UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Deploiement logicielle

Sommaire

- Rappel : Architecture Applicative
- Introduction au Déploiement d'Applications
- Environnements de déploiement
- Gestion des Versions et Déploiement
- Continu (CI-CD)
- Conteneurisation

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Architecture Applicative

Architecture Applicative

Definition

- L'architecture logicielle se réfère à la structuration fondamentale d'un système logiciel.
- Elle consiste en la définition des différents composants logiciels (ou modules), les relations qu'ils entretiennent entre eux, et les principes de conception qui guident leur organisation et leur intégration au sein du système.
- Imaginez l'architecture logicielle comme le plan d'un bâtiment, décrivant non seulement les pièces (composants logiciels) mais aussi comment elles sont connectées (communications) et où elles sont placées (déploiement).

Architecture Applicative

Pourquoi est-elle importante ?

L'architecture est la colonne vertébrale d'un projet logiciel, elle affecte directement la qualité et la performance du système fini, en termes de:

Maintenabilité: Une bonne architecture permet de modifier facilement le système pour répondre à des exigences changeantes sans introduire de défauts.

Performance: Les choix architecturaux influencent comment le logiciel performe sous charge, son efficacité en termes de traitement et de réponse.

Sécurité: Une architecture conçue avec la sécurité en tête aide à prévenir les vulnérabilités.

Scalabilité: Elle détermine la facilité avec laquelle le système peut grandir, que ce soit en termes de gestion d'un plus grand nombre d'utilisateurs ou de traitement de davantage de données.

Architecture Applicative

Les éléments clés de l'architecture logicielle ?

Composants: Les unités de logiciel qui effectuent des tâches spécifiques. Ils peuvent être des modules, des classes, ou des paquets, selon la complexité du système.

Connecteurs: Les mécanismes par lesquels les composants communiquent, comme les appels de procédure à distance ou les services web.

Contraintes: Les décisions de conception qui limitent les choix architecturaux, comme l'utilisation de certains modèles de conception, langages de programmation, ou plateformes technologiques.

Architecture Applicative

Comment est-elle mise en pratique ?

Planification et conception: Avant de commencer le développement, les architectes logiciels déterminent quels composants sont nécessaires, comment ils vont interagir, et sur quelles technologies ils seront basés.

Implémentation et déploiement: Les développeurs construisent le système en suivant le plan architectural, et les ingénieurs de déploiement placent les composants sur l'infrastructure appropriée.

Évaluation et évolution: Une fois le système en opération, son architecture est régulièrement évaluée pour s'assurer qu'elle répond toujours aux besoins du projet, avec des ajustements faits au besoin pour adapter ou améliorer le système.

Architecture Applicative

Quel type d'architecture logicielle ?

- Architectures monolithiques
- Architectures de microservices
- Architectures orientées événements
- Architectures orientées services

Architecture Applicative

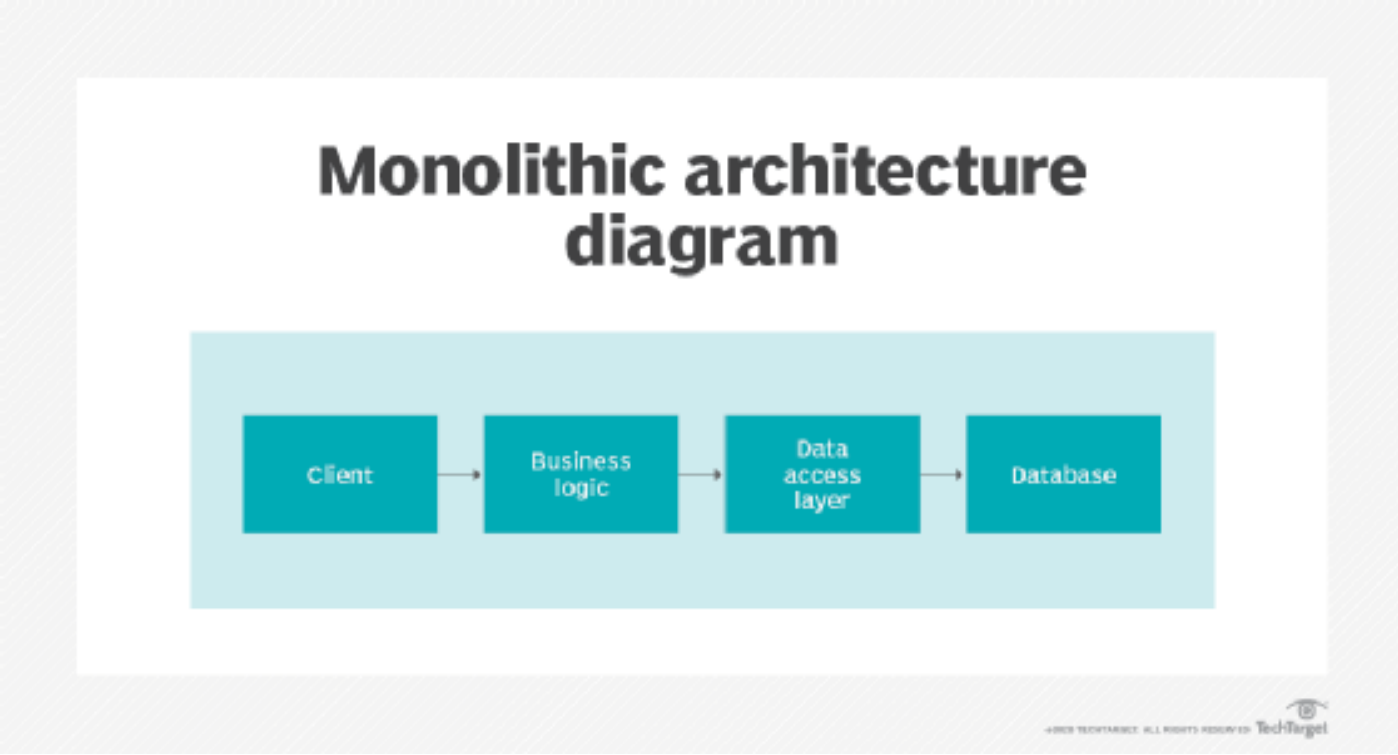
Quel type d'architecture logicielle ?

