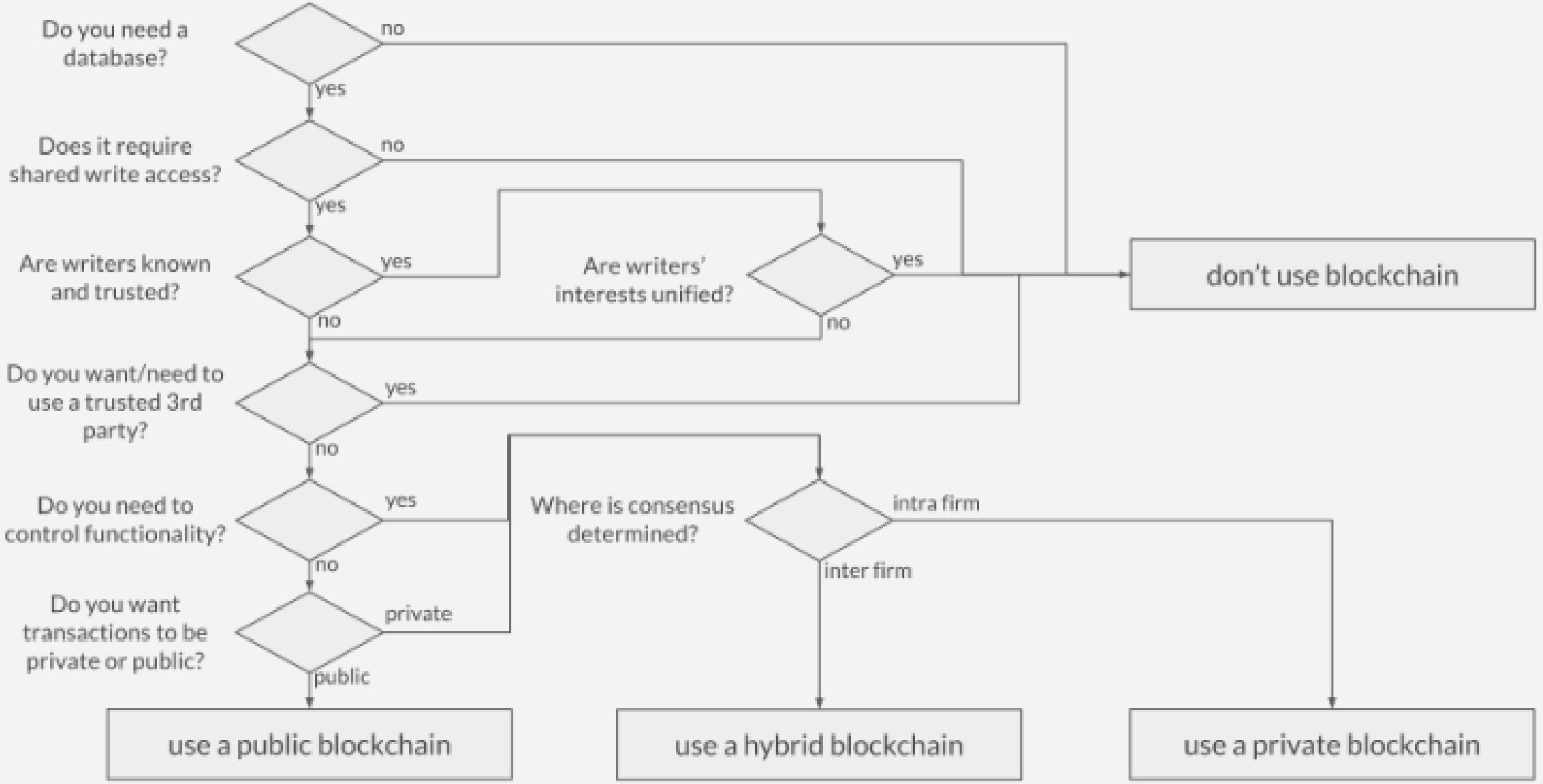
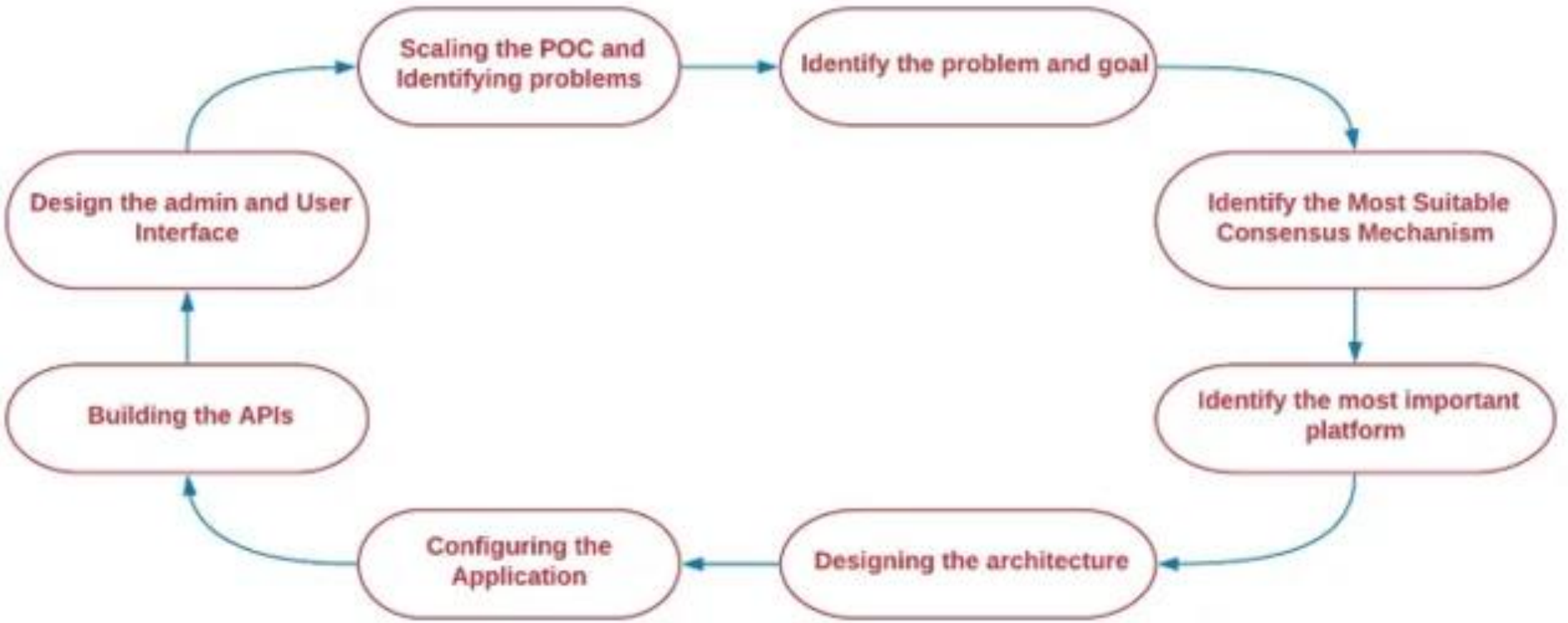
Les étapes d'implémentation d'un projet blockchain

