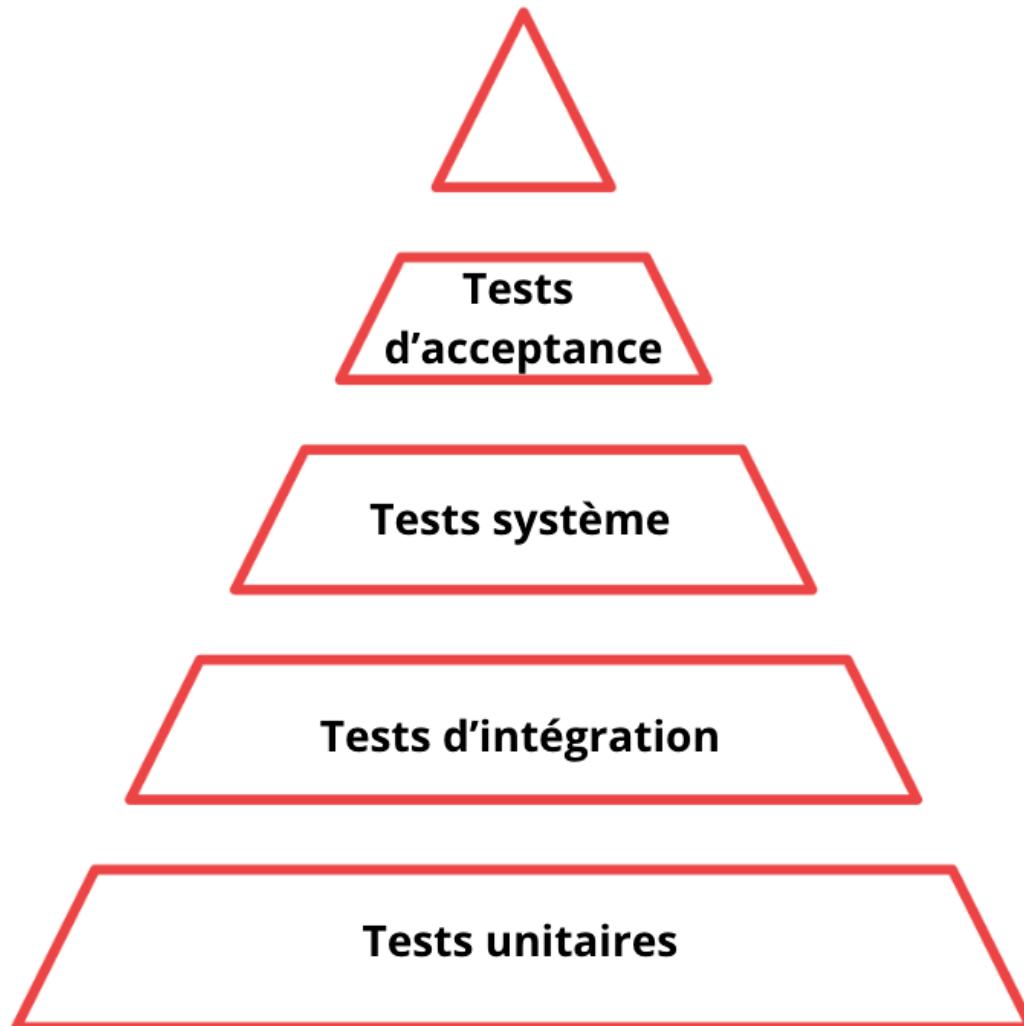
Tests fonctionnels et techniques

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



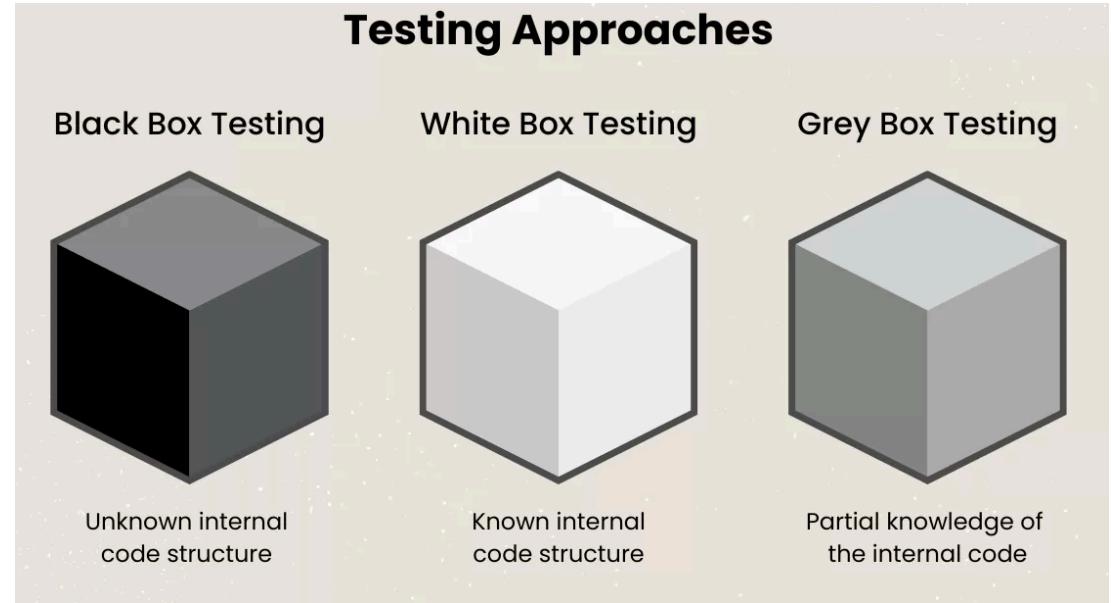
Audit de sécurité

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

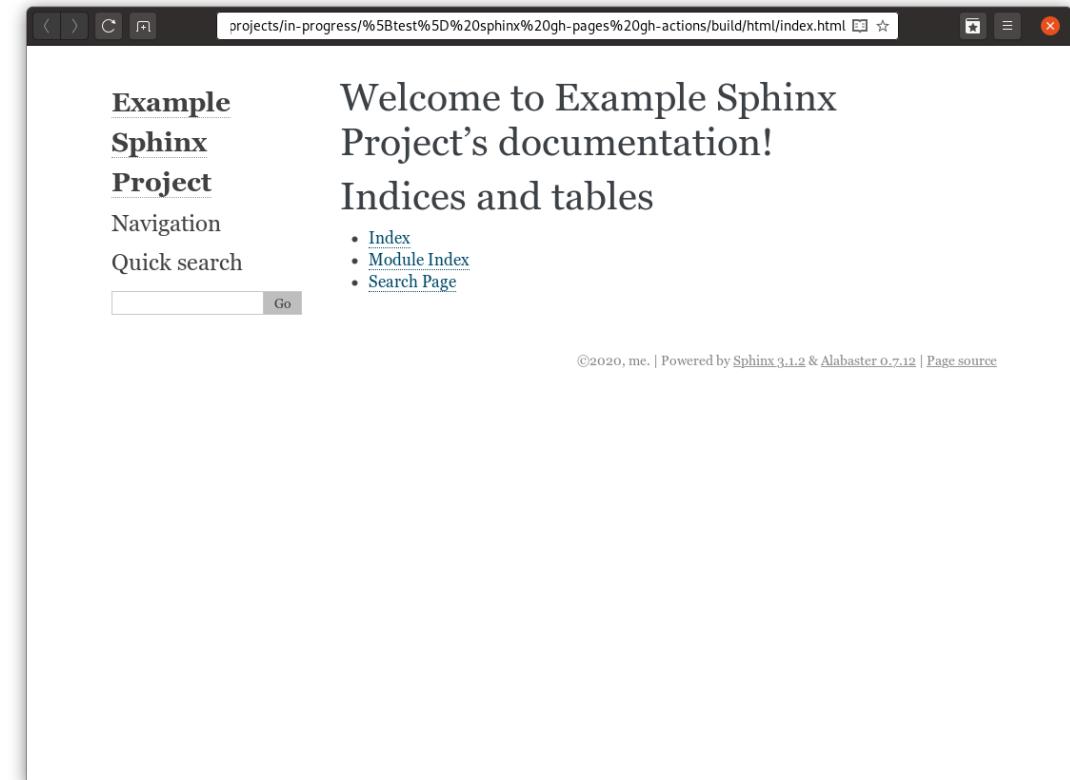
Types de documentation

1. Technique

- Architecture du système
- Configuration des services

2. Fonctionnelle :

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



Sphinx documentation

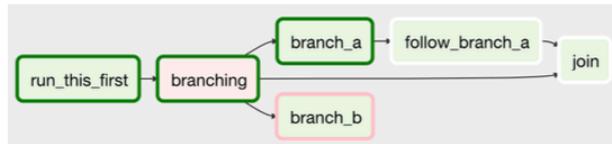
Orchestration et pratique DevOps



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

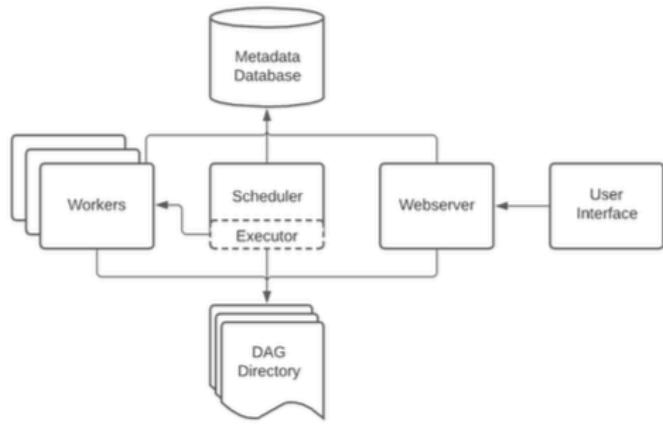


La page principale affiche la liste des DAGs

A screenshot of the Apache 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 12 listed DAGs. Each row in the table includes a small preview icon, the DAG name, the owner (airflow), the number of runs, the schedule, the last run date and time, recent task counts, and action buttons. The table has columns for "DAG", "Owner", "Runs", "Schedule", "Last Run", "Recent Tasks", "Actions", and "Links".

DAG	Owner	Runs	Schedule	Last Run	Recent Tasks	Actions	Links
example_bash_operator	airflow	1	*****	2020-10-26, 21:06:11	1	...	
example_branch_operator_v3	airflow	0	*****		0	...	
example_branch_operator	airflow	0	daily	2020-10-23, 14:08:17	0	0	...
example_complex	airflow	0	None	2020-10-26, 21:08:04	0	0	...
example_external_task_marker_child	airflow	0	None	2020-10-26, 21:07:33	0	0	...
example_external_task_marker_parent	airflow	0	None	2020-10-26, 21:08:34	0	0	...
example_kubernetes_executor	airflow	0	None		0	0	...
example_kubernetes_executor_config	airflow	0	None	2020-10-26, 21:07:40	0	0	...
example_nested_branch_dag	airflow	0	daily	2020-10-26, 21:07:37	0	0	...
example_passing_params_via_xcom	airflow	0	*****		0	0	...

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.)