Architectures monolithiques

- L'architecture monolithique est composée d'une unique pile qui rassemble toutes les fonctionnalités au sein d'une seule application.
- Chaque changement apporté au code implique de republier l'application tout entière ce qui est forcément un frein aux mises à jour, sans même parler d'ajout de nouvelles fonctionnalités.
- Les architectures modernes, au contraire, essaient de décomposer les services en fonctionnalités faiblement couplées pour fournir davantage d'agilité et de flexibilité.

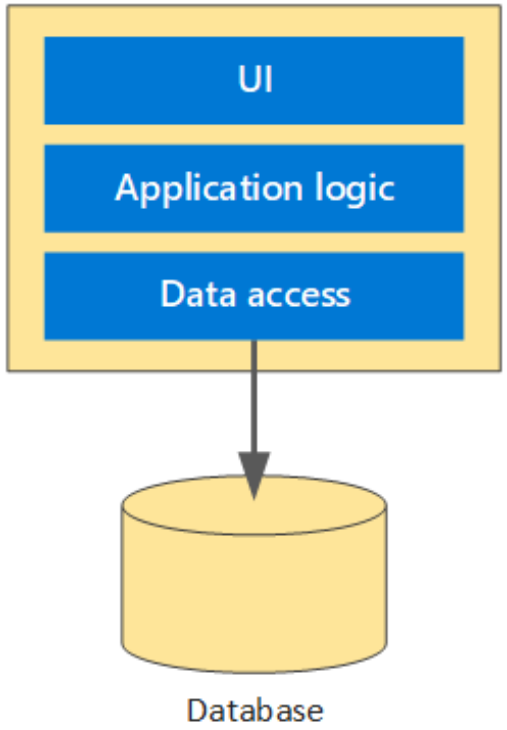
Architecture Applicative

Quel type d'architecture logicielle?



Architecture Applicative

Quel type d'architecture logicielle?



Architecture Applicative

Quel type d'architecture logicielle ?

Architectures de microservices

- Une architecture **microservices** a pour objet de diviser une application en fonctionnalités encapsulées au sein de services autonomes.
- Chacun de ces services est géré et évolue indépendamment des autres services.
- Les **Microservices** peuvent être mis à jour, étendus et déployés indépendamment les uns des autres et, par conséquent, beaucoup plus rapidement tout en limitant le risque d'une mise en péril l'ensemble de l'applicatif.
- Les **Microservices** peuvent communiquer entre eux sans état, à l'aide d'interfaces de programmation d'application (API) indépendantes de tout langage.

