



GitLab

GITLAB POUR DEVOPS

Par Frazer SADO

PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-Cl

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Monitoring

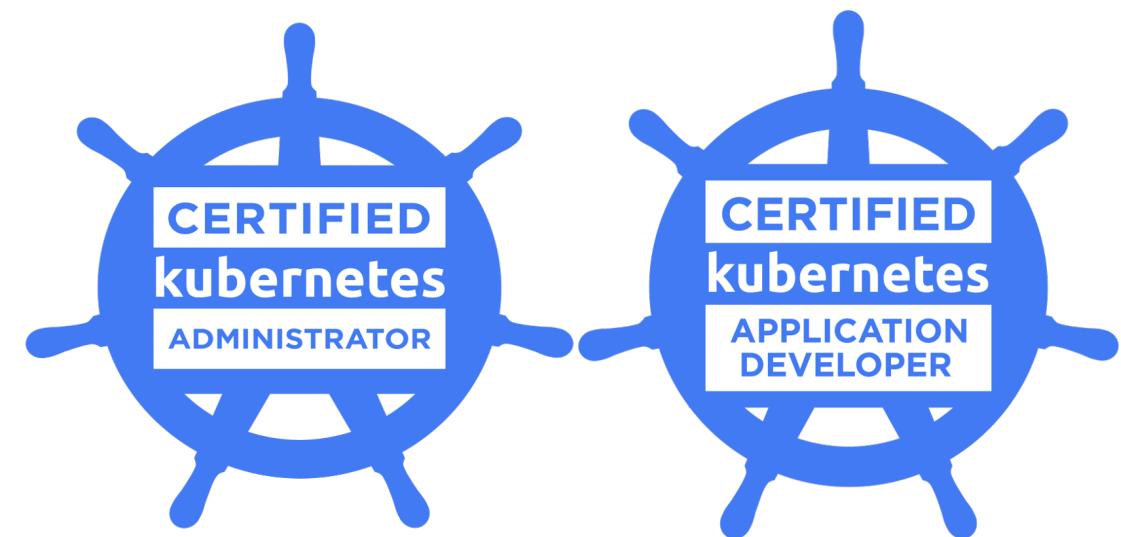
GitOps

Mini-Projet

PRÉSENTATION DU FORMATEUR

FRAZER SADO (Formateur et consultant DevOps)

- EAZYTRAINING
- INTEGRAL
- INTELCIA
- ATOS
- SERVITEL



PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-Cl

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Monitoring

GitOps

Mini-Projet



INTRODUCTION A GITLAB ET AU CI/CD: PRÉREQUIS SOUHAITÉS

PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-Cl

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Monitoring

GitOps

Mini-Projet



VCS/SCM : VERSION CONTROL SYSTEM OU SOURCE CODE MANAGEMENT

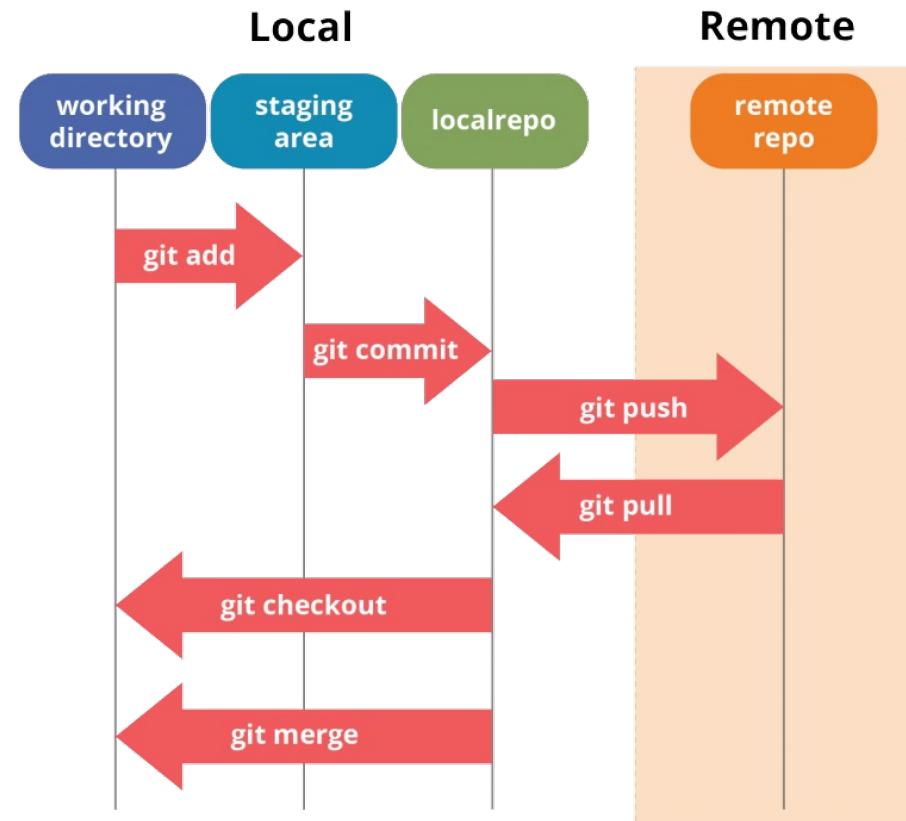
Git est un logiciel de gestion de versions décentralisé. Logiciel Open Source créé par Linus Torvalds et également logiciel de gestion de versions le plus populaire à ce jour.