Do you even need Blockchain?



Blockchain Development Life cycle



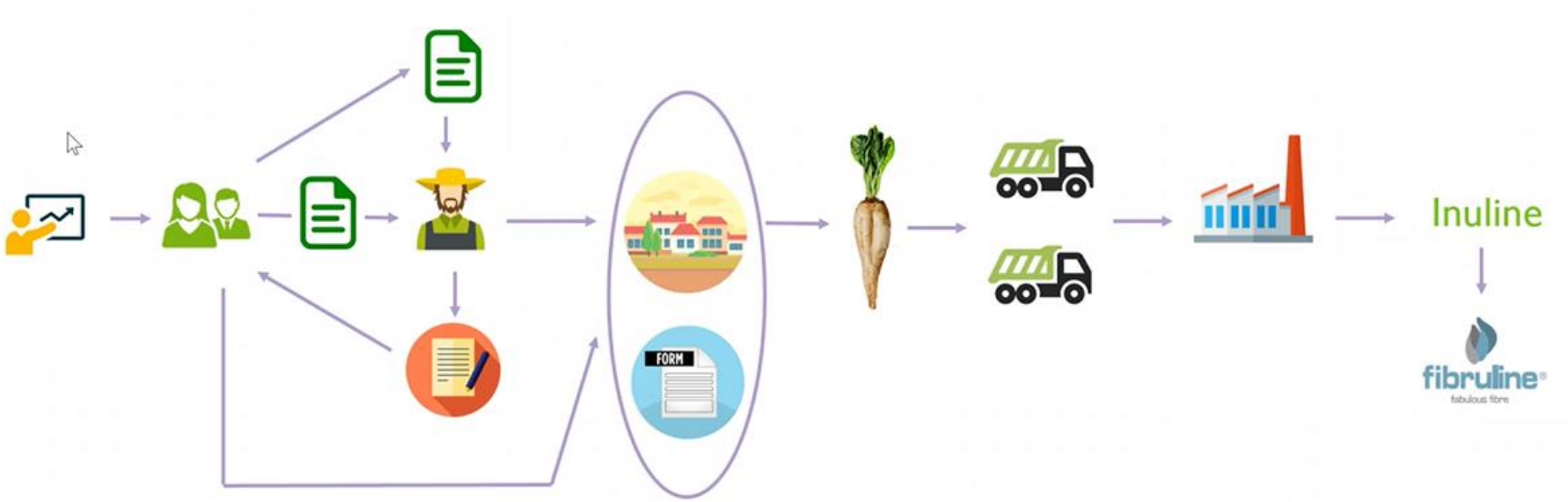
Blockchain Development Life Cycle

1. Définition des objectifs et analyse des besoins

- Identifier les besoins spécifiques du projet
- Déterminer si la blockchain est réellement nécessaire (transparence, immuabilité, décentralisation).
- Définir les cas d'utilisation précis : gestion de transactions, suivi de la chaîne d'approvisionnement, tokenisation, contrats intelligents, etc.
- Choisir le type de blockchain : publique, privée ou hybride

Exemple

Imaginons que vous souhaitiez développer une application blockchain **pour le suivi de la chaîne d'approvisionnement de produits alimentaires**. L'objectif serait de garantir la traçabilité des produits, de la ferme au consommateur. Vous détermineriez que la transparence, l'immuabilité des données et la sécurité sont des besoins clés pour votre application.



Algorithme : Définition des objectifs et analyse des besoins

- **Étape 1 : Identification du Problème ou du Besoin**
  - **Entrée** : Problème à résoudre ou besoin à satisfaire.
  - **Sortie** : Compréhension claire du problème à résoudre.

- **1.1** Décrire le problème spécifique auquel l'application blockchain répond (par exemple, traçabilité des produits alimentaires, gestion des transactions, etc.).
- **1.2** Identifier les parties prenantes (utilisateurs finaux, entreprises, régulateurs, etc.).