Architecture Applicative

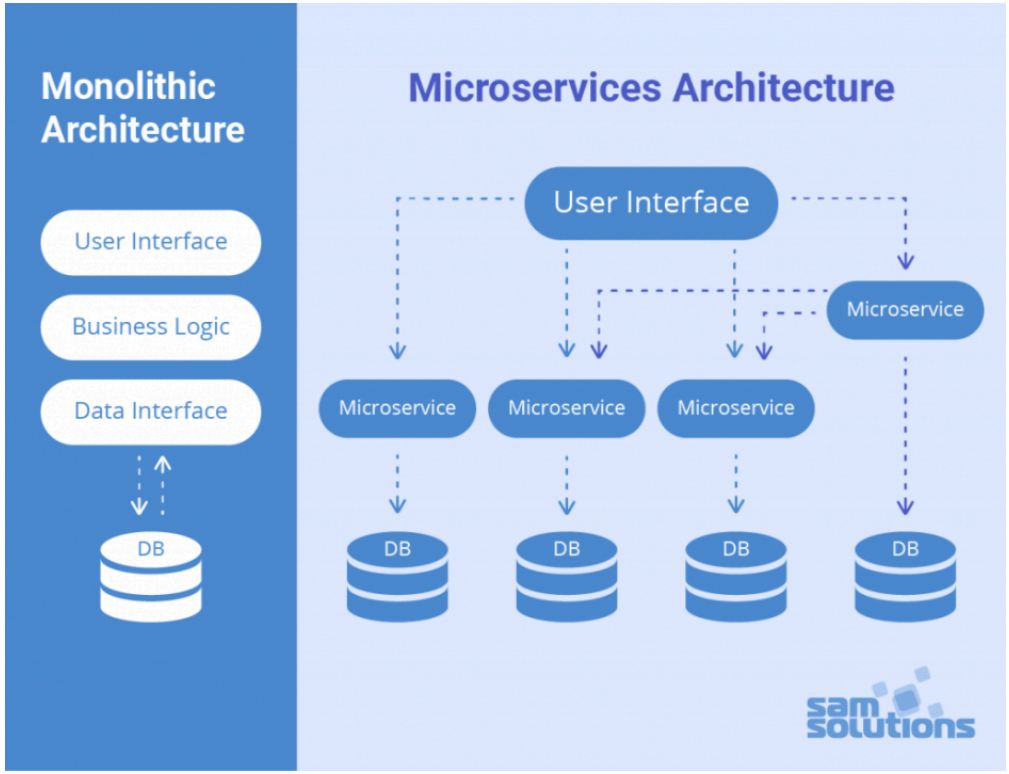
Quel type d'architecture logicielle ?

Architectures de microservices

- Le choix technologique dans une architecture Microservices est donc totalement ouvert.
- Il dépend majoritairement des besoins, de la taille des équipes et de leurs compétences.
- Ces architectures d'applications faiblement couplées qui reposent sur des Microservices avec des APIs et des pratiques DevOps constituent la base des applications cloud-native.
- Le développement cloud-native est une manière d'accélérer la création des nouvelles applications, d'optimiser les applications existantes, d'harmoniser le développement et d'automatiser la gestion pour l'ensemble des clouds privés, publics et hybrides.

