

Déploiement d'une application

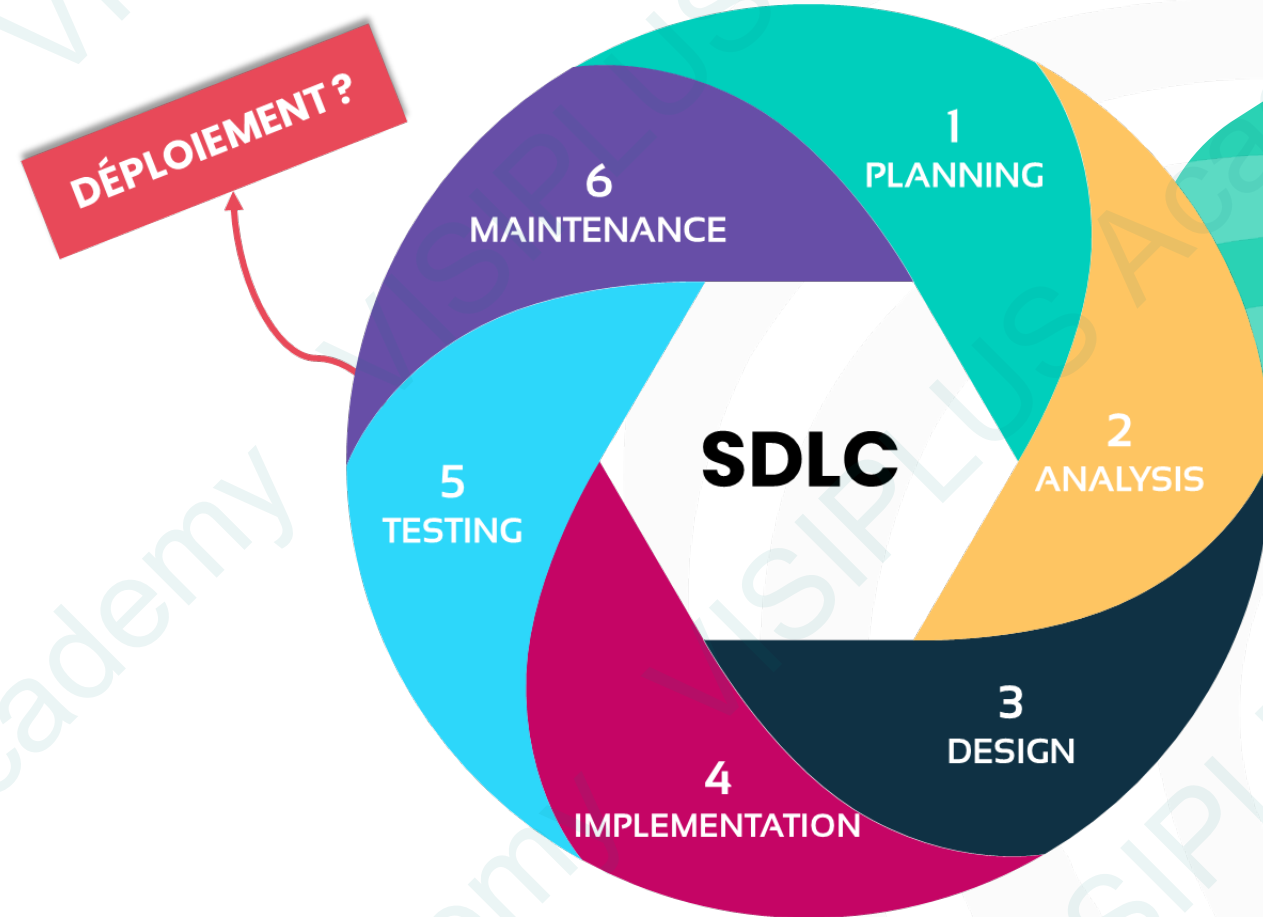


Chapitre 1 :

Le déploiement dans le cycle de vie d'une application



Rappel sur le **cycle de vie** du développement logiciel (SDLC)

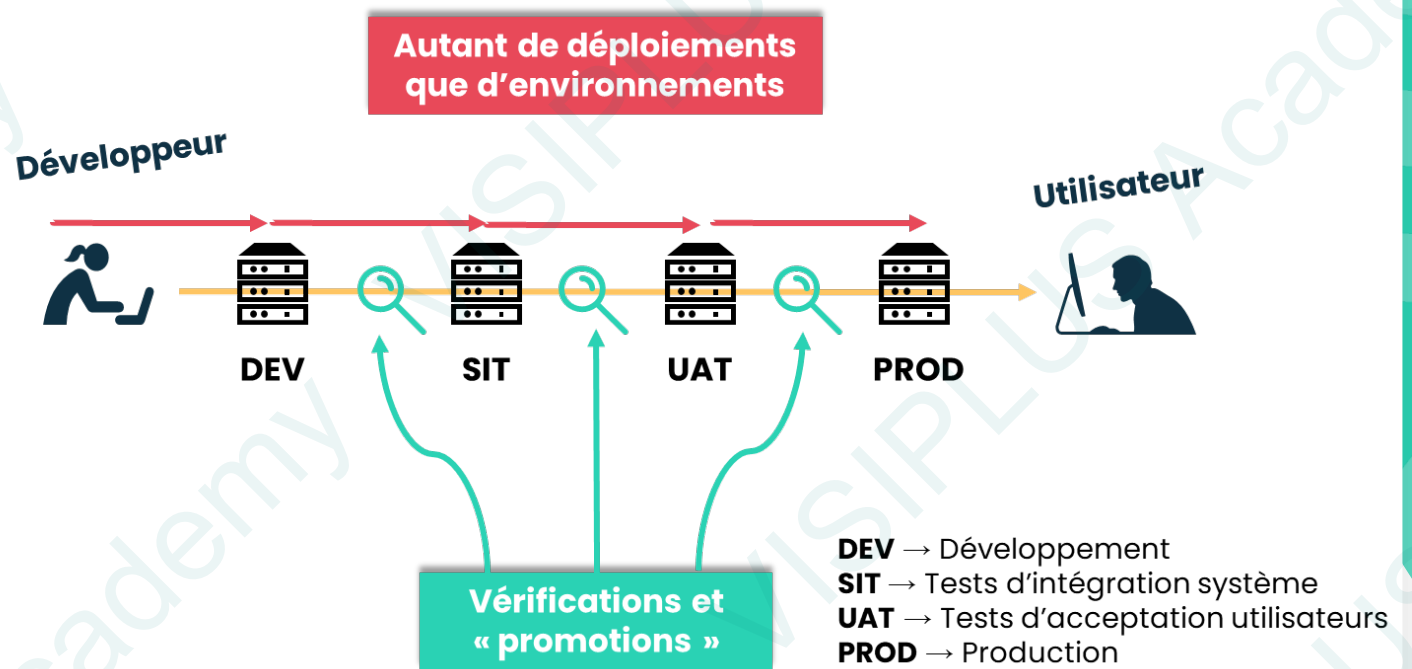


Déploiement
d'une application

Déploiement d'une application

Page 4

Les environnements de déploiement



Déploiement d'une application

Page 5

Deux enjeux pour le déploiement : quelques chiffres vertigineux des GAFAM

Accélérer la mise sur le marché

- ▼ Amazon
 - ▶ 1 déploiement toutes les ... 11 secondes !?

Réduire les pannes coûteuses

- ▼ Amazon
 - ▶ 2013 : panne de 30 minutes → 2 M\$!
- ▼ Facebook
 - ▶ 2021 : 13,7 M\$/heure de panne !

Déploiement d'une application

Page 6

Les pratiques de livraison de logiciels : **Agile**

Des clients moins
patients et plus
exigeants

Collaboration,
interactions et petits
déploiements
fréquents

Le MVP (produit
minimum viable)

Moins de planning,
mais approches
itératives proches
du client

Les pratiques de livraison de logiciels :

Intégration continue (CI)

Une ligne de développement commune entre développeurs

Cette base commune possède des tests et scripts de construction automatisés

Chaque développeur partage son code au plus tôt pour intégration rapide

Un système automatisé contrôle et livre des artefacts prêts à déployer

Déploiement d'une application

Page 7

Les pratiques de livraison de logiciels :

Déploiement continu (CD)

Déployer
automatiquement
l'artefact produit par
le CI

Grâce à des scripts
et des
environnements
programmables
(API)

De nouveaux tests
d'intégration et de
bout-en-bout
peuvent être lancés

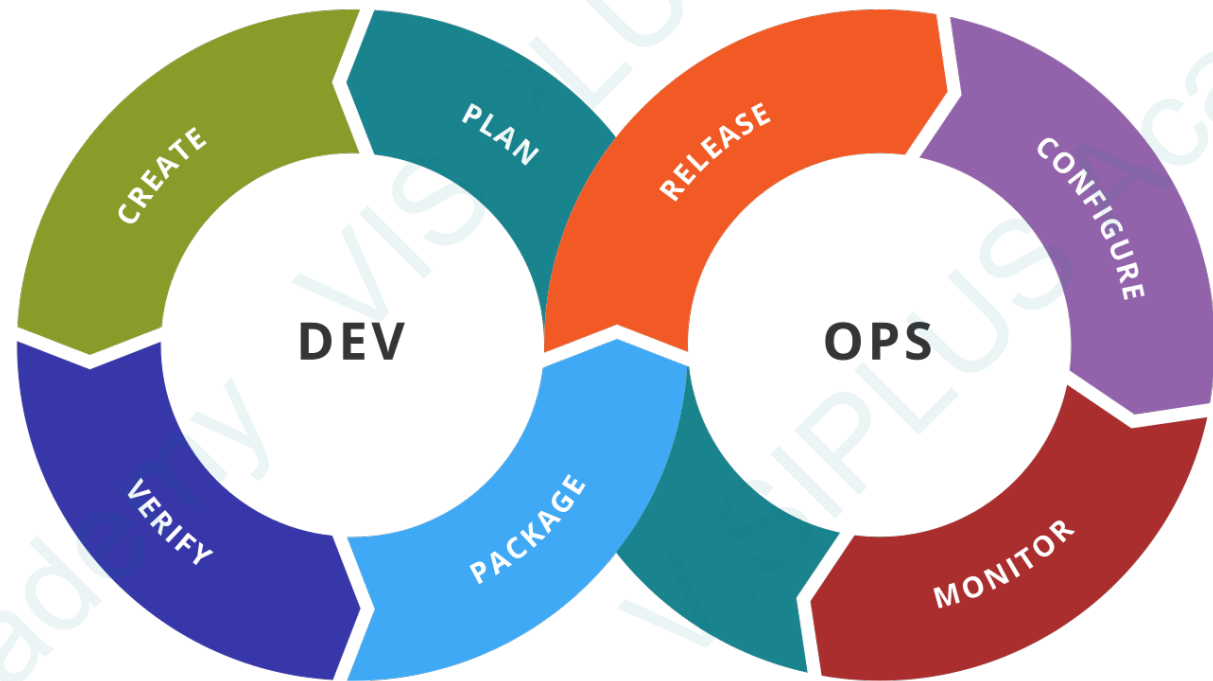
Ne signifie pas
forcément
automatisation de la
mise en production :
prise de décision,
sécurité, etc.

Déploiement d'une application

Page 8

Une nouvelle approche du SDLC

Le DevOps



Déploiement
d'une application

Page 9

Pourquoi le DevOps ?

Des **objectifs antagonistes**

Déploiement
d'une application

Page 10

Dev



Faire **évoluer** l'application
le plus
souvent/rapidement
possible

Ops



Maintenir l'application
de la manière **la plus**
stable possible

Mur de la confusion

Pourquoi le DevOps ?

Des **objectifs antagonistes**

Comment unifier le développement logiciel (dev)
et l'administration des infrastructures
informatiques (ops) ?

Déploiement
d'une application

Page 11

Un acronyme pour caractériser le DevOps : **CALMS**

Culture

Automation

Lean

Measure

Share

Déploiement
d'une application

Page 12

Déploiement d'une application

Page 13

Ce qu'il faut **retenir**



- ▼ Mise en production = Série de plusieurs déploiements par « promotion »
- ▼ Le déploiement doit supporter l'accélération de l'apport de valeur tout en limitant le risque de panne
- ▼ Possible grâce à l'automatisation, amenée par l'agilité, le CI/CD et le DevOps
- ▼ Le déploiement est à la croisée des métiers informatiques, la culture de la collaboration s'impose !