Algorithme : Définition des objectifs et analyse des besoins

- **Étape 2 : Analyse des Parties Prenantes**
  - **Entrée** : Parties prenantes identifiées.
  - **Sortie** : Liste des besoins et des attentes des parties prenantes
- **2.1** Recueillir les attentes des parties prenantes via des entretiens, des enquêtes ou des groupes de discussion.
- **2.2** Classer les besoins en fonction de leur priorité (critique, important, secondaire).

1. Définition des objectifs et analyse des besoins

- **Étape 3 : Définition des Objectifs**
  - **Entrée** : Problème, parties prenantes, attentes collectées.
  - **Sortie** : Objectifs clairs et mesurables pour l'application blockchain.
- **3.1** Définir des objectifs SMART pour l'application (Spécifiques, Mesurables, Atteignables, Réalistes, Temporels).
- **3.2** Prioriser les objectifs en fonction de leur impact sur le problème.

1. Définition des objectifs et analyse des besoins

- **Étape 4 : Spécification des Fonctionnalités et Exigences**
  - **Entrée** : Objectifs définis, parties prenantes.
  - **Sortie** : Liste des fonctionnalités et des exigences techniques.
- **4.1** Identifier les fonctionnalités nécessaires pour atteindre les objectifs (par exemple, gestion des utilisateurs, validation des transactions, auditabilité, etc.).
- **4.2** Identifier les exigences techniques liées à la blockchain (ex. : type de blockchain, sécurité, scalabilité).
- **4.3** Prendre en compte les exigences légales et éthiques liées à la confidentialité des données et à la transparence.

1. Définition des objectifs et analyse des besoins

- **Étape 5 : Évaluation des Ressources et Contraintes**
  - **Entrée** : Liste des fonctionnalités, exigences techniques.
  - **Sortie** : Plan des ressources nécessaires.
- **5.1** Identifier les ressources disponibles (temps, budget, compétences techniques).
- **5.2** Identifier les contraintes (juridiques, technologiques, budgétaires).

1. Définition des objectifs et analyse des besoins

- **Étape 6 : Validation des Objectifs et des Besoins**
  - **Entrée** : Objectifs, besoins, exigences, ressources.
  - **Sortie** : Confirmation de l’alignement des objectifs et des besoins avec les parties prenantes.
- **6.1** Valider les objectifs et les fonctionnalités avec les parties prenantes pour s’assurer qu'ils répondent aux attentes.
- **6.2** Ajuster si nécessaire en fonction des retours reçus.

Canvas : Modèle d’analyse des besoins pour une application blockchain

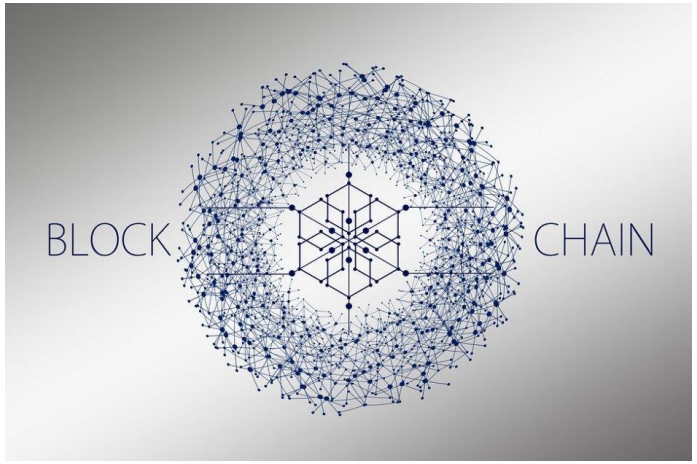
- Voici un **canvas** structuré pour aider à définir les objectifs et à analyser les besoins d'une application blockchain. Ce modèle vous permet de répondre à des questions clés à chaque étape.

Composant	Description
Problème à résoudre	Quelle problématique précise cette application va-t-elle résoudre ? (ex. : traçabilité des produits, transactions sécurisées)
Parties prenantes	Qui sont les utilisateurs finaux, les entreprises, les régulateurs, etc. ? Liste des acteurs du système blockchain.
Besoins des parties prenantes	Qu'attendent les parties prenantes de cette application ? Quelles sont leurs priorités et préoccupations principales ?
Objectifs principaux	Quels sont les objectifs spécifiques de l’application ? Utilisez des objectifs SMART (ex. : garantir la transparence des transactions, faciliter l’auditabilité).
Fonctionnalités clés	Quelles sont les fonctionnalités essentielles de l’application ? (Ex. : création de comptes, validation de transactions, reporting).
Exigences techniques	Quels sont les besoins techniques (type de blockchain, capacité de scalabilité, gestion des données) ?
Exigences légales et éthiques	Quelles sont les contraintes légales concernant la protection des données, la transparence ou la confidentialité ?
Ressources nécessaires	Quelles ressources humaines, matérielles et financières seront nécessaires pour la réalisation du projet ?
Contraintes	Quelles sont les contraintes (budget, temps, technologies disponibles) à prendre en compte ?
Risques identifiés	Quels risques sont associés au projet ? (risques techniques, légaux, liés à la scalabilité, etc.)
Validation	Comment valider les besoins et objectifs avec les parties prenantes ? Méthodes de validation (entretiens, tests utilisateurs, etc.).

2. Choix de la plateforme blockchain

•Sélectionner une plateforme adaptée en fonction des besoins :

- •**Ethereum** (contrats intelligents).
- •**Hyperledger Fabric** (solutions privées pour entreprises).
- •**Solana**, **Binance Smart Chain**, **Polygon** (performances élevées).



•Comparer les avantages en termes de scalabilité, vitesse, coût des transactions et facilité de développement.