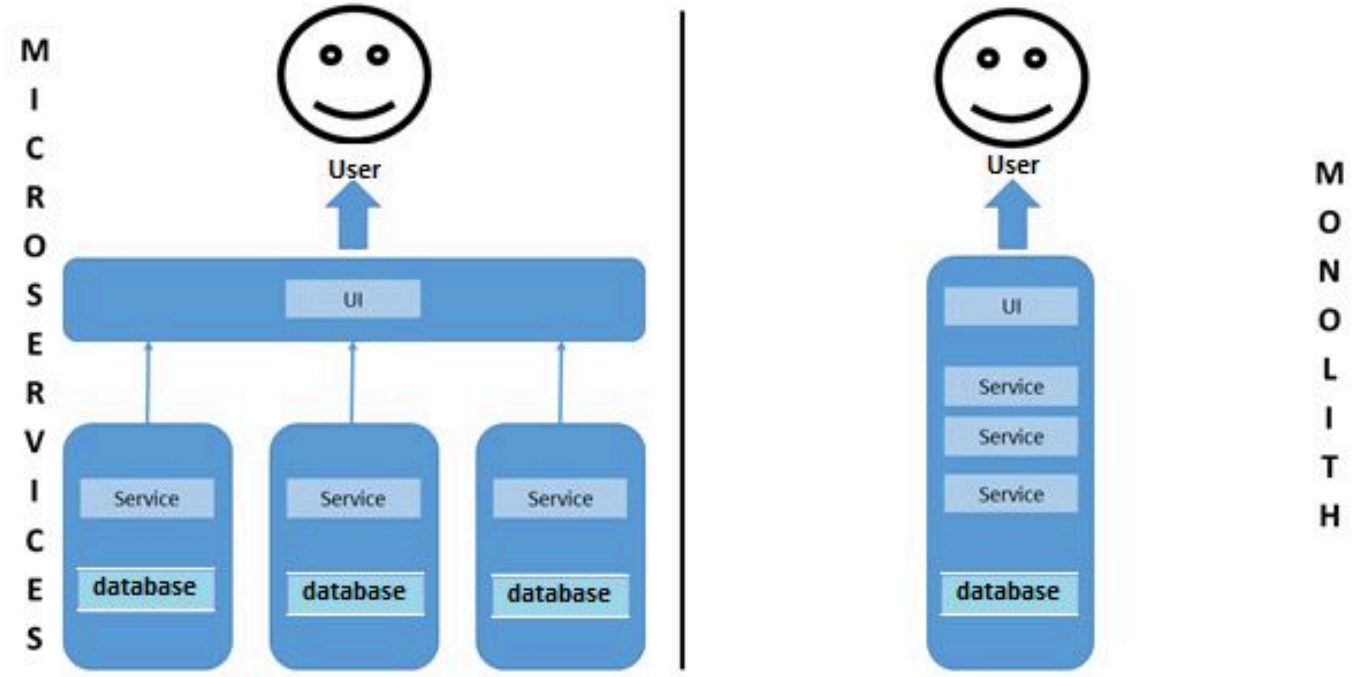
Architecture Applicative

Quel type d'architecture logicielle?



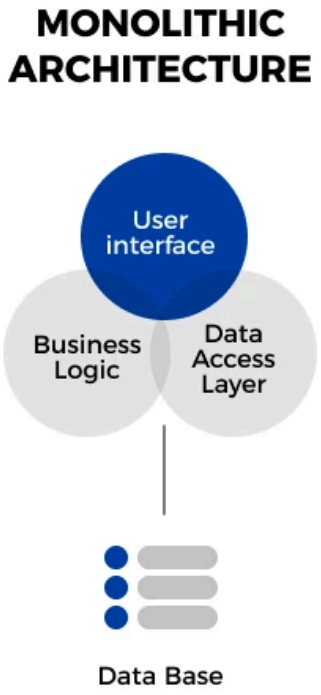
Architecture Applicative

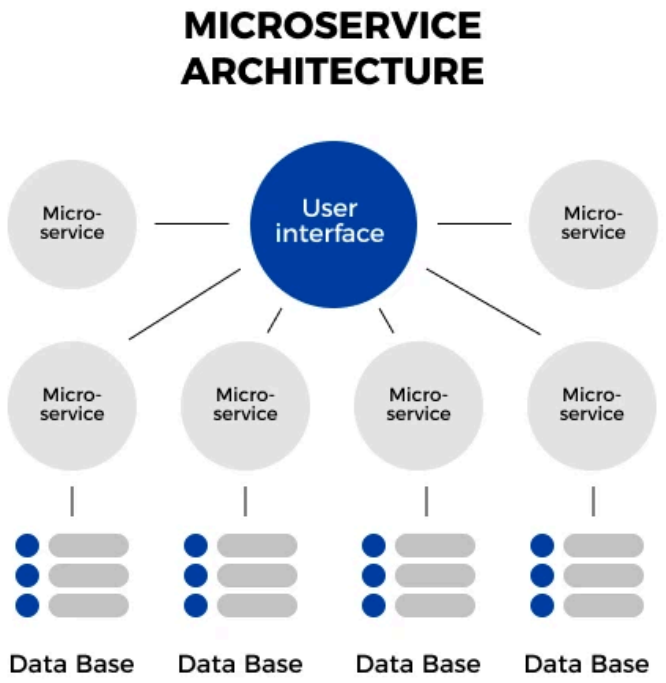
Quel type d'architecture logicielle?



Architecture Applicative

Quel type d'architecture logicielle?





Architecture Applicative

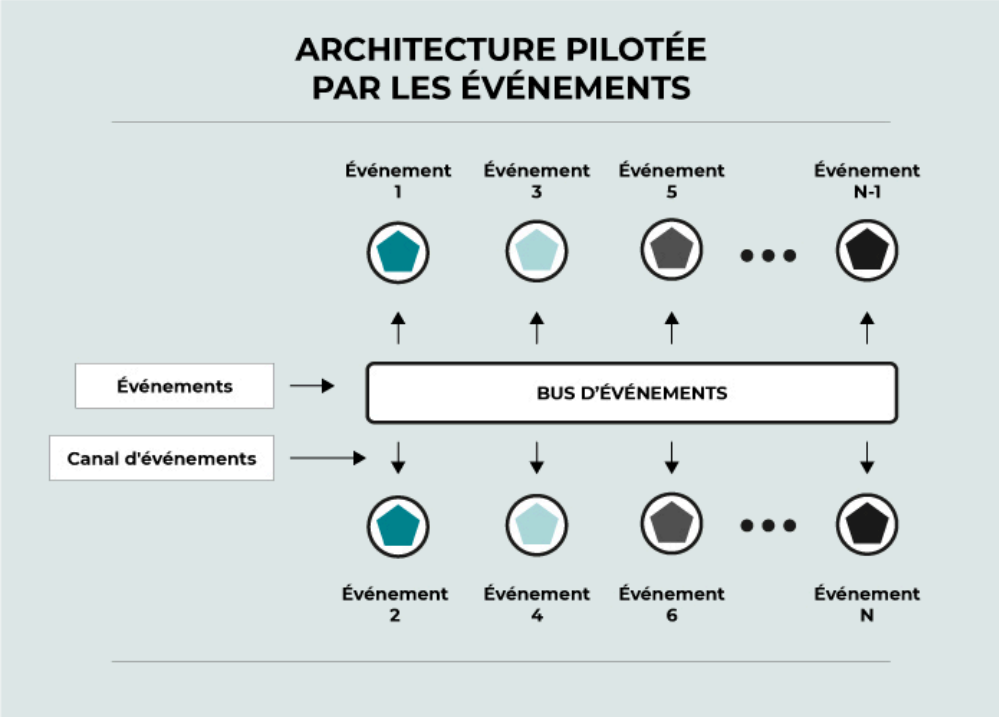
Quel type d'architecture logicielle ?

