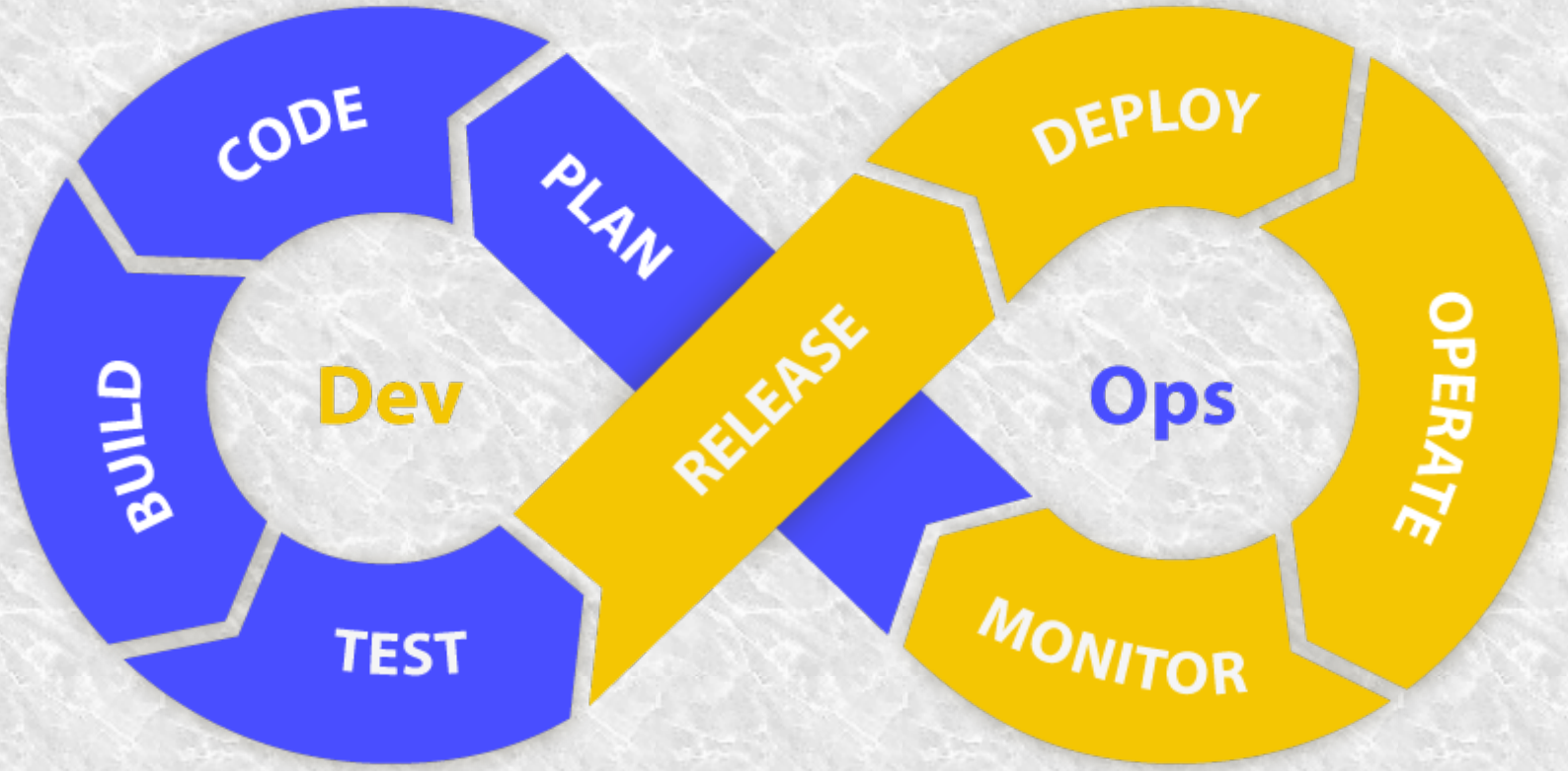
Concepts et outils DevOps



Brahim HAMDI

brahim.hamdi.consult@gmail.com

DevOps, CyberOps, LPIC1/2/3, NCLA, CCNA R&S, MTA-98366, MTA-98367

Novembre 2019

Plan de la formation

- **Module 1 : Développement moderne d'applications**
- **Module 2 : Gestion des versions**
- **Module 3 : Intégration continue**
- **Module 4 : Déploiement et orchestration de conteneurs**
- **Module 5 : Gestion de configuration**

Module 1

Développement moderne d’applications

Plan

- De Agile vers DevOps
- L’architecture à micro-services
- Les API REST
- Plateformes et concepts de données
- Les plateformes cloud
- Les stratégies de déploiement

De Agile vers DevOps

Agile c’est quoi ?

- Méthodologie de gestion de projets de développement complexes.
- Développement itératif, incrémental et évolutif.
- Feed-back client, collaboration, auto-organisation des dev team, cycle de développement rapide, amélioration continue, etc ...
- implémenté via un ensemble de méthodes : scrum, XP, RAD, FDD, Safe, kanban, etc ...
- Scrum : la méthode la plus courante.
  - Découpage de projet en « sprints »
  - Peut être combiné avec autres méthodes (XP par exemple)

De Agile vers DevOps

Limites de l’agilité

- Collaboration limitée aux équipes de dev.
- Feed-back client, mais pas des autres équipes internes.
- Cycle de développement court, mais durée résultante importante.
- Environnement de développement différent de l’environnement de production
  - Problèmes liés à l’environnement de production
  - Conflits avec les équipes d’exploitation (ops).
- L’automatisation n’est pas bien supportée.

De Agile vers DevOps

DevOps c’est quoi ?

- Méthodologie, approche et culture.
- Réunir les équipes de développement et d’exploitation.
- Feed-back des équipes internes.
- Gérer le processus de d’ingénierie logiciel de bout en bout.
- Amélioration continue, Intégration continue, livraison continue, déploiement continue, supervision continue, etc ...
- Compétence de toutes les équipes
- Automatisation partielle ou totale du processus de production.
- Supporte l’agilité (extension d’agile).

De Agile vers DevOps

Agile vs DevOps

Paramètre	Agile	DevOps
Qualité	Bonne application répond aux exigences du cahier des charges. Amélioration et adaptation aux besoins durant cycle de développement.	L'automatisation crée une meilleure qualité. Les développeurs doivent appliquer les bonnes pratiques de codage et suivre l'architecture adéquate.
Automatisation	n'insiste pas sur l'automatisation (bien que cela aide).	L'automatisation est l'objectif principal.
Objectif	Comble le fossé entre les besoins des clients et les équipes de développement et de test	résout l'écart entre développement + tests et opérations.
Se concentrer sur	Besoins fonctionnels et non fonctionnels	Plus sur l'opérationnel et le commercial.

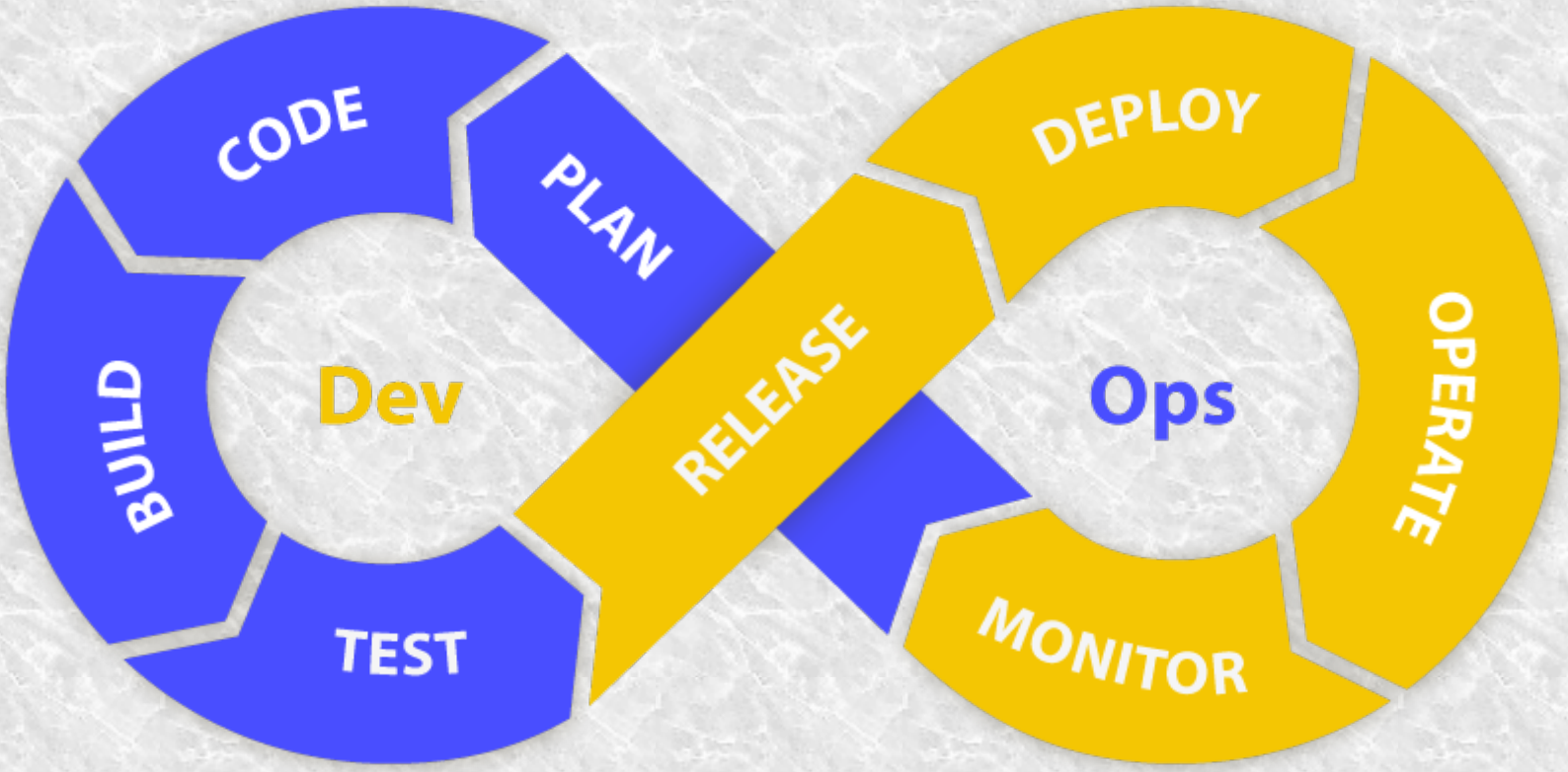
De Agile vers DevOps

Temps de travail

Action	Agile	DevOps
Programmation	60 %	25 %
Tests	20 %	10 %
Intégration	10 %	10 %
Documentation	10 %	15 %
Automatisation des tests	0 %	10 %
Automatisation de déploiement	0 %	10 %
Qualité/Validation	0 %	20 %

