MAGELLAN

POLYTECH TOURS · COURS · 2026

CI / CD

De l'intégration au déploiement continu

Pipelines · cycle de vie d'une release · stratégies de déploiement · culture DevOps

Andrey Pividori
Intervenant Magellan · Cycle ingénieur 4e & 5e année

INTERVENANT

Qui suis-je ?

Andrey Pividori

Directeur de projets IT

Pilotage · budget · gouvernance QSRC
(Qualité, Sécurité, Risque, Conformité)

Intervenant Magellan

AU QUOTIDIEN

- **Pilotage de projets -** roadmap, jalons et tenue des délais.
- **Budget & risques -** suivi financier et maîtrise des risques.
- **Gouvernance QSRC -** qualité, sécurité, conformité, audits.

POURQUOI CE COURS

- **Un retour d'expérience terrain -** comment les organisations industrialisent leurs industrialisent leurs livraisons, en gardant la maîtrise du risque.
- **Un pont théorie ↔ pratique -** le cours pose les concepts, le TP les fait vivre.

ORGANISATION

Comment se déroule ce cours

4 h

Cours magistral

Concepts, schémas, échanges

4 h

Travaux pratiques

Construire sa propre pipeline

7+1

Modules

7 modules + annexe Git & sécurité

Évaluation : note globale basée sur la restitution des TP. Le support de cours est interactif et téléchargeable en ligne (PDF), ligne (PDF), n'hésitez pas à le rouvrir pendant le TP.

SOMMAIRE

Le programme en 7 modules

OBJECTIFS PÉDAGOGIQUES

À la fin de ce cours, vous saurez…

Distinguer CI, Continuous Delivery & Deployment

et expliquer l'utilité d'une pipeline.

Décrire le cycle de vie d'une release

du développement au déploiement, niveaux de recette inclus.

Comparer les stratégies de déploiement

rolling, blue/green, canary, feature flags.

Comprendre les architectures actif/passif & actif/actif

et leur lien direct avec la livraison.

Situer les méthodes de dev et mesurer la performance

via les quatre métriques DORA.

MODULE 1 · 15 MIN

Introduction & objectifs

01

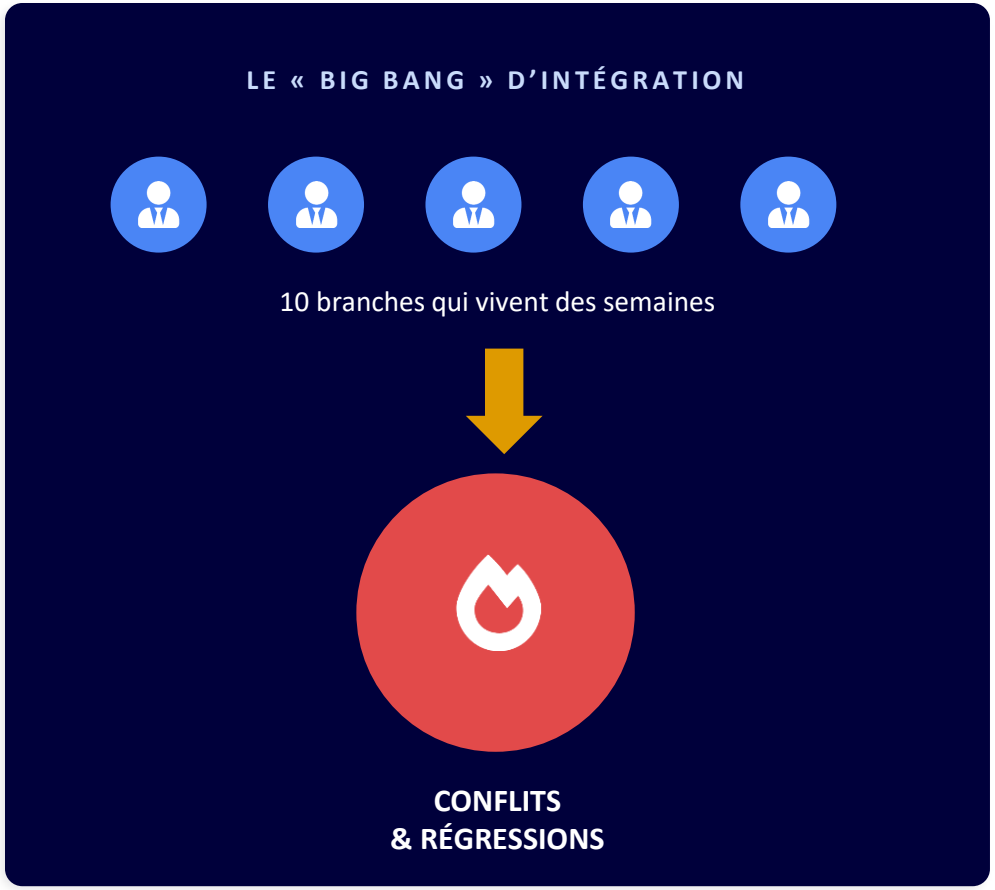
Pourquoi la CI/CD existe

15 min

L'enfer de l'intégration

- **Il y a 20 ans,** 10 développeurs travaillent chacun de leur côté pendant 3 mois sur leur copie du code.
- **Le jour de l'integration :** on rassemble tout le monde. Des milliers de lignes en conflit, des comportements incompatibles.
- **La mise en production :** un vendredi soir, à plusieurs, en croisant les doigts.

« integration hell » : des mois de travail isolé qui s'effondrent une fois une fois réunis.



Un modèle qui coûte cher

Le coût du délai

Des mois entre le code écrit et la valeur livrée à l'utilisateur.

Le coût du risque

Plus on accumule de changements, plus la livraison est grosse… et risquée. Quelle modif, parmi des centaines, a causé la panne ?

Le coût humain

La peur de déployer. On déploie moins → les lots grossissent → le risque monte → la peur monte.

Un cercle vicieux : déployer fait peur → on déploie moins souvent → les lots grossissent → le risque augmente → la peur augmente. Et ainsi de suite.

L'IDÉE FONDATRICE

« Si quelque chose fait mal, faites-le plus souvent. »

L'intégration fait mal ? Intégrez en continu, plusieurs fois par jour, par tout petits incréments.

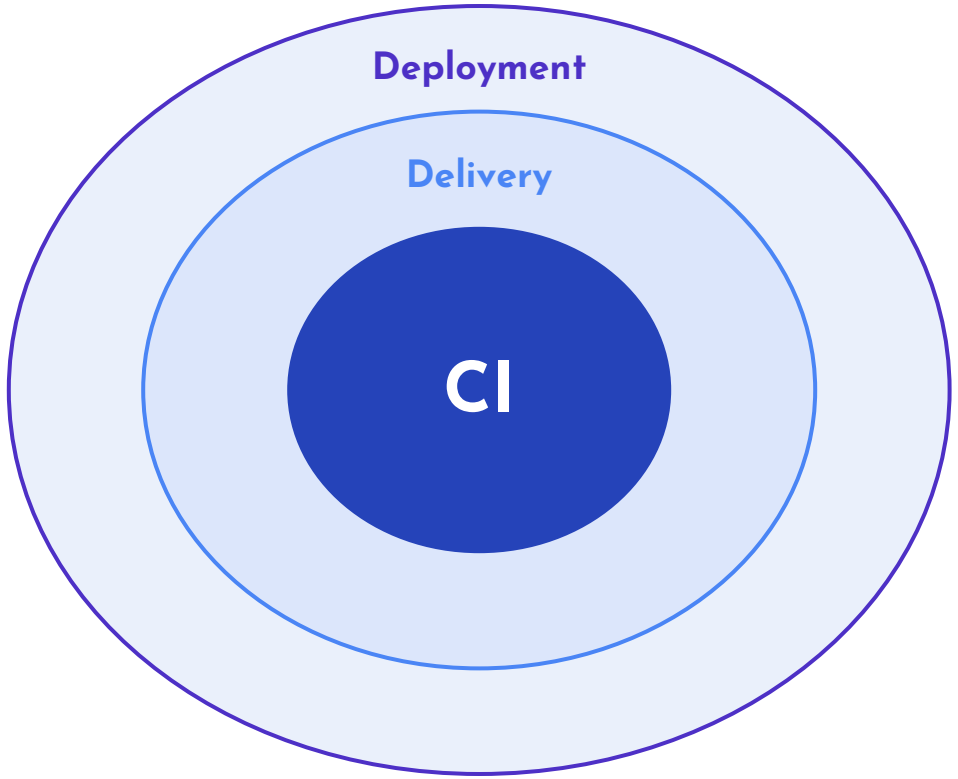
Le déploiement fait peur ? Déployez si souvent, et de façon si automatisée, que cela devienne un non-événement.