Architectures orientées événements

- Une architecture orientée événements est un modèle d'architecture logicielle asynchrone et faiblement couplée.
- Elle est donc adaptée aux architectures d'applications modernes distribuées
- Dans un système orienté événements, la structure centrale repose sur le traitement et la persistance des événements, à savoir tout phénomène ou changement d'état significatif.
- Une architecture Microservices peut être orientée événement.

Architecture Applicative

Quel type d'architecture logicielle?



Architecture Applicative

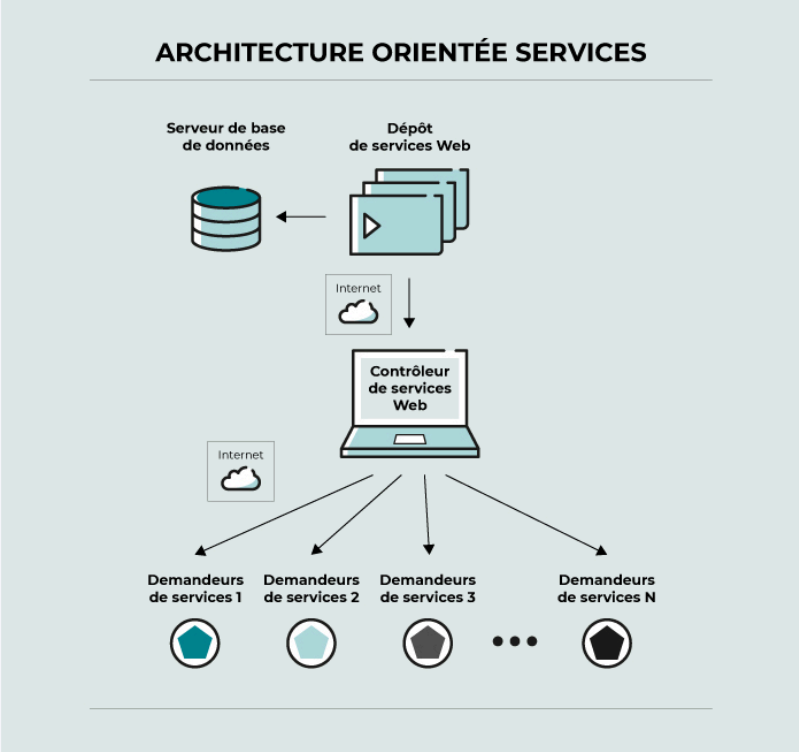
Quel type d'architecture logicielle ?

Architectures orientées services

- L'architecture orientée services (SOA) est un modèle de conception largement utilisé, similaire au modèle d'architecture de Microservices.
- Alors que les **Microservices** structurent une application comme une série de services distincts à usage unique, la **SOA** regroupe des services modulaires qui « dialoguent » entre eux pour prendre en charge les applications et leur déploiement par l'intermédiaire d'un service Bus.
- Les files d'attente de messages assurent donc la coordination entre ces applications distribuées.
- Elles peuvent considérablement simplifier le codage des applications découplées, fluidifier les pics de charge, tout en améliorant les performances, la fiabilité et l'évolutivité.

Architecture Applicative

Quel type d'architecture logicielle?



UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Environnements de déploiement

Environnements de déploiement

La Définition ?

- Les **environnements de déploiement** sont des contextes configurés spécifiquement pour tester, déployer et exécuter des applications logicielles.

- Ces **environnements** sont cruciaux pour le cycle de développement des logiciels, permettant aux équipes de s'assurer que le logiciel fonctionne comme prévu avant qu'il ne soit mis à disposition des utilisateurs finaux

Environnements de déploiement

Les types d'environnements les plus courants ?

Développement : C'est ***l'environnement*** où les développeurs écrivent et testent initialement leur code. Les modifications sont fréquentes et cet environnement est généralement très dynamique.

