

Introduction au DevOps

Introduction

Jolan PHILIPPE

9 Septembre 2026

Université d'Orléans

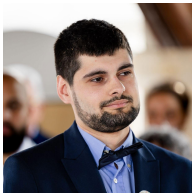


Jolan Philippe, Maître de conférence
Université d'Orléans, équipe LMV
Bureau A13 (RDC du LIFO).
($\lambda x. \lambda y. x @ y$) jolan.philippe1@univ-orleans.fr

Travaux de recherche: vérification formelle de

- Programmes parallèles distribués
- Transformation de modèles dans les SI
- Déploiement d'infrastructure

Plus de détails sur <https://jolanphilippe.github.io/>



Nicolas Hiot, ATER
Université d'Orléans, équipe Pamda
Bureau B07 (1er étage du LIFO).
(λx.λy.x@y) nicolas.hiot univ-orleans.fr

Ce qu'on va aborder dans ce cours

La philosophie DevOps

- C'est quoi Dev ? C'est quoi Ops ?
- C'est quoi DevOps ?
- Un context particulier : le Cloud

Infrastructure-as-code

- Conteneurisation avec Docker
- Provisionnement avec Terraform
- Configuration et gestion avec Ansible
- Orchestration avec Docker Swarm

DevOps késako ?

Cycle de développement en cascade

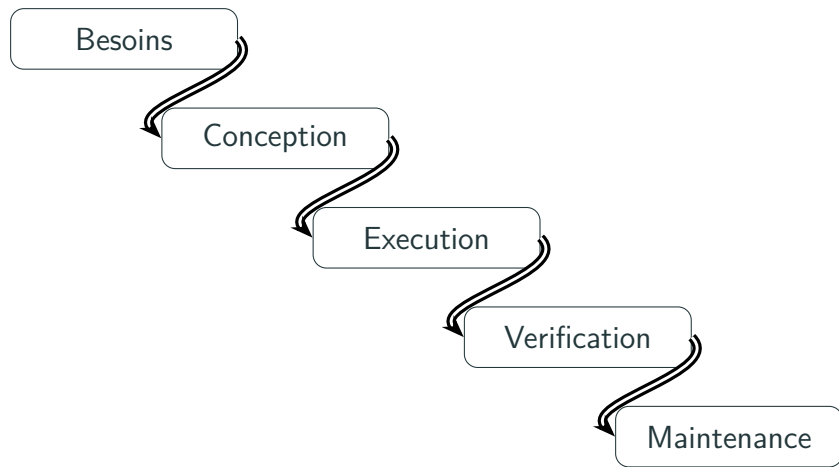
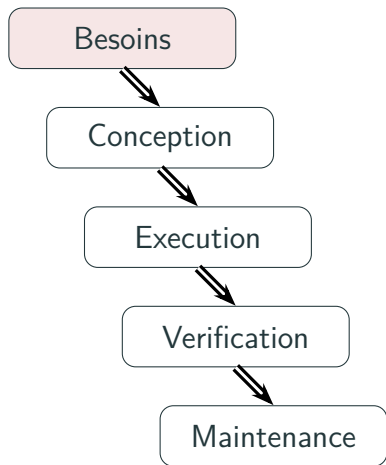


Figure 1: Développement logiciel agile

Cycle de développement en cascade - Exemple : Développer une appli CampusUO

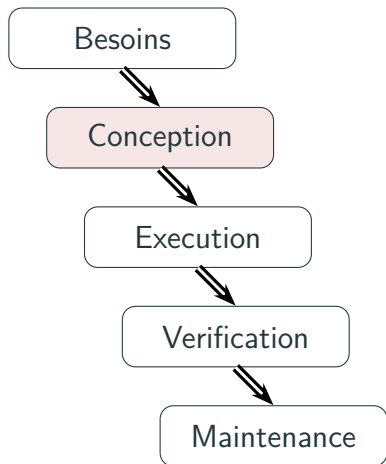


Besoins

- Compte individuel par étudiant
- Plan interactif du campus avec GPS
- Intégration des services Univ. Orléans (ADE, emails, Celene)
- Dossier scolarité avec docs (attestation étudiant, carte univ)

Figure 2: Développement logiciel agile

Cycle de développement en cascade - Exemple : Développer une appli CampusUO

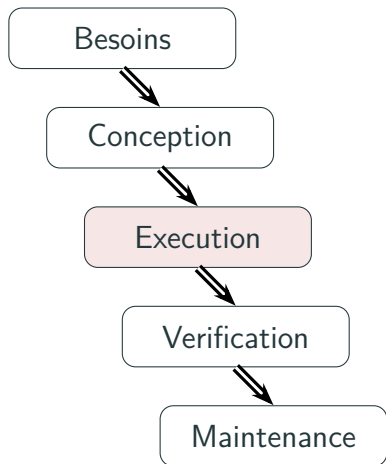


Conception

- Modélisation de l'application orientée services
- Modélisation de la base de données
- Liste des fonctionnalités par service
- Conception des vues et design UI/UX
- Architecture MVC, interaction entre les composants

Figure 3: Développement logiciel agile

Cycle de développement en cascade - Exemple : Développer une appli CampusUO

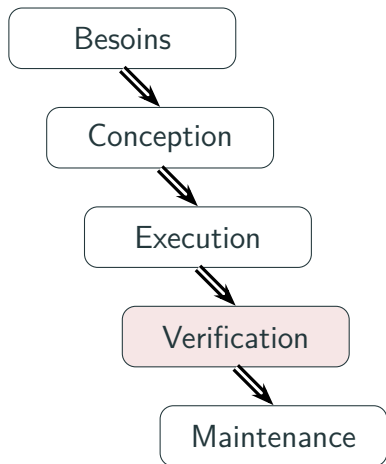


Execution

- Coder les services et les APIs
- Coder les vues
- Faire une intégration entre les composants MVC
- Déployer l'application sur un serveur

Figure 4: Développement logiciel agile

Cycle de développement en cascade - Exemple : Développer une appli CampusUO

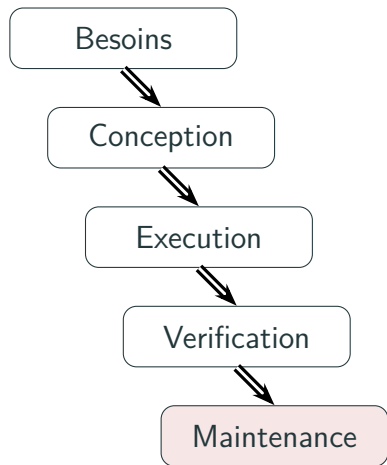


Verification

- Tests unitaires sur la partie modèle; services; etc
- Tests d'intégration entre les services
- Tests end-to-end avec l'application complète
- Tests de charge sur l'application déployée
- etc.