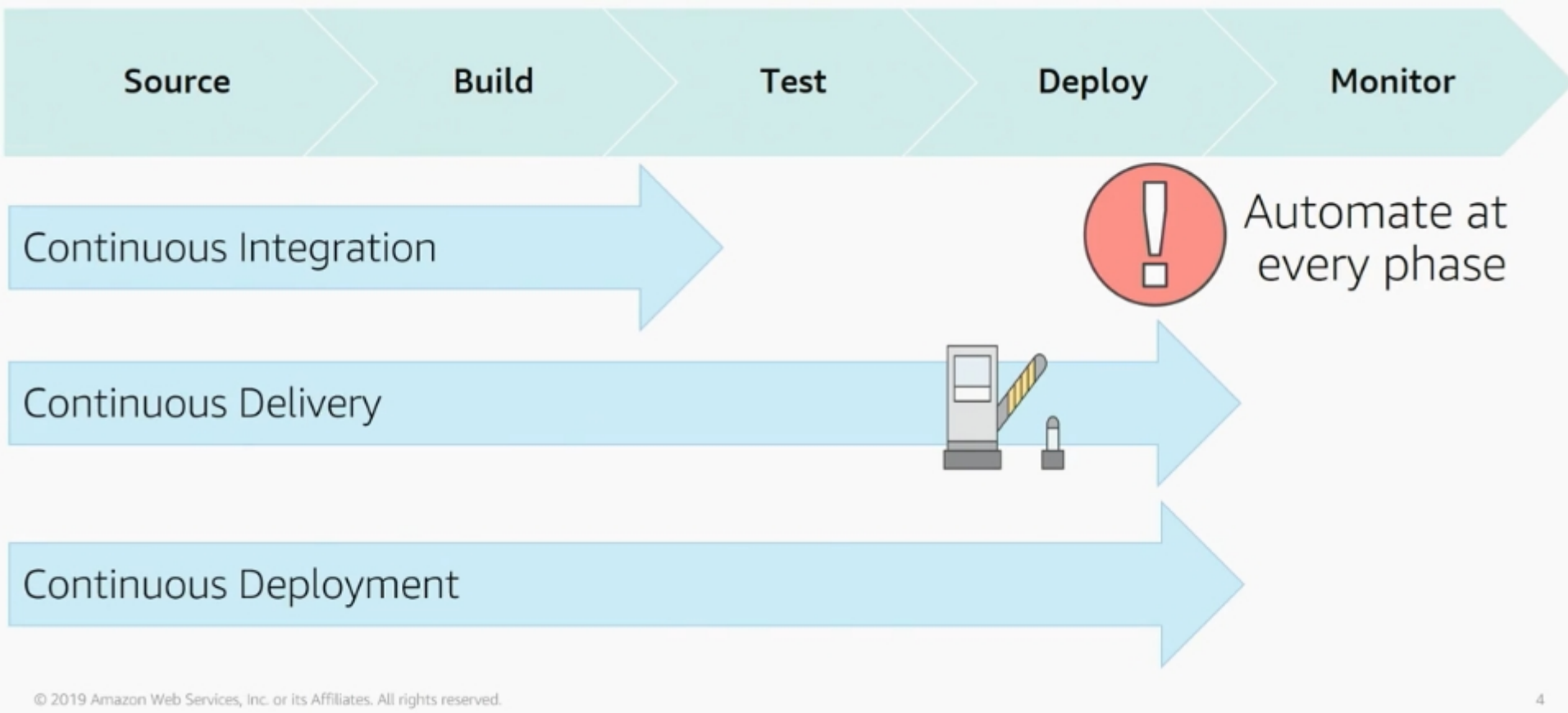
Pipeline CI/CD

Cycle de vie d’un logiciel



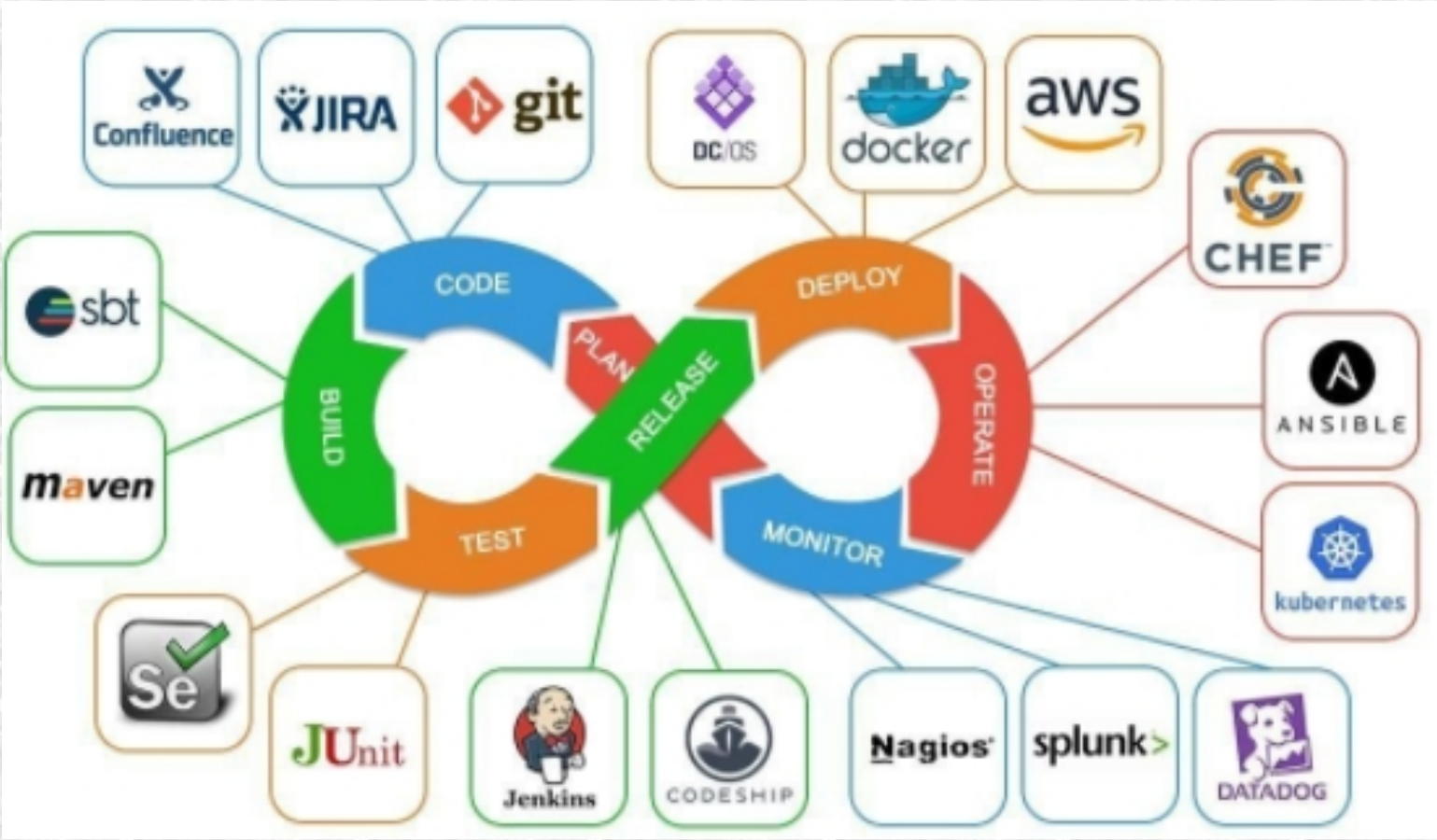
Pipeline CI/CD

Qu’est ce que le CI/CD



Pipeline CI/CD

Outils DevOps



Architecture microservices

Architecture des applications

- Importance de l'architecture des applications
  - Evolution
  - Distribution
  - Speed to market
- Architectures des applications :
  - Architecture monolithique
  - Architecture SOA
  - Architecture de microservices
  - Architecture hybride

Architecture microservices

Architecture monolithique

- Synonyme à l'architecture n-tiers.
- Trois couches
  - Couche présentation
  - Couche logique de traitement
  - Couche d'accès aux données
- Problèmes massifs de couplage
  - Chaque compilation, test et déploiement
    - Simple changement de code : réserver des ressources pour l'ensemble de l'application.
  - Problème de performance lié à une partie ⇒ Problème de performance sur toute l'application.

Architecture microservices

Architecture SOA

- Service-based architecture
- Découper l’application en petits modules
- Bon moyen de découplage et de communication
- Séparer les éléments internes et externes du système.
- Les services communiquent à travers le bus ESB.
  - Plus de composants ajoutés au système ⇒ ESB de plus en plus grand.

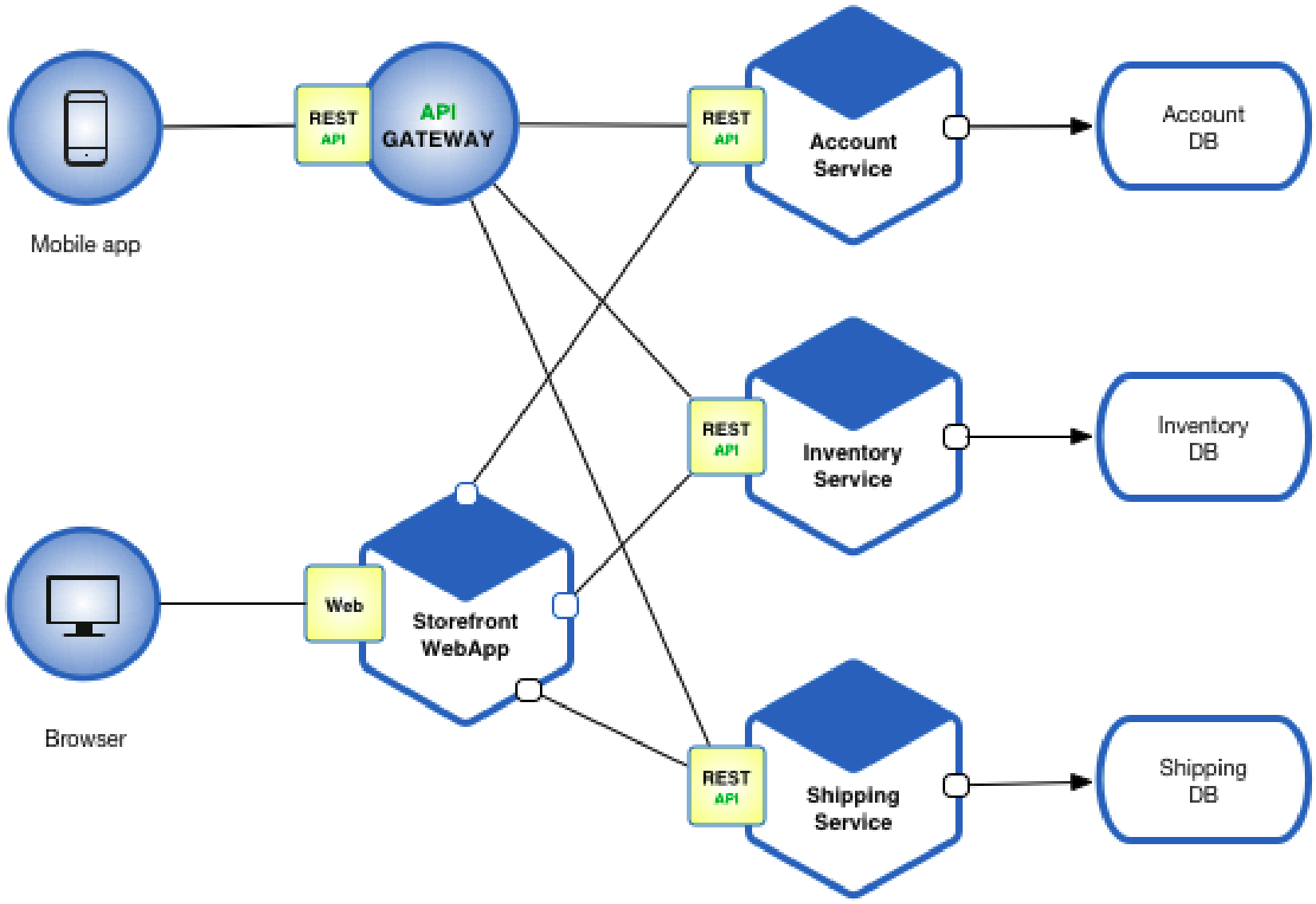
Architecture microservices

Architecture à microservices

- Extension de SOA.
- Description fine des services ⇒ les microservices
- Apporte une série de bonnes pratiques :
  - Déploiement quotidien
  - Réduire « time de market » à quelques jours
  - Interconnexion entre applications internes et externes
- Des langages et technologies différentes dans la même architecture.
- Communication via des API REST
- Bien adaptée au cloud natif.

Architecture microservices

Architecture microservices



API REST

Qu’est ce qu’une API?

- Application Program Interface
- Les APIs sont partout
- Contrat fournit d’une application à une autre
- Requête et réponse structurée

API REST

Qu’est ce qu’une REST ?

- REpresentational State Transfer.
- Style d’architecture pour la conception d’applications en réseau
- S’appuie sur un protocole client-serveur sans état, généralement HTTP
- Traite les objets serveur comme des ressources pouvant être crées ou détruites
- Utilisé par n’importe quel langage de programmation.

API REST

Les méthodes HTTP

- GET: Récupérer les données d'une ressource spécifiée
- POST: Soumettre les données à traiter à une ressource spécifique.
- PUT: met à jour une ressource spécifiée
- DELETE: Supprimer une ressource spécifiée
- HEAD: Identique à GET mais ne renvoie pas de corps.
- OPTIONS: Retourne les méthodes HTTP supportées
- PATCH: Mise à jour partielle des ressources

API REST
les Endpoints

- *L'URI/URL où l'API / service peut être accédé par une application cliente.*
- *Exemples:*
  - *GET https://monsite.com/api/users*
  - *GET https://monsite.com/api/users/1*

    ou

    GET https://monsite.com/api/users/details/1
  - *POST https://monsite.com/api/users*
  - *PUT https://monsite.com/api/users OU https://monsite.com/api/users/update/1*
  - *DELETE https://monsite.com/api/users/1*

    ou

    DELETE https://monsite.com/api/users/delete/1

API REST

HTTP : codes d'état

- *1xx Informational response*
  - *100 Continue*
  - *101 Switching Protocols*
  - *102 Processing*
  - *103 Early Hints*
- *2xx Success*
  - *200 OK*
  - *201 Created*
  - *202 Accepted*
  - *203 Non-Authoritative Information*
  - *204 No Content*
  - *205 Reset Content*
  - *206 Partial Content*
  - *207 Multi-Status*
- *3xx Redirection*
  - *301 Moved Permanently*
  - *302 Found (Moved temporarily)*
  - *304 Not Modified*
  - *305 Use Proxy*
  - *307 Temporary Redirect*
  - *308 Permanent Redirect*
- *4xx Client errors*
  - *400 Bad Request*
  - *401 Unauthorized*
  - *402 Payment Required*
  - *403 Forbidden*
  - *404 Not Found*
  - *407 Proxy Authentication Required*
- *5xx Server errors*
  - *500 Internal Server Error*
  - *501 Not Implemented*
  - *502 Bad Gateway*
  - *503 Service Unavailable*
  - *504 Gateway Timeout*
  - *505 HTTP Version Not Supported*

Qu’est ce que JSON?

- JSON : JavaScript Object Notation
- Format léger d’échange de données
- Facile à lire et à écrire pour les humains
- Facile à analyser et à générer par les machines.
- Les réponses du serveur doivent toujours être au format JSON et cohérentes.
- Toujours contenir des méta-données et, éventuellement, des données

API REST

Exemples

- Exemple 1 – Demander la liste des livres :

Protocol: GET

URI: /api/v1/books

Request body: EMPTY

```
{
  meta: {
  },
  data: [{
    id: 24,
    title: 'Gestion de projets Agile',
    author: 'Jim Highsmith'
  }, {
    id: 25,
    title: 'Integration continue',
    author: 'Martin Fowler'
  }]
}
```

API REST

Exemples

- Exemple 2 – Demander un livre :

Protocol: GET

URI: /api/v1/books/id/24

Request body: EMPTY

```
{
  meta: {
  },
  data: {
    id: 24,
    title: 'Agile Project Management'
    author: 'Jim Highsmith'
  }
}
```

API REST

Exemples

- Exemple 3 – Demander la création d'un livre :

Protocol: POST

URI: /api/v1/books

Request body:

```
{
  id: 26,
  title: 'DevOps Handbook',
  author: 'Gene Kim'
}
```

Réponse avec code état 201

```
{
meta: {
  },
  data: {
    uri: /api/v1/books/id/26
  }
}
```

API REST
Exemples

- Exemple 4 – Demander la suppression d'un livre :

Protocol: DELETE

URI: /api/v1/books

Request body:

```
{
  id: 24
}
```

Réponse avec code état 201

```
{
meta: {
  },
  data: {
  }
}
```

Base de données relationnelle

- Basée sur le modèle relationnel de données.
- Utilisent le langage SQL.
- Données organisées dans des tables.
- Chaque ligne d'une table a sa propre clé unique (clé primaire).
- Les lignes d'une table peuvent être liées à des lignes d'autres tables (clé étrangère).
- MySQL (MariaDB), Oracle, Postgres, etc ...

Base de données NoSQL

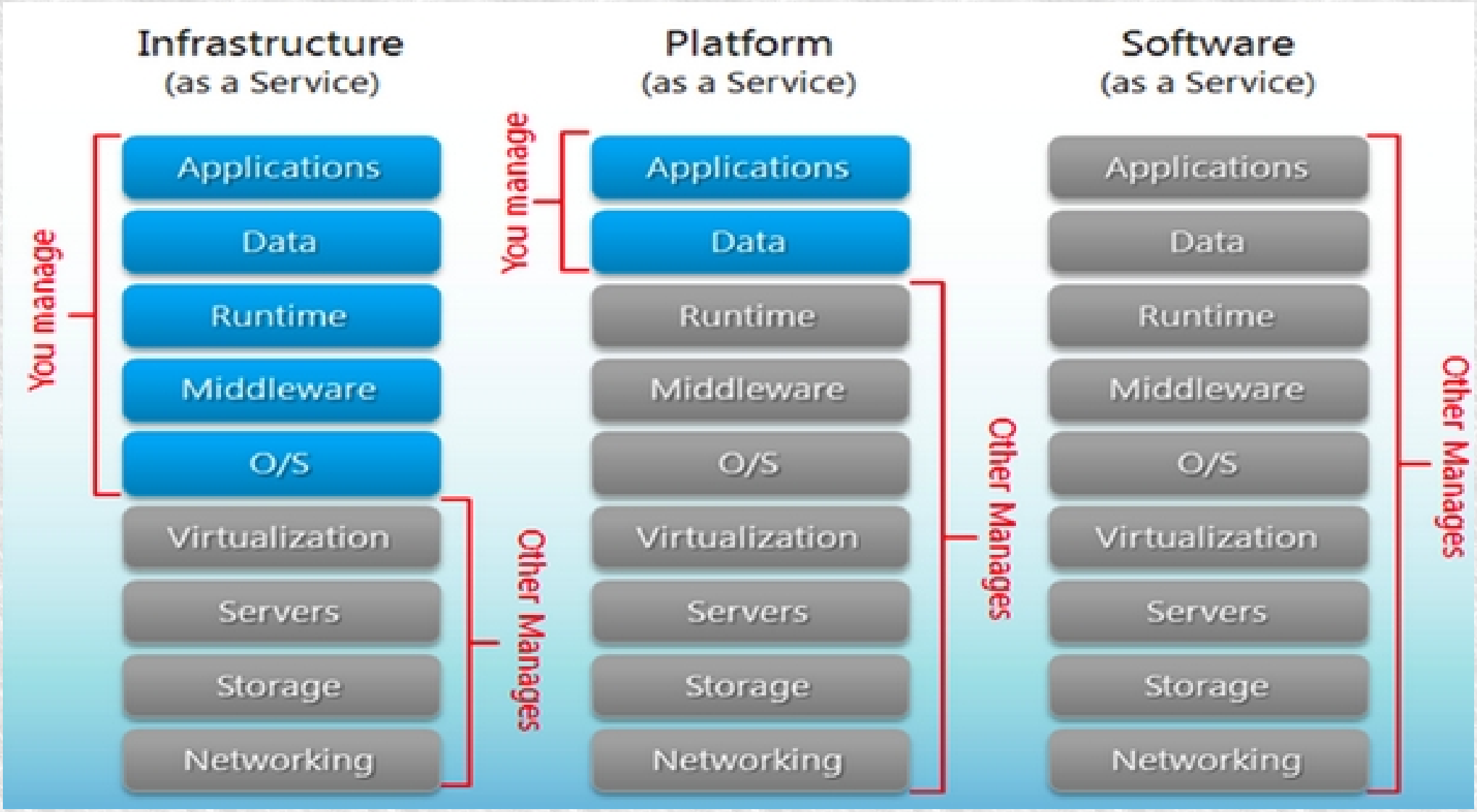
- Autres mécanisme de stockage et récupération de données
- De plus en plus utilisé dans le Big Data et les applications Web temps réel.
- Propriétés :
  - Simplicité de conception
  - Evolution plus simple en cluster de machines (problème pour les bases de données relationnelles)
  - Un contrôle plus fin sur la disponibilité.
  - Certaines opérations plus rapides (que la base de données relationnelle)
- Exemples de bases de données NoSQL:
  - MongoDB (très populaire)
  - Cassandra (Facebook, Linkedin, Netflix)
  - DynamoDB (Amazon)

Stockage objet

- Un objet est composé de :
  - La donnée
  - L’Identifiant unique : chemin d'accès de la donnée.
  - Les métadonnées : Type de données (vidéo, fichier, etc...), sa structure, etc.
    - faire le lien entre plusieurs objets qui comportent ces mêmes métadonnées.
- Avantages :
  - Coût
  - Facilement scalables
  - Interaction via des API REST.
- Usage :
  - Données web statiques (images, vidéos, mail, etc …)
  - Données de sauvegardes
  - Systèmes d’archivage

Les plateformes cloud

Services cloud



Les plateformes cloud

Solutions IaaS

- Solutions privées
  - Openstack
  - Opennebula
  - Cloud Stack
  - Eucalyptus
- Fournisseurs publiques
  - Amazon Web Services (EC2)
  - Microsoft Azure (virtual machine)
  - Google Cloud

Les plateformes cloud

Solutions PaaS

- AWS Lambda
- Google Cloud Functions
- Azure Web Apps
- Oracle Cloud PaaS
- OpenShift
- Cloud Foundry
- Etc ...