Test : Souvent appelé environnement ***QA*** (Quality Assurance), il est utilisé pour tester le logiciel de manière systématique et contrôlée pour s'assurer qu'il répond aux exigences spécifiées. Les tests peuvent inclure des tests unitaires, des tests d'intégration, des tests de performance, et plus encore.

Environnements de déploiement

Les types d'environnements les plus courants ?

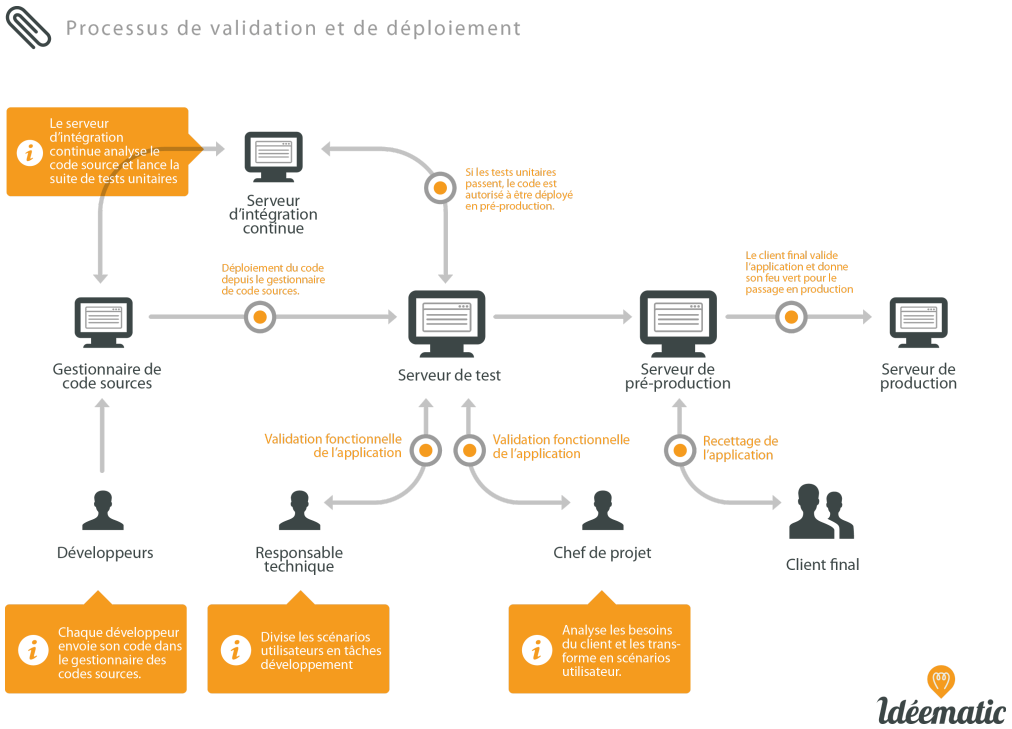
Staging (Préproduction) : Cet ***environnement*** simule le plus possible ***l'environnement de production*** pour permettre des tests finaux avant le déploiement du logiciel. Cela aide à identifier les problèmes qui ne se manifestent que dans une configuration qui imite la production.

Production : C'est ***l'environnement*** où le logiciel est finalement déployé et accessible aux utilisateurs finaux. Il est essentiel que cet ***environnement*** soit stable, sécurisé et performant.

Environnements supplémentaires : Selon les besoins, d'autres ***environnements*** comme ceux pour la formation, la démonstration ou le développement spécifique à une branche du projet peuvent également être configurés.

Environnements de déploiement

Process deploiement



Gestion des Versions et Déploiement

Gestion des versions

1. **Systèmes de contrôle de version :** Outils comme Git, SVN ou Mercurial qui aident les développeurs à suivre les modifications apportées au code source et à collaborer plus efficacement.
2. **Numérotation des versions :** Utilisation de schémas de numérotation pour identifier clairement les différentes versions d'un produit. Les modèles courants incluent la version majeure, mineure, et les patches (par exemple, version 2.3.1).
3. **Branchements et fusions :** Création de branches pour développer des fonctionnalités spécifiques ou corriger des bugs, qui seront ensuite fusionnées avec la branche principale ou d'autres branches.

Gestion des Versions et Déploiement

Déploiement

- **Intégration continue (CI) :** Automatisation de la compilation et des tests du code à chaque commit pour s'assurer que le code est toujours prêt à être déployé.

- **Livraison continue (CD) :** Automatisation de la livraison du code aux environnements de test ou de production après les étapes de CI, minimisant les efforts manuels requis pour déployer les changements.

- **Gestion des environnements :** Configuration et maintenance des environnements de développement, de test et de production qui peuvent différer en termes de configuration et de ressources.