Chapitre 2 : **Le diagramme UML de** **déploiement**

Rappel sur l'objectif d'un diagramme de déploiement

Représenter les éléments physiques d'un système,
leurs interactions ainsi que les logiciels qui s'y
exécutent

Déploiement d'une application

Page 15

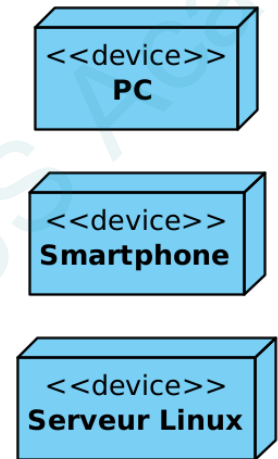
Déploiement d'une application

Page 16

L'élément principal : **Node**

Un node avec le stéréotype `<<device>>` est un équipement physique

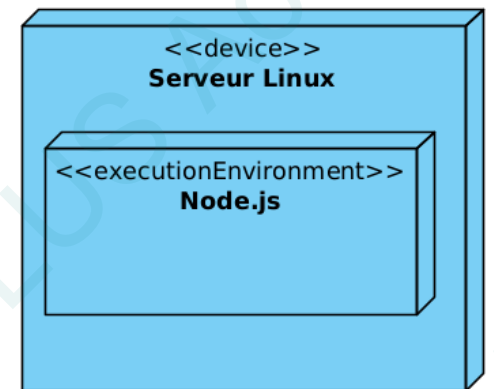
Le nom est soit générique, soit spécifique pour un contexte donné



L'environnement d'exécution avec Node

Pour spécifier un environnement d'exécution (OS, virtualisation, conteneurisation, interpréteur de langage...)

Utiliser un objet Node, inclus dans un Node device, avec le stéréotype <<exécution Environment>>

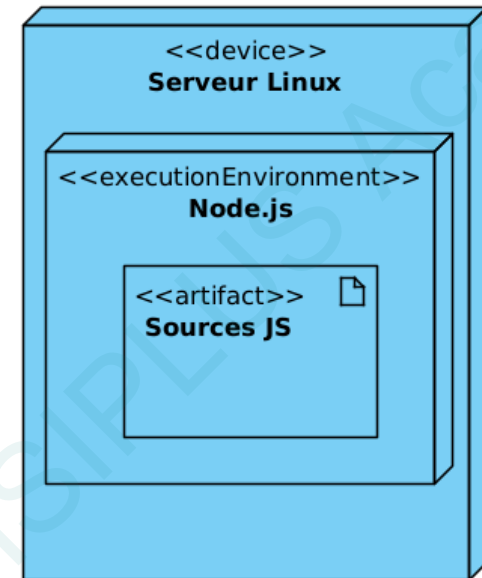


Déploiement d'une application

Page 18

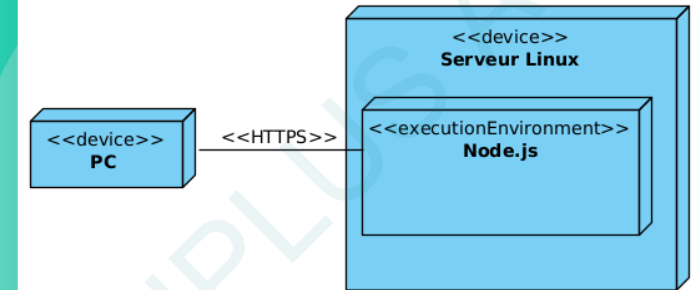
L'artefact à exécuter

Un document
stéréotypé
<<artifact>>
représente ce qui va
être exécuté : un
binaire, des sources à
interpréter, etc.



Interactions entre **objets Node**

Une association entre deux nodes représente une communication réseau. Le stéréotype est le protocole de communication.



Déploiement d'une application

Déploiement d'une application

Page 20

SCREENCAST

Diagrammes de
déploiement
avec Visual
Paradigm

Scénario simple

Scénario
complet

Déploiement d'une application

Page 21

Ce qu'il faut **retenir**

- ▼ Le diagramme de déploiement permet de faire comprendre l'architecture physique hébergeant le produit développé
- ▼ Visualisation des terminaux, communications réseaux, et serveurs
- ▼ Le degré de détail de la description dépend de la cible qui va lire la documentation et de la phase d'avancement dans le projet





Chapitre 3 : **Le diagramme UML de** **composants**

Qu'est-ce qu'un **composant** ?

Un composant est responsable d'un ensemble de fonctions. Il est autonome et peu « couplé » : son remplacement n'affecte pas le reste du produit

<<component>>
Client ECommerce

<<component>>
Module commande

<<component>>
Base de données produits

Un composant est associé aux autres via des interfaces

- ▼ Une interface permet de communiquer entre composants avec un faible couplage
- ▼ Exemples :
 - ▶ API Client – API serveur
 - ▶ Interface – Classe en POO

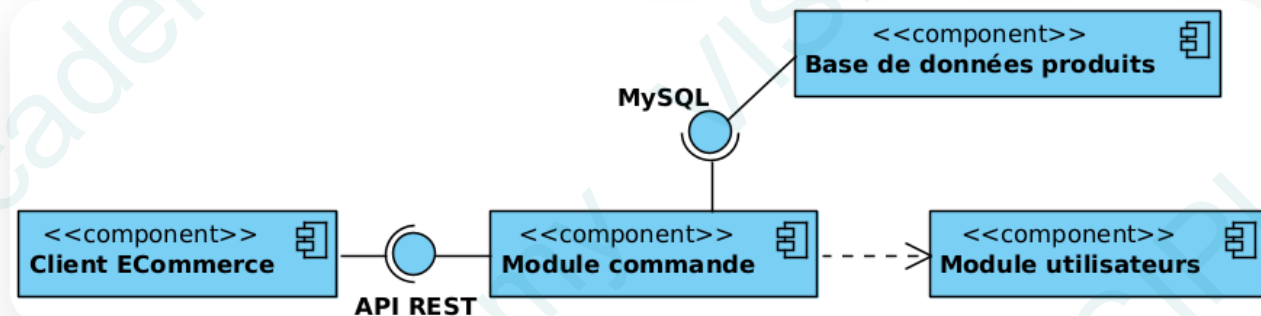


Déploiement d'une application

Page 25

Utiliser les interfaces

- ▼ Les interfaces explicitent les dépendances entre composants
- ▼ Mais il est possible d'utiliser la relation UML « dependency » pour une dépendance simplifiée

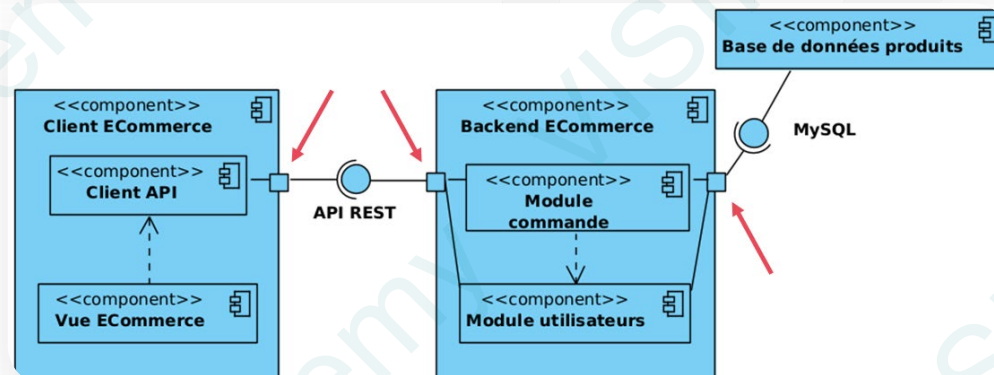


Déploiement d'une application

Page 26

Utiliser les ports

- ▼ Les ports (symbole carré) définissent une interaction entre un composant et le monde « extérieur »
- ▼ Utile pour distinguer les relations internes entre sous-composants et « vraies » interfaces



SCREENCAST

Déploiement d'une application

Page 27

Diagrammes de
composants avec
Visual Paradigm

Exemple avec une
architecture Node +
React

Déploiement d'une application

Page 28

Ce qu'il faut **retenir**



- ▼ Le diagramme de composants permet de détailler des ensembles de fonctions autonomes et peu couplées
- ▼ Les interfaces permettent de montrer les relations entre composants
- ▼ Les ports permettent de distinguer les dépendances externes des détails internes d'implémentation

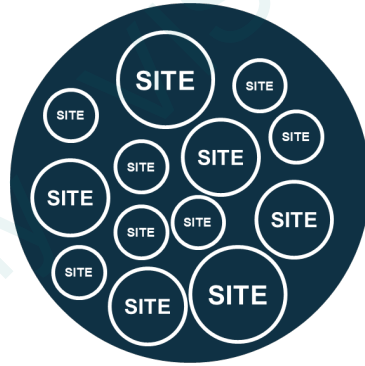


Chapitre 4 : Typologie des hébergements classiques

Déploiement d'une application

Page 30

L'hébergement mutualisé



Fournir de multiples sites web sur un même serveur

Historiquement orienté PHP/MySQL : sites vitrine, blogs, CMS

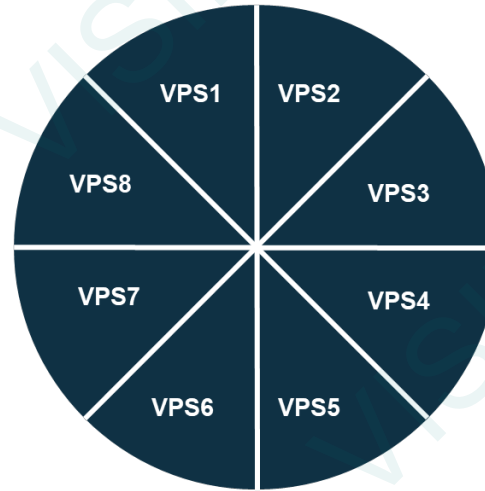
Extension vers d'autres technologies : Python, Node, etc.

Performances limitées et non garanties selon le partage

Contrôle limité via une interface d'administration : cPanel

L'hébergement VPS

Virtual Private Server



Fournir plusieurs serveurs virtuels (VM) sur un même serveur

Performance garantie par la technologie de virtualisation : portions de CPU, de RAM et de réseau contrôlées

Administration comme un vrai serveur, en lignes de commandes distantes : ssh

Déploiement d'une application

Page 31

Déploiement d'une application

Page 32

L'hébergement dédié ou « **bare-metal** »

Intel Xeon
5315-Y 64Go
RAM

Intel Xeon E-
2288G
32Go RAM

Intel Xeon
Gold 6330
128Go RAM

Fournir des serveurs
physiques

Performance de la
machine garantie
(mais fixée)

Administration
complète à faire par
le client

Déploiement d'une application

Page 33

SCREENCAST

- ▼ Exemples types d'hébergement classiques

Déploiement d'une application

Page 34

Ce qu'il faut **retenir**

- ▼ Plus le client souhaite un contrôle et une garantie de performance de son hébergement, plus c'est coûteux
- ▼ Hébergements mutualisés pour des sites web avec peu de besoins/trafic
- ▼ Hébergements VPS pour des sites web à plus fort trafic et un accès admin à la VM
- ▼ Hébergements dédiés pour des besoins spécifiques nécessitant des machines physiques





Chapitre 5 :

Typologie des hébergements Cloud



Qu'est-ce qu'un hébergement **Cloud** – dans les nuages ?