Stratégies de déploiement cloud

Les stratégies les plus courantes :

- Déploiement « Blue/Green » :
  - mise en place de deux infrastructures parallèles - l'ancien et le nouvel environnement.
  - transition d'utilisateurs de l'ancien au nouveau, après une période de test bêta.
- Déploiement « Canary »
  - Identique au modèle de déploiement Blue/Green,
  - le test Canary exécute en même temps le nouveau et l'ancien code.
  - Cependant, un petit groupe d'utilisateurs ont accès à la nouvelle version pour effectuer des tests en direct.

Module 2

Gestion des versions

Plan

- Git
- Commandes de base
- Les branches Git
- Les dépôts distants

Introduction

- SCM : Système de gestion de version
- Permet de maintenir et gérer toutes les versions d'un ensemble de fichiers.
- Pourquoi un système de gestion de version ?
  - Revenir aisément à une version précédente.
  - Suivre l'évolution du projet au cours du temps.
  - Permettre le travail en parallèle sur des parties disjointes du projet et gérer les modifications concurrentes.
  - Faciliter la détection et la correction d'erreurs.

Types de SCM

- SCM centralisé
  - Structure, gestion et utilisation simples
  - Très sensible aux pannes
  - Inadapté aux très grands projets
  - Exemples : CVS, SVN (Subversion)
- SCM distribué
  - Moins sensible aux pannes
  - Adapté aux très grands projets
  - Gestion et utilisation plus compliqués
  - Exemples : Git, Mercurial

Git

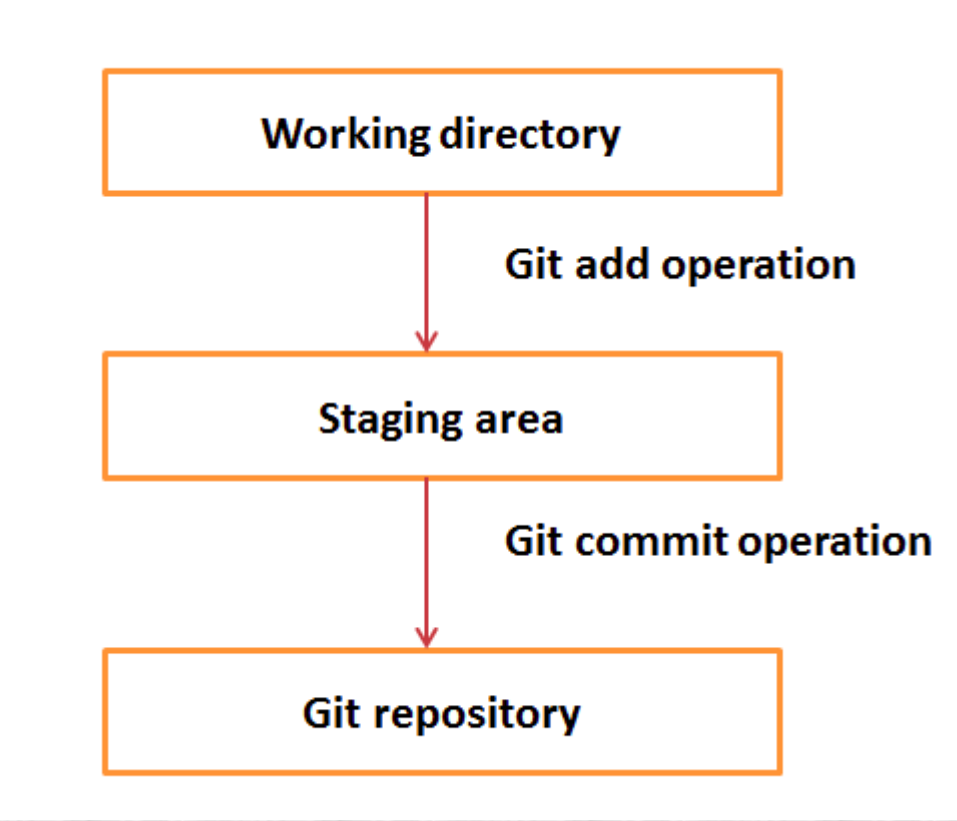
- Système de gestion de versioitns distribué
- Historique
  - 1991 – 2002 : Noyau Linux développé sans utilisation d’un SCM.
  - 2002 – 2005 : La communauté utilise BitKeeper (DVCS propriétaire)
  - 2005 : BitKeeper retire la possibilité d’utiliser gratuitement son produit.
  - Après quelques mois de son développement : Git héberge le développement du noyau Linux.

Les dépôts

- Dépôt Git : une sorte de système de fichiers, enregistrant les versions de fichiers d'un projet.
  - A des moments précis au cours du temps
  - Sous forme d'instantannés.
- Deux types de dépôts :
  - Dépôt local : sur la machine du développeur
  - Dépôt distant : sur un serveur distant (GitHub, GitLab, etc …)

Git
le dépôt local

- 3 sections d'un projet Git :
  - Le répertoire de travail
    - Extraction unique d'une version du projet depuis la base de de données du dépôt.
  - La zone de transit / index
    - Simple fichier contenant des informations à propos de ce qui sera pris en compte lors de la prochaine soumission.
  - Le dépôt Git
    - Dossier « .git »
    - Contient les métadonnées et la base de données des objets du projet



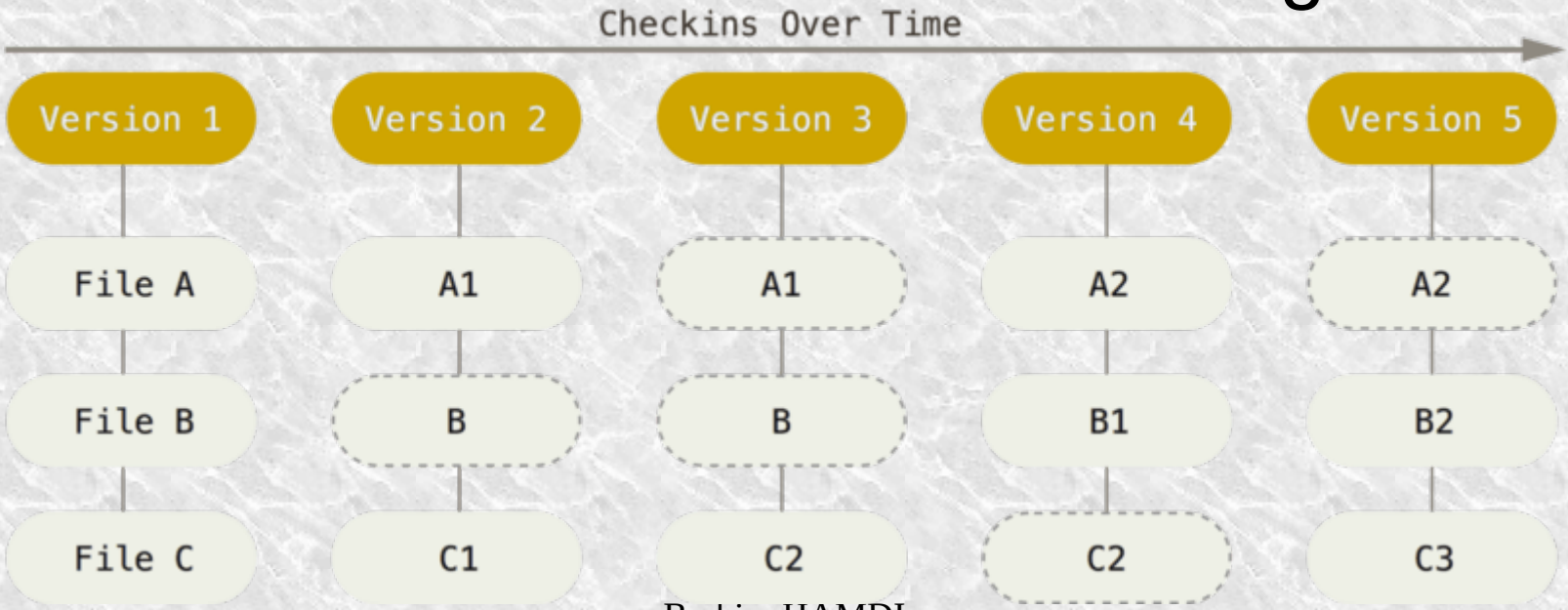
Etat d’un fichier

- 4 états d’un fichier dans Git :
  - Non versionné : Fichier n'étant pas ou plus géré par Git ;
  - Non modifié : Fichier sauvegardé de manière sûre dans sa version courante dans la base de données du dépôt ;
  - Modifié : Fichier ayant subi des modications depuis la dernière fois qu'il a été soumis ;
  - Indexé (staged) : idem, sauf qu'il en sera pris un instantané dans sa version courante lors de la prochaine soumission (commit).

Git

Les commits

- A chaque commit, Git prend un instantané (snapshot) de l'espace de travail.
- Il enregistre une référence à chaque snapshot.
- Git ne stocke pas le fichier à nouveau, mais mais crée une référence vers le fichier d'origine



Gestion de l'intégrité

- Impossible de perdre des données ou corrompre un fichier sans que Git puisse le détecter.
- Tout objet Git doit être vérifié par une somme de contrôle avant d'être enregistré.
- SHA-1 est utilisée pour calculer la somme de contrôle.
- L'intégrité de données fait partie de la philosophie de Git.
- Les hash sont aussi utilisés pour identifier les objets Git.

Commandes de base

Initialiser et afficher l’état d’un dépôt

- Avant tout :

  $ git config --global user.name "Brahim Hamdi"

  $ git config --global user.email brahim.hamdi.consult@gmail.com

- Initialiser un dépôt

  $ git init

- Afficher l'état des fichiers du répertoire courant

  $ git status

  - Untracked les : Fichiers non versionnés.
  - Changes to be committed : modifications (ajout, suppression, changements) chargées en zone de transit (staging area), ou indexées.
  - Changes not staged for commit : modifications n'ayant pas été chargées en zone de transit (ou indexées).

Commandes de base

Indexer et annuler les modifications

- Indexer l'ajout ou les changements d'un fichier

  $ git add [-p] <fichier>

- Annuler les modifications indexées d'un fichier

  $ git reset <fichier>

- Annuler les modications non encore indexées d'un fichier

  $ git checkout [--] <fichier>

- Indexer la suppression d'un fichier

  $ git rm <fichier>

- Déversionner un fichier

  $ git rm --cached <fichier>

Commit et détails des modifications

- Afficher le détail des modifications non indexées

  $ git diff

- Afficher le détail des modifications indexées

  $ git diff --staged

- Soumettre les modifications indexées en zone de transit

  $ git commit

- Voir l'historique des soumissions

  $ git log

Les branches

C’est quoi et pourquoi ?

- Une branche est une ligne d'évolution du projet
- Elle diverge de la ligne d'évolution courante
- Les branches se poursuivant indépendamment l'une de l'autre.
- Une branche dans Git est tout simplement un pointeur vers un objet « commit ».
- Utilisées pour :
  - Permettre à un développeur de travailler sur une fonctionnalité indépendante du projet.
  - Pouvoir tester différentes implémentations d'une même fonctionnalité de manière indépendante.
  - Ayant toujours la capacité de revenir à une version stable que l'on peut continuer à maintenir indépendamment.

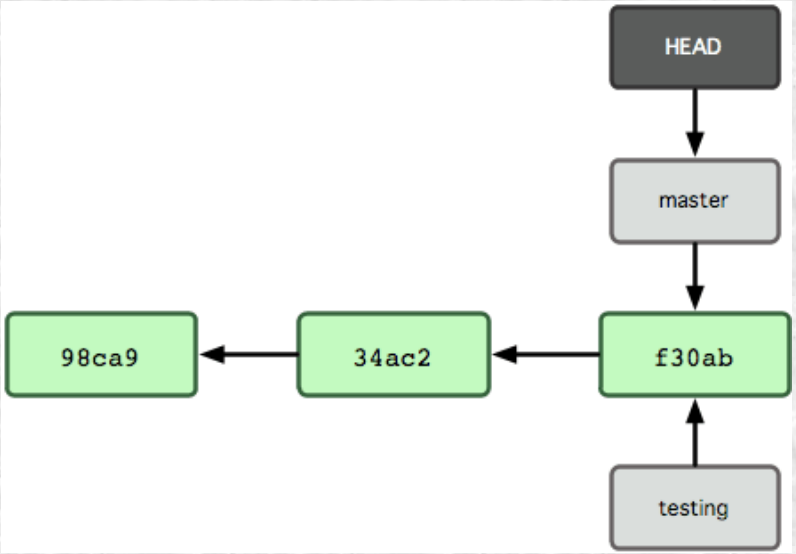
Les branches

Créer une branche

- Par défaut, il en existe une seule branche master.
- Créer une nouvelle branche

  $ git branch <branche>

- HEAD est un pointeur spécial vers la branche sur laquelle on travaille actuellement (extraite dans le répertoire de travail).



Gerer les branches

- Voir les branches du dépôt local

  $ git branch

- Supprimer une branche

  $ git branch -d <branche>

- Passer à une branche donnée (mise à jour de l'index et du répertoire de travail, ainsi que du pointeur HEAD)

  $ git checkout <branche>

Fusionner les branches

- Fusionner les modications d'une branche donnée dans la branche courante (HEAD)

  $ git merge <branche>

- Si l'objet commit pointé par <branche> est déjà un ancêtre de l'objet commit courant (HEAD), alors rien n'est fait.
- Si l'objet commit pointé par <branche> est un descendant de l'objet commit courant, seul le pointeur de la branche courante est déplacé sur l'objet commit concerné par la fusion.

Résolution de conflits

- En cas de conit empêchant la fusion :
  - Aucun objet commit de fusion n'est créé, mais le processus est mis en pause.
  - *git status* donne les fichiers n'ayant pas pu être fusionnés (listés en tant que unmerged).
  - Git ajoute des marqueurs de résolution de conflits à tout fichier sujet à conflits afin que ceux-ci puissent être résolus à la main.
  - Pour marquer les conflits dans un fichier <fichier> comme résolus, il faut faire *git add <fichier>* .
    - On peut, après résolution de tous les conflits, soumettre les modifications sous forme d'objet commit de fusion avec *git commit* et terminer ainsi le processus de fusion.