Trois sigles à ne pas confondre

Sigle	Nom	Définition courte
CI	**Continuous Integration (Intégration continue)**	Chaque modification est fusionnée fréquemment dans une branche commune, puis compilée et testée automatiquement.
CDel	**Continuous Delivery (Livraison continue)**	Tout code qui passe la CI est automatiquement amené dans un état déployable. La mise en prod reste déclenchée par un humain (un clic).
CDep	**Continuous Deployment (Déploiement continu)**	Tout code qui passe l'ensemble des tests est déployé en production automatiquement, sans intervention humaine.

La vraie frontière : la présence — ou l'absence — d'un **bouton humain** avant la production.

Un emboîtement, pas trois mondes séparés



CI ⊂ Delivery ⊂ Deployment

- **Tout Deployment** fait nécessairement du Delivery.
- **Tout Delivery** fait nécessairement de la CI.
- L'inverse n'est pas vrai : on peut faire de la CI sans déployer en continu.

Faire du Continuous Deployment suppose donc d'avoir d'abord rendu la CI et la livraison fiables. On construit de l'intérieur vers l'extérieur.

MODULE 2 · 35 MIN

Cycle de vie d'une release

Environnements, recettes & gouvernance

35 min

02

Qu'est-ce qu'une « release » ?

Une release = un **lot cohérent de modifications**, figé et identifié par un **numéro de version**, que l'on fait progresser d'un environnement à l'autre jusqu'en production.

Un lot, pas une ligne

On ne livre pas chaque modification isolée, mais un ensemble testé et cohérent.

Identifiée

Un numéro de version unique permet d'en parler, de la tracer et d'y revenir.

Promue telle quelle

Le même artefact traverse les environnements, on ne le reconstruit jamais.

Une chaîne, du poste de dev à la prod

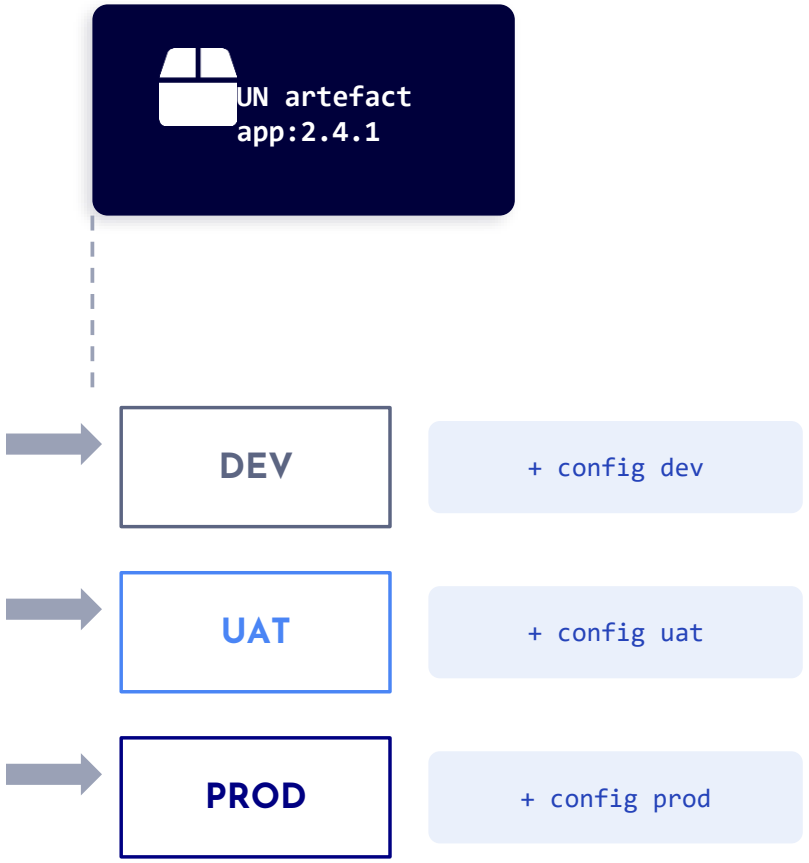
Fidélité à la prod →

DEV	**Développement**	Le poste du développeur. Code en cours, instable par nature.
CI	**Intégration**	Build et tests automatiques à chaque fusion. Le cœur de la CI.
STG	**Recette interne (staging)**	Validation par les équipes QA. Réplique technique de la prod.
UAT	**Recette externe**	Validation par le client / le métier sur des cas réels.
PP	**Pré-production**	Réplique fidèle de la prod. Derniers contrôles avant le grand saut.
PROD	**Production**	Les utilisateurs réels. Le seul environnement qui compte vraiment.

Build once, deploy anywhere

- **On construit l'artefact une seule fois :** en début de pipeline. pipeline. C'est exactement ce binaire-là qui ira jusqu'en prod. jusqu'en prod.
- **Ce qui change d'un environnement à l'autre** , c'est uniquement la configuration (URL de base de données, clés, variables).
- **La config est externalisée,** injectée au démarrage, jamais recompilée dans l'artefact.

Pourquoi ? Si on recompilait pour chaque environnement, on ne on ne testerait jamais vraiment ce qui part en prod. Tester un un artefact ≠ tester un autre.



Le parcours d'une release, étape par étape

Les portes de qualité (*quality gates*) : **entre deux étapes, on ne passe que si la précédente est validée.** Un échec arrête le parcours : on parcours : on corrige avant d'aller plus loin.

« Bien construit » vs « le bon produit »

Recette interne

environnement : staging

« A-t-on bien construit le produit ? »

- Menée par les équipes QA / techniques.
- Vérifie que le logiciel fonctionne correctement, sans régression.
- Regard tourné vers la conformité technique.

Recette externe

environnement : UAT

« A-t-on construit le bon produit ? »

- Menée par le client ou le métier.
- Vérifie que le logiciel répond au besoin réel.
- Regard tourné vers la valeur d'usage.

SemVer : Le sens du versionning

2 . 4 . 1

MAJEUR MINEUR CORRECTIF

MAJEUR

Changement incompatible

Casse la compatibilité : du code existant devra devra être adapté.

`1.9.0 → 2.0.0`

MINEUR

Nouvelle fonctionnalité

Ajout rétrocompatible : rien ne casse pour l'existant.

`2.3.5 → 2.4.0`

CORRECTIF

Correction de bug

Corrige sans rien ajouter ni casser. Rétrocompatible.

`2.4.0 → 2.4.1`

Trois types de changements

Standard

Pré-approuvé, routinier, à faible risque. Pas
Pas de validation au cas par cas.

Rapide

Normal

Évalué au cas par cas, soumis à un comité (CAB) qui arbitre le risque.

Plus lent

Urgent

Correctif critique en prod. Procédure accélérée, validée a posteriori.