Déploiement d'une application

Robustesse :
installation sur
un réseau de
serveurs et non
plus un serveur
unique

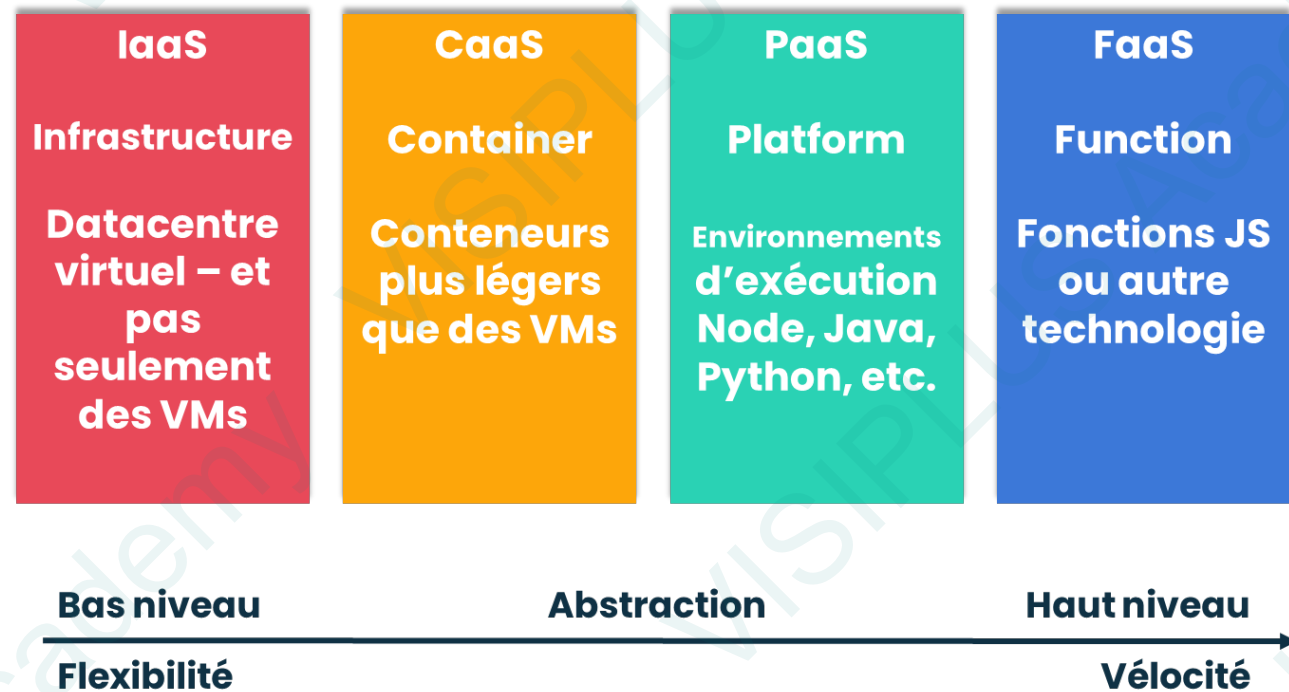
Élasticité :
paiement « à
l'usage » (as a
Service). Mettre
dynamiquement
plus de
ressources en cas
de pics de
demande

Automatisation :
les ressources
demandées sont
modifiables
dynamiquement
à la main ou via
une API !

Déploiement d'une application

Page 37

L'univers des « **as a Service** » pour les développeurs



SCREENCAST

▼ Revue des offres « as a Service »

Déploiement
d'une application

Page 38

Déploiement d'une application

Page 39

Ce qu'il faut **retenir**

- ▼ Le Cloud est un modèle économique de paiement d'hébergement « à l'usage »
- ▼ Il répond aux besoins du marché :
 - ▶ Gestion des pics de demandes sur les produits web
 - ▶ Économie des coûts de maintien d'une infrastructure physique
 - ▶ Automatisation du provisionnement des ressources à déployer : serveurs, environnements d'exécution, applications





Chapitre 6 :

Structure d'une architecture répartie

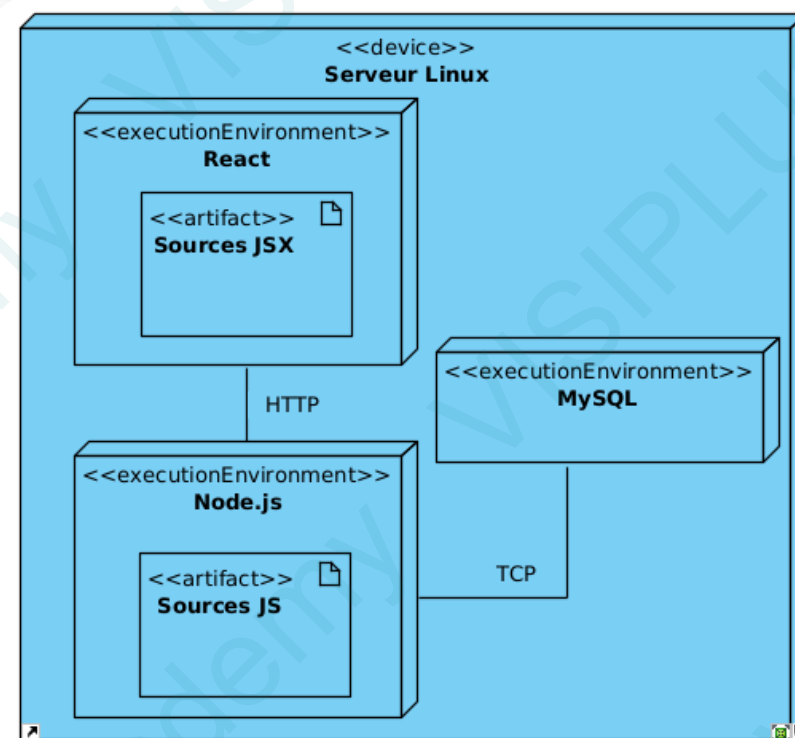


Enjeu : comprendre une **architecture de production**

En tant que **dev**, vous **développez**

Déploiement d'une application

Page 41

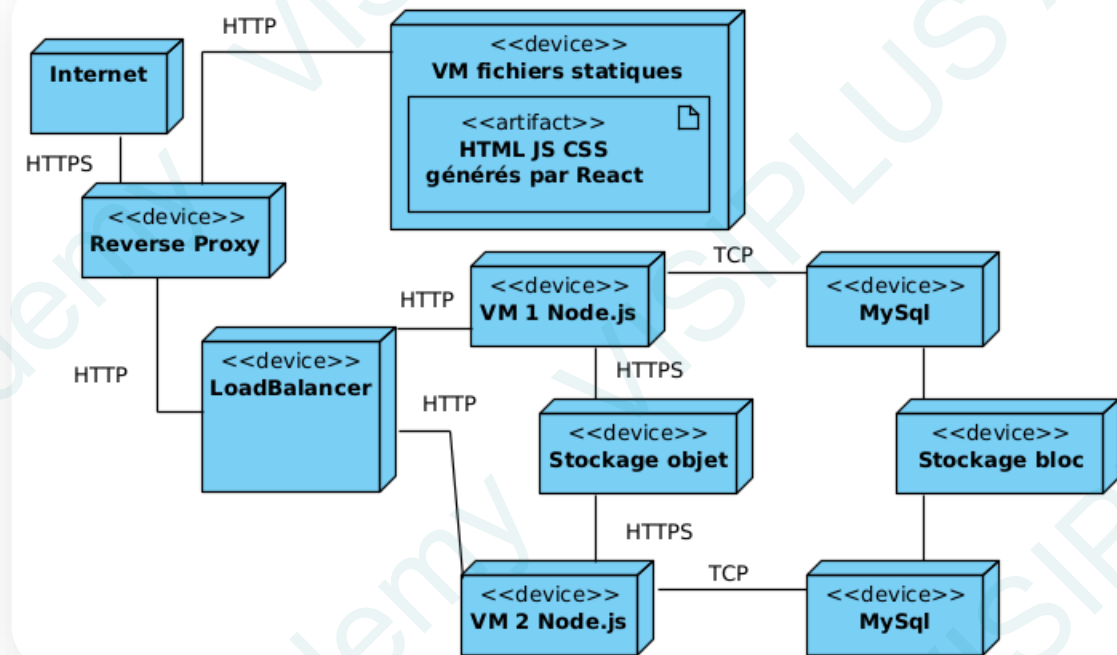


Enjeu : comprendre une architecture de production

En tant que **devops**, vous **déployez**

Déploiement d'une application

Page 42



Déploiement d'une application

Page 43

SCREENCAST

- ▼ Comprendre les nœuds d'une architecture de production

Déploiement d'une application

Page 44

Ce qu'il faut **retenir**

- ▼ La machine virtuelle (VM) permet d'obtenir plusieurs ordinateurs/systèmes sur un seul serveur physique
- ▼ À une VM est associée une activité : serveur d'application, base de données, cache, etc.
- ▼ Le stockage de données est séparé de la VM : montage de données blocs ou service de stockage objet
- ▼ Des services permettent d'augmenter les performances : Cache, Load Balancers, etc.
- ▼ Il est possible de créer des réseaux virtuels (VLAN) pour plus de sécurité





Chapitre 7 : Déploiement sur un service PaaS

Enjeu : comprendre une **architecture de production**

Déploiement d'une application



Offre PaaS :
Heroku fournit
l'environnement
d'exécution
(machines et
middlewares)

Permettre de
déployer très
simplement une
application ...
avec
un git push !

SCREENCAST

- ▼ Déploiement d'une application React/Node sur Heroku

Déploiement d'une application

Page 47

Déploiement d'une application

Page 48

Ce qu'il faut **retenir**



- ▼ Avec Heroku, le déploiement d'une application s'effectue avec Git push sur leur dépôt distant. Potentiellement automatisable !
- ▼ Avec une architecture Node + React :
 - ▶ Construire le frontend → fichiers statiques
 - ▶ Servir les fichiers frontend par Node Express
 - ▶ Lancer le backend Node



Chapitre 8 :

Le versionnement sémantique d'une application



Déploiement d'une application

Page 50

Pourquoi **versionner** ?

Identification forte du déploiement, traçabilité nécessaire pour les clients et les partenaires

Rigueur encore plus importante lorsque le produit possède une API publique

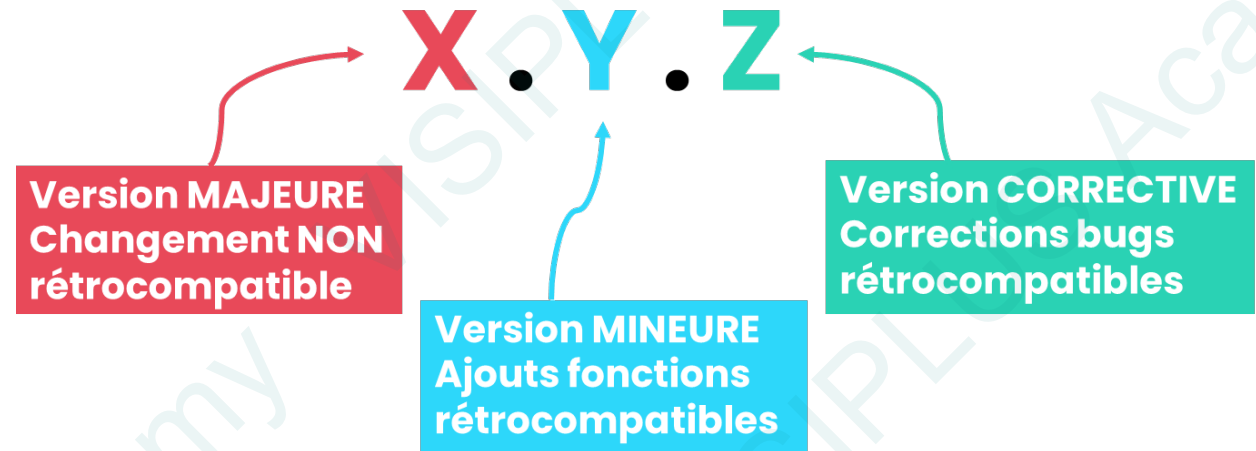
Créer des versions selon une méthode connue et partagée par tout le monde

⇒ Semantic Versionning – semver pour les intimes !