2. Choix de la plateforme blockchain

- **Exemple :**
  Pour votre application de traçabilité alimentaire, vous pourriez choisir **Ethereum** si vous souhaitez **utiliser des contrats intelligents** pour automatiser des actions comme la certification des produits, ou **Hyperledger Fabric** si vous préférez une solution privée et plus adaptée aux entreprises ayant des partenaires multiples.

Algorithme pour le choix de la plateforme blockchain

❑ **Définir les objectifs du projet**

- Quel est le but de l'application ? (ex : sécurité, transparence, traçabilité, décentralisation)
- Quel type de transaction sera effectué ? (ex : micro-paiements, contrats intelligents)
- Qui seront les utilisateurs ? (ex : entreprises, utilisateurs finaux, consortiums)

❑ **Analyser les besoins techniques**

- **Scalabilité** : Quelle est la capacité nécessaire pour supporter un grand volume de transactions ?
- **Performance** : Quels sont les délais de confirmation des transactions ? Quelle latence est acceptable ?
- **Sécurité** : Quel niveau de sécurité est requis ? (ex : résistance aux attaques, intégrité des données)
- **Interoperabilité** : La plateforme doit-elle interagir avec d'autres systèmes ou blockchains ?
- **Consensus** : Quel mécanisme de consensus est le plus adapté ? (ex : Proof of Work, Proof of Stake, etc.)

Algorithme pour le choix de la plateforme blockchain

❑ **Évaluer les plateformes possibles**

Recherche des plateformes populaires (ex : Ethereum, Hyperledger, Binance Smart Chain, Polkadot, etc.)

Comparer les caractéristiques techniques, le coût, la communauté de développeurs, et la documentation disponible.

❑ **Comparer la personnalisation et la flexibilité**

Est-ce que la plateforme permet de personnaliser les contrats intelligents et d'intégrer des fonctionnalités spécifiques ?

La plateforme offre-t-elle des outils de développement adéquats ?

❑ **Analyser les coûts**

Quel est le coût de la transaction sur la plateforme ?

Quels sont les coûts de développement, de déploiement et de maintenance ?

❑ **Choisir la plateforme**

En fonction des objectifs du projet, des critères techniques, des coûts et de la flexibilité, choisir la plateforme la plus adaptée.

Canevas pour le choix de la plateforme blockchain

Critères	Détails
Objectifs du projet	- Sécurité ? Transparence ? Traçabilité ? Décentralisation ?
Type de transactions	- Micro-paiements ? Contrats intelligents ? Autres types de transactions ?
Utilisateurs cibles	- Entreprises ? Utilisateurs finaux ? Consortiums ?
Scalabilité	- Besoin de supporter un grand volume de transactions ?
Performance	- Temps de confirmation des transactions acceptable ?
Sécurité	- Niveau de sécurité requis ? (ex : attaques possibles, cryptage)
Interoperabilité	- Nécessité d'interagir avec d'autres plateformes ?
Mécanisme de consensus	- Quel mécanisme est préférable ? (ex : Proof of Work, Proof of Stake, etc.)
Plateformes possibles	- Ethereum, Hyperledger, Binance Smart Chain, Polkadot, etc.
Personnalisation et flexibilité	- Capacité à personnaliser les contrats intelligents et autres fonctionnalités
Coût des transactions	- Coût par transaction sur la plateforme choisie ?
Coût de développement	- Coût du développement et de la maintenance sur cette plateforme
Documentation et communauté	- Présence d’une bonne documentation et d’une communauté active pour le support
Choix final	- Sélectionner la plateforme la mieux adaptée aux besoins du projet.

3. Conception de l’architecture

- Décider des composants principaux :
  - **Contrats intelligents** (Smart Contracts) : écrire des règles et logiques métier.
  - **Noeuds** : décider combien et comment ils seront déployés.
  - **Protocoles de consensus** : preuve de travail (PoW), preuve d’enjeu (PoS), etc.
- Planifier l’intégration avec d'autres systèmes (base de données, API, front-end).

3. Conception de l’architecture

- **Exemple :**
  Vous décidez de mettre en place un réseau privé basé sur **Hyperledger Fabric** avec plusieurs nœuds représentant différents acteurs de la chaîne d'approvisionnement (fermes, transformateurs, distributeurs). Vous planifiez également de déployer des **contrats intelligents** pour certifier chaque étape de la production.

Algorithme pour la conception de l'architecture de l'application blockchain

•**Définir les objectifs de l'architecture**
•Quels sont les buts spécifiques pour l'architecture ? (ex : sécurité, scalabilité, performance)
•Quels sont les composants clés nécessaires à l'architecture (ex : contrats intelligents, nœuds, API) ?
•**Conception des composants principaux**
•**Contrats intelligents** :
    •Quels sont les cas d'utilisation du contrat intelligent ? (ex : automatisation des paiements, validation de transactions)
    •Écrire la logique métier sous forme de contrats intelligents.
•**Nœuds** :
    •Déterminer combien de nœuds seront déployés dans le réseau.
    •Choisir les types de nœuds : validateurs, nœuds complets, nœuds légers.
    •Où et comment seront déployés les nœuds (cloud, serveurs physiques) ?
•**Protocoles de consensus** :
    •Choisir le mécanisme de consensus adapté à votre projet (ex : Proof of Work, Proof of Stake, etc.).
    •Analyser les avantages et inconvénients de chaque mécanisme pour choisir celui qui répond aux besoins (scalabilité, sécurité, etc.).
•**Planifier l'intégration avec d'autres systèmes**
•**Base de données** : Choisir comment intégrer la blockchain avec une base de données externe (si nécessaire).
•**API** : Définir les API pour l'interaction entre la blockchain et les applications front-end, ou d'autres systèmes externes.
•**Front-end** : Planifier l'interaction du front-end avec les contrats intelligents et les nœuds via les API.
•**Sécuriser l'architecture**
•Mettre en place des mécanismes de sécurité pour les nœuds, les contrats intelligents et les communications API.
•Protéger les clés privées, les données sensibles et assurer la confidentialité des transactions.
•**Tester l'architecture**
•Effectuer des tests de performance, de sécurité et de compatibilité pour s'assurer que l'architecture fonctionne comme prévu.
•Tester les intégrations avec d'autres systèmes pour vérifier la fluidité des échanges entre la blockchain et les systèmes externes.