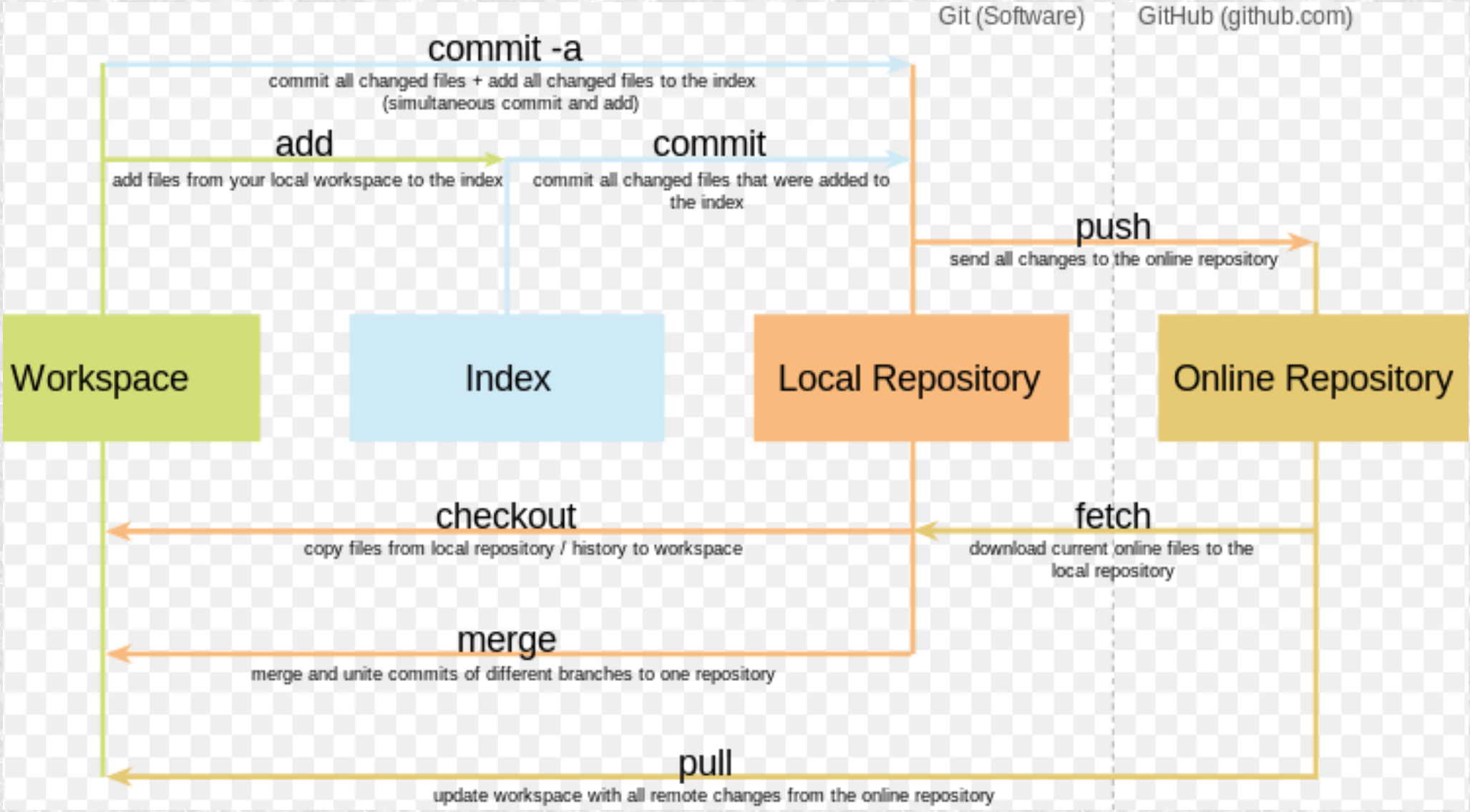
Les dépôts distants

Pourquoi et comment ?

- Pour collaborer, il est nécessaire de communiquer et d'échanger avec un ou plusieurs dépôts distants hébergeant le même projet.
  - typiquement des dépôts publics associés à une personne, une équipe ou tout le projet.
- Les données des dépôts distants (objets commit et instantanés) sont entièrement copiées dans le dépôt local
  - pour chaque branche <branche> d'un dépôt distant <dépôt> est maintenue une branche locale <dépôt>/<branche> non modifiable
  - permettant de suivre la position de <branche> sur <dépôt> localement.

Les dépôts distants

Commandes



Les dépôts distants

Commandes

- Cloner un dépôt distant (automatiquement nommé origin)

  $ git clone <URL> [<répertoire>]

- Récupérer les modications d'un dépôt distant

  $ git fetch <dépôt>

- Ajouter un dépôt distant

  $ git remote add <nom> <URL>

- Mettre à jour un dépôt distant donné pour une certaine branche locale

  $ git push <dépôt> <branche>

  - Attention : Fonctionne uniquement si le dernier objet commit de la branche concernée du dépôt distant a été récupéré et intégré dans la branche en question du dépôt local

- Combiner git fetch et git merge

  git pull <dépôt> <branche>

- Combiner git fetch et git rebase

  git pull --rebase <dépôt> <branche>

Module 3
Intégration continue

Plan

- Présentation de l’intégration continue
- Ordonnancement avec Jenkins
- Outils d’intégration continue

Présentation de l'IC
Intégration continue

- Ensemble de pratiques utilisés afin de s'assurer que chaque modification ne produit pas des problèmes de régression.
- Principal but : Identifier rapidement les bugs avant la mise en production du logiciel.
- Vision plus complète sur les points faibles et points forts du code et de l'équipe.
- Gagner en réactivité face aux problèmes pouvant être présents dans les différentes phases du projet.

Présentation de l'IC

Fonctionnement l'IC

- Repose sur la mise en place d'une brique d'outils pour automatiser plusieurs tâches.
  - Compilation automatique du code
  - Tests unitaires et fonctionnels,
  - Validation du produit en fonction de plusieurs critères,
  - Tests de performances
    - pour procéder à certaines optimisations
- Tout au long du projet, cette brique va exécuter un ensemble de tâches et de tests.
- Les résultats consultables par l'équipe de développeurs
  - pour comprendre ce qui pose problème dans les dernières modifications de code.

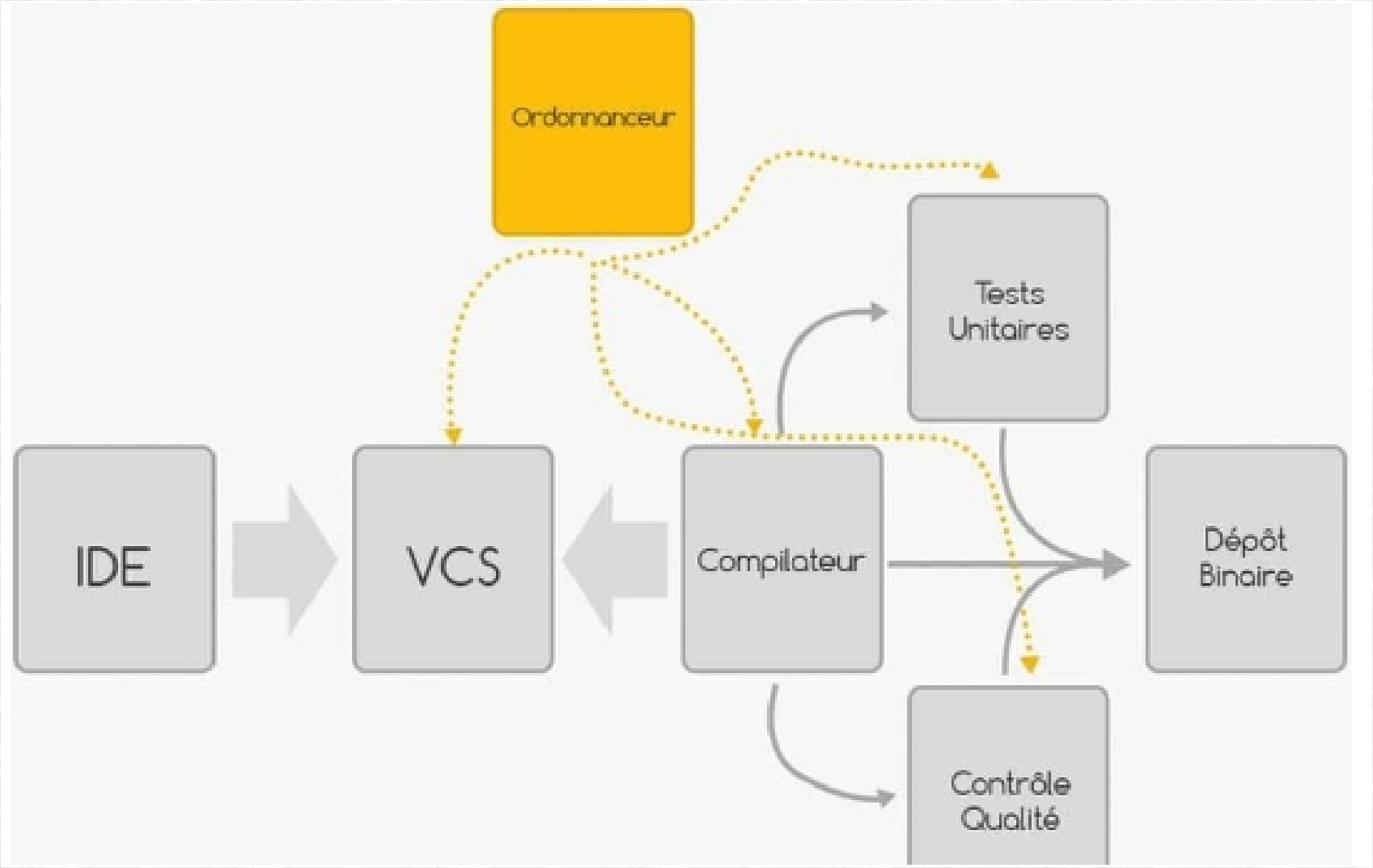
Présentation de l'IC

Etapes de l'IC

- Le développeur exporte les modifications du code vers le serveur SCM (VCS).
- L'ordonnanceur constate qu'une nouvelle version est disponible
- Il lance une compilation sur l'une des machines prévues à cet effet.
- Le code étant compilé, des tests unitaires sont lancés sur des machines de tests.
- En parallèle la qualité du code est vérifiée.
- Si la qualité du code et les tests unitaires sont satisfaisants, le(s) nouveau(x) binaire(s) est (sont) envoyé(s) dans le dépôt.
  - Ils peuvent être déployés par la suite dans d'autres environnements
- En cas d'échec, une notification est générée au chef de projet et/ou à l'équipe de développement

Présentation de l'IC

Etapes de l'IC



Composants d'une plateforme d'IC

- L'ordonnanceur :
  - ordonne les tâches pour qu'elles se réalisent dans le bon ordre.
  - Remonte également les erreurs aux utilisateurs.
  - Jenkins est le plus utilisé
- Le gestionnaire de code source (VCS) :
  - L'ensemble des sources sous différentes versions.
  - Git est le CVS classique
- Le compilateur :
  - Traduire le code source en langage compréhensible par la machine, ou par un interpréteur.
  - Maven est le plus utilisé pour le code java.
- Les tests unitaires :
  - Effectuer l'ensemble des tests unitaires définis dans un projet.
  - JUnit pour le code java
- Le contrôle qualité :
  - Chargé d'assurer la qualité du code en fonction de plusieurs paramètres.
  - L'outil le plus classique est SonarQube
- Le dépôt de binaire :
  - permet le stockage des différentes versions d'un projet après compilation.
  - Il peut aussi servir comme source pour des composants utilisés dans son propre projet.
  - Nexus peut servir comme moyen de dépôt binaire

Ordonnancement avec Jenkins

Jenkins

- Outil d'intégration continue
- Open source développé en java
- Tester et rapporter les changement en temps réel
- Intégrer et automatiser toutes les étapes de cycle de développement du logiciel

  ⇒ Le pipeline
- l'IC est assurée par le biais des plugins.
- Nécessite l'installation des plugins des outils utilisés.

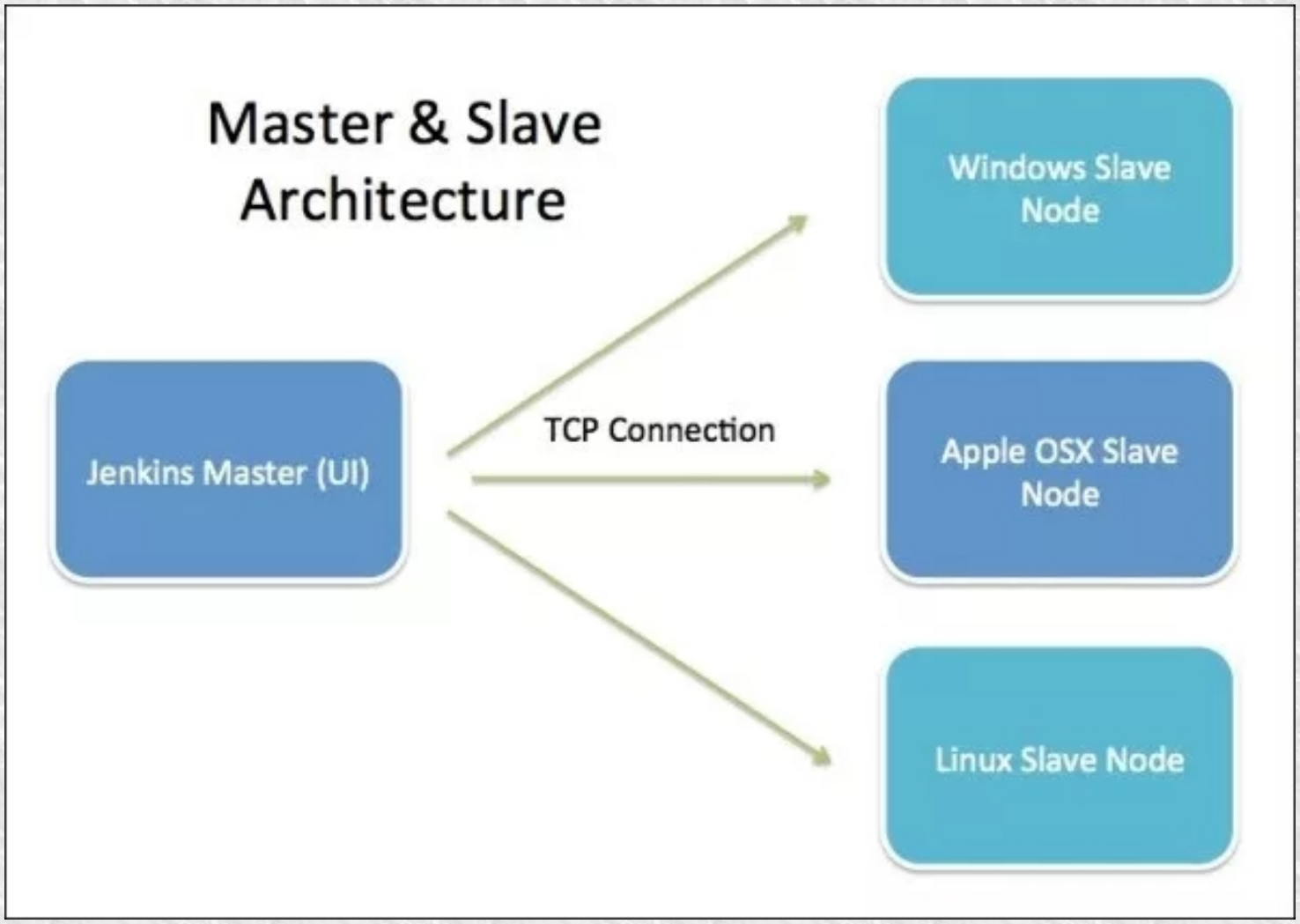
Ordonnancement avec Jenkins

Jenkins - architecture

- Jenkins propose une architecture distribuée
  - Appelée aussi master/slave
- Permet de répartir les builds sur des machines
  - Peuvent être des OS différents
- Jenkins peut exécuter le même test en parallèle.
  - Ce qui aide à récupérer rapidement les résultats désirés.
- Les résultats des jobs sont collectés et combinés sur le master.