Figure 5: Développement logiciel agile

Cycle de développement en cascade - Exemple : Développer une appli CampusUO



Maintenance

- Mise à jour des protocoles (e.g., HTTP 1.1 vers HTTP 2)
- Debugger

Figure 6: Développement logiciel agile

C'est trop **monolithique**

Chaque mise à jour nécessite de reparcourir la cascade

Personne ne code sans bug

Les tests arrivent trop tardivement, et non incrémentalement

Windows 10 - Update causant la suppression de fichier

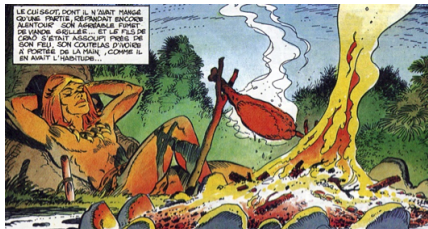
- **Problème:** patch effaçant des fichiers utilisateurs lors de redirections de dossiers
- **Prévention:**
 - Automatiser les tests, notamment en mockant la migration
 - Déploiement progressif avec surveillance d'état du système

Firefox - Expiration d'un certificat désactivant les extensions

- **Problème:** expiration d'un certificat de signature d'extensions
- **Prévention:**
 - Automatiser des tests réguliers avec utilisation d'une horloge
 - Surveillance automatisé des dates d'expirations avec des patches automatiques

Test-First Design

- 12000 av JC: OgTheVenerable, tenant un morceau de viande crue, il décrit à ses compagnons préhistoriques comment faire du feu, définissant ainsi leurs attentes et la marche à suivre pour cette tâche. Le morceau de viande carbonisée qu'il tenait ensuite à la main est généralement considéré comme le premier exemple réussi.



Test-First Design

- 10/12/1815 – 17/11/1852: Ada Lovelace crée involontairement le besoin de développement piloté par les tests: Elle a utilisé les informations et les formules fournies par Babbage pour calculer (e.g., polynomes, nombres de Bernouilli). Elle pose les resultats d'abord, donne les resultats des operations intermediaire, et a déterminé où les calculs seraient effectués par la machine et où les résultats seraient affichés.

<http://www.bookrags.com/research/lovelace-ada-byron-king-countess-of-csci-01/>



Test-First Design

- dans les années 50 : Les programmeurs de la NASA du programme spatial Mercury utilisent une forme de test-first design pour programmer des cartes perforées. Ils programment des résultats de scénario connus (trajectoire, vitesse, altitude, etc.) et comparaient les valeurs avec les résultats des programmes.



Test Driven Development (Kent Beck, 1999)

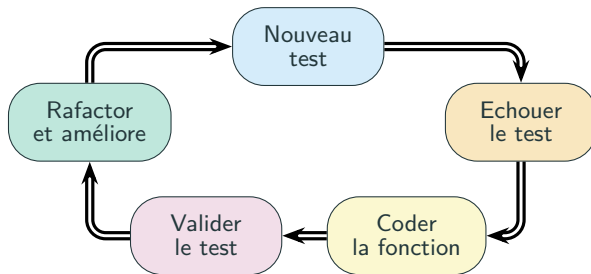


Figure 7: Cycle de Test Driven Development

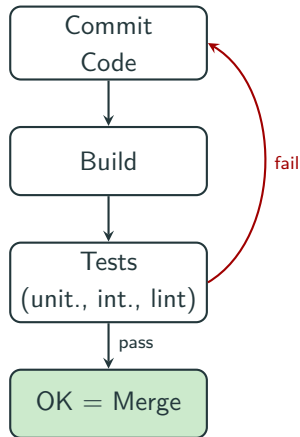
Les trois lois ~~de la robotique~~ du TDD - Robert C. Martin

- "You may not write production code until you have written a failing unit test"
- "You may not write more of a unit test than is sufficient to fail, and not compiling is failing."
- "You may not write more production code than is sufficient to pass the currently failing test."

Intégration continue et tests automatisés

Qu'est-ce que la CI ?

- Fusionner le code **très fréquemment** (plusieurs fois/jour).
- Chaque commit déclenche **build + tests automatiques**.
- But : **détecter tôt** les erreurs et stabiliser le produit.



Prépare la suite : tests au déploiement.

Pourquoi automatiser les tests ?

- **Réduire les régressions** liées aux patches/maintenance.
- **Accélérer le feedback** aux développeurs.
- **Standardiser la qualité** (critères objectifs dans la pipeline).
- **Déployer plus souvent** avec confiance (vers CI/CD).

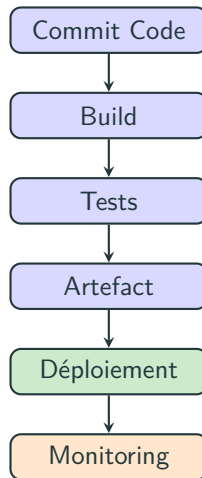
Tester au déploiement avec Jenkins

Jenkins

- **But** : valider automatiquement chaque changement de code
- **Déclencheurs** : webhooks Git (push/PR)
- **Boucle de feedback** : rapide, standardisée, versionnée.