Canevas pour la conception de l'architecture d’une application blockchain

Critères	Détails
Objectifs de l’architecture	- Sécurité, scalabilité, performance ?
Composants principaux	- Contrats intelligents, nœuds, protocoles de consensus, intégration avec d’autres systèmes
Contrats intelligents	- Définir les règles métiers et les cas d'utilisation. Exemples de contrats à écrire (ex : paiement automatisé)
Nœuds	- Combien de nœuds seront déployés ?
	- Types de nœuds : validateurs, nœuds complets, nœuds légers ?
	- Où seront déployés les nœuds (cloud, serveurs physiques) ?
Protocoles de consensus	- Choisir entre PoW, PoS, ou un autre mécanisme de consensus adapté aux objectifs du projet.
Intégration avec d'autres systèmes	- Base de données : Comment la blockchain interagit-elle avec une base de données externe ?
	- API : API nécessaires pour communiquer entre le front-end, les contrats intelligents, et autres systèmes
	- Front-end : Interaction avec les contrats intelligents via API et interface utilisateur.
Sécurité de l’architecture	- Mécanismes de sécurité pour nœuds, contrats intelligents, API, et communications.
Tests de l’architecture	- Tests de performance, de sécurité et de compatibilité pour s'assurer de la robustesse de l'architecture.

4. Développement

- **Développement des contrats intelligents** :
  - Écriture avec des langages spécifiques comme **Solidity**, **Vyper** (Ethereum) ou **Chaincode** (Hyperledger).
  - Tester les contrats sur des réseaux de test (testnet).
- **Back-end** :
  - Développer les fonctions qui interagissent avec la blockchain.
  - Utiliser des outils comme **Web3.js**, **Ethers.js**.
- **Front-end** :
  - Développer une interface utilisateur conviviale.
  - Intégrer des portefeuilles numériques (ex. : **Metamask**).
- Utiliser des frameworks comme **Truffle** ou **Hardhat** pour faciliter le développement.

4. Développement

- **Exemple :**
  Vous commencez à développer les **contrats intelligents** en **Solidity** pour automatiser la vérification des produits à chaque étape du processus. Par exemple, un contrat intelligent pourrait garantir qu'un produit ne soit validé pour la vente que si toutes les étapes de production ont été validées par les acteurs de la chaîne.

Algorithme pour *le développement* d'une application blockchain

- **Développement des contrats intelligents**
  - **Choisir un langage** : Choisir le langage adapté pour écrire les contrats intelligents, comme Solidity (pour Ethereum), Vyper, ou Chaincode (pour Hyperledger).
  - **Écrire les contrats intelligents** : Implémenter la logique métier sous forme de contrats intelligents. Par exemple, définir des règles pour les transactions, la gestion des tokens, ou les processus d'automatisation.
  - **Tester sur des réseaux de test (testnet)** : Avant de déployer sur le réseau principal (mainnet), tester les contrats intelligents sur des réseaux de test pour s'assurer qu'ils fonctionnent correctemen

Algorithme pour le développement d’une application blockchain

- **Développement du back-end**
- **Fonctions d'interaction avec la blockchain** : Créer des fonctions qui permettent à l’application back-end d’interagir avec la blockchain (ex : effectuer des transactions, lire des données sur la blockchain).
- **Choisir des outils de développement** : Utiliser des bibliothèques comme Web3.js ou Ethers.js pour interagir avec des contrats intelligents depuis le back-end.
- **Sécurisation et gestion des clés privées** : Mettre en place des mécanismes de sécurité pour gérer les clés privées des utilisateurs et les transactions.

Algorithme pour le développement d'une application blockchain

- **Développement du front-end**
  - **Interface utilisateur** : Créer une interface utilisateur simple et conviviale pour permettre aux utilisateurs d'interagir avec la blockchain via l'application.
  - **Intégration des portefeuilles numériques** : Intégrer des portefeuilles comme Metamask pour gérer les transactions sur la blockchain.
  - **Frameworks pour faciliter le développement** : Utiliser des outils comme Truffle ou Hardhat pour faciliter le processus de développement et de déploiement des contrats intelligents.

Algorithme pour le développement d'une application blockchain

1. **Tests de l'application**
    1. **Tests des contrats intelligents** : Effectuer des tests sur des testnets pour vérifier la fonctionnalité des contrats intelligents avant le déploiement sur mainnet.
    2. **Tests de l'intégration front-end/back-end** : Tester l'interaction entre le front-end, le back-end et la blockchain pour s'assurer que l'application fonctionne correctement.
2. **Déploiement**
    1. Déployer les contrats intelligents sur le réseau principal (mainnet) après avoir effectué tous les tests nécessaires.
    2. Mettre en ligne l'application front-end et back-end.

5. Tests approfondis

- Tester la sécurité des contrats intelligents (vérification des vulnérabilités comme les attaques réseaux).
- Vérifier les performances : scalabilité, temps de traitement des transactions.
- Effectuer des audits de sécurité pour garantir la robustesse du système.