Exceptionnel

L'effet d'une bonne pipeline : automatisée, testée et réversible, elle transforme un changement **« normal »** (risqué, qui passe en CAB) en changement **« standard »** (routinier, pré-approuvé). C'est ainsi qu'on déploie souvent sans tout ré-arbitrer.

ITIL et la pipeline ne jouent pas le même rôle

ITIL

Le cadre organisationnel

Le « quoi » et le « pourquoi » de la gouvernance.

- Qui décide et qui approuve.
- Qui est informé, comment on trace.
- Comment on gère le risque et les MEP.

La pipeline

La mise en œuvre technique

Le « comment » concret et répétable.

- Automatise build, tests, déploiement.
- Exécute le parcours de façon fiable.
- Rend chaque release reproductible.

À retenir : la pipeline est l'outil qui exécute ce qu'ITIL encadre. L'un ne remplace pas l'autre — ils se complètent.

MODULE 3 · 55 MIN

La pipeline CI/CD

03

Anatomie, stratégies de branches & outils

55 min

Le vocabulaire de la pipeline

Trigger

Le déclencheur : l'événement qui lance la la pipeline (un push, une pull request, un tag). tag).

Stage

Une phase logique de la pipeline, par exemple : exemple : build, test, deploy.

Job

Une unité de travail dans un stage, exécutée sur un runner.

Step / Script

Une commande élémentaire au sein d'un job d'un job (compiler, lancer les tests…).

Runner / Agent

La machine (souvent éphémère) qui exécute concrètement les jobs.

Artefact

Le produit d'un job (binaire, image conteneur) transmis aux étapes suivantes.

Quatre grandes étapes

1

Build

Compiler le code en un artefact artefact déployable.

2

Tests fonctionnels

Le logiciel fait-il ce qu'on attend ?

3

Tests techniques

Tient-il la charge ? Est-il sûr ?

4

Déploiement

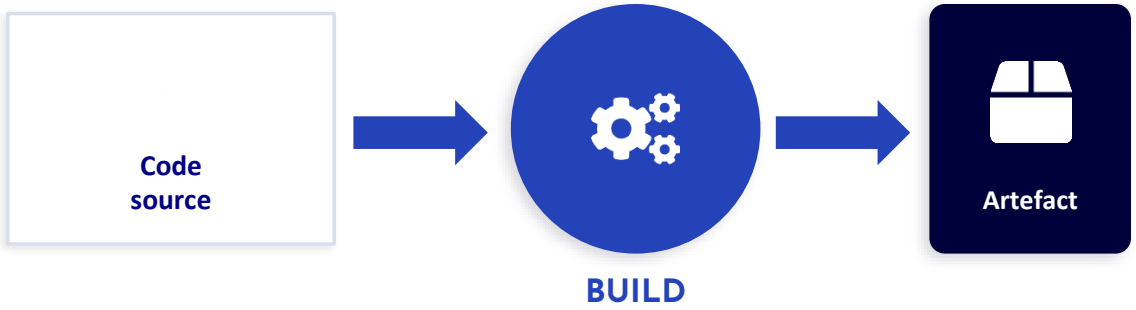
Mettre l'artefact à disposition. disposition.

Chaque étape est une porte. Si elle échoue, la pipeline s'arrête : inutile de tester un code qui ne compile pas, ou de déployer un code qui déployer un code qui ne passe pas les tests. On échoue tôt, et pour pas cher.

ÉTAPE 1 · BUILD

Produire l'artefact, une seule fois

- **Compiler** le code source en un artefact exécutable et autonome.
- **L'artefact = une image de conteneur** le plus souvent : immuable, embarquant tout ce qu'il faut pour tourner.
- **« Build once »** : c'est précisément cet artefact-là qui traversera tous les environnements jusqu'en prod.
- **En cas d'échec, on s'arrête** : rien ne sert d'aller plus loin avec un code qui ne compile pas.



Pourquoi une image de conteneur ?
Parce qu'elle fige l'application ET son environnement d'exécution. On élimine le fameux « ça marche sur ma machine » : ce qui tourne en test tourne à l'identique en prod.

Le logiciel fait-il ce qu'on attend ?

On vérifie le comportement métier de l'application, à différents niveaux de granularité :

Tests unitaires

Vérifient une fonction isolée. Nombreux, très rapides (la base de tout).

Tests d'intégration

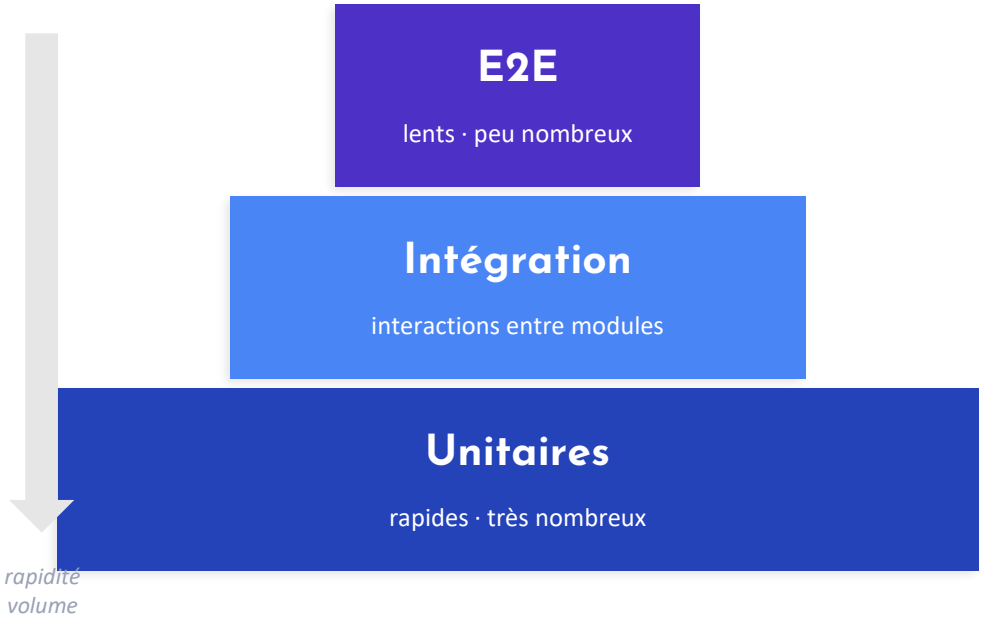
Vérifient que plusieurs composants fonctionnent bien ensemble.

Tests de bout en bout

Rejouent un parcours utilisateur complet. Lents mais réalistes.

Cypress / Playwright / Selenium

La pyramide des tests



L'anti-pattern : le cornet de glace

Proportions inversées : beaucoup de tests E2E lents, peu de tests unitaires. unitaires. Résultat : une pipeline lente, fragile et coûteuse à maintenir.

L'analyse statique

SonarQube, linters… Ils inspectent le code sans l'exécuter pour détecter bugs, détecter bugs, vulnérabilités et « code smells ». Une porte de qualité supplémentaire, et très rapide.

ÉTAPE 3 · TESTS TECHNIQUES

Au-delà du fonctionnel : tenue & sûreté

Performance & charge

Comment se comporte l'application sous forte charge ? Temps de réponse, montée en charge.

`JMeter · k6 · Gatling`

Sécurité

Recherche de vulnérabilités, à plusieurs niveaux du code à l'exécution.