Déploiement d'une application

Page 51

Comment versionner ? Semver en bref



**Npm utilise semver pour gérer
package.json et package-lock.json**

SCREENCAST

Déploiement d'une application

Page 52

Parcours du référentiel
semver.org

Exploitation de
package.json et
package-lock.json par
npm


Déploiement d'une application

Page 53

Ce qu'il faut **retenir**

- ▼ Utiliser des versions identifie clairement un produit déployé
- ▼ Semver donne les règles du jeu de versionnement pour tous les développeurs
- ▼ npm install exploite package.json ⇒ Des dépendances peuvent être mises à jour
- ▼ Symbole ^ et ~ guident npm install
- ▼ npm ci exploite package-lock.json uniquement
- ▼ npm version change la version de votre code selon semver





Chapitre 9 : Les workflows de versionnement de code

Déploiement d'une application

Page 55

Bien déployer nécessite
un **flux de code bien organisé**

Git permet de
travailler en
équipe grâce un
mécanisme de
branche efficace
et flexible

Mais le flux
développement
→ intégration
→ déploiement a
besoin d'être
harmonisé

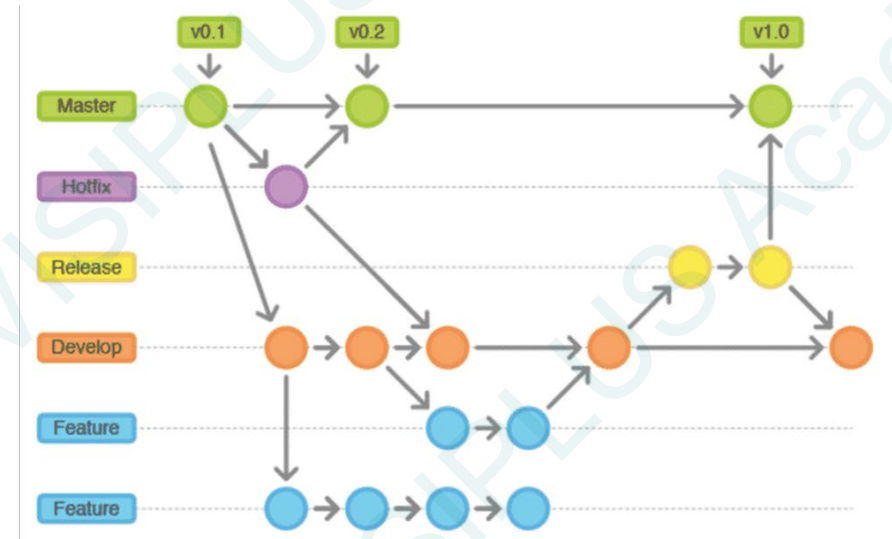
Quand créer une
branche et
quand faire un
merge ? Quelle
branche permet
d'intégrer ou de
déployer ?

Déploiement d'une application

Page 56

Git flow : la référence « historique »

master ou **main**
la prod + tags
versions

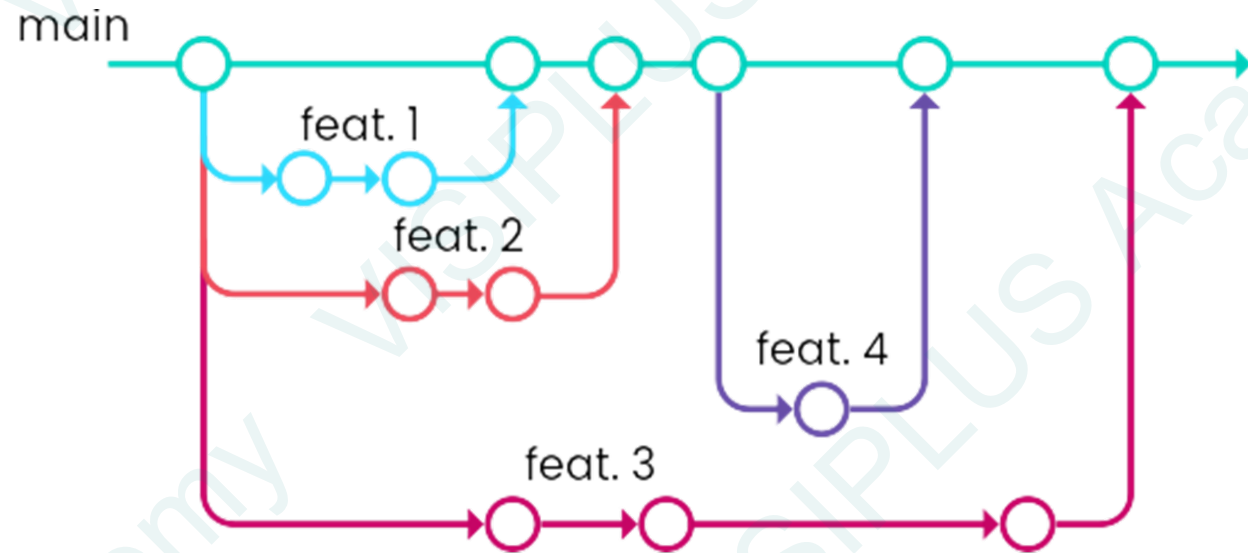


Branches de dev/validation/hotfix à **durée de vie longue**
Très structurant, mais **perte d'efficacité** avec l'usage
des pull/merge requests et du CI/CD

Déploiement d'une application

Page 57

Github flow : pragmatisme et simplicité

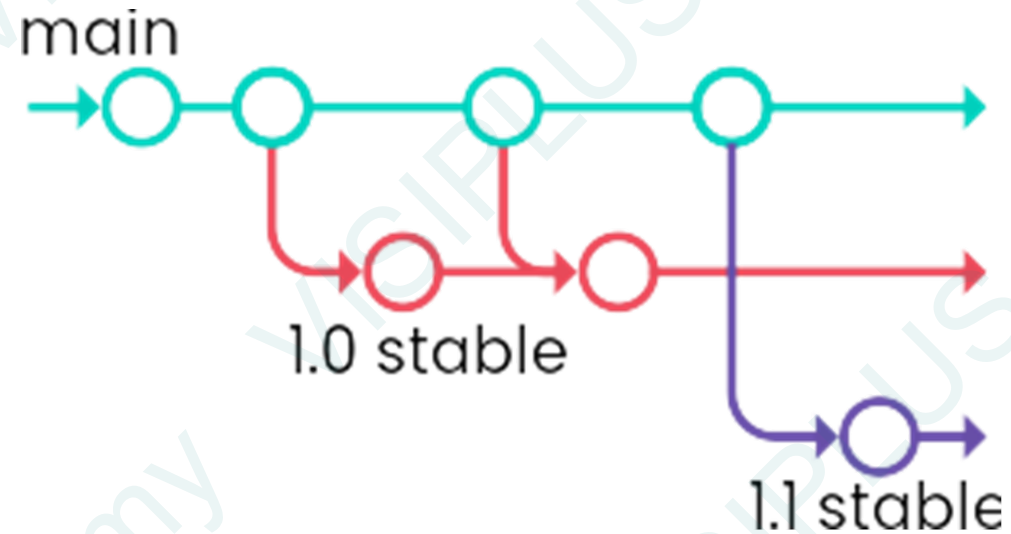


main : Branche validée **susceptible d'être déployée**
Et **branches courtes** de features, avec appui des **pull-requests** pour la validation

Déploiement d'une application

Page 58

Gitlab flow : prise en compte de **versions long terme**



main : branche avec features validées par **merge-requests**. Mais déploiement via **branches de versions stables**. Utilisation de **cherry-picks** pour les hotfixes.

Déploiement d'une application

Page 59

SCREENCAST

- ▼ Comparaison entre workflows Git flow et Github flow

Déploiement d'une application

Page 60

Ce qu'il faut **retenir**

- ▼ Choisir un workflow Git est essentiel pour fiabiliser le travail de l'équipe et déployer rapidement
- ▼ Git Flow est historiquement connu et toujours utilisé
- ▼ Github flow et gitlab flow se présentent comme des alternatives plus simples, à condition d'utiliser les pull/merge-requests



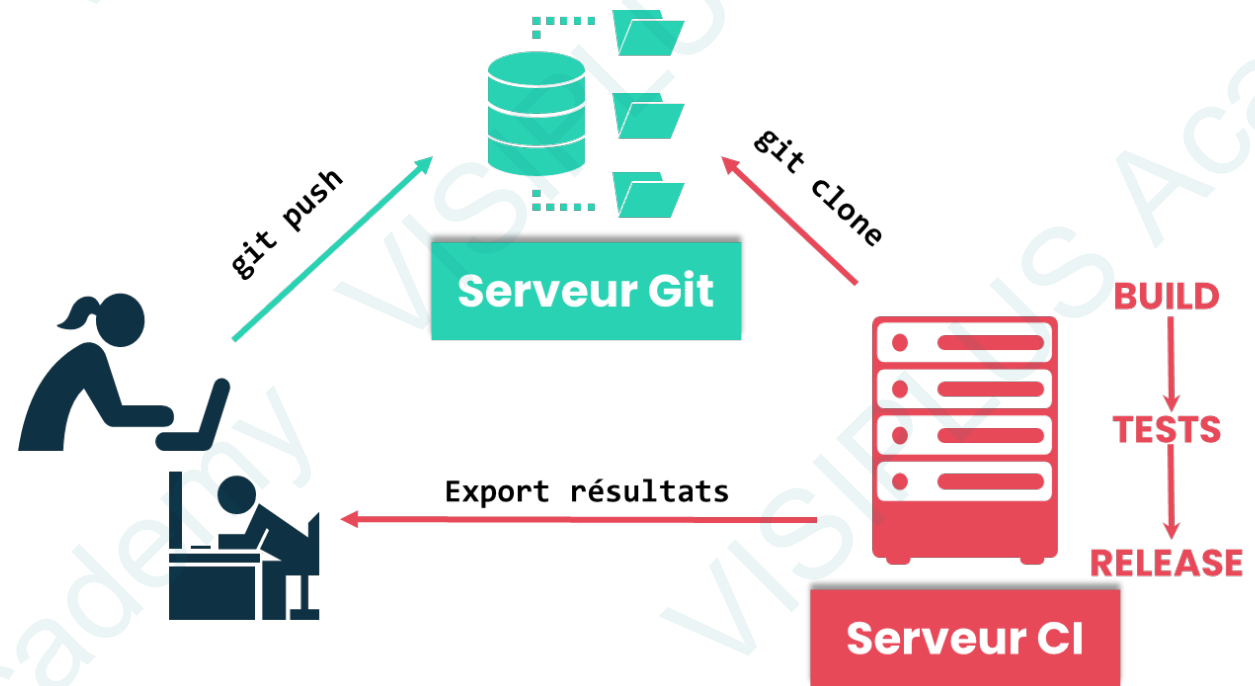


Chapitre 10 : Intégration continue d'une application

Déploiement d'une application

Page 62

Principe de l'intégration continue



Déploiement d'une application

Page 63

SCREENCAST

L'intégration
continue avec
Gitlab

Structure d'un
pipeline Gitlab

La syntaxe pour
écrire un pipeline
Gitlab

Déploiement d'une application

Page 64

Ce qu'il faut **retenir**



- ▼ La construction et la vérification automatisées d'artefacts à déployer s'effectuent par de l'intégration continue (CI)
- ▼ Gitlab CI décompose un pipeline en stages et en jobs
- ▼ Des rules permettent de configurer les actions à lancer en fonction du workflow : branches et merge-requests



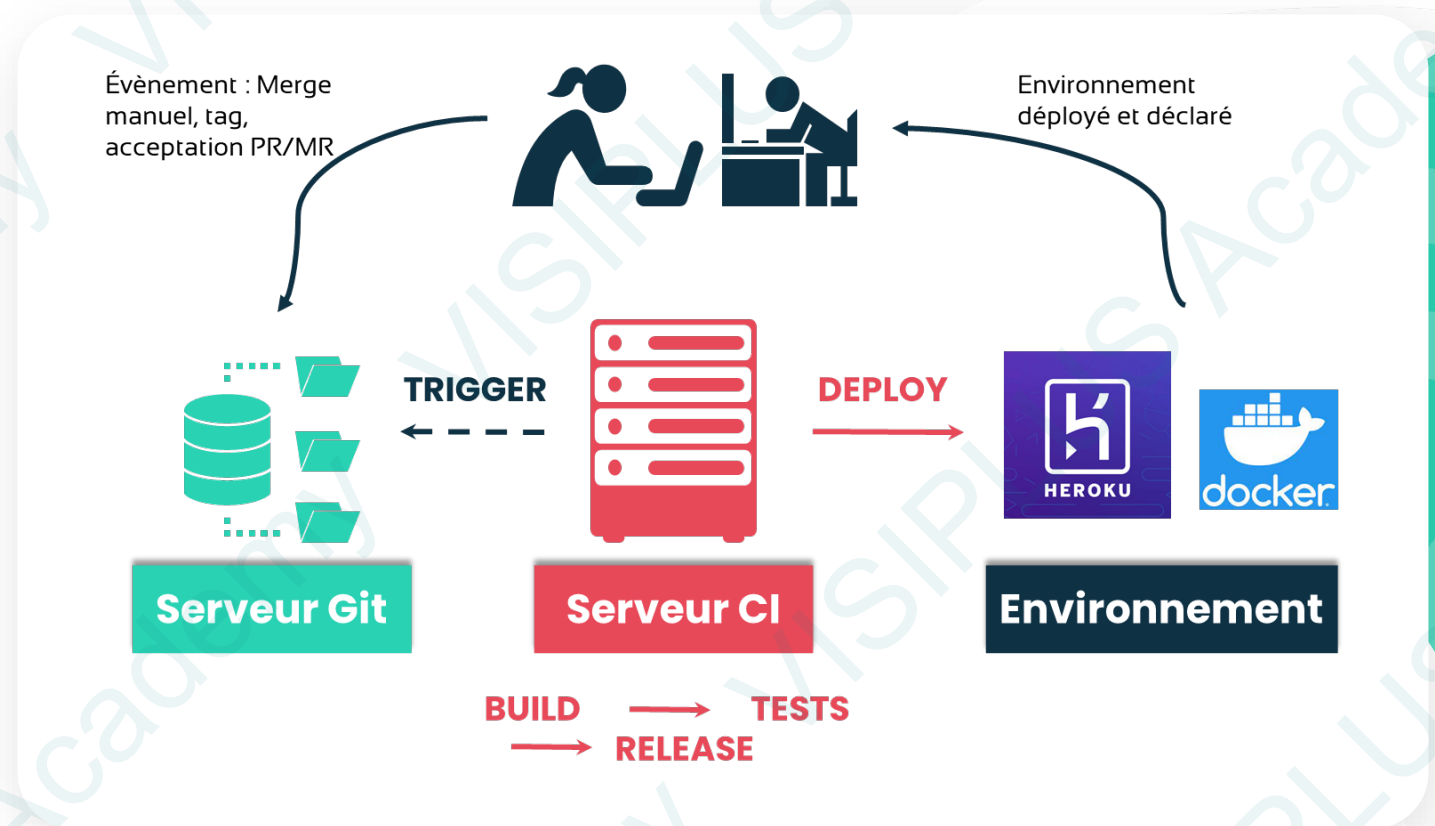
Chapitre 11: Déploiement continu d'une application