5. Tests approfondis

- **Exemple :**
  Vous testez vos contrats intelligents sur un réseau de test **Rinkeby** (testnet Ethereum) pour valider que le contrat fonctionne correctement dans différents scénarios : par exemple, tester ce qui se passe si un acteur tente de manipuler les informations de la chaîne d'approvisionnement.

6. Déploiement

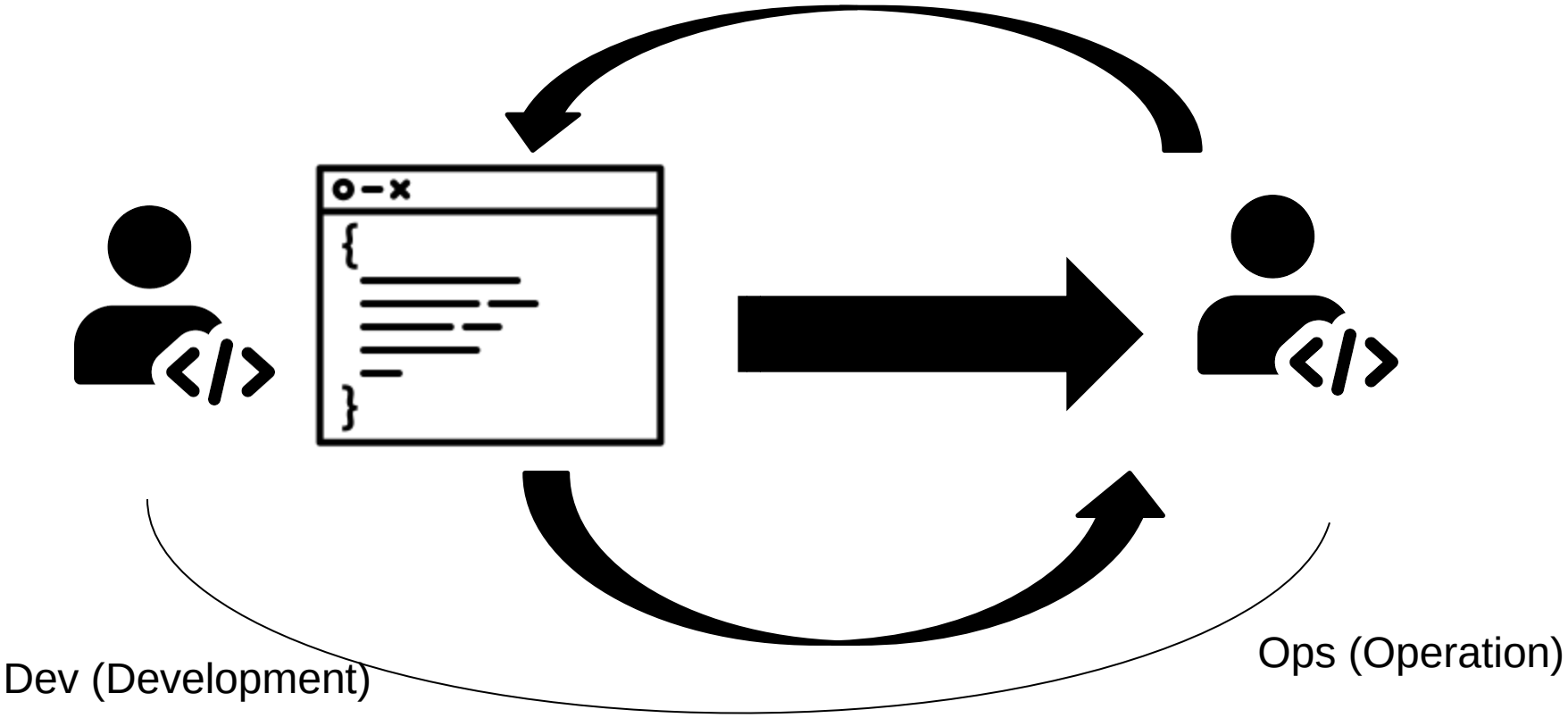
- Déployer sur le réseau blockchain choisi (Mainnet pour un lancement officiel).
- Configurer et gérer les noeuds si nécessaire.
- Mettre en place un monitoring des performances et des transactions.

6. Déploiement

- **Exemple :**
  Une fois les tests concluants, vous déployez votre solution sur le **Mainnet Ethereum** (réseau principal) pour qu'il soit utilisé en production par les différents acteurs de la chaîne d'approvisionnement. Vous installez également des **portefeuilles numériques** pour que les utilisateurs puissent interagir avec l'application.

DevOps:
Development (Dev) and IT Operations (Ops)

DevOps



DevOps

- Le concept de DevOps a émergé au cours **des années 2000** en réponse aux défis **de collaboration** entre les équipes de développement (Dev) et les équipes opérationnelles (Ops) dans le domaine du développement logiciel.

- L'idée centrale de DevOps est **de promouvoir une collaboration** étroite et une intégration continue entre les équipes de développement et d'exploitation, afin d'améliorer l'efficacité du cycle de vie du développement logiciel,

Les phases importantes d'un cycle DevOps

- ❑ Continuous Development
- ❑ Continuous Testing
- ❑ Continuous Deployment
- ❑ Continuous Monitoring
- ❑ Continuous Feedback

DevOps

Créer un environnement

Test

Déployer

Automatiser

1- Bash/powerShell
2- Doker
3- Ansible
4- Jenkins
   githib actions
   gitlab
5- kubernets
6. Terraform
7- Cloud: Azure,ACP, AWS
8- Python: Langage go