Ordonnancement avec Jenkins

Jenkins - architecture



Ordonnancement avec Jenkins

Jenkins pipeline

- Jenkins 2 est livré avec une fonctionnalité appelée Pipelines
  - Très extensible lorsqu'on doit définir un environnement d'intégration continue.
- Un pipeline est un moyen de définir certaines étapes de Jenkins à l'aide de code.
  - Pour automatiser le processus de déploiement de logiciels.
- Il utilise un DSL (Domain Specific Language) avec deux syntaxes différentes:
  - Pipeline déclaratif
  - Pipeline scripté

Ordonnancement avec Jenkins

Exemple

```
def gitUrl = 'git://github.com/jenkinsci/job-dsl-plugin.git'
job('PROJ-unit-tests') {
    scm {
        git(gitUrl)
    }
    triggers {
        scm('*/15 * * * *')
    }
    steps {
        maven('-e clean test')
    }
}

job('PROJ-sonar') {
    scm {
        git(gitUrl)
    }
    triggers {
        cron('15 13 * * *')
    }
    steps {
        maven('sonar:sonar')
    }
}
```

```
job('PROJ-integration-tests') {
    scm {
        git(gitUrl)
    }
    triggers {
        cron('15 1,13 * * *')
    }
    steps {
        maven('-e clean integration-test')
    }
}

job('PROJ-release') {
    scm {
        git(gitUrl)
    }
    // no trigger
    authorization {
        // limit builds to just Jack and Jill
        permission('hudson.model.Item.Build', 'jill')
        permission('hudson.model.Item.Build', 'jack')
    }
    steps {
        maven('-B release:prepare release:perform')
        shell('cleanup.sh')
    }
}
```

Outils d'intégration continue

Maven

- Apache Maven : outil de gestion de projet java.
- Permet l'automatisation des tâches suivantes :
  - Téléchargement des dépendances
  - Compilation
  - Exécution des tests
  - Compression du code compilé (jar, war, zip, …)
  - Déploiement de ces artefacts
- Composé d'un noyau léger
  - Les fonctionnalités des projets sous forme de plugins
- Le dépôt (local ou distant) contient des artéfacts :
  - Livrables
  - Dépendances
  - Plugins

Outils d’intégration continue

SonarQube

- Logiciel libre, précédemment appelé Sonar, permettant de mesurer la qualité du code source.
- Supporte plus de 25 langages
- Reporting sur :
  - Identification des duplications de code
  - respect des règles de programmation
  - détection des bugs potentiels
  - évaluation de la couverture de code par les tests unitaires
  - analyse de la répartition de la complexité
  - analyse du design et de l'architecture d'une application
- Analyses entièrement automatisées :
  - intégration avec Maven et serveurs d'intégration continue comme Jenkins.
- Extensible par des plugins pour augmenter les fonctionnalités
  - Checkstyle
  - PMD
  - FindBugs
  - Etc ...

Outils d’intégration continue

Nexus

- Gestionnaire d’objets binaires (logiciels pour les paquets, artefacts et leurs métadonnées).
- Centralise la gestion de ces objets générés par les développeurs internes ou externes.
- Répond à la problématique de complexité liée aux objets binaires
  - diversité des types d'objets binaires
  - leur position dans l'ensemble du flux de travail
  - dépendances entre eux.

Module 4

Déploiement et orchestration de conteneurs

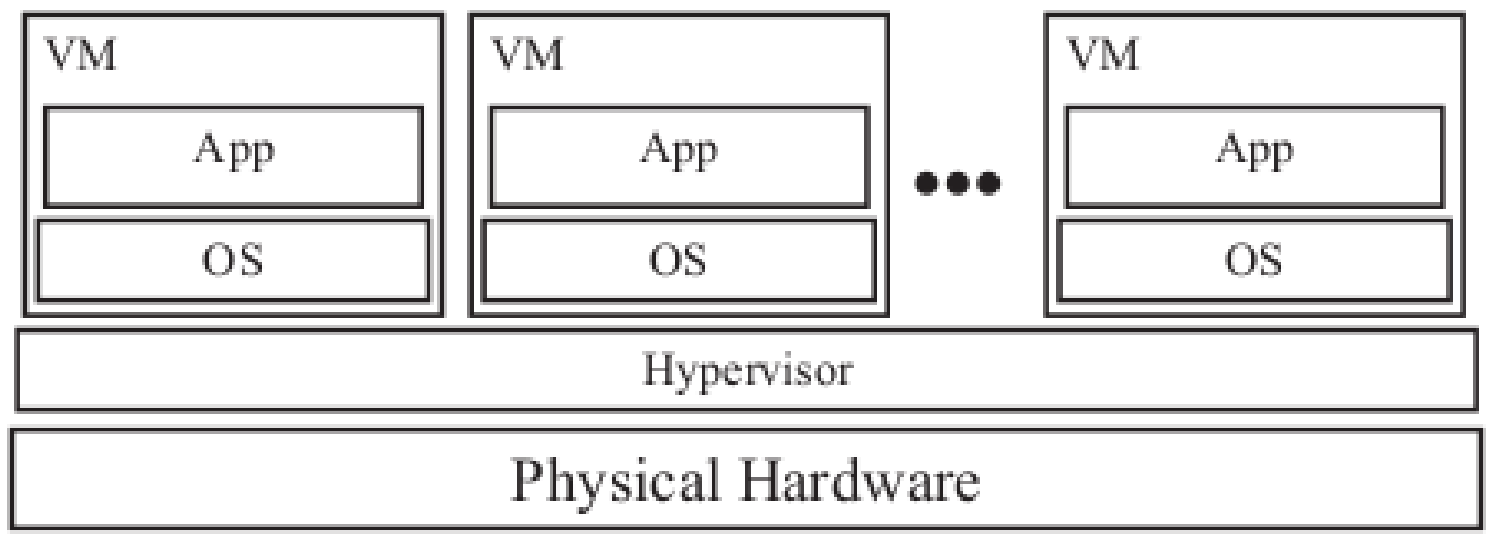
Plan

- Docker et les conteneurs
- Connecter un conteneur au réseau
- Les volumes persistants
- Kubernetes

Advantages de la virtualisation

- Minimiser les coûts
- Flexibilité
- Scalabilité
- Automatisation
- Approvisionnement / administration simplifiés

Inconvénients de la virtualisation



Chaque machine virtuelle nécessite un système d'exploitation (OS)

- Chaque OS nécessite une licence
- Chaque système d'exploitation a ses propres coûts de traitement et de stockage
- Besoin de maintenance, mises à jour

Docker et les conteneurs

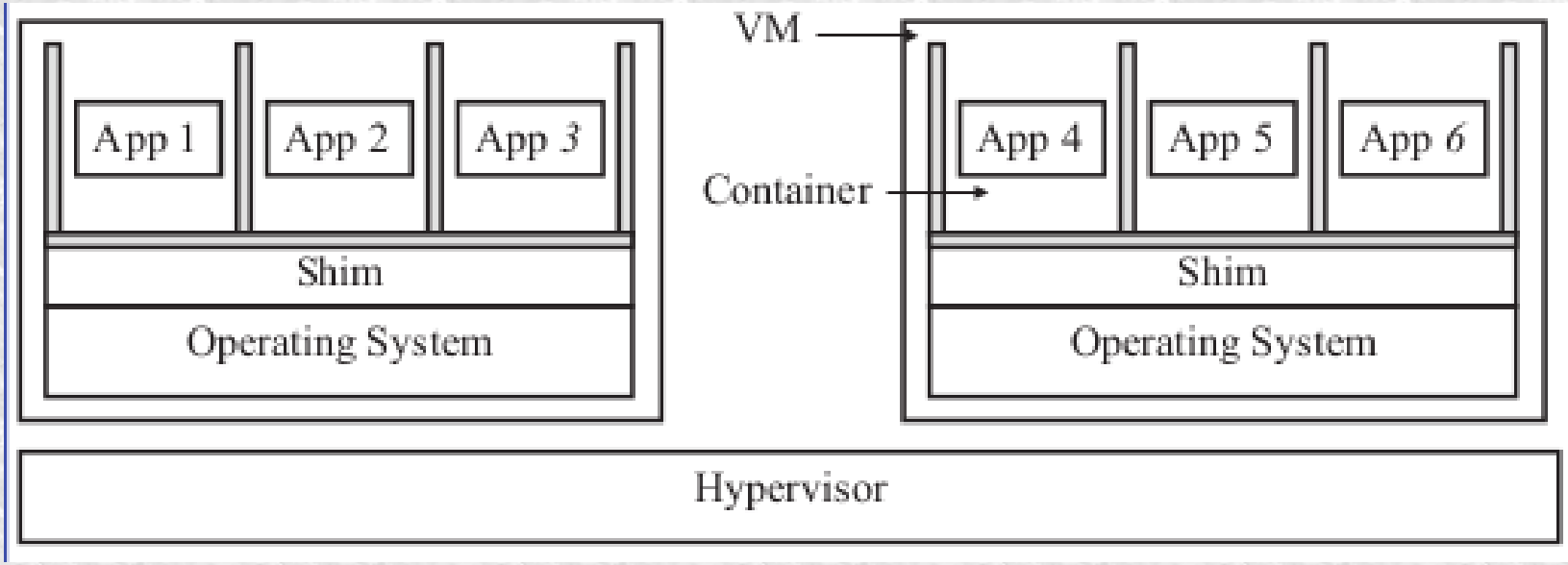
Solution

- Exécuter de nombreuses applications sur la même machine virtuel
  - Les applications partagent l'OS et la charge.
  - Doivent être isolées les une des autres.
  - Pas d'accès aux ressources de chacune sans autorisation explicite.
  - Comme des appartements dans un complexe

⇒ **Conteneurs**

Docker et les conteneurs

Les conteneurs



- En plus des avantages des VM,
- Plusieurs conteneurs s'exécutent sous le même OS d'une machine virtuelle / physique.
- Les conteneurs sont isolés ⇒ ne peuvent pas interférer les uns avec les autres
  - Système de fichiers / données propre, propre réseau ⇒ Portable

Docker et les conteneurs

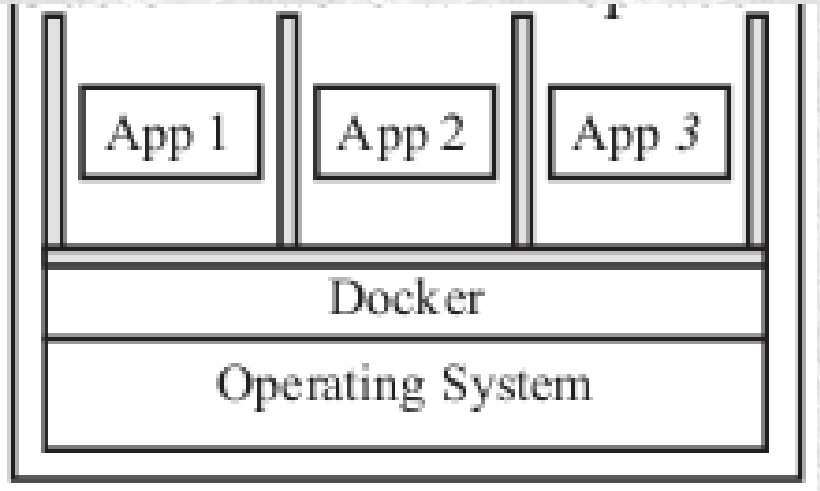
Conteneurs vs VM

Critère	VM	Conteneur
Taille image	3x	x
Temps démarrage	> 10s	~1s
Charge	> 10%	< 5 %
Sécurité	basse-moyenne	moyenne-élevée
Flexibilité OS	Excellent	Pas d'OS

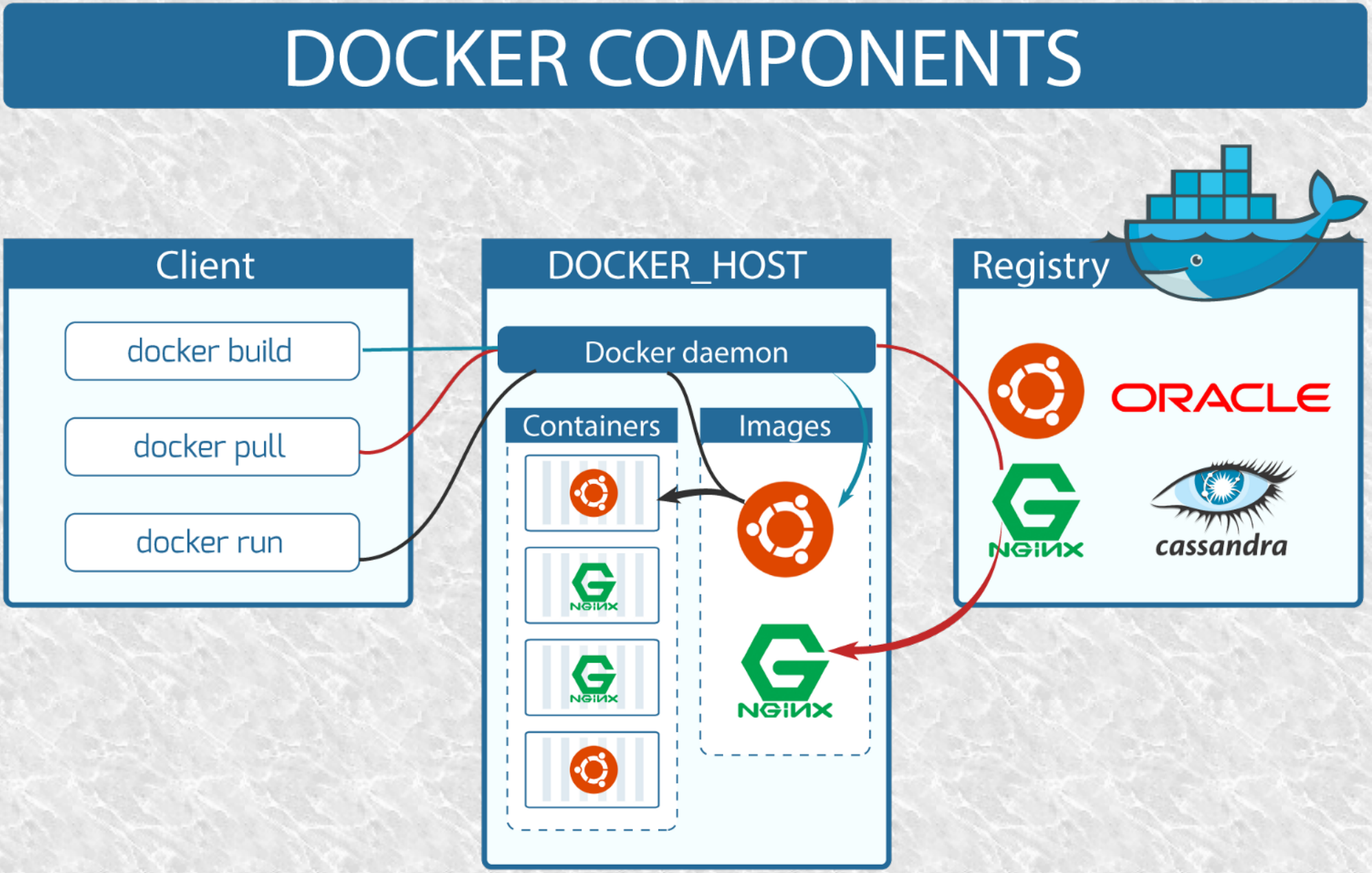
Docker et les conteneurs

Docker

- Fournit l'isolement entre les conteneurs
- Les aide à partager le système d'exploitation
- Docker = Docke Worker (Gérer les conteneurs)
- Développé initialement par Docker.com
- Téléchargeable sous Linux, Windows et Mac



Docker et les conteneurs

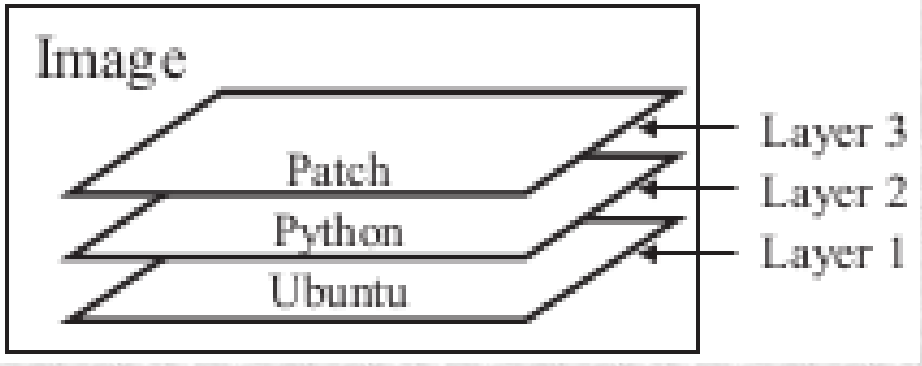


Le Registry et les images

- Les conteneurs sont construits à partir d'images et peuvent être sauvegardés en tant qu'images.
- Les images sont stockées dans des registres
  - Registre local sur le même hôte
  - Registre public : Docker Hub Registry
  - Registre privé
- Tout composant non trouvé dans le registre local est téléchargé à partir d’un emplacement spécifique.
- Docker Hub : Registre officiel de Docker
- Chaque image comporte plusieurs balises, par exemple, v2, dernière, ...
- Chaque image est identifiée par son hachage 256 bits