Déploiement d'une application

Page 66

Principe du **déploiement continu**



SCREENCAST

Déploiement d'une application

Page 67

Déploiement sur
Heroku avec Gitlab

Comprendre les
environnements dans
Gitlab

Déploiement d'une application

Page 68

Ce qu'il faut **retenir**



- ▼ Le déploiement continu permet de déployer via un système automatisé sur la base d'un évènement lié à Git
- ▼ Les jobs de déploiement possèdent donc des règles de déclenchement spécifiques
- ▼ Gitlab recense les environnements déployés grâce aux infos de CI/CD

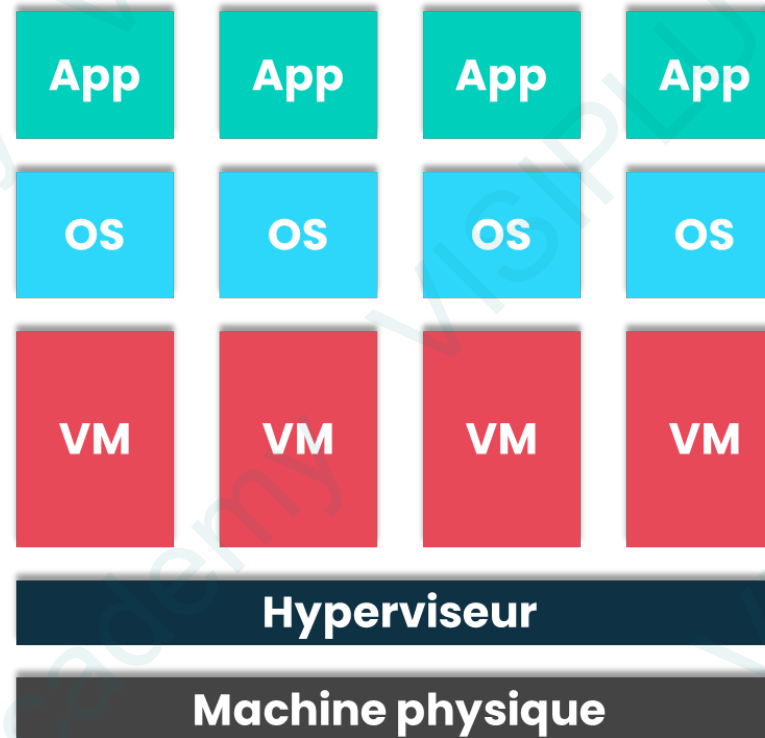


Chapitre 12: Les principes de la conteneurisation

Déploiement d'une application

Page 70

VM vs conteneur : Architecture VM



Chaque VM possède son OS :

→ **surconsommation** de ressources

→ **OS** à mettre à jour

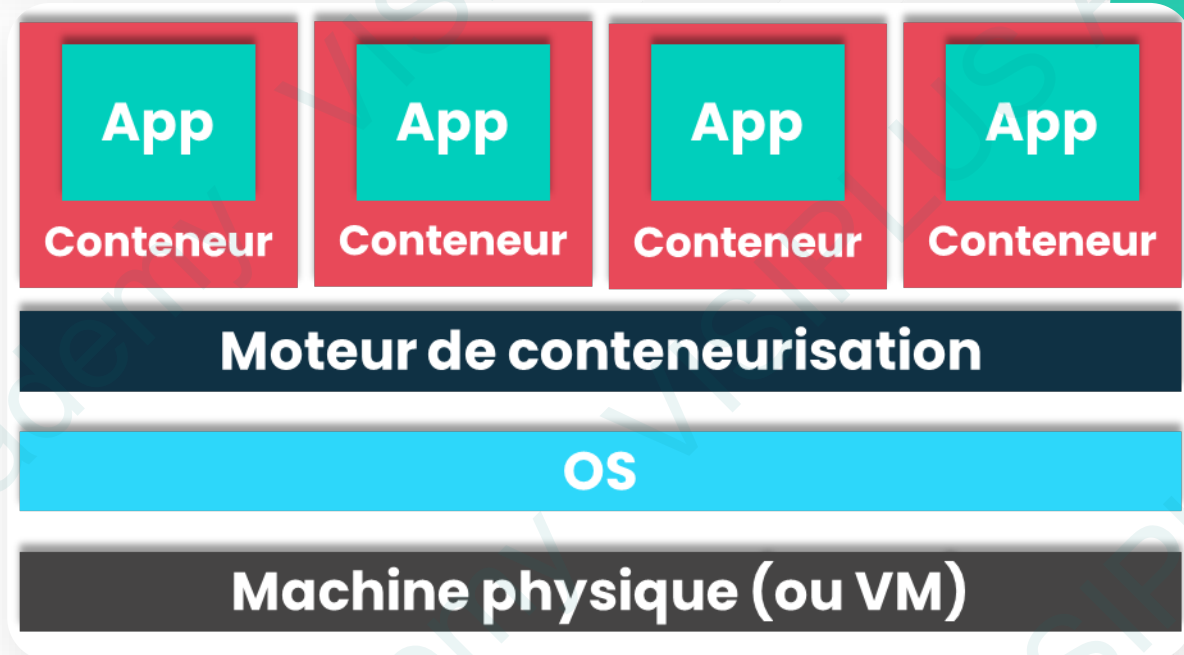
Déploiement d'une application

Page 71

VM vs conteneur

Architecture conteneur

Un conteneur s'appuie sur l'OS hôte. Il fournit une configuration spécifique et le moteur de conteneurisation s'assure de l'isolation entre conteneurs



Docker : pour commencer dans le monde des conteneurs



Docker permet de
créer et gérer des
conteneurs

En ligne de
commande docker

Sous Windows avec
Docker Desktop

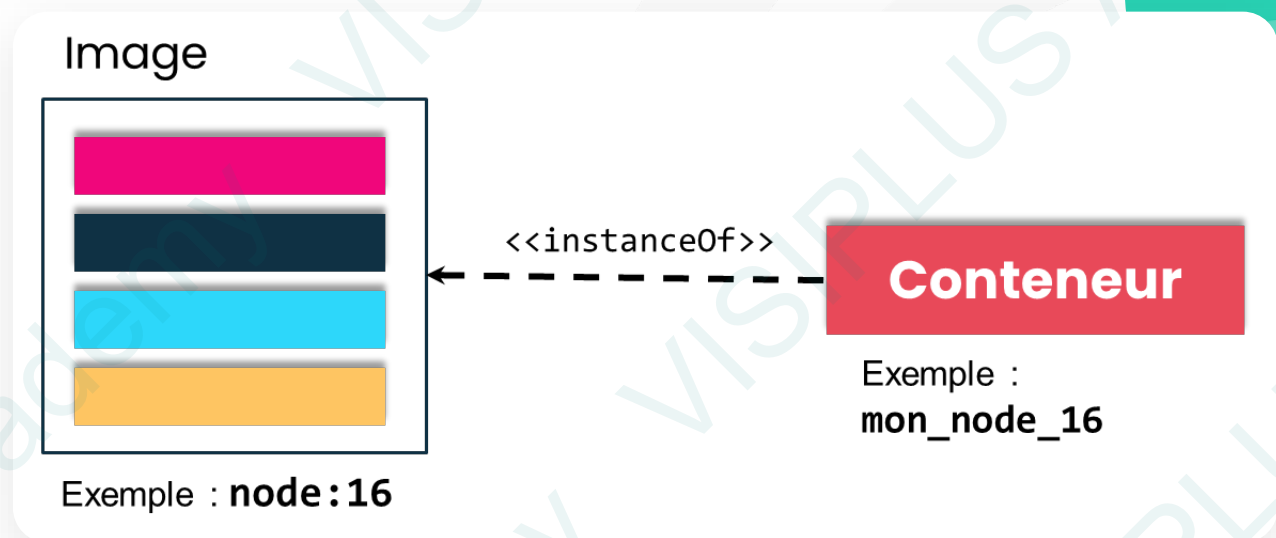
Offre packagée :
client + API + moteur
de conteneurs

Déploiement d'une application

Page 72

Les images de conteneurs

- ▼ Une image contient la configuration à appliquer pour un conteneur : système de fichiers, environnement, etc.
- ▼ Un conteneur est une « instance » d'une image



SCREENCAST

Déploiement d'une application

Page 74

Découverte de Docker

Commandes
élémentaires de
Docker

Déploiement d'une application

Page 75

Ce qu'il faut **retenir**

- ▼ Avec les conteneurs, hébergement de plusieurs services sans dupliquer l'OS
- ▼ Docker crée et gère des conteneurs, possible sur son propre PC
- ▼ docker run permet de lancer des conteneurs basés sur des images
- ▼ docker exec permet d'entrer dans le conteneur





Chapitre 13: Création d'image et déploiement de conteneur

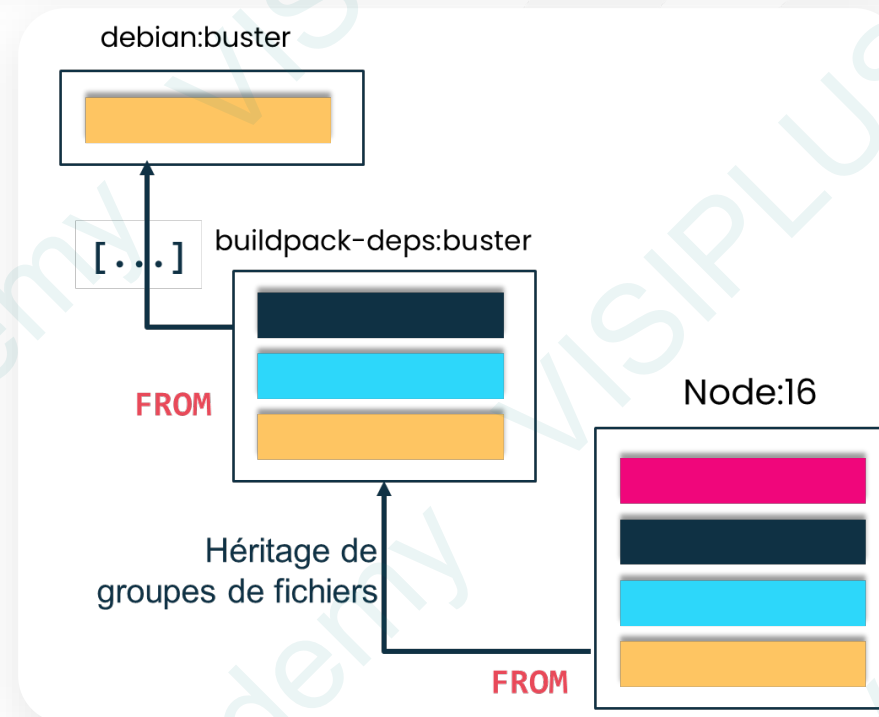


Déploiement d'une application

Page 77

Comment est structurée une image ?