`SAST · DAST · SCA`

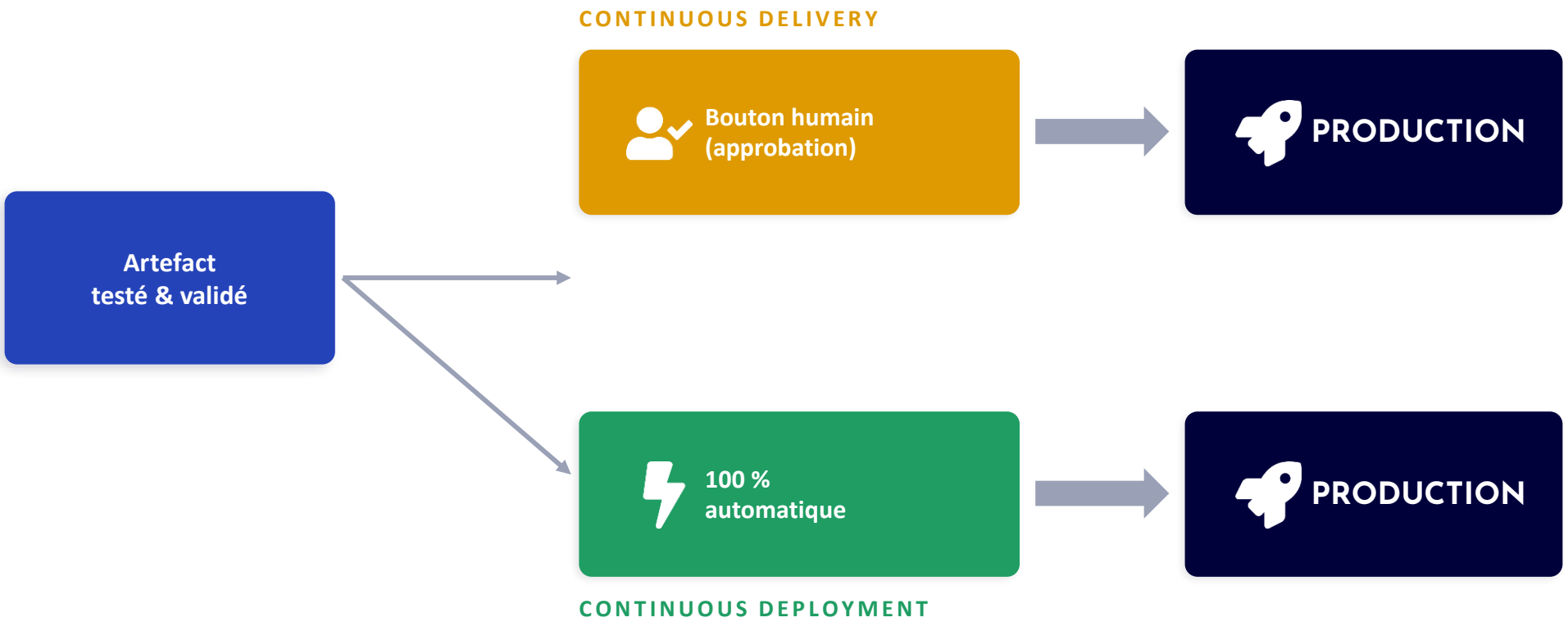
Infrastructure (IaC)

L'environnement est-il conforme et reproductible ? Infra décrite comme du code. code.

`Terraform · Ansible`

Sécurité (3 angles) : **SAST** (analyse du code) · **DAST** (application en marche) · **SCA** (dépendances tierces).

La frontière Delivery / Deployment



La seule différence : un humain appuie sur le bouton (Delivery), ou personne (Deployment). Tout le reste de la pipeline est identique.

Stratégies de branches

GitFlow

Lourd

Beaucoup de branches longues (develop, release, hotfix…). Recrée l'« integration hell » qu'on cherche à fuir.

GitHub Flow

Simple

Une branche main + des branches de fonctionnalité courtes, fusionnées par pull request.

Trunk-Based

Aligné CI/CD

Tout le monde intègre sur le tronc commun commun via des branches très courtes (< 1 (< 1 jour). Le plus proche de l'esprit CI.
CI.

La règle d'or : plus les branches sont courtes, plus l'intégration est continue. Pour livrer du code incomplet sans bloquer : **feature flags** & **branch by abstraction**.

Les outils du marché

GitHub Actions

Intégré à GitHub. Workflows décrits en YAML dans *.github/workflows/*.

GitLab CI

Intégré à GitLab. Un seul fichier *.gitlab-ci.yml* à la racine du dépôt.

Jenkins

L'historique, très répandu et extrêmement flexible (Jenkinsfile, plugins).

Le concept commun « pipeline as code » : la définition de la pipeline est un fichier versionné, vivant DANS le dépôt, à côté du code. Elle est relue, historisée et revue comme n'importe quel code.

GitHub Actions, ligne à ligne

.github/workflows/ci-cd.yml

```yaml
name: CI/CD
on:
  push:
    branches: [ main ]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - run: docker build -t app:$SHA .

  test:
    needs: build        # ne tourne que si build OK
    steps: [ ... npm test ... ]

  deploy:
    needs: test         # ni si les tests cassent
    environment: production
    steps: [ ... ./deploy.sh ... ]
```

`needs:`

Crée la porte de qualité. Un job attend que le précédent réussisse, sinon il ne démarre pas. C'est l'enchaînement build → test → deploy.

`environment:`

`production` peut exiger une approbation manuelle. manuelle. C'est le **bouton humain** → on est en en Continuous Delivery.

GitLab CI — une ligne change tout

.gitlab-ci.yml

```yaml
stages: [ build, test, deploy ]

build:
  stage: build
  script: [ docker build ... ]

test:
  stage: test
  script: [ npm test ]   # porte qualité

deploy:
  stage: deploy
  script: [ ./deploy.sh ]
  rules:
    - when: manual      # Delivery : un humain
    # - when: on_success → Deployment auto
```

`when: manual`

Le job deploy attend un clic dans l'interface avant de partir. → Continuous Delivery (le bouton humain).

`when: on_success`

Le job part seul dès que tout est vert. → **Continuous Deployment.** *Une seule ligne sépare les deux régimes.*

MODULE 4 · 40 MIN

Stratégies de déploiement

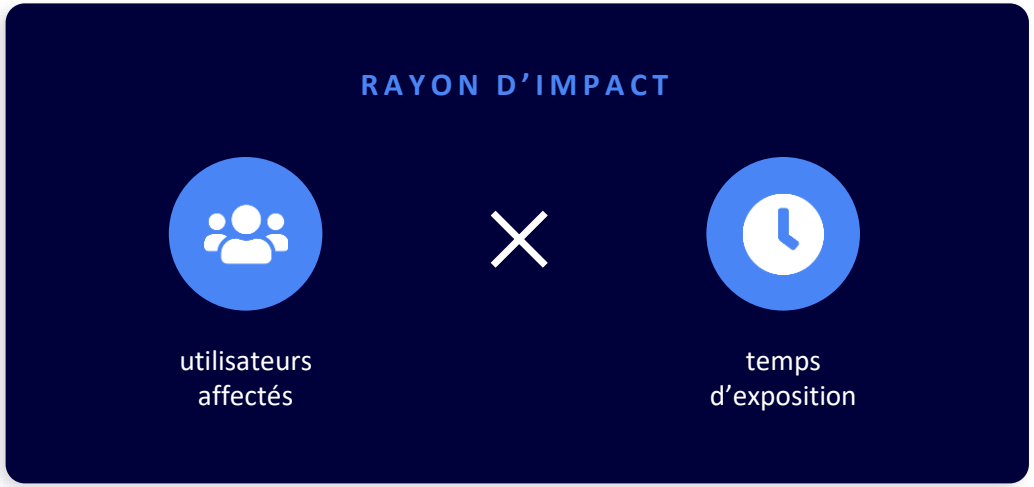
Mettre en production sans tout casser

40 min

04

Déployer, c'est accepter un risque

- **Tout déploiement est un changement** , et tout changement peut mal tourner.
- **On ne supprime pas le risque** , on le maîtrise en réduisant son ampleur.
- **Le bon critère de choix :** le rayon d'impact (blast radius).