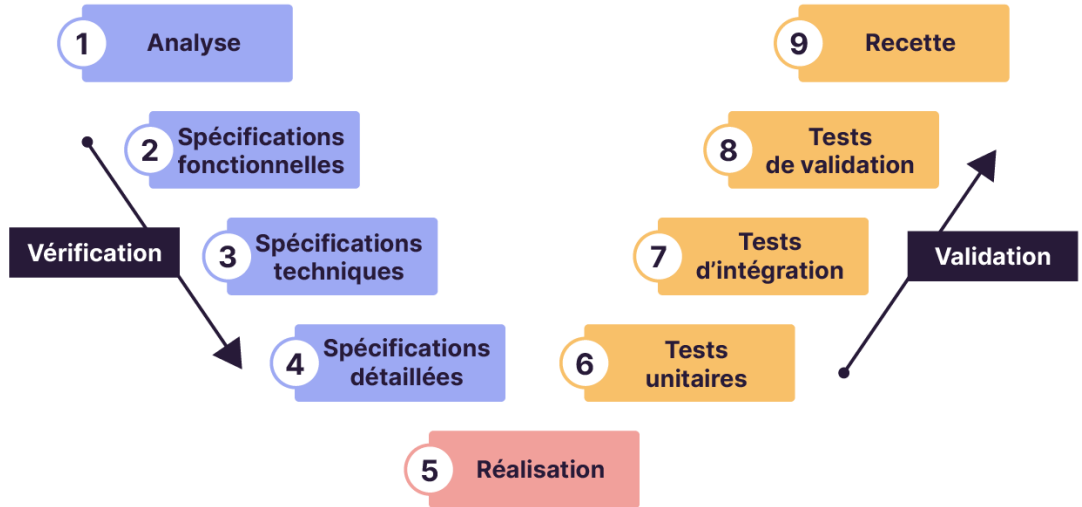
Automatiser la CI

- **Build** reproductible
- **Tests** automatisés (unitaires, intégration, lint, ...)
- **Qualité** : seuils (tests, couverture, lint)
- **Artefacts** : packaging, signature, dépôt binaire
- **Rapports** : notifications (chat/mail)



Jenkins orchestre chaque étape du pipeline.

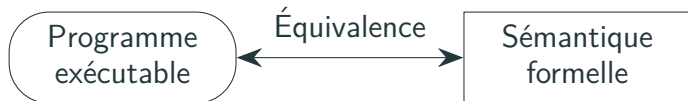
Le cycle en V



Correction par construction



Correction à posteriori



*Program testing can be used to show the presence of bugs,
but never to show their absence. - E. W. Dijkstra*

Qu'est-ce que Ops ?

- La partie **Operations** du cycle DevOps.
- Responsable de la **mise en production**, du **monitoring**, de la **sécurité**.
- Objectif : **assurer la disponibilité et la fiabilité** du service.

Missions clés

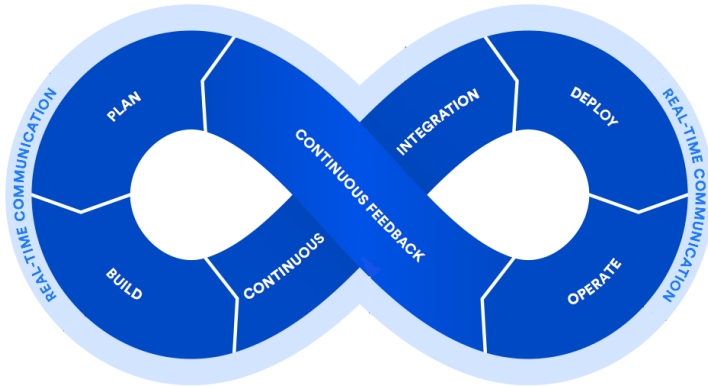
- Déploiement et exploitation des applications.
- Supervision, alertes, gestion des incidents.
- Automatisation des tâches d'infra (Infra as Code, scripts).

Pourquoi automatiser le déploiement ?

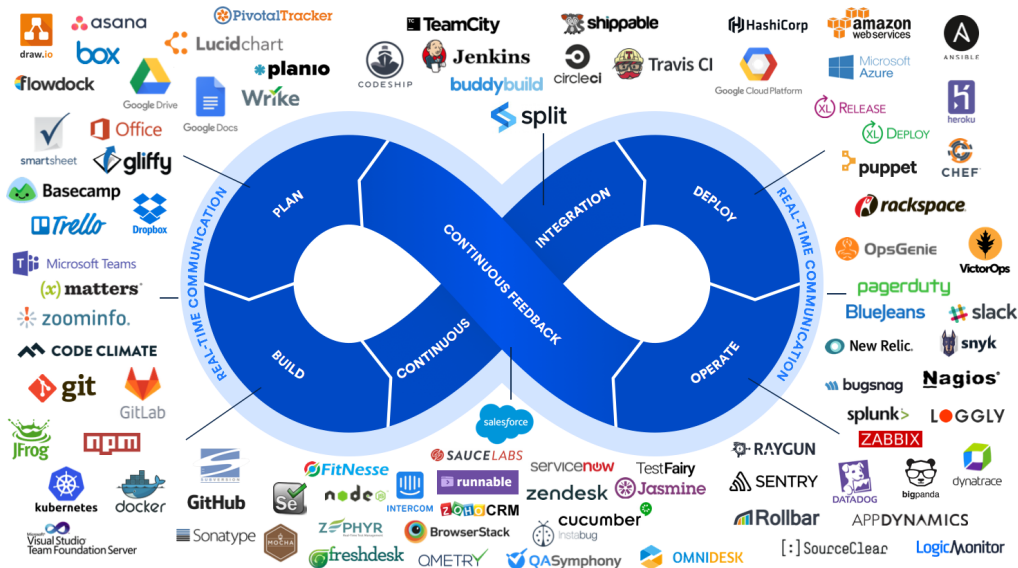
- **Réduire les erreurs humaines** liées aux procédures manuelles.
- **Déployer rapidement et fréquemment** en production.
- **Uniformiser** les environnements.
- **Sécuriser les mises en production** avec rollback rapide.