Gestion des Versions et Déploiement

Interaction entre gestion des versions et déploiement

- La gestion des versions et le déploiement sont interconnectés.
- La bonne gestion des versions assure que l'on sait exactement quelle version du logiciel est en production, en test, ou en développement à tout moment.
- Cela permet un déploiement plus sûr et plus systématique des nouvelles versions du logiciel.
- Par exemple, une branche stable de la gestion de versions peut être automatiquement déployée dans l'environnement de production grâce à des pipelines de CI/CD.

UTOPIOS®
TRANSFORMATION DIGITALE INTELLIGENTE

Intégration Continue - Deploiement Continu (CI-CD)

CI-CD

Définition

L'approche CI/CD permet d'augmenter la fréquence de distribution des applications grâce à l'introduction de l'automatisation au niveau des étapes de développement des applications.

CI-CD

Définition

- **Intégration continue**
  - Une méthode de développement logiciel dans laquelle le logiciel est reconstruit et test´e `a chaque modification apportée par un programmeur.
- **Livraison continue**
  - La livraison continue est une approche dans laquelle l'intégration continue associée à des techniques de déploiement automatiques assurent une mise en production rapide et fiable du logiciel.
- **Déploiement continu**
  - Le déploiement continu est une approche dans laquelle chaque modification apportée par un programmeur passe automatiquement toute la chaîne allant des tests à la mise en production. Il n'y a plus d'intervention humaine.

CI-CD

Les avantages CI-CD

- **Accélérer le Time-to-Market** (le temps de développement et de mise en production d'une fonctionnalité).
- **Réduire les erreurs** lors des livraisons.
- **Assurer une continuité** de service des applications.

CI-CD

Les principes etapes du CI-CD

1. **Récupération :** code source récupéré du gestionnaire
2. **Création :** compilation de l'application
3. **Test :** test du code et notamment tests automatisés
4. **Lancement :** distribution de l'application au référentiel
5. **Déploiement :** déploiement du code en production
6. **Validation et conformité :** ces étapes de validation sont à adapter en fonction des besoins de l'entreprise