Si un déploiement échoue, combien d'utilisateurs sont touchés, et touchés, et pendant combien de temps ? Toutes les stratégies cherchent cherchent à réduire l'un, l'autre, ou les deux.

Deux leviers :
réduire la part d'utilisateurs exposés (canary, flags) et / ou raccourcir le temps d'indisponibilité (blue/green, rollback rapide).

Recreate — on coupe, on remplace

Avantage

Simple à mettre en œuvre ; aucune coexistence de versions à gérer.

À surveiller

Interruption de service pendant le redémarrage : à réserver aux cas qui la tolèrent.

Rolling – remplacement progressif

Les instances sont remplacées une par une — le service reste debout

Défaut de Kubernetes. Pendant la transition, v1 et v2 servent en même temps.

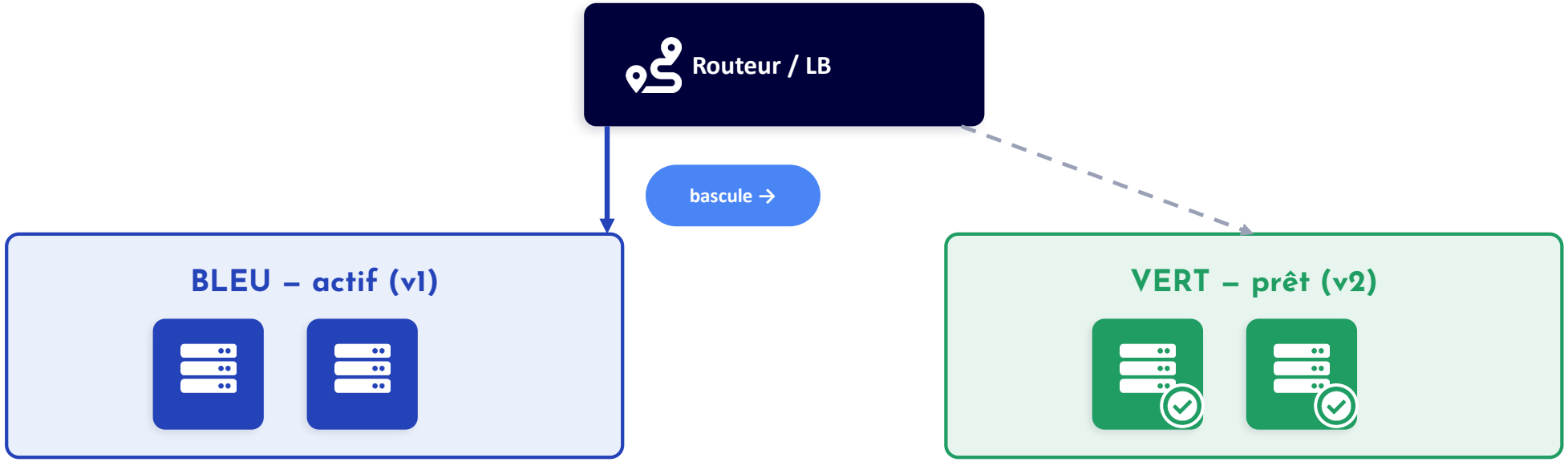
Avantage

Aucune interruption ; pas de double infrastructure à financer.

À surveiller

v1 et v2 coexistent : les deux versions doivent rester compatibles (base de données, API).

Blue / Green — deux environnements jumeaux



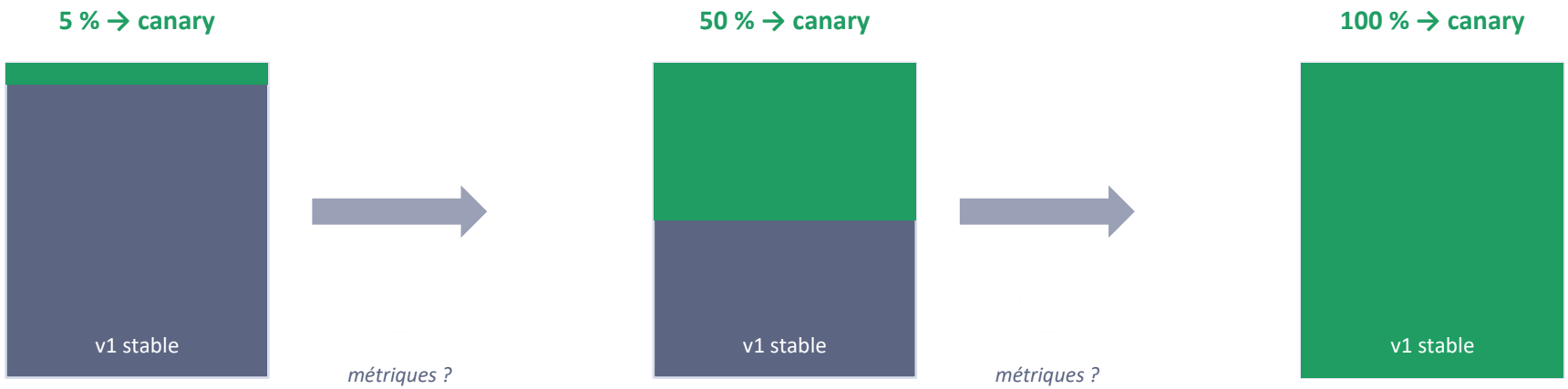
Avantage

Bascule et rollback quasi instantanés : on repointe simplement le routeur.

À surveiller

Coûte deux fois l'infrastructure. C'est un schéma actif / passif (voir passif (voir module 5).

Canary – exposer une fraction d'abord



Avantage

Rayon d'impact minimal : on valide sur de vrais utilisateurs, en petit nombre.

À surveiller

Plus complexe à orchestrer ; exige une vraie observabilité (métriques, alertes).

Feature flags — découpler déployer et activer

Déployer ≠ releaser. Le code part en prod désactivé ; on l'active — ou on le coupe (kill switch) — par configuration, sans redéployer.

Avantage

Découple déploiement et activation ; kill switch immédiat en cas de souci.

À surveiller

Chaque flag est de la dette technique : à nettoyer une fois la fonctionnalité stabilisée.

Comparatif des stratégies

Stratégie	Interruption	Coût infra	Rayon d'impact	Rollback	Complexité
Recreate	Oui	Faible	Élevé	Lent	Faible
Rolling	Non	Faible	Moyen	Moyen	Moyenne
Blue / Green	Non	Élevé ×2	Faible	Instantané	Moyenne
Canary	Non	Moyen	Très faible	Rapide	Élevée
Feature flags	Non	Faible	Très faible	Instantané	Moyenne

Il n'y a pas de « meilleure » stratégie — il y a celle qui correspond à votre rayon d'impact acceptable et à ce que vous pouvez financer et opérer.

Le rollback : savoir revenir en arrière

> « Un déploiement n'est jamais terminé tant qu'on ne sait pas revenir en arrière. »

Le principe

- Pouvoir restaurer la version précédente vite et de façon sûre.
- Les stratégies y aident : blue/green (repointer), canary (stopper la montée).
- À préparer et à tester AVANT, pas à improviser le jour de l'incident.

Le piège : les données

- Revenir au code est facile ; revenir sur une migration de base de données l'est beaucoup moins.
- Règle d'or : des migrations rétrocompatibles, qui n'empêchent pas l'ancienne version de fonctionner.
- Sans cela, le rollback est illusoire.

MODULE 5 · 30 MIN

Architecture & résilience

05

Concevoir pour que ça tienne — et pour s’en remettre

30 min

Haute disponibilité & reprise après sinistre

Haute disponibilité

HA : High Availability

« Le service ne tombe pas. »

- Redondance : aucun point unique de défaillance (SPOF).
- Bascule automatique en cas de panne d'un composant.
- Objectif : la continuité, au quotidien.