Le CD rend le déploiement **fiable, répétable et rapide**, en s'appuyant sur l'automatisation et l'intégration avec la CI.

Mission du DevOps : Intégration continue et déploiement continu (CI/CD)



Mission du DevOps : Intégration continue et déploiement continu (CI/CD)



`https://roadmap.sh/devops`

Infrastructure-as-code

Architecture orientée services (SOA)

Définition

- **Approche d'architecture logicielle** organisée autour de **services**.
- Un service = une **fonctionnalité autonome** exposée via une interface standard.
- Communication via des protocoles réseau (SOAP, REST, gRCP/RMI, messages).

Architecture orientée services (SOA)

Définition

- **Approche d'architecture logicielle** organisée autour de **services**.
- Un service = une **fonctionnalité autonome** exposée via une interface standard.
- Communication via des protocoles réseau (SOAP, REST, gRCP/RMI, messages).

Principes

- **Réutilisabilité** : services mutualisés dans plusieurs applications.
- **Interopérabilité** : indépendants des langages et plateformes.
- **Faible couplage** : évolution indépendante des services.
- **Composition** : services combinés pour former des processus métiers.

Utilisation dans le cloud

- Les services sont **déployés et consommés à la demande**.
- Mise en œuvre via des **APIs exposées en ligne**.
- Intégration fréquente avec des **microservices** et du **serverless**.
- Supportée par des **plateformes cloud** (providers) : AWS, Azure, GCP.

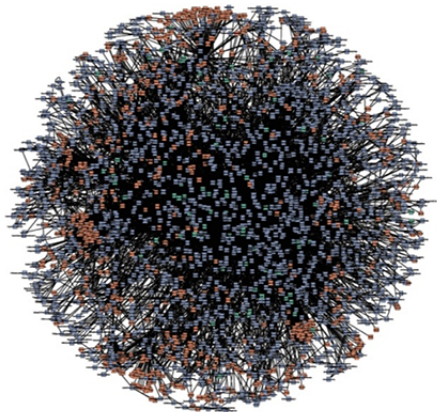
Utilisation dans le cloud

- Les services sont **déployés et consommés à la demande**.
- Mise en œuvre via des **APIs exposées en ligne**.
- Intégration fréquente avec des **microservices** et du **serverless**.
- Supportée par des **plateformes cloud** (providers) : AWS, Azure, GCP.

Avantages pour le cloud

- **Élasticité** : montée en charge dynamique des services.
- **Scalabilité mondiale** : services accessibles partout.
- **Agilité accrue** : déploiement rapide de nouveaux services.
- **Coût à l'usage** : facturation en fonction de la consommation.

Réalité: Des Deathstars



amazon.com



Que faut-il faire ?

- Préparer **ses services** pour fonctionner **de façon atomique**
- Préparer **ses ressources** (nœuds physiques/virtuels, VM/containers, réseau, règles de sécurité, secrets/identités).
- **Installer, configurer et démarrer** les services sur ces ressources .
- **Adapter l'infrastructure** aux événements : autoscaling, auto-healing, etc.

Challenges

- **Gérer les dépendances** : induit un ordre partiel sur les opérations
- **Déployer sans faute** : Avoir des opérations de déploiement atomiques (déployer, mise à jour, supprimer) sûres.
- **Robustesse** : tolérance aux pannes, passage à l'échelle, rollbacks rapides, etc.
- **Sécurité** : secrets/identités, durcissement, moindre privilège, conformité.
- **Impact** : coûts, empreinte carbone, dette opérationnelle, incidents clients.

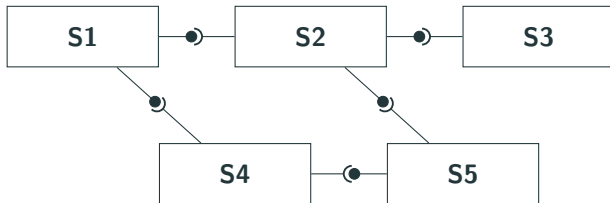
Gérer les dépendances



Dans ce cas, Service 1 expose son API, qui est utilisée par Service 2.

Service 1 fournit des informations, un service, etc. via son port.

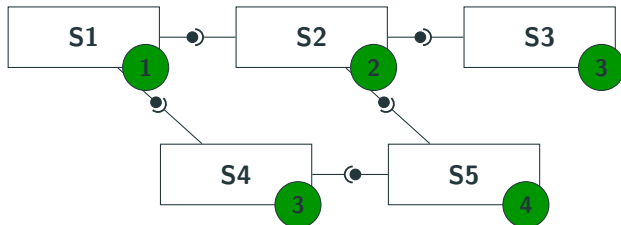
On parle de provide port pour Service 1 et de use port pour Service 2



Dépendances

- **S2** dépend de **S1**
- **S3** dépend de **S2**
- **S4** dépend de **S1** et **S5**
- **S5** dépend de **S2**

Déployer sans faute



Relation d'ordre partiel strict

- Deploy **S1** > Deploy **S2**
- Deploy **S2** > Deploy **S3**
- Deploy **S2** > Deploy **S5**
- Deploy **S4** > Deploy **S1**
- Deploy **S4** > Deploy **S5**

Exemple de trace: Deploy **S1**; Deploy **S2**; Deploy **S5**; Deploy **S4**; Deploy **S3**;

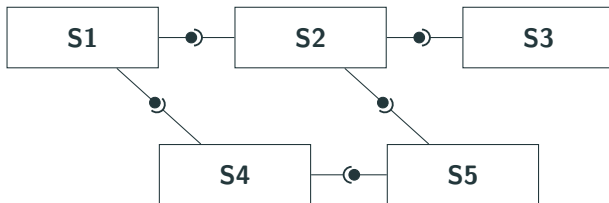
Rappel

- **Irréflexivité** : $\forall a \in E, \neg(a < a)$
- **Transitivité** : $\forall a, b, c \in E, (a < b \wedge b < c) \Rightarrow a < c$
- **Asymétrie** : $\forall a, b \in E, (a < b) \Rightarrow \neg(b < a)$