- ▼ Une image est structurée en couches de fichiers
- ▼ Elle peut hériter des fichiers d'une autre image



Déploiement d'une application

Page 78

Structure d'un Dockerfile

```
FROM <autre_image>:<version>
```

Héritage d'image

```
# instructions  
# ADD, RUN, EXPOSE ...
```

Instructions de
configuration

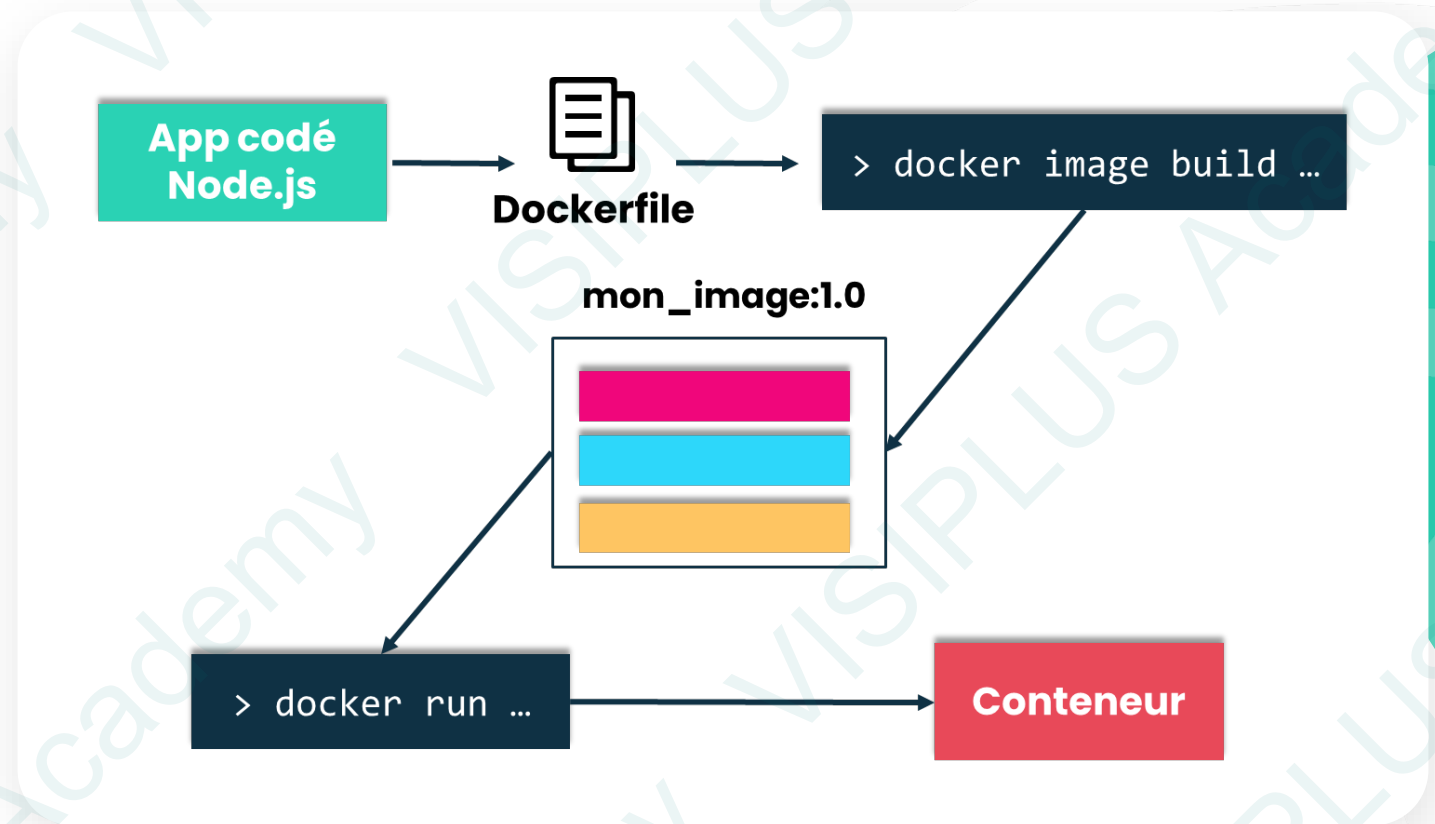
```
ENTRYPOINT ["cmd", "arg1"]
```

Commande de
lancement

Déploiement d'une application

Page 79

Workflow création d'image : déploiement conteneur



SCREENCAST

Déploiement d'une application

Page 80

Conteneur
personnalisé « Hello
World »

Conteneur pour
application Node

Déploiement d'une application

Page 81

Ce qu'il faut **retenir**



- ▼ Utiliser les conteneurs s'effectue en trois temps
 - ▶ Créer Dockerfile. Héritage avec FROM
 - ▶ Construire l'image
 - ▶ Lancer un conteneur instanciant cette image
- ▼ ADD/COPY : copier une ressource sur l'image
- ▼ RUN : commande + couche de fichiers
- ▼ ENV : variable d'environnement
- ▼ CMD/ENTRYPOINT : commande finale conteneur



Chapitre 14:

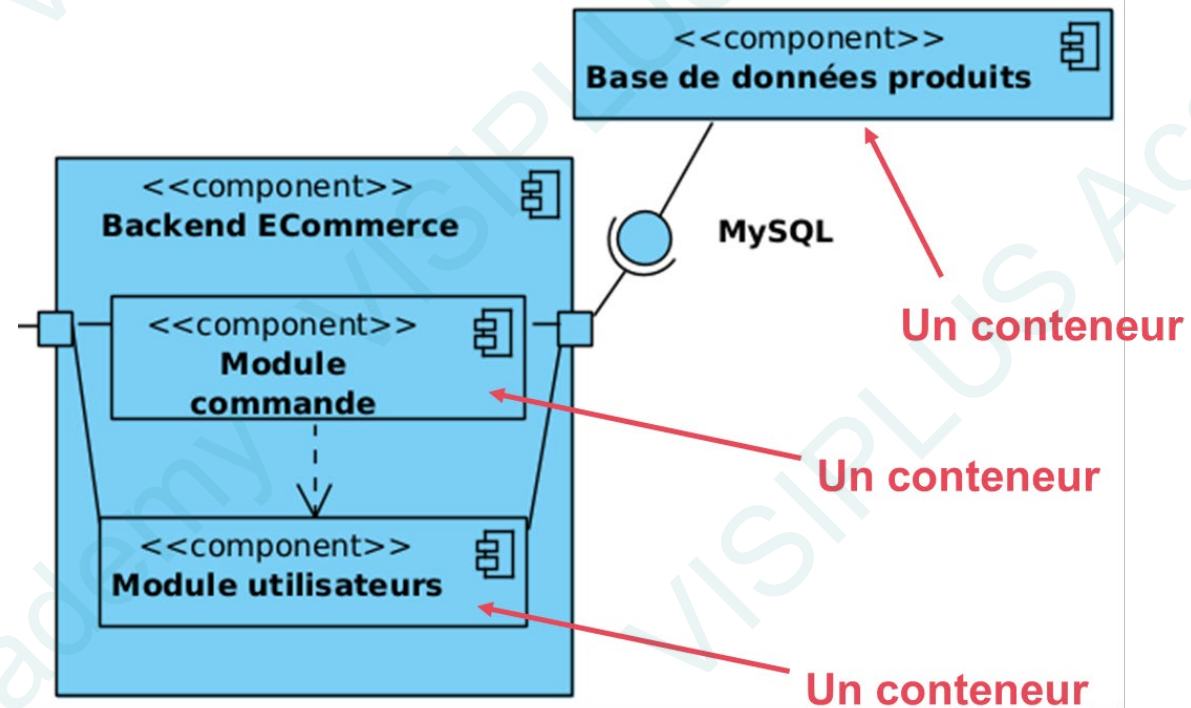
Déploiement de conteneurs multiples



Déploiement d'une application

Page 83

Un conteneur = un service



Déploiement d'une application

Page 84

Plusieurs conteneurs en une action ? Docker-compose !

Identification de conteneurs

Base de données,
image open-source

```
services:
  database:
    image: mysql:8.0
    environment:
      - MYSQL_USER=visiplus
      - MYSQL_PASSWORD=visipluspassword
```

App créée à partir
d'un Dockerfile local

```
  app:
    depends_on:
      - mysqldb
    build: .
    ports:
      - 3000:3000
    environment:
      - DB_HOST=mysqldb
      - DB_USER=visiplus
      - DB_PASSWORD=visipluspassword
```

App lancée seulement
après avoir lancé
database

Déploiement d'une application

Page 85

SCREENCAST

- ▼ Déployer une application Node React MySQL en une seule fois avec Docker-compose

Déploiement d'une application

Page 86

Ce qu'il faut **retenir**

- ▼ Docker-compose déploie une application multi-services de manière atomique
- ▼ Un fichier docker-compose.yml définit les services et leurs dépendances au format YAML
- ▼ **services** : liste des services à déployer
- ▼ **image** : instancier une image construite
- ▼ **build** : construire à la volée l'image
- ▼ **depends_on** : attendre (partiellement) un conteneur avant d'en lancer un autre





Chapitre 15 :

Gestion des volumes et des réseaux avec les conteneurs



Les **volumes** dans Docker

Extraire les données écrites par le conteneur sur un emplacement partageable et sécurisable

Exemple : dossier de données d'une base de données (/var/lib/mysql) ou d'une application

Avec Docker-compose, possibilité de nommer le volume pour le partager et le retrouver sur l'hôte

```
volumes:  
  - db:/var/lib/mysql
```