DevOps

1- **Bash/powerShell:** interagir avec le système d'exploitation

2- **Doker**: plateforme open source permettant d'automatiser le déploiement,

3- **Ansible**: gestion de configuration et d'automatisation des tâches

4- **Jenkins:** automatise le processus CI/CD

- GitHub actions

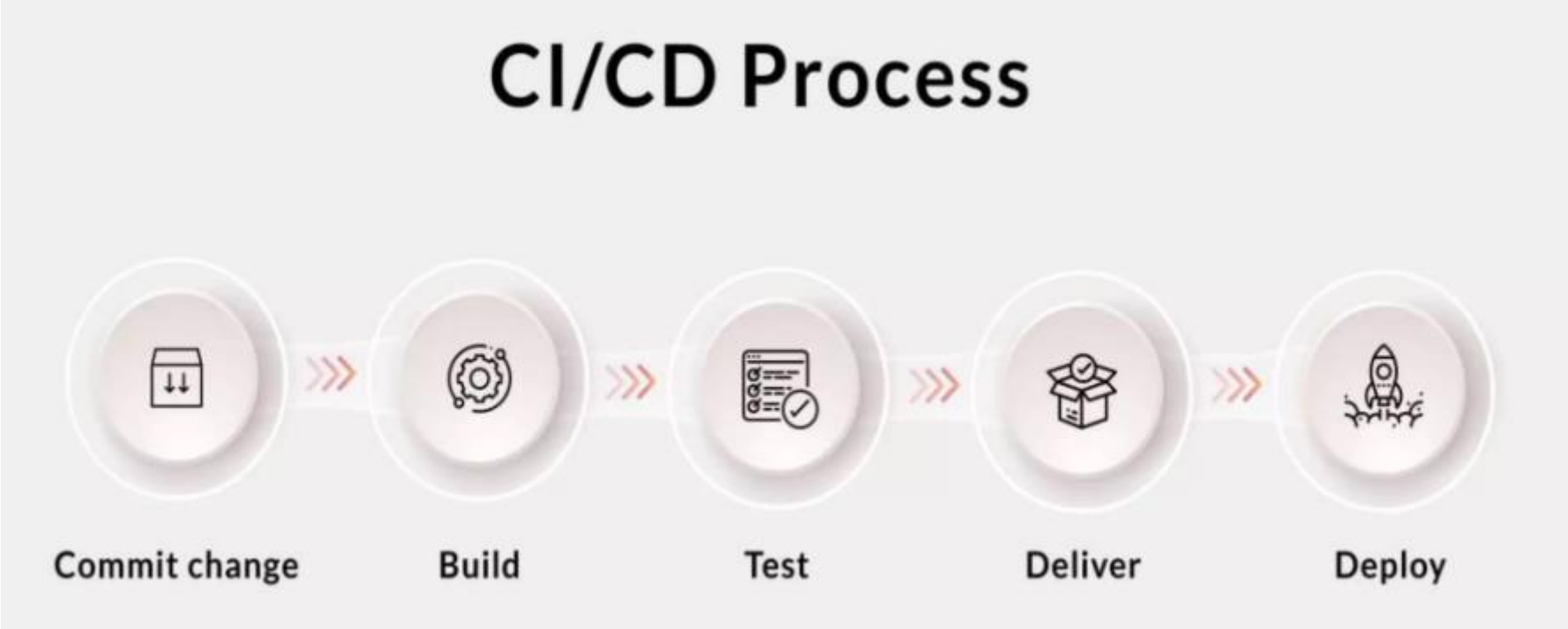
5- **kubernets:** gestion d'orchestration de conteneurs qui facilite le déploiement