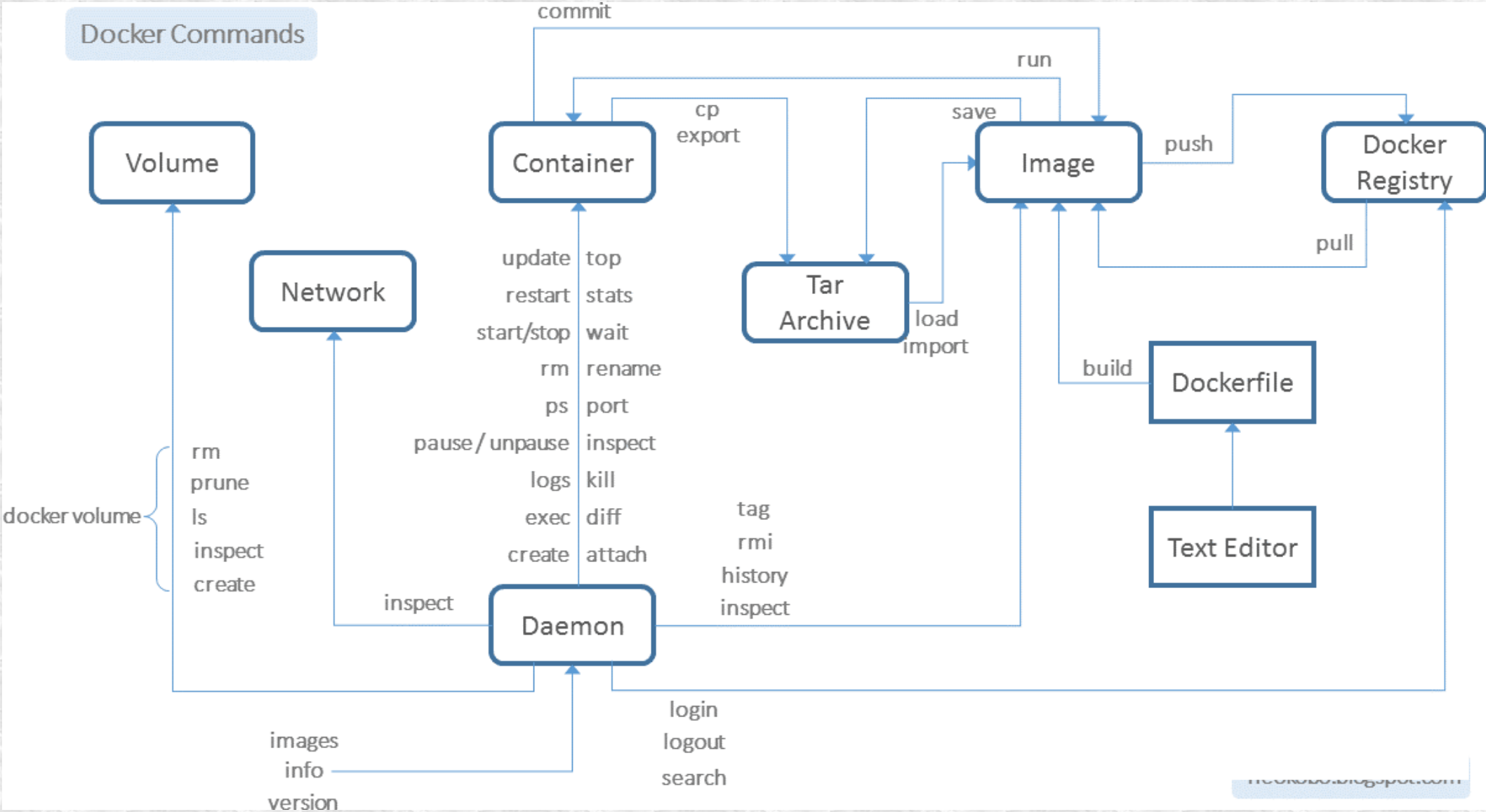
Les couches de l'image

- Chaque image est composée de plusieurs couches
- Elle est construite couche par couche (Dockerfile).
- Chaque couche a son propre hachage sur 256 bits
- Les couches peuvent être partagées entre plusieurs conteneurs
- Exemple :



Docker et les conteneurs

commandes



Exemples

- Créer et démarrer un conteneur :

  docker create -t -i alpine sh

  docker start -a -i 067f4b9e2eda

- Lister toutes les images

  docker images

- Les conteneurs en cours d'exécution

  docker ps

- Tous les conteneurs (arrêtés et en cours d'exécution)

  docker ps -a

Docker et les conteneurs

Exemples

- Informations détaillées sur un conteneur :

  docker inspect 067f4b9e2eda

- Informations sur la configuration réseau :

  docker inspect --format='{{.NetworkSettings}}'\ 067f4b9e2eda

- Adresse IP du conteneur

  *docker inspect --\
  format='{{range .NetworkSettings.Networks}}\{{.IPAddress}}
  {{end}}' 067f4b9e2eda*

- L’adresse MAC :

  *docker inspect --format='{{range .NetworkSettings.Networks}}\
  {{.MacAddress}}{{end}}' 067f4b9e2eda*

Exemples

- Lancer un conteneur en se connectant à son terminal :

  docker run --name alpine1 -it alpine:latest

- Lancer un terminal non attaché au terminal:

  docker run -d nginx:latest

- Executer une commande sur un conteneur :

  docker exec -t alpine1 cat /etc/os-release

- Se connecter un conteneur (avec shell)

  docker exec -it alpine1 sh

Docker et les conteneurs

Exemples

- Arrêter un conteneur :

  docker stop alpine1

- Supprimer un ou plusieurs conteneurs arrêtés :

  docker rm alpine1

- Supprimer tous les conteneurs arrêtés :

  docker container prune

- Supprimer les images non utilisées

  docker image prune

- Envoyer un signal à un conteneur

  docker kill 067f4b9e2eda

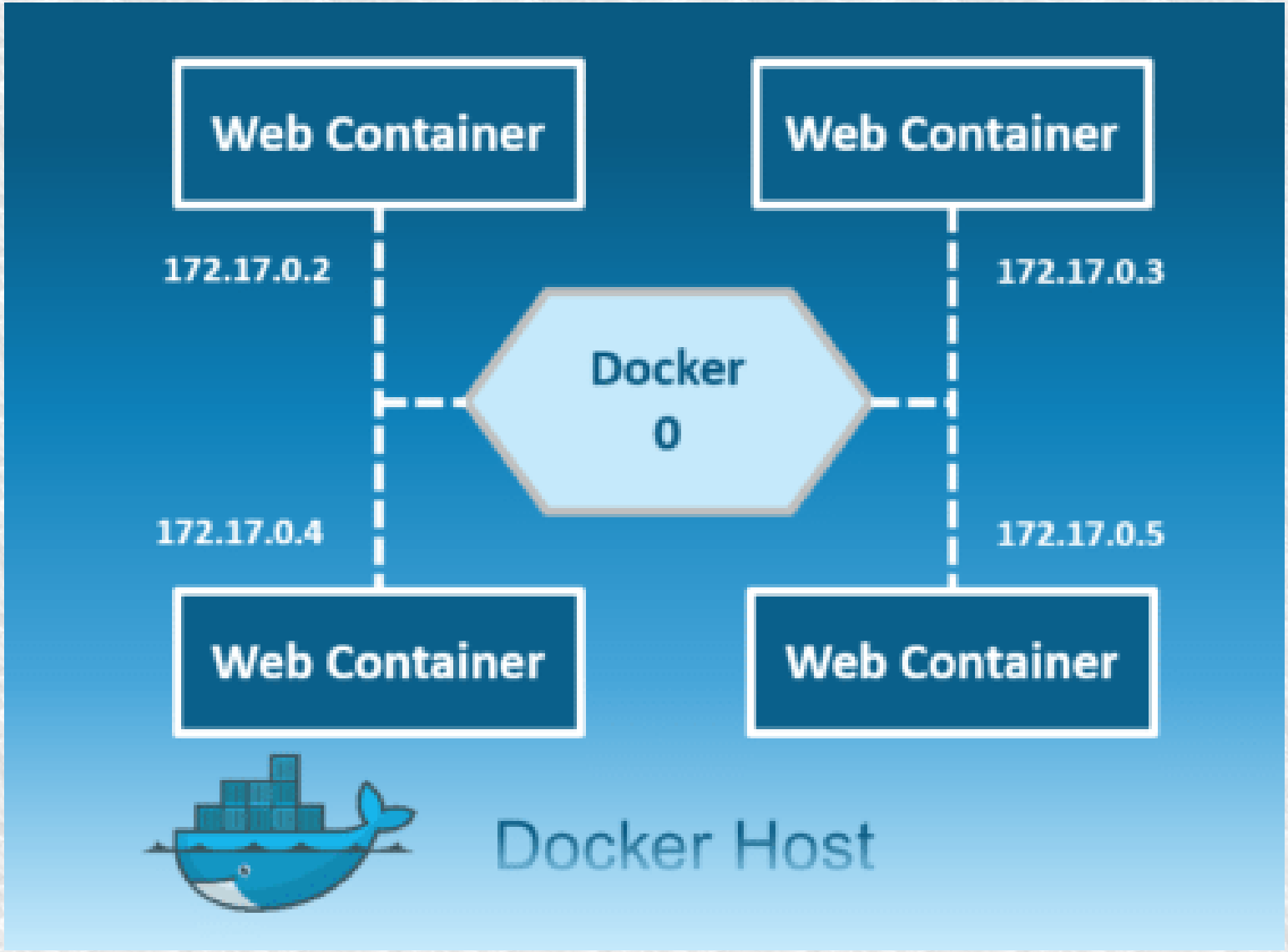
Connecter un conteneur au réseau

Network drivers

- CNM – Container Networking Model : Sous-système réseau de Docker
- Il est basé sur les « Network Drivers ».
- Plusieurs « drivers network » existent :
  - Bridge: le pilote réseau par défaut, au niveau du même OS.
  - Host: Le conteneur prend les mêmes paramètres réseau de l'OS du hôte.
  - Overlay: Connecter des conteneurs de différentes machines du cluster.
  - Macvlan: Attribuer une adresse MAC à un conteneur, le faisant apparaître comme un périphérique physique sur le réseau
  - None: désactivertous les réseaux. Habituellement utilisé avec un « Network Driver » personnalisé.