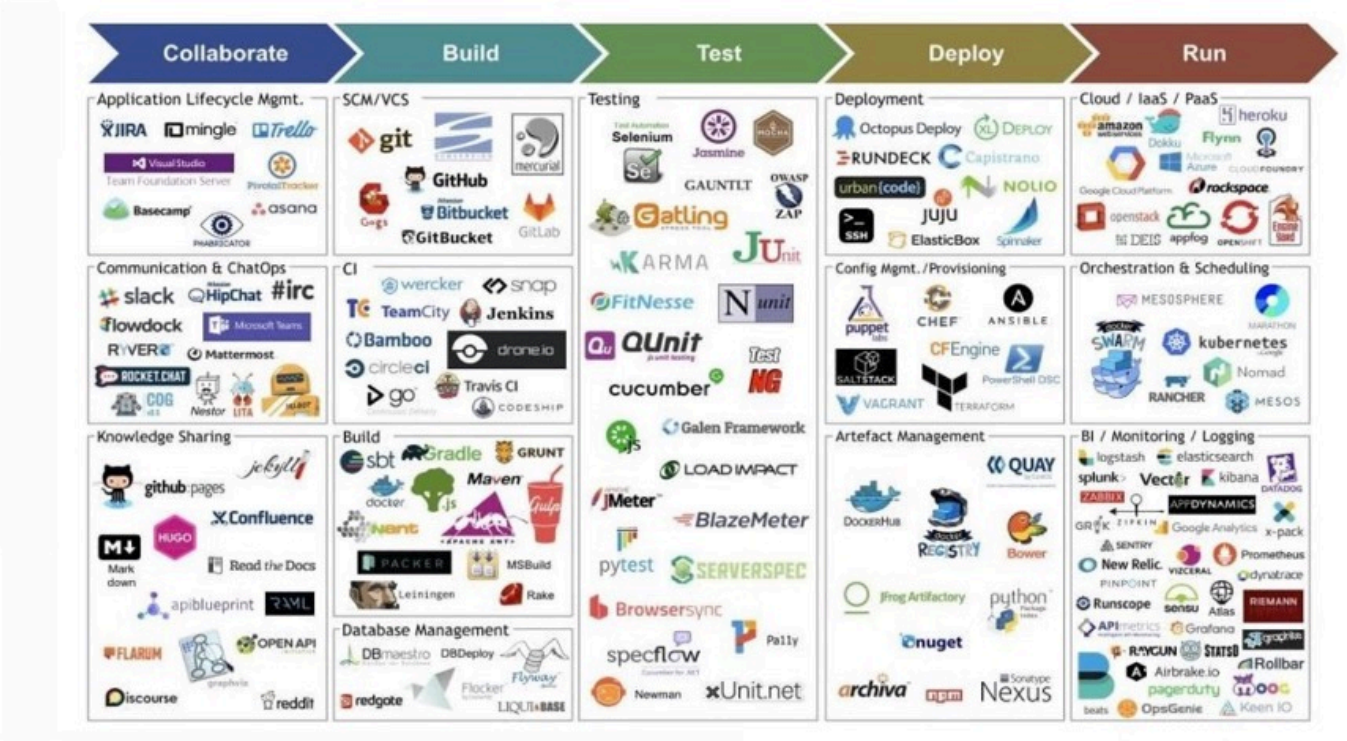
CI-CD

Les outils CI-CD

- **Versionning-commit de code** : *SVN, Git, TFS, GitHub, Tuleap*
- **Qualité de code** : SonarQube, IntelliJ, Eclipse
- **Sécurité** : *SonarQube, Checkmarx, dependency-check-maven*
- **Tests** : *JUnit, Mockito, Selenium, Postman, Gatling, Jmeter*
- **Build** : *Maven, NPM*
- **Intégration/Déploiement/Livraison continus** : *Jenkins, Teamcity, Bamboo, TFS*
- **Gestion des releases** : *Nexus, Artifactory*
- **Déploiement** : *Ansible, Docker, OpenShift, Azure, AWS, VmWare*
- **Performance, monitoring** : *Nagios, Zabbix, Spring boot Actuator*
- **Logs** : *Splunk, Logstash, Elasticsearch*

CI-CD

Les outils CI-CD

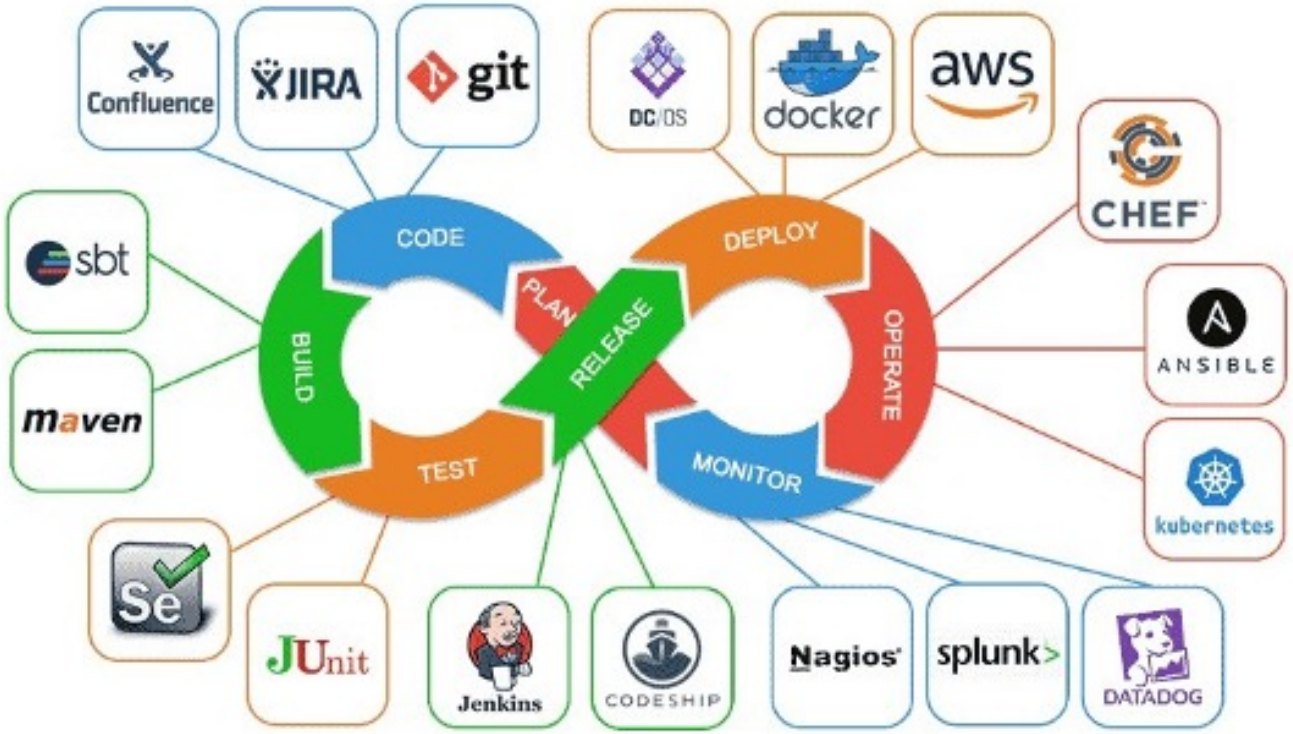


CI-CD

Les outils CI-CD

CI-CD

Les outils CI-CD



CI-CD

Les outils CI-CD