Replace sans faute

Scenario: Replace **S2**

C'est à dire : Destroy **S2** ; Deploy **S2**



Deploy

- Deploy **S1** > Deploy **S2**
- Deploy **S2** > Deploy **S3**
- Deploy **S2** > Deploy **S5**
- Deploy **S4** > Deploy **S1**
- Deploy **S4** > Deploy **S5**

Destroy

- Destroy **S1** < Destroy **S2**
- Destroy **S1** < Destroy **S4**
- Destroy **S2** < Destroy **S3**
- Destroy **S2** < Destroy **S5**
- Destroy **S5** < Destroy **S4**

Replace sans faute: Graphe de dépendance

$\forall S$, Replace **S** := Destroy **S**; Deploy **S**

Destroy

- Destroy **S1** < Destroy **S2**
- Destroy **S2** < Destroy **S3**
- Destroy **S2** < Destroy **S5**
- Destroy **S5** < Destroy **S4**
- Destroy **S1** < Destroy **S4**

Deploy

- Deploy **S1** > Deploy **S2**
- Deploy **S2** > Deploy **S3**
- Deploy **S2** > Deploy **S5**
- Deploy **S5** > Deploy **S4**
- Deploy **S4** > Deploy **S1**

Replace **S2**



Replace sans faute: Graphe de dépendance

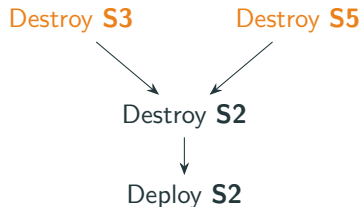
$\forall S$, Replace **S** := Destroy **S**; Deploy **S**

Destroy

- Destroy **S1** < Destroy **S2**
- Destroy **S2** < Destroy **S3**
- Destroy **S2** < Destroy **S5**
- Destroy **S5** < Destroy **S4**
- Destroy **S1** < Destroy **S4**

Deploy

- Deploy **S1** > Deploy **S2**
- Deploy **S2** > Deploy **S3**
- Deploy **S2** > Deploy **S5**
- Deploy **S5** > Deploy **S4**
- Deploy **S4** > Deploy **S1**



Replace sans faute: Graphe de dépendance

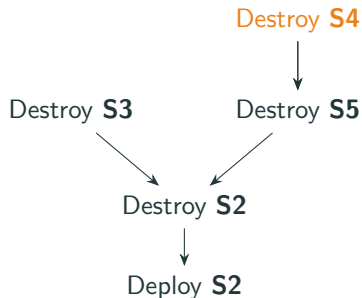
$\forall S$, Replace **S** := Destroy **S**; Deploy **S**

Destroy

- Destroy **S1** < Destroy **S2**
- Destroy **S2** < Destroy **S3**
- Destroy **S2** < Destroy **S5**
- Destroy **S5** < Destroy **S4**
- Destroy **S1** < Destroy **S4**

Deploy

- Deploy **S1** > Deploy **S2**
- Deploy **S2** > Deploy **S3**
- Deploy **S2** > Deploy **S5**
- Deploy **S5** > Deploy **S4**
- Deploy **S4** > Deploy **S1**



Replace sans faute: Graphe de dépendance

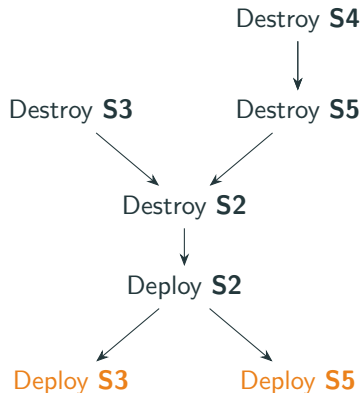
$\forall \mathbf{S}$, Replace $\mathbf{S} := \text{Destroy } \mathbf{S}; \text{Deploy } \mathbf{S}$

Destroy

- Destroy $\mathbf{S1} < \text{Destroy } \mathbf{S2}$
- Destroy $\mathbf{S2} < \text{Destroy } \mathbf{S3}$
- Destroy $\mathbf{S2} < \text{Destroy } \mathbf{S5}$
- Destroy $\mathbf{S5} < \text{Destroy } \mathbf{S4}$
- Destroy $\mathbf{S1} < \text{Destroy } \mathbf{S4}$

Deploy

- Deploy $\mathbf{S1} > \text{Deploy } \mathbf{S2}$
- Deploy $\mathbf{S2} > \text{Deploy } \mathbf{S3}$
- Deploy $\mathbf{S2} > \text{Deploy } \mathbf{S5}$
- Deploy $\mathbf{S5} > \text{Deploy } \mathbf{S4}$
- Deploy $\mathbf{S4} > \text{Deploy } \mathbf{S1}$



Replace sans faute: Graphe de dépendance

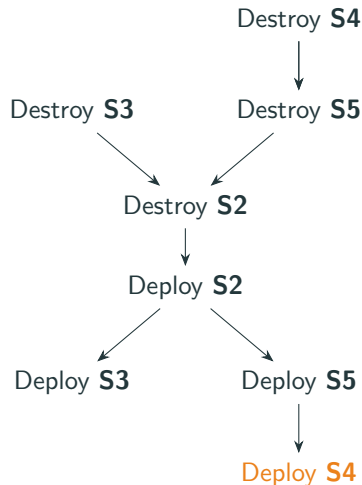
$\forall \mathbf{S}$, Replace $\mathbf{S} := \text{Destroy } \mathbf{S}; \text{Deploy } \mathbf{S}$