Reprise après sinistre

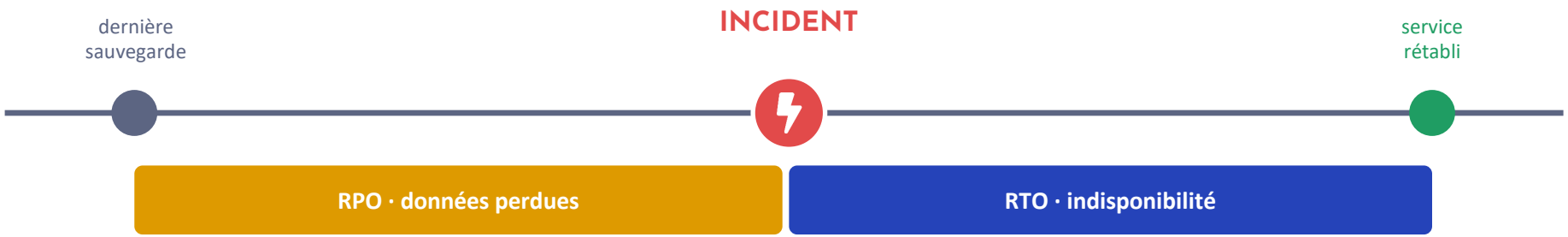
DR : Disaster Recovery

« Quand tout tombe, on s'en remet. »

- Sinistre majeur : perte d'un datacenter, d'une région.
- Restauration depuis sauvegardes ou site de secours.
- Objectif : la remise en service, après la catastrophe.

On a besoin des deux : la HA absorbe les pannes du quotidien sans qu'on le remarque ; la DR organise le relèvement face aux catastrophes rares catastrophes rares mais graves.

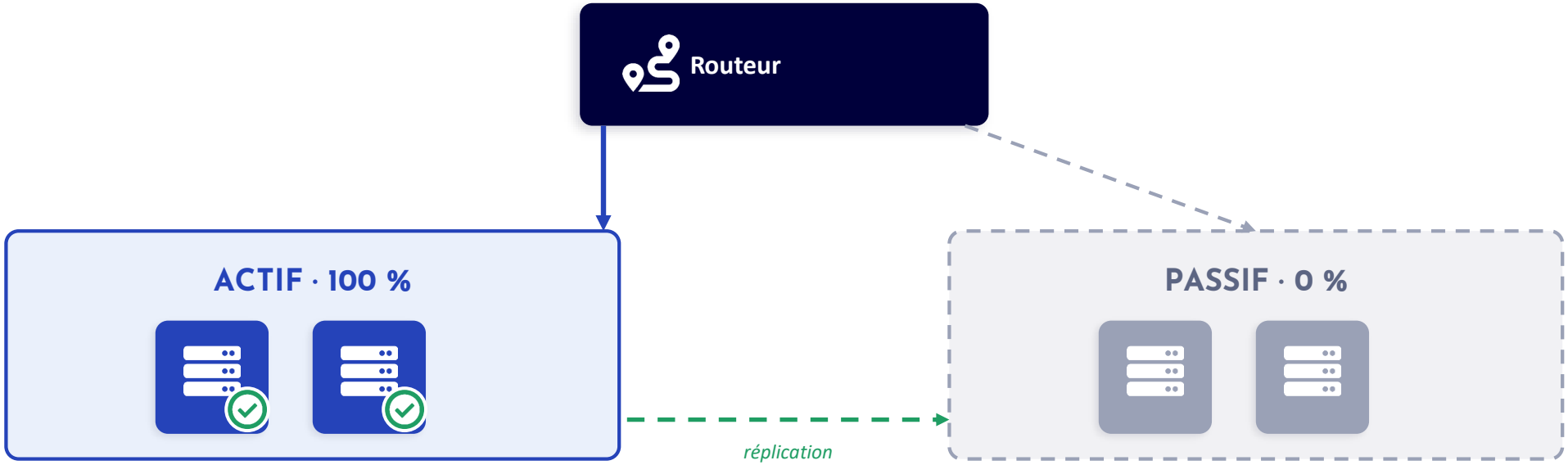
RTO & RPO – chiffrer la résilience



RPO : Recovery Point Objective
Quelle quantité de données acceptez-vous de perdre ? Détermine la fréquence des sauvegardes.

RTO : Recovery Time Objective
Combien de temps d'indisponibilité tolérez-vous ? Dimensionne le dispositif de reprise.

Actif / passif — un nœud sert, l'autre attend



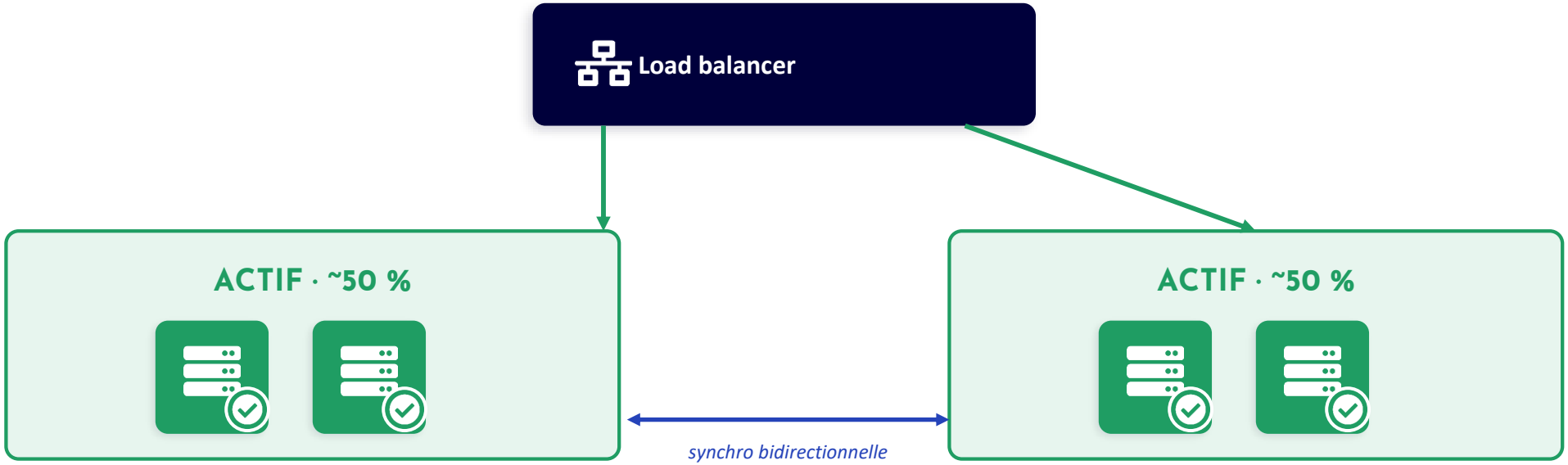
Avantage

Cohérence des données simple : un seul nœud écrit à la fois.

À surveiller

Le passif est payé mais inactif ; le failover prend un peu de temps temps (RTO > 0).

Actif / actif — les deux servent en même temps



Avantage

RTO proche de zéro ; les ressources sont toutes utilisées.

À surveiller

Synchronisation bidirectionnelle complexe — la cohérence des des données devient un vrai défi (théorème CAP).

Déploiement & résilience : les mêmes briques

Les stratégies du module 4 ne sont que ces topologies, appliquées au moment de livrer :

Blue / Green	→ **Actif / passif**	*on bascule d'un jumeau à l'autre*
Rolling · Canary	→ **Actif / actif**	*plusieurs instances actives en parallèle*

TROIS BRIQUES COMMUNES

- **Redondance**
- **Réplication**
- **Reroutage**

MODULE 6 · 25 MIN

Culture DevOps & métriques DORA

06

La technique ne suffit pas, il faut la culture qui va avec

25 min

Attention à l'homonymie : le DORA réglementaire (Digital Operational Resilience Act, secteur bancaire) n'a rien à voir avec les métriques DORA. les métriques DORA. Même sigle, sujets différents.