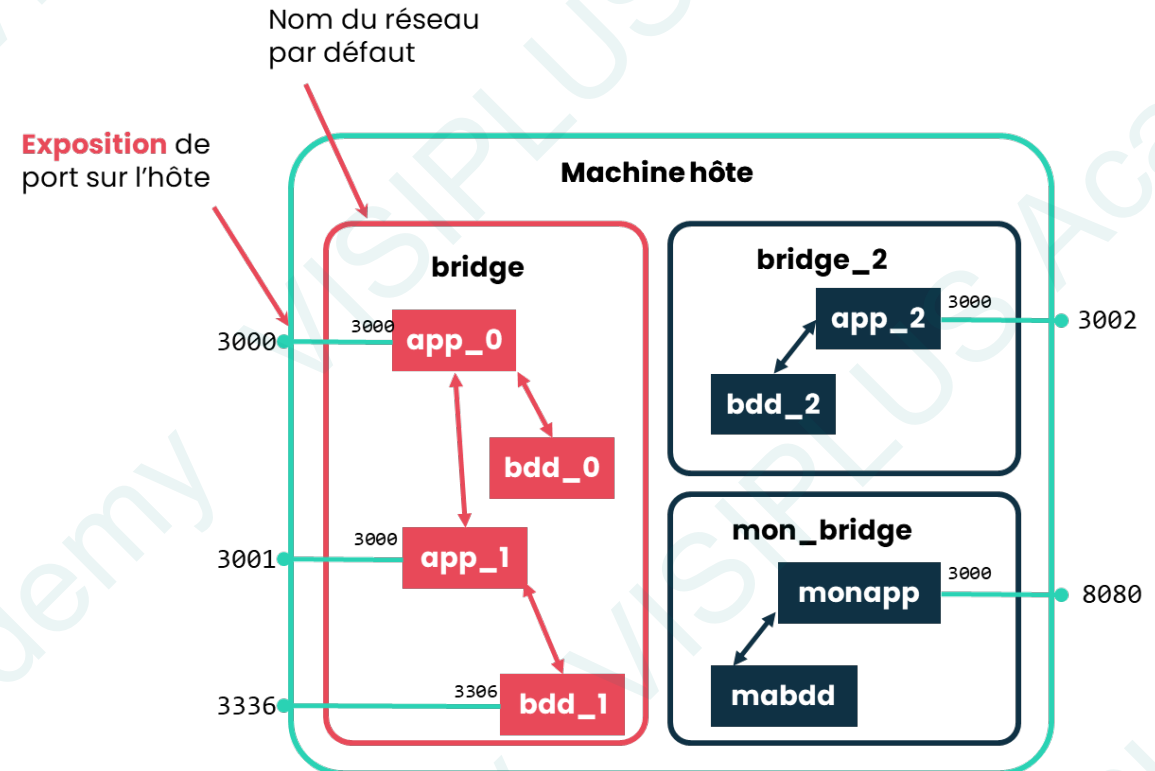
Nom du volume

Point de montage sur le conteneur

Les réseaux dans Docker : conteneur **en mode bridge**

Déploiement d'une application

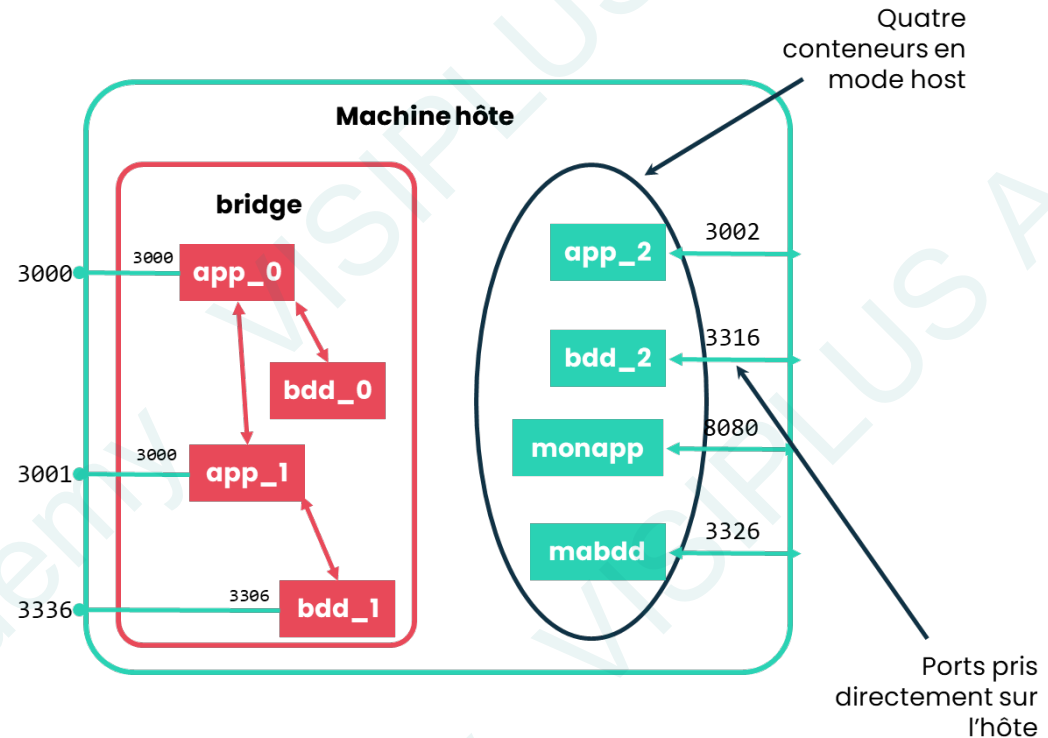
Page 89



Déploiement d'une application

Page 90

Les réseaux dans Docker : le mode host



Déploiement d'une application

Page 91

SCREENCAST

- ▼ Expérimentation des volumes et des réseaux Docker

Déploiement d'une application

Page 92

Ce qu'il faut **retenir**

- ▼ Le volume stocke de manière efficace et plus sécurisable un dossier de données
- ▼ En mode bridge, un service lancé n'est pas accessible directement depuis l'hôte ou l'extérieur
- ▼ Il doit exposer un port qui peut être différent de celui du service
- ▼ Plusieurs services déployés dans un docker-compose peuvent communiquer directement via leur nom de conteneur





Chapitre 16 : Orchestration de conteneurs

Déploiement d'une application

Page 94

Déployer en **haute disponibilité**

Haute
disponibilité :
assurer une durée
de
fonctionnement
continue (uptime)
supérieure au
service fourni par
une seule
machine

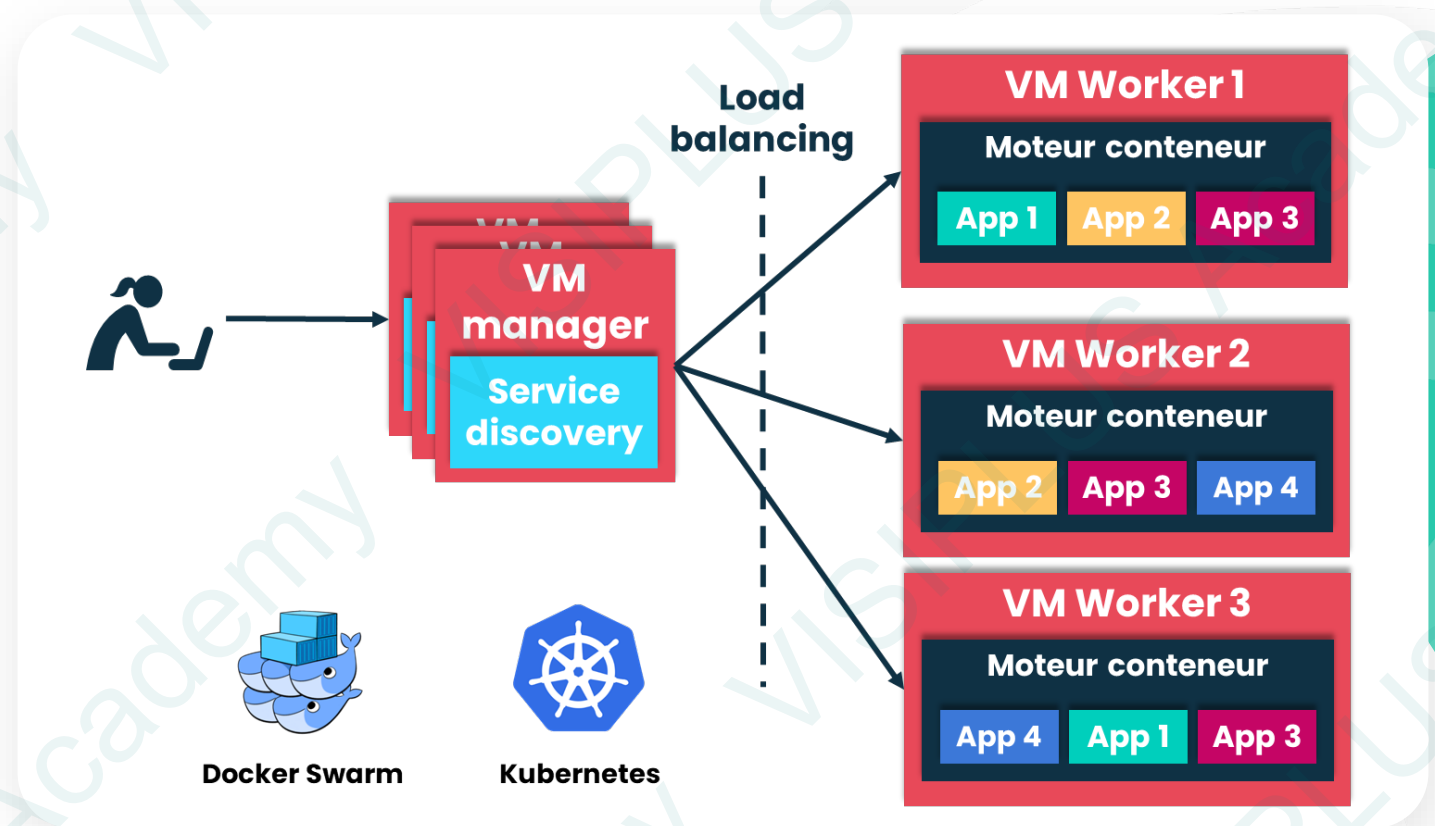
Mécanisme pour
fournir le service
malgré les
pannes et les
mises à jour

Service à
déployer sur
plusieurs nœuds,
mais transparent
pour l'utilisateur :
le cluster !

Déploiement d'une application

Page 95

Principe d'un cluster



SCREENCAST

Déploiement d'une application

Page 96

Release d'une image
Docker sur gitlab

Découverte de Docker
Swarm





Déploiement d'une application

Page 97

Ce qu'il faut **retenir**

- ▼ Un cluster permet de déployer un service en haute disponibilité : les mises à jour et certaines pannes n'affectent pas le service
- ▼ Une VM « manager » gère l'accès au service en effectuant du load balancing
- ▼ Une VM « worker » héberge un replica de service
- ▼ Docker Swarm permet de réutiliser le format Docker-compose, enrichi de directives de déploiement



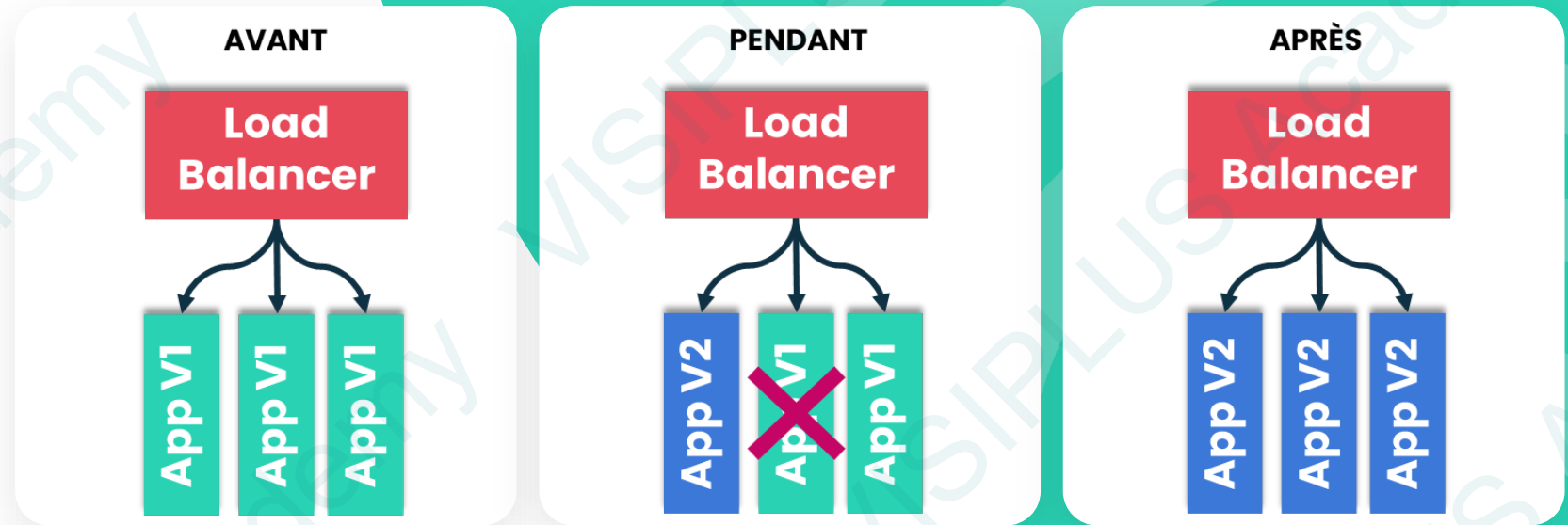


Chapitre 17: Déploiement de type « rolling update »

Déploiement d'une application

Page 99

Rolling update :
progressif **nœud par nœud**



Déploiement d'une application

Page 100

SCREENCAST

▼ Rolling update avec Docker Swarm

Déploiement d'une application

Page 101

Ce qu'il faut **retenir**

- ▼ Une mise à jour déployée en mode rolling update s'effectue replica par replica
- ▼ Elle expose deux versions du produit durant un même laps de temps
- ▼ Mode par défaut avec Docker Swarm, paramétrable avec `--update-delay`