Destroy

- Destroy $\mathbf{S1} < \text{Destroy } \mathbf{S2}$
- Destroy $\mathbf{S2} < \text{Destroy } \mathbf{S3}$
- Destroy $\mathbf{S2} < \text{Destroy } \mathbf{S5}$
- Destroy $\mathbf{S5} < \text{Destroy } \mathbf{S4}$
- Destroy $\mathbf{S1} < \text{Destroy } \mathbf{S4}$

Deploy

- Deploy $\mathbf{S1} > \text{Deploy } \mathbf{S2}$
- Deploy $\mathbf{S2} > \text{Deploy } \mathbf{S3}$
- Deploy $\mathbf{S2} > \text{Deploy } \mathbf{S5}$
- Deploy $\mathbf{S5} > \text{Deploy } \mathbf{S4}$
- Deploy $\mathbf{S4} > \text{Deploy } \mathbf{S1}$



Replace sans faute: Graphe de dépendance

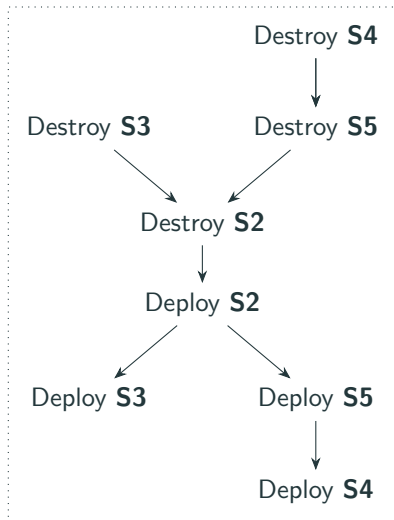
$\forall S$, Replace **S** := Destroy **S**; Deploy **S**

Destroy

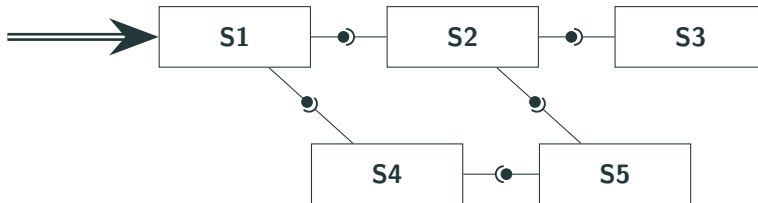
- Destroy **S1** < Destroy **S2**
- Destroy **S2** < Destroy **S3**
- Destroy **S2** < Destroy **S5**
- Destroy **S5** < Destroy **S4**
- Destroy **S1** < Destroy **S4**

Deploy

- Deploy **S1** > Deploy **S2**
- Deploy **S2** > Deploy **S3**
- Deploy **S2** > Deploy **S5**
- Deploy **S5** > Deploy **S4**
- Deploy **S4** > Deploy **S1**



Graphe de dépendance pour Replace **S2**



Robustesse

- **Déclencheur** : Un service (e.g., S2) tombe
- **Action** : Ce service est remplacé

Sécurité

- L'utilisateur n'a accès qu'au service façade (e.g., S1)
- Les services permettent d'utiliser JRMP (Java Remote Method Protocol) entre eux

Élaborer des stratégies

Pour avoir des impacts

- Qualité de service (DevOps)
- Financier (FinOps)
- Sécurité (SecOps)
- Énergétique (GreenOps)

Exemple

Une migration d'une partie de l'infrastructure

- 1 Garder deux environnements et basculer progressivement les services.
⇒ Coûteux financièrement mais assure une continuité de service
- 2 Supprimer l'ancien environnement progressivement et remplacer les ressources au fur et à mesure
⇒ Peu réduire le coût financier et énergétique au prix d'une interruption de service

Challenges

- **Gérer les dépendances** : induit un ordre partiel sur les opérations
- **Déployer sans faute** : Avoir des opérations de déploiement atomiques (déployer, mise à jour, supprimer) sûres.
- **Robustesse** : tolérance aux pannes, passage à l'échelle, rollbacks rapides, etc.
- **Sécurité** : secrets/identités, durcissement, moindre privilège, conformité.
- **Impact** : coûts, empreinte carbone, dette opérationnelle, incidents clients.

⇒ **Automatiser** avec l'**infrastructure-as-code**

Définition

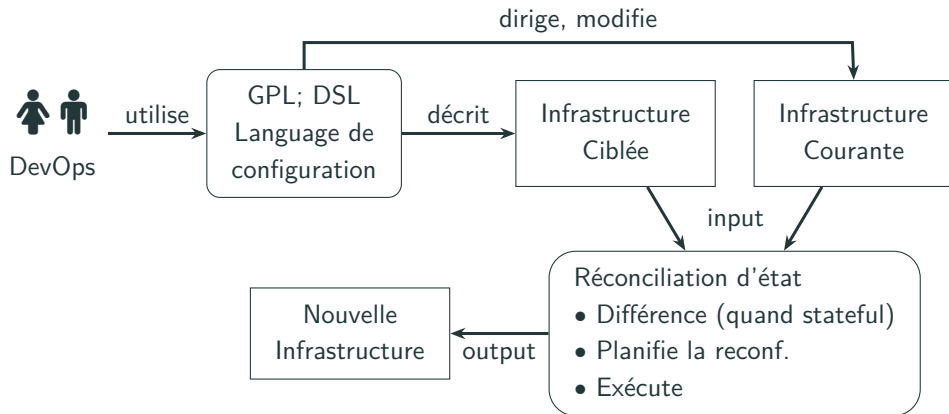
- **Infrastructure as Code** = pratique consistant à **décrire et gérer l'infrastructure** (réseau, serveurs, VM, containers, règles) au moyen de code en utilisant
 - des **langages génériques**, ou GPL (General Purpose Language), avec des bibliothèques
 - des **langages dédiés**, ou DSL (Domain Specific Language)
 - des **langages de configuration** (YAML ou JSON par exemple)
- L'infrastructure est donc **déclarative et reproductible**, plutôt que créée manuellement.

IaC et DevOps : comment ça marche ?