Des cycles longs aux itérations courtes

Cascade / cycle en V

un seul gros lot, livré à la toute fin

Agile

des incréments livrés en continu

Scrum : on avance par sprints, à cadence fixe (1 à 4 semaines).

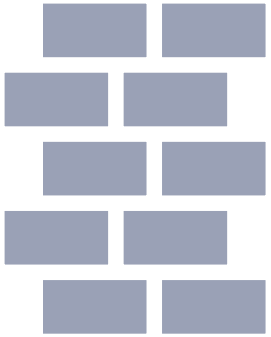
Kanban : un flux continu, en limitant le travail en cours (WIP).

Le lien essentiel : l'agilité crée la **demande** de livrer souvent ; la CI/CD en fournit le **moyen technique**. L'une réclame des livraisons fréquentes, l'autre les rend possibles sans douleur.

Le mur de la confusion

DÉVELOPPEMENT

veut le CHANGEMENT
(livrer des fonctionnalités)



MUR

EXPLOITATION

veut la STABILITÉ
(ne rien casser)

DevOps abat le mur

Un objectif commun aux deux équipes : livrer vite ET sûrement. Plus deux camps qui se renvoient la responsabilité, mais une chaîne de bout en bout, partagée.

Quatre principes pour abattre le mur

Responsabilité partagée

« You build it, you run it » : qui développe assume aussi l'exploitation.

Automatisation

Supprimer les gestes manuels, sources d'erreurs et de lenteur.

Mesure

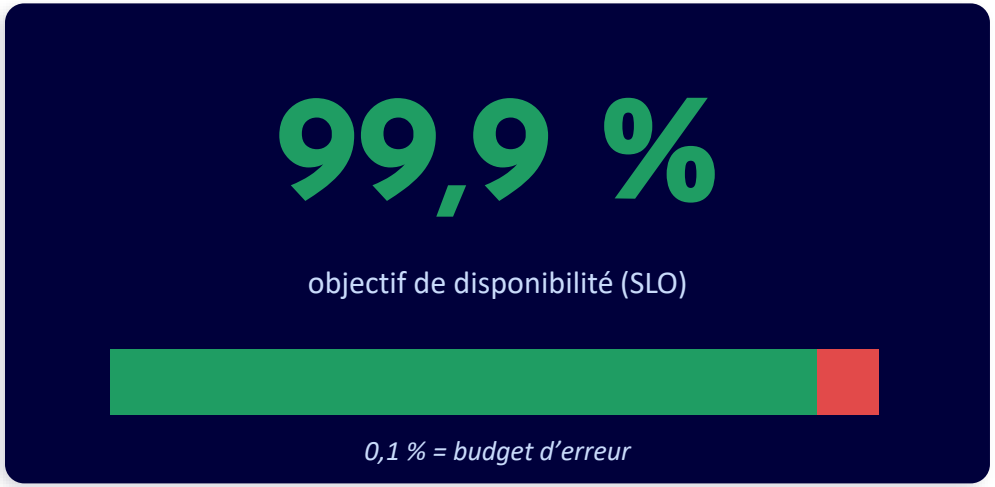
Décider sur des faits et des indicateurs, pas sur des opinions.

Amélioration continue

Apprendre des incidents via des post-mortems sans blâme.

SRE & budget d'erreur

- **SRE (Site Reliability Engineering) :** l'approche née chez Google : appliquer appliquer l'ingénierie logicielle à l'exploitation.
- **On fixe un objectif de fiabilité (SLO),** par exemple 99,9 % de disponibilité.
- **Le budget d'erreur,** c'est le complément : les 0,1 % d'indisponibilité « autorisés ».



À quoi ça sert ? Tant qu'il reste du budget d'erreur, l'équipe peut continuer à livrer vite et prendre des risques mesurés. Une fois le budget épuisé, on lève le pied sur les nouvelles fonctionnalités pour consolider la fiabilité. **Le budget d'erreur arbitre objectivement la tension vitesse / stabilité.**

Les quatre métriques DORA

DORA = DevOps Research and Assessment. Quatre indicateurs, regroupés en deux familles.

VITESSE

Fréquence de déploiement

À quelle fréquence met-on en production ?

Délai de livraison (Lead Time)

Combien de temps entre un commit et sa mise en prod ?

STABILITÉ

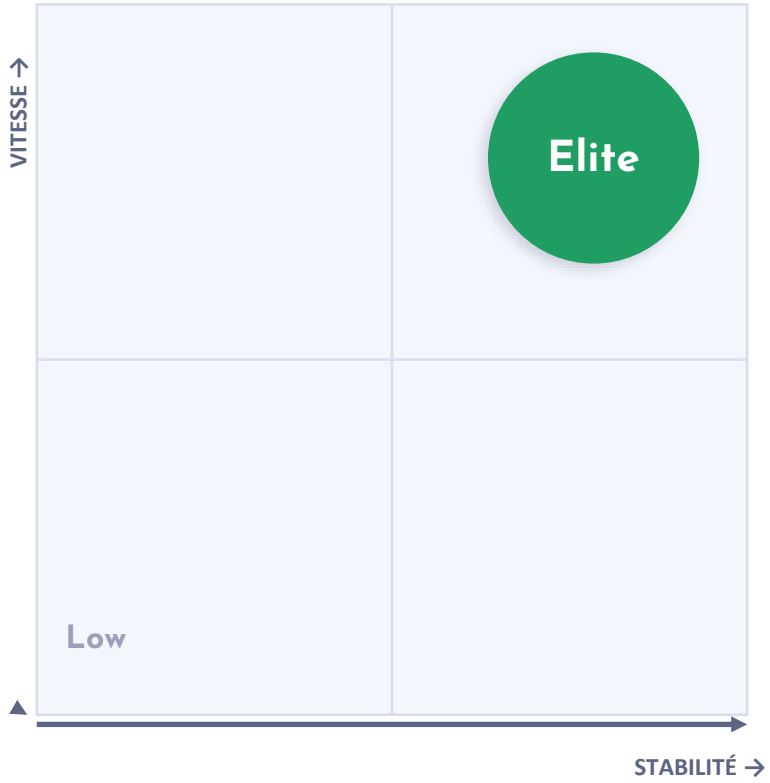
Taux d'échec des changements

Quelle part des déploiements provoque un incident ?

Temps de rétablissement (MTTR)

Combien de temps pour réparer après une panne ?

Vitesse ET stabilité – pas un compromis



Le résultat le plus contre-intuitif de l'étude DORA

- **Vitesse et stabilité ne s'opposent pas :** les équipes « Elite » excellent sur les deux à la fois.
- **Le secret :** de petits lots, livrés souvent, automatisés et outillés.
- **Livrer souvent réduit le risque,** au lieu de l'augmenter.

On boucle sur l'idée fondatrice

« Si quelque chose fait mal, faites-le plus souvent. » Déployer souvent, par petits incréments, c'est exactement ce qui rend la livraison à la fois rapide et sûre.

MODULE 7 · 15 MIN

Synthèse & lien avec le TP

Ce qu’il faut retenir, et ce qui vous attend

15 min

07

Sept idées à emporter

1. **Si quelque chose fait mal, faites-le plus souvent.** l'idée fondatrice de toute la CI/CD.
2. **CI ⊂ Delivery ⊂ Deployment.** la seule vraie frontière, c'est le bouton humain.
3. **Build once, deploy anywhere.** un artefact unique ; seule la configuration change.
4. **La pipeline est une suite de portes de qualité.** on échoue tôt, et pour pas cher.
5. **On choisit sa stratégie selon le rayon d'impact.** il n'y a pas de « meilleure » stratégie absolue.
6. **Déploiement et résilience : les mêmes briques.** redondance, réplication, reroutage.
7. **Vitesse ET stabilité vont de pair.** mesurées par les quatre métriques DORA.