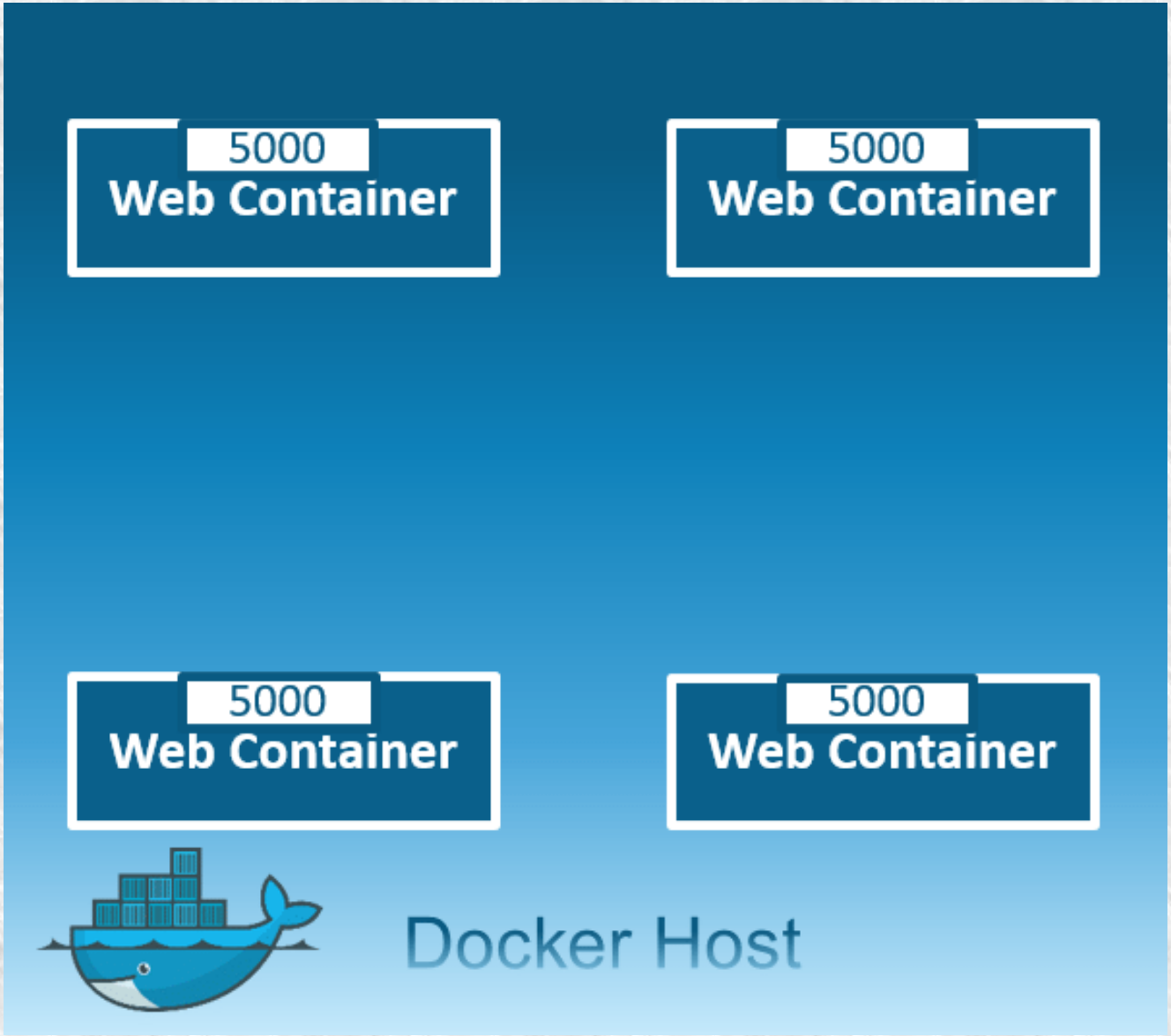
Connecter un conteneur au réseau

Bridge Network



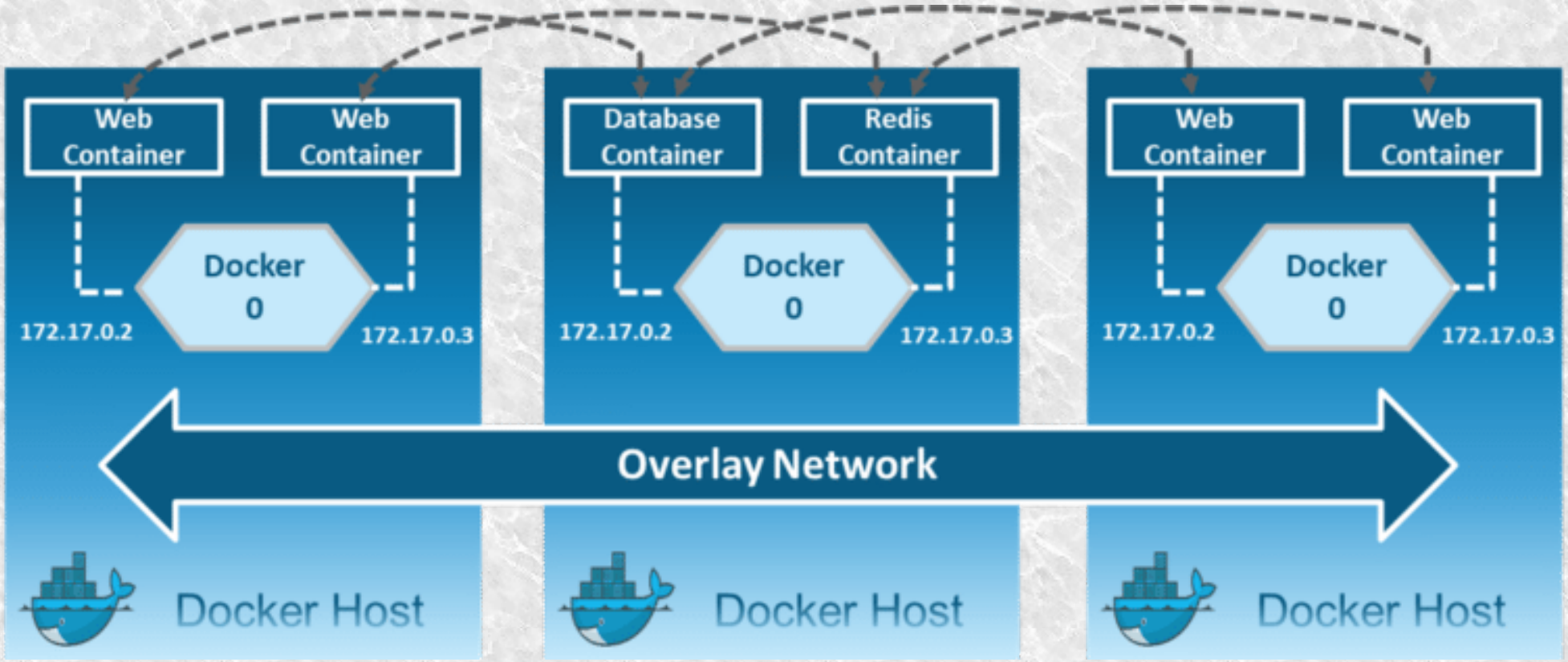
Connecter un conteneur au réseau

Host network



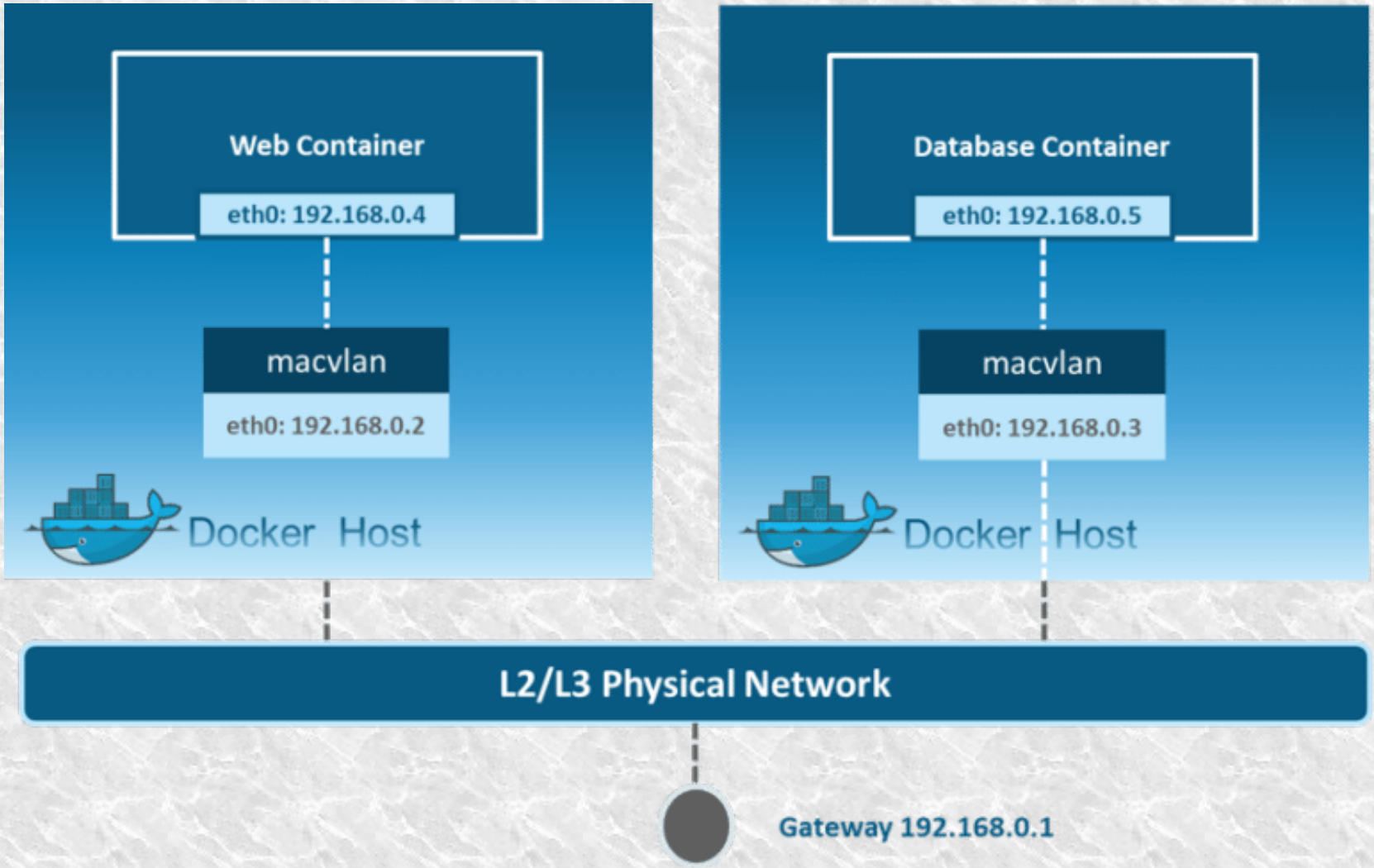
Connecter un conteneur au réseau

Overlay network



Connecter un conteneur au réseau

Macvlan network



Connecter un conteneur au réseau

Exemples de commandes

docker network create my-net

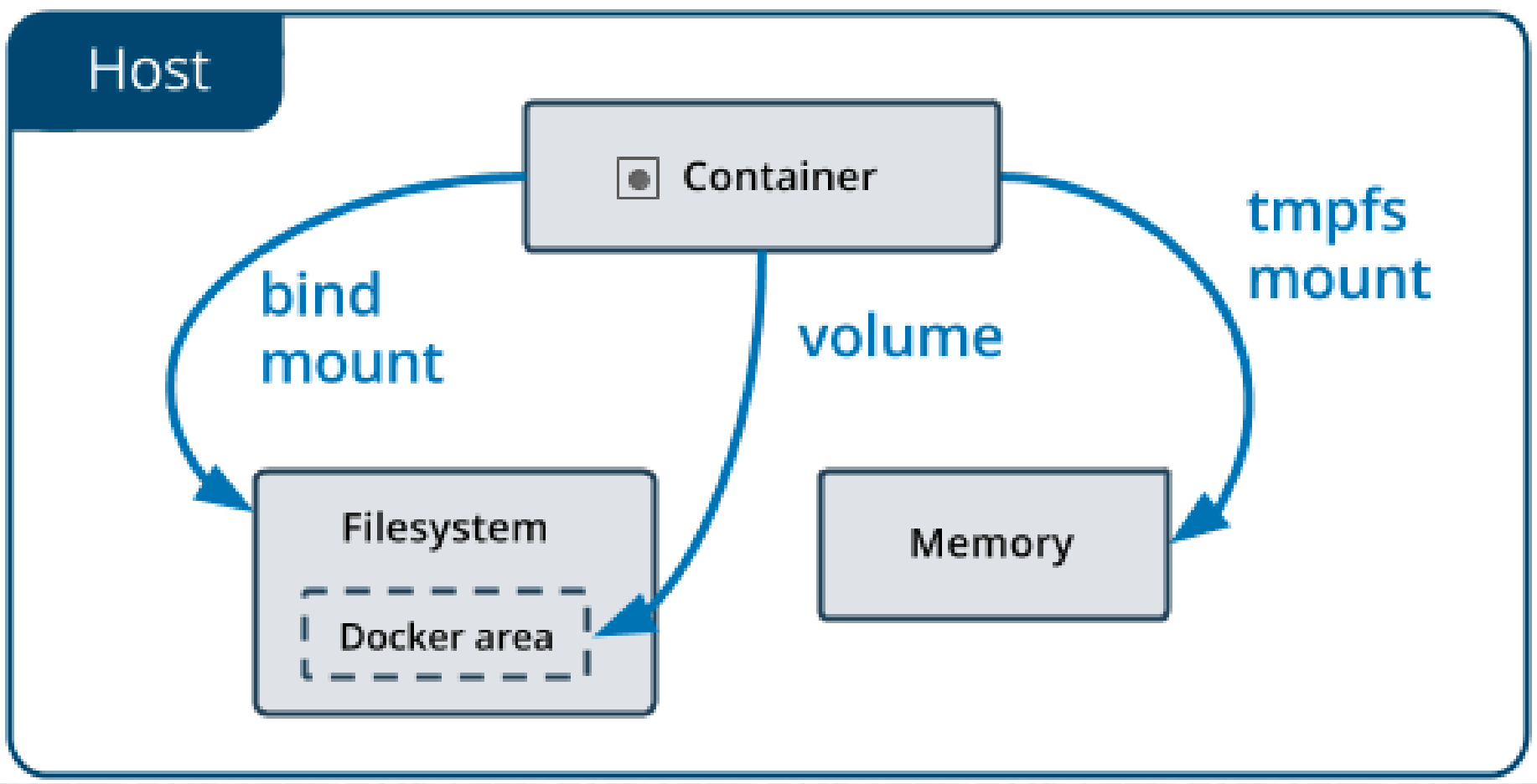
docker network rm my-net

docker create --name my-nginx --network my-net --publish 8080:80 \ nginx:latest

docker network connect my-net my-nginx

docker network disconnect my-net my-nginx

Mécanismes de stockage de docker



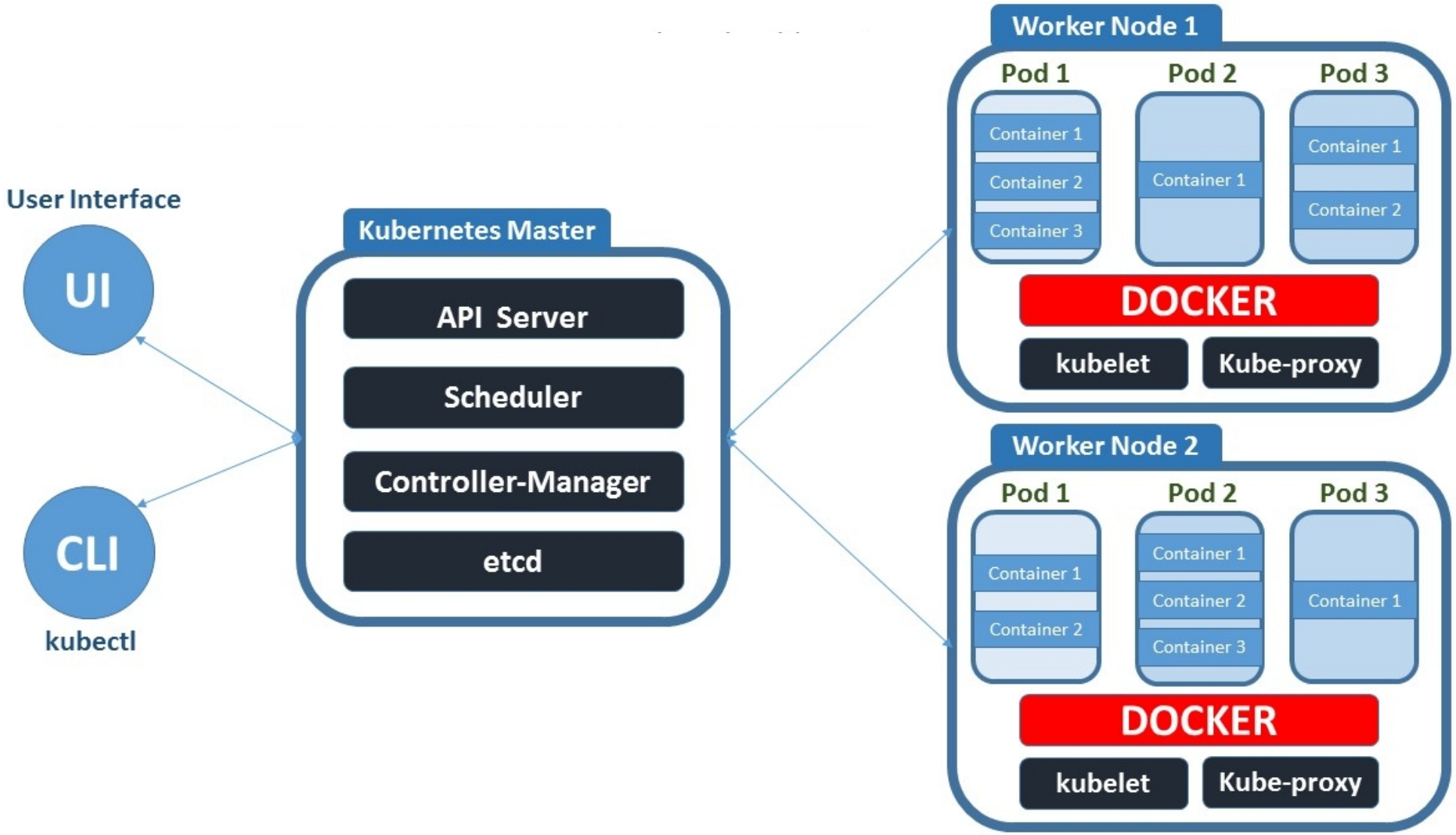
Gestion des volumes

- Volumes créés et gérés par Docker.
- Certains cas d'utilisation de volumes incluent:
  - Partage de données entre plusieurs conteneurs.
  - Stockage de données du conteneur sur un hôte distant ou un fournisseur de cloud, plutôt que localement.
  - Sauvegarder, restaurer ou migrer les données d'un hôte Docker vers un autre.
- Commandes :
  - docker volume create my-vol
  - docker volume ls
  - docker volume inspect my-vol
  - docker volume rm my-vol
  - docker run -v /dbdata --name dbstore2 ubuntu /bin/bash
  - docker run -d --name devtest --mount source=myvol2,target=/app \
    nginx:latest

Qu’est ce que Kubernetes?

- Un projet open source hautement collaboratif conçu à l'origine par Google
- Parfois appelé:
  - Kube
  - K8s
- Démarrer, arrêter, mettre à jour et gérer un cluster de machines exécutant des conteneurs de manière cohérente et maintenable.
- Particulièrement adapté aux architectures applicatives évolutives ou «microservices»
- K8s > (docker swarm + docker-composer)
- Il n'expose pas toutes les "fonctionnalités" la ligne decommande Docker.
- Minikube: un outil facilitant l’exécution locale de Kubernetes.