Dans la pratique

- Le code d'infrastructure est **stocké dans Git**, versionné comme le code applicatif.
- Les outils (Terraform, Ansible, Pulumi, CloudFormation, etc.) **appliquent le code** pour provisionner ou mettre à jour les ressources.
- Intégration dans les **pipelines CI/CD** : validation, tests, déploiement automatisé.
- Résultat : la création/évolution de l'infra suit le même cycle que le code des Devs
plan → *review* → *merge* → *apply*

IaC et DevOps : comment ça marche ?



Virtualization et Conteneurisation

Customiser, configurer
tester l'application
et la conteneuriser

Virtualization et Conteneurisation

Customiser, configurer
tester l'application
et la conteneuriser

Provisionnement d'infrastructure

Demander ressources
physiques ou virtuelles;
configurer le réseau;
et règles sécurité

Virtualization et Conteneurisation

Customiser, configurer
tester l'application
et la conteneuriser

Provisionnement d'infrastructure

Demander ressources
physiques ou virtuelles;
configurer le réseau;
et règles sécurité

Installation et Configuration

Installer les services
(app + deps)
configurer les services;
et les intégrer

Virtualization et Conteneurisation

Customiser, configurer
tester l'application
et la conteneuriser

Provisionnement d'infrastructure

Demander ressources
physiques ou virtuelles;
configurer le réseau;
et règles sécurité

Installation et Configuration

Installer les services
(app + deps)
configurer les services;
et les intégrer

Orchestration de cycle de vie

Upgrades auto;
Backup et recovery;
Surveillance;
Passage à l'échelle

Les rôles des outils d'IaC... Dans un exemple concret

On veut une **base de données maitre**, et **des réplicats** de cette base de données. Ces services sont chacun sur **une VM différente, sur un noeud différent**. Cependant, tous ces **noeuds sont connecté en réseau**. Dans le cas où un **réplicat tombe en panne, il faut en redéployer un**.

Les rôles des outils d'IaC... Dans un exemple concret

On veut une **base de données maitre**, et **des réplicats** de cette base de données. Ces services sont chacun sur **une VM différente, sur un noeud différent**. Cependant, tous ces **noeuds sont connecté en réseau**. Dans le cas où un **réplicat tombe en panne, il faut en redéployer un**.

**Virtualization et
Conteneurisation**

**Provisionnement
d'infrastructure**

**Installation et
Configuration**

**Orchestration
de cycle de vie**

Les rôles des outils d'IaC... Dans un exemple concret

On veut une **base de données maitre**, et **des réplicats** de cette base de données. Ces services sont chacun sur **une VM différente, sur un noeud différent**. Cependant, tous ces **noeuds sont connecté en réseau**. Dans le cas où un **réplicat tombe en panne, il faut en redéployer un**.

Virtualization et Conteneurisation

Créer un conteneur
(Docker) pour démarrer
une base de données

Provisionnement d'infrastructure

Installation et Configuration

Orchestration de cycle de vie

Les rôles des outils d'IaC... Dans un exemple concret

On veut une **base de données maitre**, et **des réplicats** de cette base de données. Ces services sont chacun sur **une VM différente, sur un noeud différent**. Cependant, tous ces **noeuds sont connecté en réseau**. Dans le cas où un **réplicat tombe en panne, il faut en redéployer un**.

Virtualization et Conteneurisation

Créer un conteneur
(Docker) pour démarrer
une base de données

Provisionnement d'infrastructure

Créer un VPC (subnet);
Demander des noeuds;
Définir des VMs / cluster;
Firewall

Installation et Configuration

Orchestration de cycle de vie

Les rôles des outils d'IaC... Dans un exemple concret

On veut une **base de données maitre**, et **des réplicats** de cette base de données. Ces services sont chacun sur **une VM différente, sur un noeud différent**. Cependant, tous ces **noeuds sont connecté en réseau**. Dans le cas où un **réplicat tombe en panne, il faut en redéployer un**.

Virtualization et Conteneurisation

Créer un conteneur
(Docker) pour démarrer
une base de données

Provisionnement d'infrastructure

Créer un VPC (subnet);
Demander des noeuds;
Définir des VMs / cluster;
Firewall

Installation et Configuration

Démarrer les conteneurs
sur les VMs; configurer
et bootstrapper les BDDs;
intégrer master et réplicats

Orchestration de cycle de vie

Les rôles des outils d'IaC... Dans un exemple concret

On veut une **base de données maitre**, et **des réplicats** de cette base de données. Ces services sont chacun sur **une VM différente, sur un noeud différent**. Cependant, tous ces **noeuds sont connecté en réseau**. Dans le cas où un **réplicat tombe en panne, il faut en redéployer un**.

Virtualization et Conteneurisation	Provisionnement d'infrastructure	Installation et Configuration	Orchestration de cycle de vie
Créer un conteneur (Docker) pour démarrer une base de données	Créer un VPC (subnet); Demander des noeuds; Définir des VMs / cluster; Firewall	Démarrer les conteneurs sur les VMs; configurer et bootstrapper les BDDs; intégrer master et réplicats	Configurer un trigger: détecter une panne Configurer une action: redéployer une BDD

Les rôles des outils d'IaC... Dans une cuisine ?

Virtualization et Conteneurisation

Former les cuisiniers:
leur apprendre des
recettes, etc.

Provisionnement d'infrastructure

Fournir plan de travail,
robots de cuisine,
four, etc. et les
assembler.

Installation et Configuration

Assigner les cuisiniers
à des postes, et à des
tâches; leur donner des
outils de coms; etc.

Orchestration de cycle de vie

Avoir un chef de brigade
prêt à changer l'organisation
pour plus d'efficacité, ou
répondre à un imprévu



Pourquoi adopter l'IaC ?

- **Interopérable** : fonctionne sur multi-cloud et on-premise.
- **Réutilisable** : modules et templates factorisables.
- **Versionnable** : suivi dans Git, rollback possible.
- **Automatisable** : intégré aux pipelines CI/CD.
- **Traçable et auditable** : chaque changement est explicite.
- **Scalable et idempotent** : état final garanti, même à grande échelle.
- **Auto-documentant** : le code décrit l'infrastructure.