Ressources

Continuous Delivery

Jez Humble & David Farley

La référence fondatrice du domaine.

Accelerate

Forsgren, Humble & Kim

La science derrière les métriques DORA.

The Phoenix Project

Gene Kim

Un roman pour comprendre DevOps de l'intérieur.

State of DevOps Report

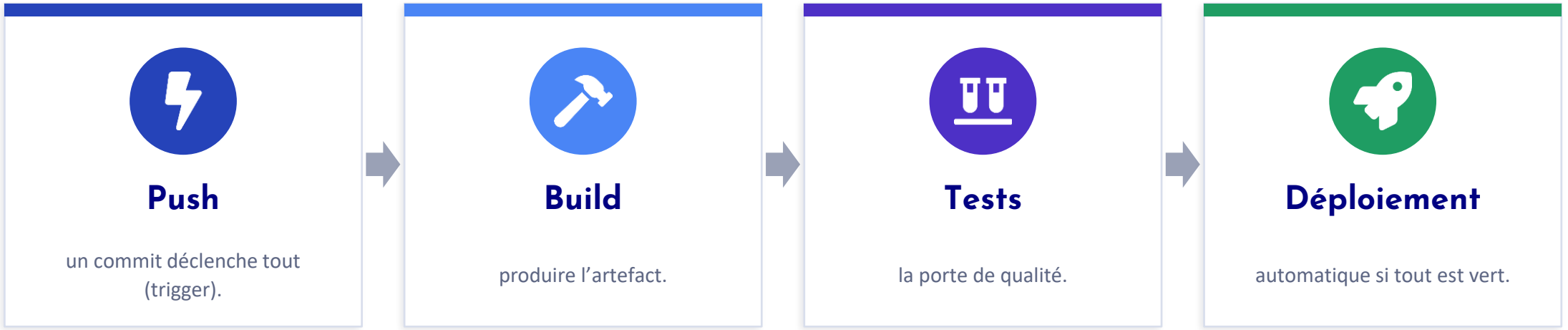
Le rapport annuel qui établit l'état de l'art, chiffres à l'appui. C'est la source des catégories de performance (de « Low » à « Elite »).

Docs officielles

GitHub Actions & **GitLab CI :** la pratique, toujours à jour. Le meilleur point de départ pour écrire vos premiers pipelines.

Place au TP : construisez votre pipeline

Vous allez mettre en œuvre, concrètement, tout ce qu'on vient de décrire :



Le sujet du TP est en ligne : **www.pividori.fr/polytech**. La théorie est posée ; place à la pratique.
Des questions avant de commencer ?

ANNEXE · PRÉREQUIS

Git — les fondamentaux, à revoir en autonomie

Git est le prérequis pratique du TP. L'annexe complète (en ligne) couvre :

Les trois zones

working directory · staging · dépôt

Branches & fusion

créer, fusionner, comprendre l'historique

Pull requests & conflits

collaborer et résoudre les conflits

Commandes de sauvetage

annuler, revenir en arrière sans paniquer

Cheat sheet

les commandes du quotidien, en un coup d'œil

Workflow complet

du clone au push, le cycle de travail

À consulter seul·e : le support détaillé et le PDF sont disponibles sur **www.pividori.fr/polytech**. Revoyez-le avant le TP si Git n'est pas TP si Git n'est pas encore un réflexe.

TLS / HTTPS & certificats X.509

- **Un certificat prouve l'identité** du serveur et permet de chiffrer les échanges.
- **La terminaison TLS** se fait souvent au load balancer ou à l'ingress, en entrée d'infrastructure.
- **Le handshake utilise l'asymétrique** pour négocier une clé de session symétrique, plus rapide.

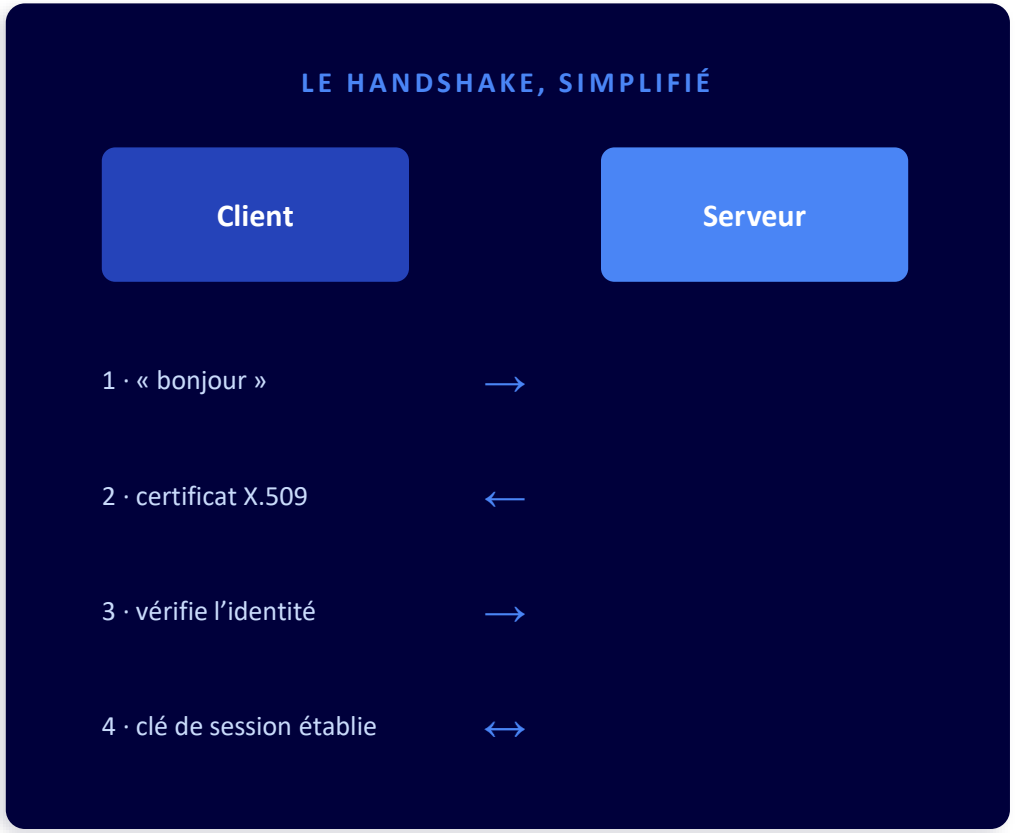
LA TRIADE ASSURÉE

Confidentialité

Intégrité

Authenticité

LE HANDSHAKE, SIMPLIFIÉ



La gestion des secrets dans la pipeline

La règle d'or : un secret (mot de passe, clé d'API, token) ne doit JAMAIS se retrouver dans le dépôt de code.

À ne jamais faire

- Committer une clé « en attendant ».
- Mettre un mot de passe en clair dans le code.
- Oublier de couvrir .env via .gitignore.

Les bons outils

- Secret stores : GitHub / GitLab secrets.
- Coffre-fort centralisé : HashiCorp Vault.
- Injection au moment de l'exécution.

Bonnes pratiques

- Rotation régulière (cryptopériode).
- Bancaire : HSM & key ceremony.
- Authentification mutuelle (mTLS).

MAGELLAN

MERCI

Des questions ?

Andrey Pividori

Intervenant Magellan · Polytech Tours · 2026

andrey@pividori.fr

www.pividori.fr