6. **Terraform:** Automatisation et la codification de l'infrastructure

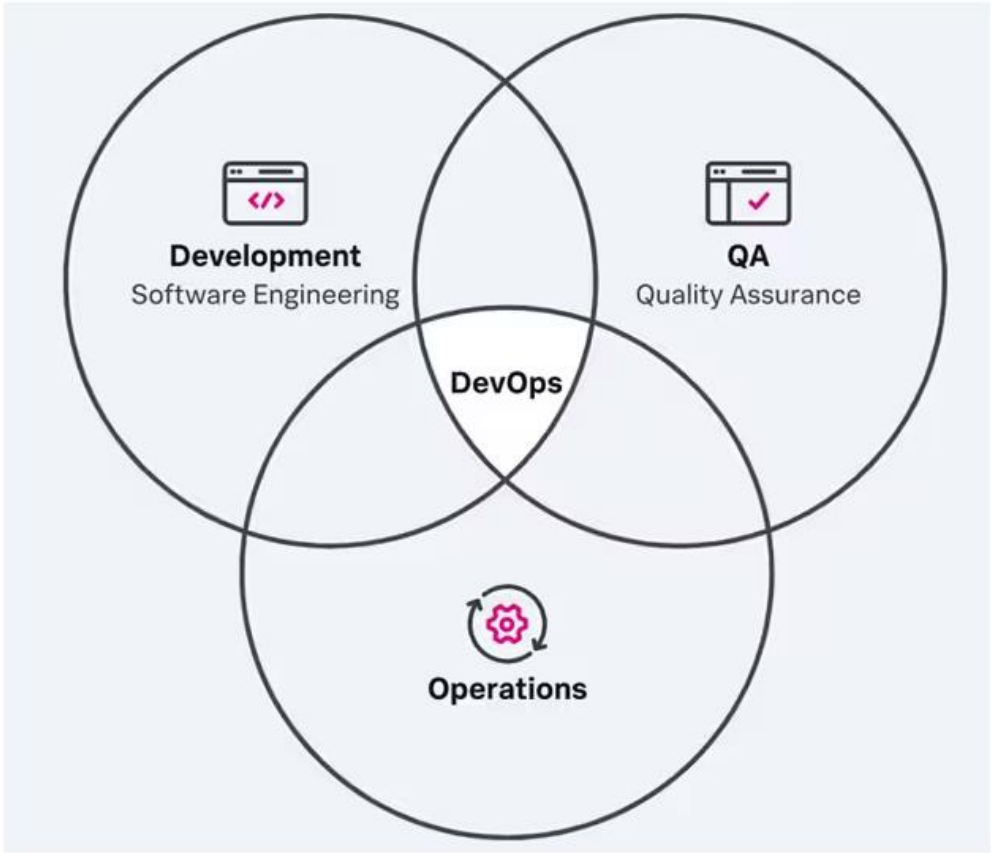
7- **Cloud**: Azure, ACP, AWS

8- **Python**: Langage go

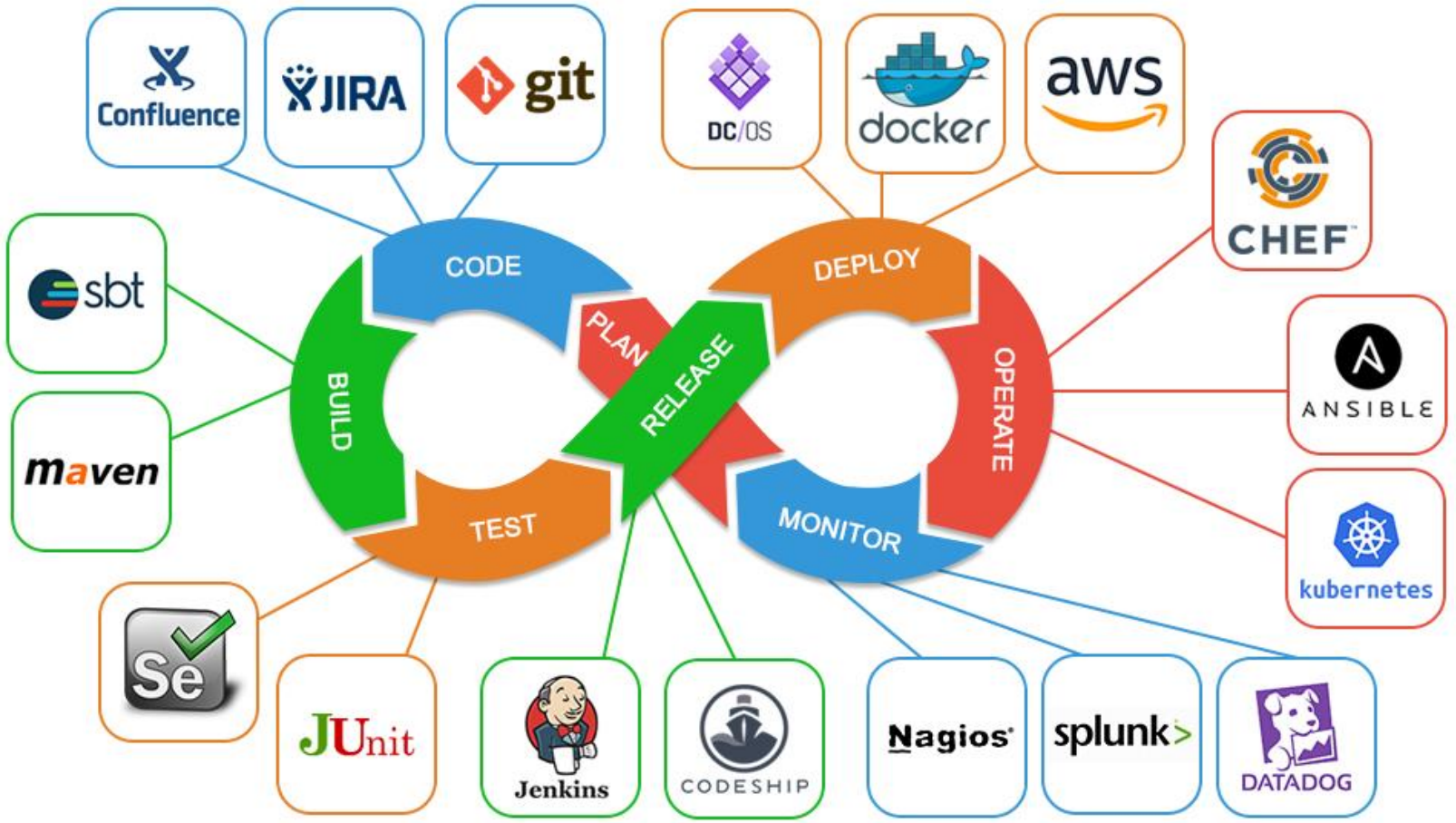
DevOps: CI/CD Pipline



Principaux rôles et responsabilités du DevOps



Création d'un pipeline DevOps



7. Maintenance et évolutions

- Superviser l’application pour détecter les anomalies ou bugs.
- Publier des mises à jour pour améliorer les performances ou intégrer de nouvelles fonctionnalités.
- S'assurer que les contrats intelligents respectent les normes de compatibilité et les mises à jour des plateformes.

7. Maintenance et évolutions

- **Exemple :**
  Après le déploiement, vous surveillez le système pour détecter d'éventuels problèmes (comme des failles de sécurité ou des erreurs de validation des contrats). Vous collectez des retours utilisateurs pour ajouter de nouvelles fonctionnalités, par exemple, l'intégration d'un mécanisme de vote décentralisé pour valider les modifications de la chaîne.

Outils et technologies recommandés :

- **Langages** : Solidity, JavaScript, Python.
- **Frameworks** : Truffle, Hardhat, Remix.
- **Bases de données** : IPFS (pour stockage décentralisé), MongoDB.
- **APIs et outils** : Infura, Alchemy, Ganache.