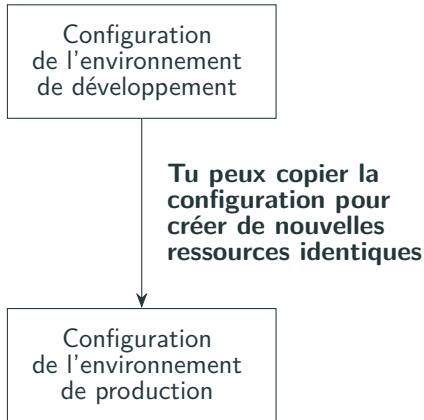
Quatres piliers

- **Reproductibilité** : Peut-on utiliser la même configuration pour reproduire un environnement ou des ressources d'infrastructure ?
- **Idempotence** : Peut-on exécuter à plusieurs reprises l'automatisation sur l'infrastructure sans modifier son état actuel ?
- **Composabilité** : Peut-on assembler n'importe quelle combinaison de ressources d'infrastructure sans reconstruire un nouveau système ?
- **Evolutivité** : Peut-on modifier votre infrastructure et minimiser les perturbations du système global ?

Avec les solutions d'Infrastructure as Code : OUI

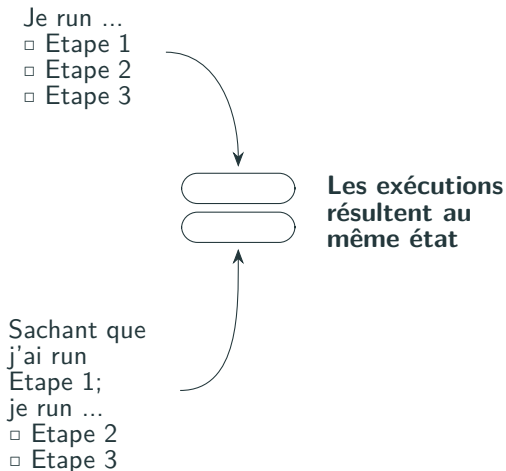
Reproductibilité

- Peut-on utiliser la même configuration pour reproduire un environnement ou des ressources d'infrastructure ?



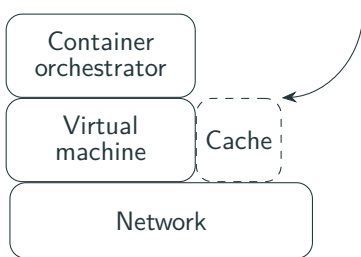
Idempotence

- Peut-on exécuter à plusieurs reprises l'automatisation sur l'infrastructure sans modifier son état actuel ?



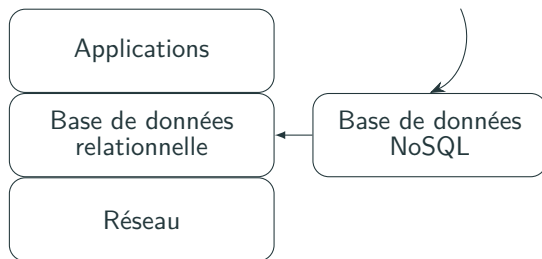
- Peut-on assembler n'importe quelle combinaison de ressources d'infrastructure sans reconstruire un nouveau système ?

**Tu peux ajouter
un cache au
système facilement**



- Peut-on modifier votre infrastructure et minimiser les perturbations du système global ?

Tu peux échanger une base de données et minimiser la perturbation de l'application



Technologie d'Infrastructure as Code

Technology	Conteneurisation	Provisioning	Conf. Mgmt	Cluster. Mgmt	Target
CONTAINERD	✓	✗	✗	✗	Agnostic
DOCKER	✓	✗	✗	✗	Agnostic
LXC	✓	✗	✗	✗	Dedicated (Linux)
PODMAN	✓	✗	✗	✗	Agnostic
CRI-O	✓	✗	✗	✗	Dedicated (Kubernetes)
AZURE RESOURCE MANAGER	✗	✓✓	✓	✓	Dedicated (Azure)
CLOUDFORMATION	✗	✓✓	✓	✓	Dedicated (AWS)
HEAT	✗	✓✓	✓	✓	Dedicated (OpenStack)
PULUMI	✗	✓✓	✓	✗	Agnostic
TERRAFORM	✗	✓✓	✓	✗	Agnostic
ANSIBLE	✗	✓	✓✓	✗	Agnostic
CFENGINE	✗	✓	✓✓	✗	Agnostic
CHEF	✗	✓	✓✓	✗	Agnostic
PUPPET	✗	✓	✓✓	✗	Agnostic
SALTSTACK	✗	✓	✓✓	✗	Agnostic
KUBERNETES	✗	✓✓	✗	✓✓	Hybrid (Containers)
NOMAD	✗	✓✓	✗	✓✓	Agnostic
DOCKER SWARM	✗	✓✓	✓	✓✓	Dedicated (Docker)
JUJU	✗	✓✓	✓✓	✓✓	Agnostic
TOSCA	✗	✓✓	✓✓	✓✓	Agnostic

Intégration continue / Déploiement continu (CI/CD)

- **CI** : build + tests à chaque commit.
- **CD** : déploiement automatisé

DevOps

- Dev = **garant de l'évolution du service** (innovation, nouvelles fonctionnalités, correction de bugs, qualité logicielle)
- Ops = **garant de la continuité de service** (disponibilité, robustesse, supervision), en lien étroit avec les Devs

Architecture orientée services (SOA)

- Approche modulaire: un service = une fonctionnalité
- Utilisation d'API pour composer
- Difficile à gérer à cause de la taille

Infrastructure-as-code

- Solution pour déployer sous forme de code
 - Préparer son application (conteneurisation)
 - Préparer ses ressources (provisionnement)
 - Installer son application (gestion de configuration)
 - Gérer des événements dans son infrastructure (orchestration)
- Reproductible, idempotent, composable et évolvable.