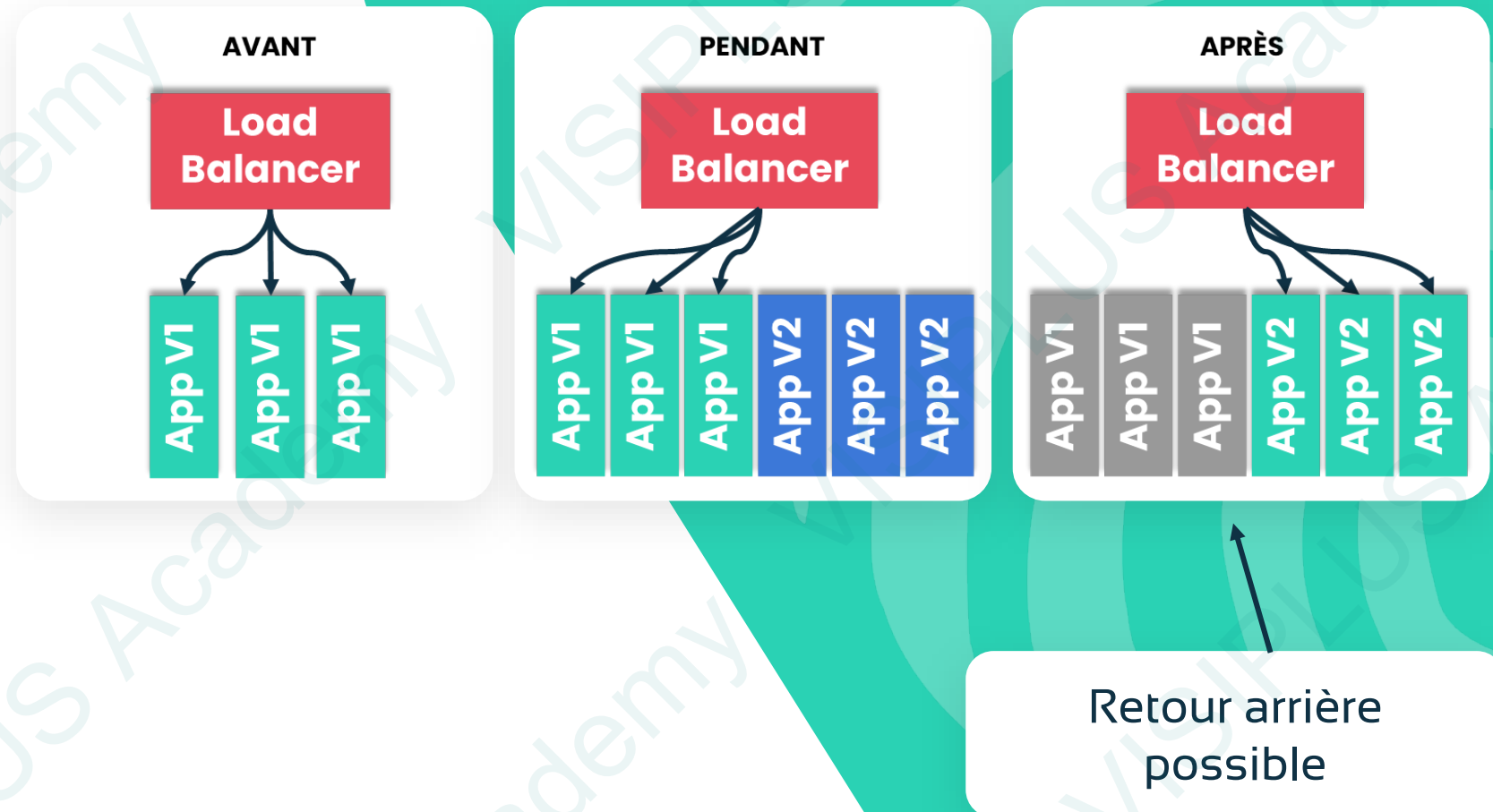


Chapitre 18 : Déploiement de type « blue-green »

Blue-green : un seul basculement

Déploiement d'une application

Page 103



Déploiement d'une application

Page 104

SCREENCAST

▼ Blue-green avec Docker Swarm

Déploiement d'une application

Page 105

Ce qu'il faut **retenir**

- ▼ Une mise à jour déployée avec le mode blue-green déploie entièrement le nouveau produit de manière cachée → blue
- ▼ Un basculement passe de l'ancienne version à la nouvelle version → green
- ▼ Un rollback (retour arrière) est possible en théorie en cas de problème

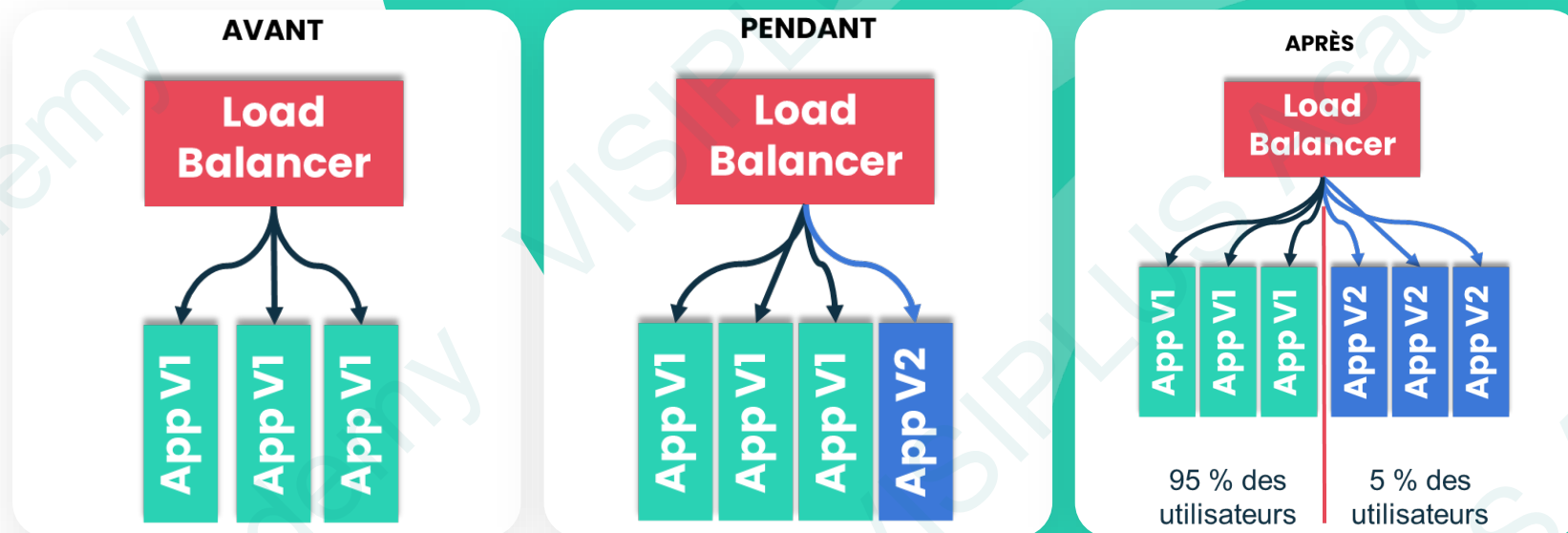


Chapitre 19 : Déploiement de type Canary

Canary : expérimentation de nouvelles fonctionnalités

Déploiement d'une application

Page 107



Déploiement d'une application

Page 108

SCREENCAST

- ▼ Simulation de Canary avec Docker Swarm

Déploiement d'une application

Page 109

Ce qu'il faut **retenir**

- ▼ Un déploiement de type Canary permet d'exposer deux versions d'un produit simultanément
- ▼ Pour expérimenter une nouvelle fonctionnalité auprès d'un panel utilisateurs
- ▼ Les données doivent être compatibles avec les deux versions du produit



Chapitre 20 : Migration de données

Déploiement d'une application

Page 111

Enjeu de la **migration de données**

Transformer la structure des données, par exemple d'une base MySQL

Un script de migration (au langage SQL) permet d'effectuer cette transformation

Ajouter, supprimer, renommer une table/colonne.
Certains changements non rétrocompatibles !

Comment migrer dans un mode déploiement sans interruption ?

Déploiement d'une application

Page 112

Rendre **rétrocompatible** toute migration

Exemple : Table Users – transformer une colonne id (un nombre) en une colonne userId (un UUID, 9d1cef37-1718-483d-883a-4bd38c9a20c1)

Version 1.0.0 :
Utilisation de id

Version 1.1.0 :
Ajouter, alimenter,
utiliser userId

Version 2.0.0 :
Supprimer id
inutilisée

Déploiement d'une application

Page 113

SCREENCAST

▼ Migrations de données en rolling update

Déploiement d'une application

Page 114

Ce qu'il faut **retenir**

- ▼ Une migration de données s'effectue avec un ou plusieurs scripts de migration
- ▼ Certains changements sont rétrocompatibles et d'autres non
- ▼ Implémenter le plus possible des changements rétrocompatibles pour assurer un rollback éventuel ou une coexistence de deux versions de produit





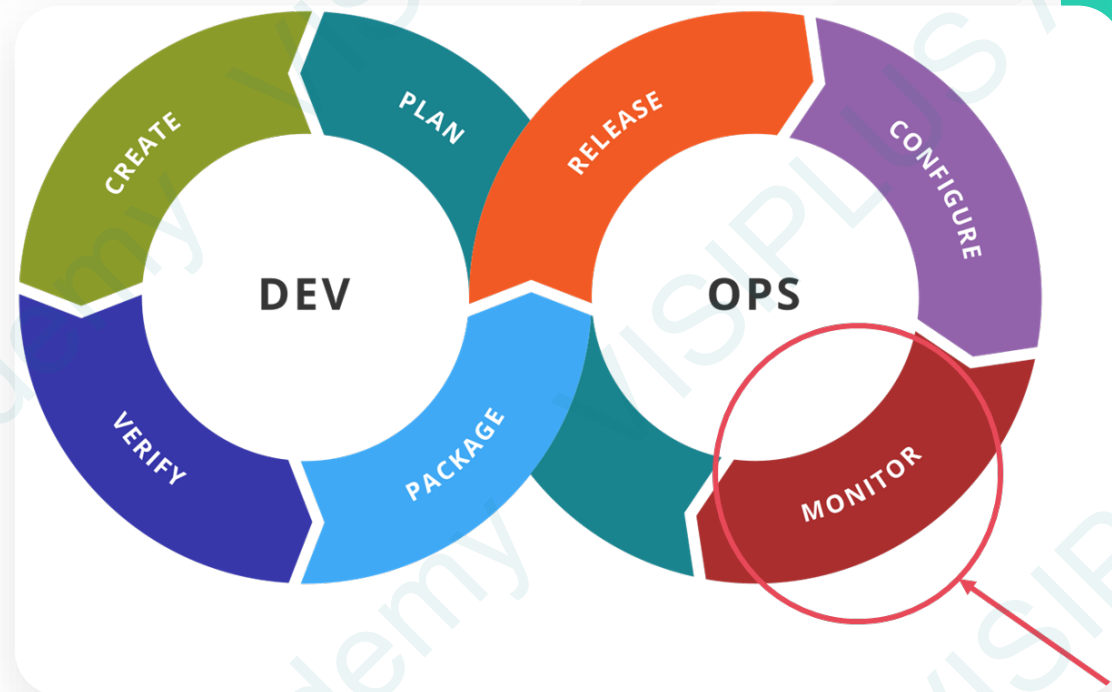
Chapitre 21: Monitoring d'une application

Déploiement d'une application

Page 116

Monitor : une étape charnière souvent sous-utilisée

- ▼ Après déploiement, le monitoring (supervision) remonte des indicateurs techniques et fonctionnels ⇒ Aide à la décision pour la phase Plan



Déploiement d'une application

Page 117

SCREENCAST

Exposer des métriques
avec Prometheus

Alerte mails avec
sonde de supervision

Déploiement d'une application

Page 118

Ce qu'il faut **retenir**

- ▼ Le monitoring permet de remonter des métriques depuis les environnements déployés
- ▼ Il permet de superviser la santé d'un environnement par des indicateurs techniques
- ▼ Il permet de remonter des indicateurs aidant à une décision fonctionnelle
- ▼ Deux formes d'indicateurs : les dashboards et les notifications d'alerte



Chapitre 22 : Signature de code

Déploiement d'une application

Page 120

Qu'est-ce que la **signature de code** ?

Confirmer l'identité
de l'auteur du code

Garantir que le code
n'a pas été modifié
depuis qu'il a été
signé

⇒ Sécuriser le
déploiement de ce
qui a été construit

Utilisation de la
cryptographie

Déploiement d'une application

Page 121

Exemples de **signature de code**

Signature des
commits Git :
utilisation de GPG

Docker Content Trust :
utilisation de Notary
avec mécanismes de
rotation de clés et de
délégation

Déploiement d'une application

Page 122

SCREENCAST

▼ Signature GPG de commits avec Gitlab

Déploiement d'une application

Page 123

Ce qu'il faut **retenir**

- ▼ La signature de code fiabilise la traçabilité du code ou d'un artefact créé par un développeur.
- ▼ Elle nécessite la création d'une paire de clés privées/publiques
- ▼ Elle nécessite aussi un registre de certificats