Kubernetes
Architecture



Kubernetes
commande kubeadm

- kubeadm effectue les actions nécessaires pour obtenir un cluster opérationnel.
- Initialiser K8s sur le nœud Master :
  - *Kubeadm init*
- Joindre un nœud Worker
  - *kubeadm join*
- Annuler toutes les modifications apportées par kubeadm init ou kubeadm join:
  - *kubeadm reset*

kubectl commande kubectl

- Gérer le cluster à partir du nœud Master.
- Syntaxe :

  kubectl [command] [TYPE] [NAME] [flags]

  - Commandes: create, get, describe, delete
  - Type : type de ressource (pod, service, depoyment, node, …)
  - NAME: nom ressource (pod1, node1, etc ...)
  - flags: Spécifier des indicateurs facultatifs

Kubernetes
kubectl - exemples

$ kubectl get pod

$ Kubectl get node -w

$ kubectl get pod -f ./pod.yml

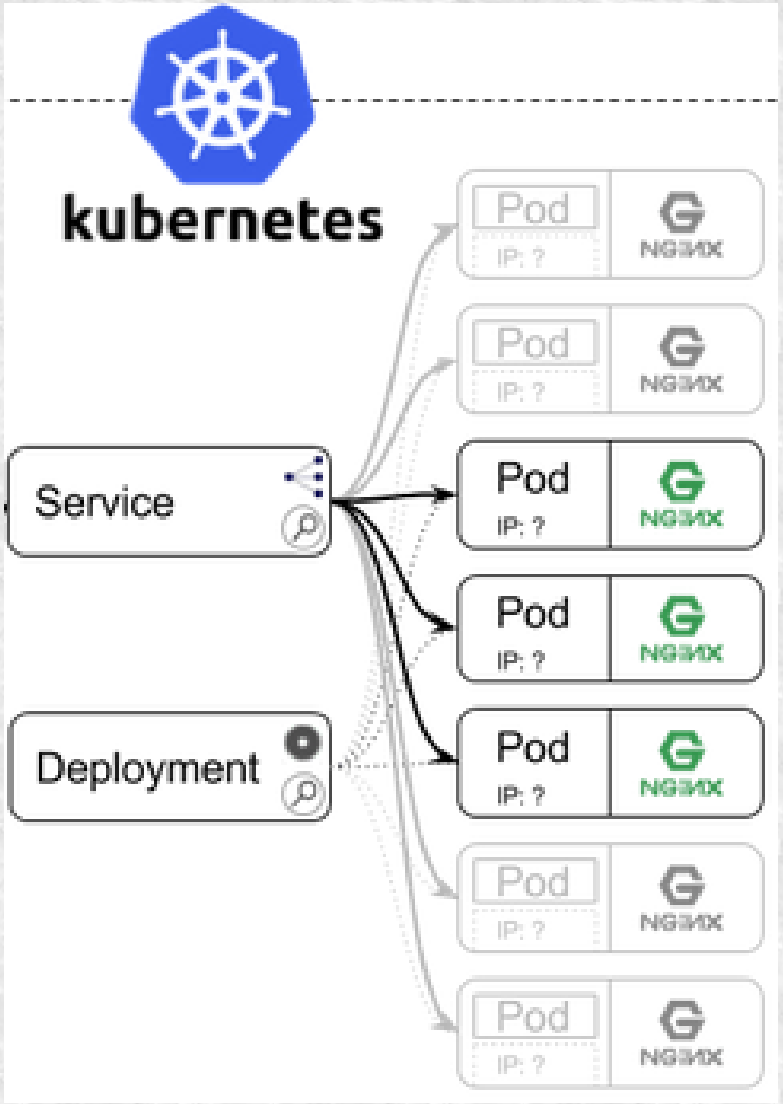
$ kubectl get pod web-pod-13je7 -o=yml

$ kubectl create -f service1.yml

$ kubectl create -f replicaset1.yml

$ kubectl create -f <directory>

Kubernetes
Service-deployment-pod



Kubernetes
Fichiers YAML – pod

```
apiVersion: v1
# Object kind
kind: Pod
metadata:
  name: single-nginx-pod
spec:
  containers:
  - name: nginx
    image: nginx
```

$ *kubectl apply -f pod.yml #* to apply yaml file

Kubernetes
Fichiers YAML – deployment

```
apiVersion: apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: webserver
  template:
    metadata:
      labels:
        app: webserver
    spec:
      containers:
      - name: nginx
        image: nginx
```

Kubernetes
Fichiers YAML – service

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: webserver
  ports:
    - port: 80
      targetPort: 80
```

Module 4

Gestion de configurations avec Ansible

Plan

- Outils de gestion de configuration
- Ansible
- Inventory
- Playbook
- Variables
- Template Jinja2
- Roles
- ansible-vault

Outils de gestion de configuration

Infrastructure as code

- IaC : ensemble de mécanismes permettant de gérer, par des fichiers descripteurs ou des scripts, une infrastructure virtuelle.
- les équipes d'exploitation peuvent gérer et approvisionner l'infrastructure via le code.
- Offre aux développeurs la possibilité d'automatiser leurs déploiements,
  - Eviter toute configuration manuelle
- une réponse aux besoins des entreprises en termes d'automatisation et la simplification des infrastructures des projets informatiques.

Outils de gestions de configuration

Solution : Ansible, Chef, Puppet

	Language	Agent	Configuration	Communication
Ansible	Python	No	YAML	OpenSSH
Chef	Ruby	Yes	Ruby	SSL
Puppet	Ruby	Yes	Puppet DSL	SSL

Outils de gestions de configuration

Pourquoi Ansible

- Agentless!
- Utilise SSH
- Utilise des fichiers YAML (syntaxe simple)
- Push-Based
- Modules intégrés
- Bonne documentation en ligne de commande

Configuration – ansible.conf

- */etc/ansible/ansible.conf*, par défaut
- Exemple :

[defaults]

Inventory = ./inventory

log_path = ./ansible.log

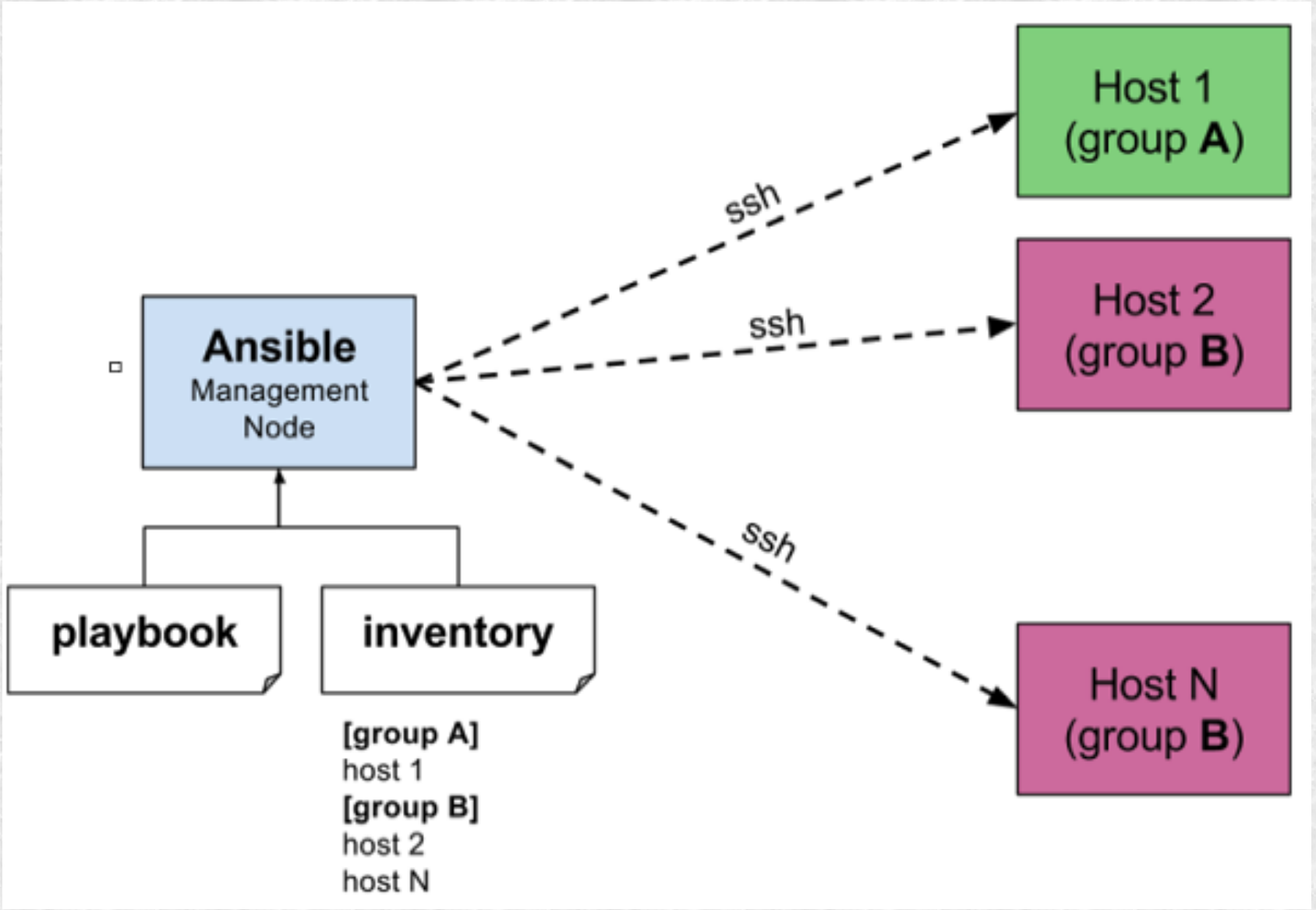
Forks = 5

ask_sudo_pass = True

ask_pass = True

Ansible

Architecture



Inventory

C’est quoi ?

- Liste d’hôtes, groupes et modèles d’hôtes dans /etc/ansible/hosts par défaut.
- Peut être dynamique ou statique
- Groupes définis entre crochets [ ] et par nom
  - Décrire les systèmes
  - Décide quels systèmes contrôler, à quel moment et dans quel but (rôles).
  - Les groupes peuvent être imbriqués avec: children
- Les hôtes peuvent être dans plus d'un groupe
  - Le serveur pourrait être à la fois un serveur Web et un serveur de dbs.

Inventory

Exemple

- **INI**-like version :

```
mail.example.com
[webservers]
foo.example.com
bar.example.com
[dbservers]
one.example.com
two.example.com
three.example.com
```

- **YAML** version :

```
all:
  hosts:
    mail.example.com:
  children:
    webservers:
      hosts:
        foo.example.com:
        bar.example.com:
    dbservers:
      hosts:
        one.example.com:
        two.example.com:
        three.example.com:
```

Inventory
sélection d’hôtes

- La sélection d’hôtes peut être effectuée en incluant/excluant des groupes et des hôtes uniques.
- La sélection peut être faite en passant par :
  - all / *
  - Nom de groupe
  - Exclusion (all:!CentOS)
  - Intersection (webservers:&staging)
  - Regex

Playbook
commandes ad hoc

- Ad-Hoc: Exécuter des commandes simples
- Taches : Tirer parti d'un module Ansible exécuté sur l'hôte cible.
- Modules:
  - Principalement écrits en Python
  - Envoyé via SSH vers l'hôte cible
  - Retour JSON, interprété par Ansible pour le résultat
  - Supprimé une fois exécuté
- Exemples :
  - Suppression de tout le répertoire et des fichiers sur le serveur abc:

    $ Ansible abc -m file -a "dest = /path/user1/new state = absent"
  - Connaître l'état actuel de toutes les machines (Gathering Facts) :

    $ Ansible all -m setup

Orchestration avec les playbooks

- Le vrai pouvoir d'Ansible vient de l'abstraction et de l'orchestration, en utilisant des « playbooks »
- Les playbooks sont les fichiers dans lesquels le code Ansible est écrit (au format YAML).
- Ensemble de tâches ordonnées, combinées avec des cibles sélectionnées.
- Ils fournissent des stratégies toutes faites pour amener des (groupes de) hôtes à un état souhaité.
- Les groupes/hôtes sont définis dans un fichier d'inventaire.
- Pour exécuter un playbook ansible:

$ ansible-playbook playbook_file.yml

Exemple de playbook

```
- name: This is a Play
  hosts: web-servers
  remote_user: brahim
  become: yes
  gather_facts: no
  vars:
      state: present

  tasks:
    - name: Install Apache
      yum: name=httpd state={{ state }}
```

Playbook
Boucles

- Plusieurs types de boucles d'usage général et spécial:
  - with_nested
  - with_dict
  - with_fileglob
  - with_together
  - with_sequence
  - until
  - with_random_choice
  - with_first_found
  - with_indexed_items
  - with_lines

Playbook

Tâches conditionnelles

- Example : Exécuter une tâche uniquement sur RedHat

```
- name: This is a Play
  hosts: web-servers
  remote_user: mberube
  become: sudo

  tasks:
     - name: install Apache
       yum: name=httpd state=installed
       when: ansible_os_family == "RedHat"
```

Playbook
Tags

- Exemple :

```
tasks:
    - yum: name={{ item }} state=installed
      with_items:
          - httpd
          - memcached
      tags:
          - packages
    - template: src=templates/src.j2 dest=/etc/foo.conf
      tags:
          - configuration
```

- Exécuter avec les tags :

```
$ ansible-playbook example.yml --tags “configuration”
$ ansible-playbook example.yml --skip-tags "notification"
```

Variables

Comprendre les variables

- L’automatisation existe pour faciliter la répétitivité
- Problème : Les systèmes ne sont pas tous identiques.
- Le comportement ou l'état des machines configurées peut changer et avoir un impact dynamique sur l'état souhaité des autres services.
- Certains fichiers de configuration peuvent exister en tant que modèles nécessitant une instanciation, en fonction de leur contexte.
- Les variables dans Ansible indiquent comment traiter les différences entre les systèmes et les états.
- Les variables permettent de "programmer" avec des conditions et des boucles.

Réglage des variables

- Les variables peuvent être définies dans plusieurs zones
  - Inventaire
  - Playbook
  - Fichiers et Rôles
  - Ligne de commande
  - Etc ...

Variables d’hôtes

- Les variables hôtes sont attribuées dans l'inventory.
- Il y a aussi des variables qui changent le chemin Ansible se comporte lors de la gestion des hôtes, par exemple:

  90.147.156.175 \

  ansible_ssh_private_key_file=~/.ssh/ansible-default.key \
  ansible_ssh_user=centos

Variables des groupes

- Les hôtes sont regroupés en fonction de leurs aspects ou de groupe souhaité.
- Ansible permet de définir des variables de groupe disponibles pour n'importe quel hôte d'un groupe.
- Les variables de groupe peuvent être définies dans l'inventaire:

```
[webservers:vars]
http_port=80
```

- Ou dans des fichiers séparés sous group_vars

group_vars/webservers → ---

```
http_port=80
```

Variables

Enregistrement et utilisation de variables

- Register : utilisé pour capturer la sortie (résultat) d'une tâche dans une variable.
  - peut ensuite utiliser pour différents scénarios, comme une instruction conditionnelle, la journalisation, etc ...
- Chaque variable enregistrée sera valide sur l'hôte distant où la tâche a été exécutée pour le reste de l'exécution du playbook.
- Exemple

```
- hosts: all
  tasks:
  - name: Ansible register variable basic example
    shell: "find *.txt"
    args:
      chdir: "/Users/mdtutorials2/Documents/Ansible"
    register: find_output

  - debug:
      var: find_output
```

Référencer un champ

- Prend en charge des structures qui mappent les clés à des valeurs.
- Exemple :

  foo:

  field1: one

  field2: two

- Ensuite référencer un champ spécifique de la structure en utilisant:
  - Un crochet : *foo['field1']*

    or
  - Un point : *foo.field1*

Variables prédéfinies

- Certaines variables sont automatiquement créées et remplies par Ansible:
  - inventory_dir
  - inventory_hostname
  - inventory_hostname_s
  - hort
  - inventory_file
  - playbook_dir
  - play_hosts
  - hostvars
  - groups
  - group_names
  - ansible_ssh_user

ansible-vault
C’est quoi ?

- Une fonctionnalité d’ansible qui conserve des données sensibles dans des fichiers cryptés
  - plutôt que du texte en clair dans des playbooks ou des rôles.
- ansible-vault est utilisé pour modifier les fichiers.
- Il peut chiffrer n'importe quel fichier utilisé par Ansible.

ansible-vault

Exemples d'utilisation

- Créer nouveaux fichiers :

  ansible-vault create foo.yml

- Chiffrer le contenu des fichiers existants :

  ansible-vault encrypt foo.yml bar.yml baz.yml

- Déchiffrer le contenu des fichiers

  ansible-vault decrypt foo.yml bar.yml baz.yml

- Editer des fichiers chiffrés

  ansible-vault edit foo.yml

- Régénérer des clés de chiffrement

  ansible-vault rekey foo.yml bar.yml baz.yml

- Create encrypted variables to embed in yaml

  encrypt_string

- Viewing Encrypted Files

  ansible-vault view foo.yml bar.yml baz.yml