Les Avantages ou fonctionnalités de GIT

- Gestion des versions de nos fichiers ou répertoires par ordinateur
- Sauvegarde et synchronisation avec un serveur distant
- Conservation de l'historique nominatif
- Possibilité de retour en arrière
- Fusion des modifications
- Visualisation des changements au cours du temps

Vocabulaire de GIT (lié au stockage)

- Working Directory
- Staging area ou index
- Local Repository
- Remote Repository



RAPPEL SUR GIT: VERSION CONTROL

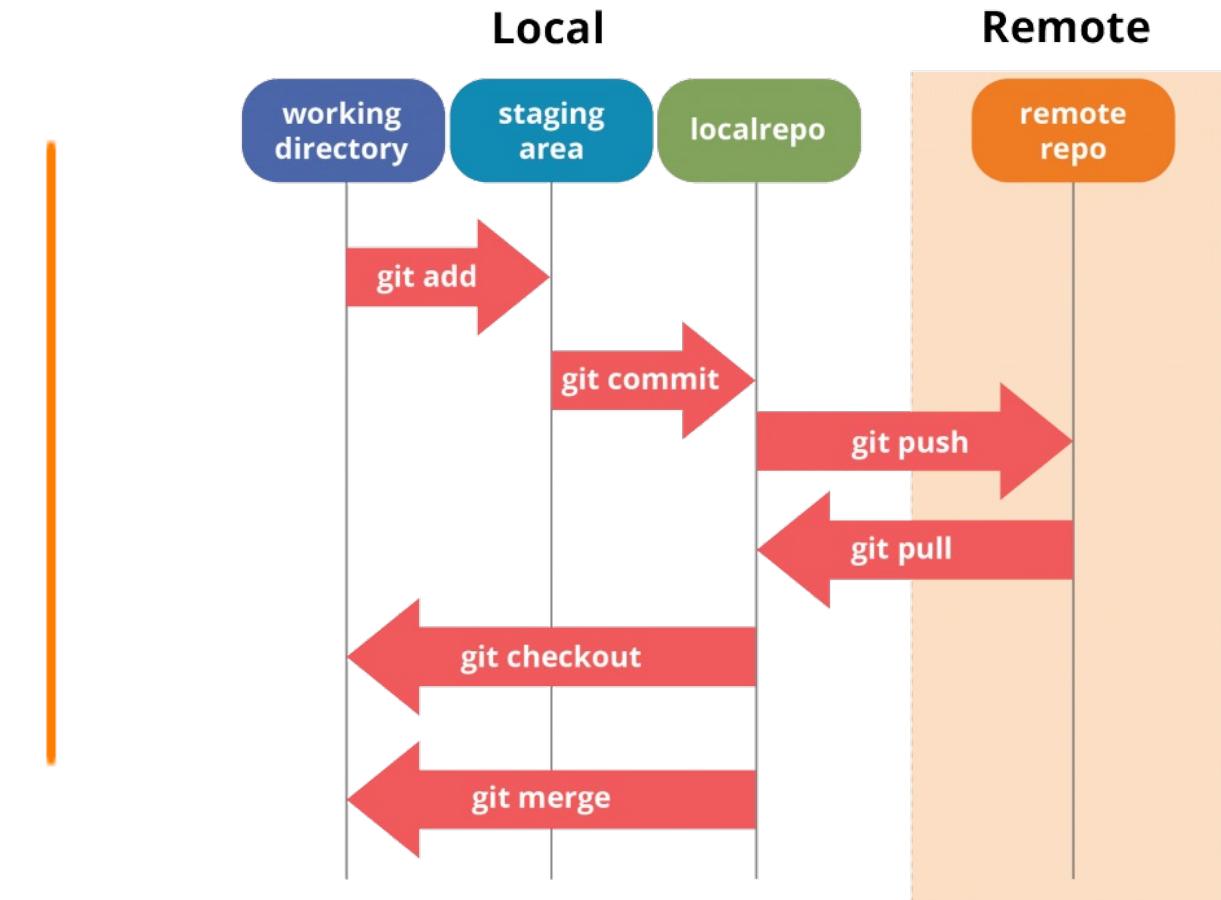


VCS/SCM : VERSION CONTROL SYSTEM OU SOURCE CODE MANAGEMENT

Git est un logiciel de gestion de versions décentralisé. Logiciel Open Source créé par Linus Torvalds et également logiciel de gestion de versions le plus populaire à ce jour.

Les Opérations de base sur Git

- Clone : git clone <url_remote_repository>
- Initialize : git init
- Add : git add <file>
- Commit : git commit -a -m <message>
- Pull : git pull origin <branch-name>
- Push : git push <remote-name> <branch-name>
- Checkout : git checkout <id_commit> ou <branch_name>
- Merge : git merge <branch_name>



RAPPEL SUR GIT: VERSION CONTROL



De loin, le système de contrôle de version moderne le plus utilisé dans le monde aujourd'hui est Git. Git est un projet open source mature et activement maintenu, développé à l'origine en 2005 par Linus Torvalds, le célèbre créateur du noyau du système d'exploitation Linux.

GitHub



GitHub est une solution web ou service web qui permet d'héberger les dépôts ou repo Git.

Il offre toutes les fonctionnalités de contrôle de version distribué et de gestion du code source de Git, ainsi que l'ajout de ses propres fonctionnalités. Il fournit un contrôle d'accès et plusieurs fonctionnalités de collaboration telles que le suivi des bugs, les demandes de fonctionnalités, la gestion des tâches et les wikis pour chaque projet.



GitLab

GitLab CE a des fonctionnalités similaires à GitHub, à la différence qu'il est possible d'auto-héberger GitLab pour votre propre usage et un accès privé au code.

RAPPEL SUR GIT: VERSION CONTROL

PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-Cl

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Monitoring

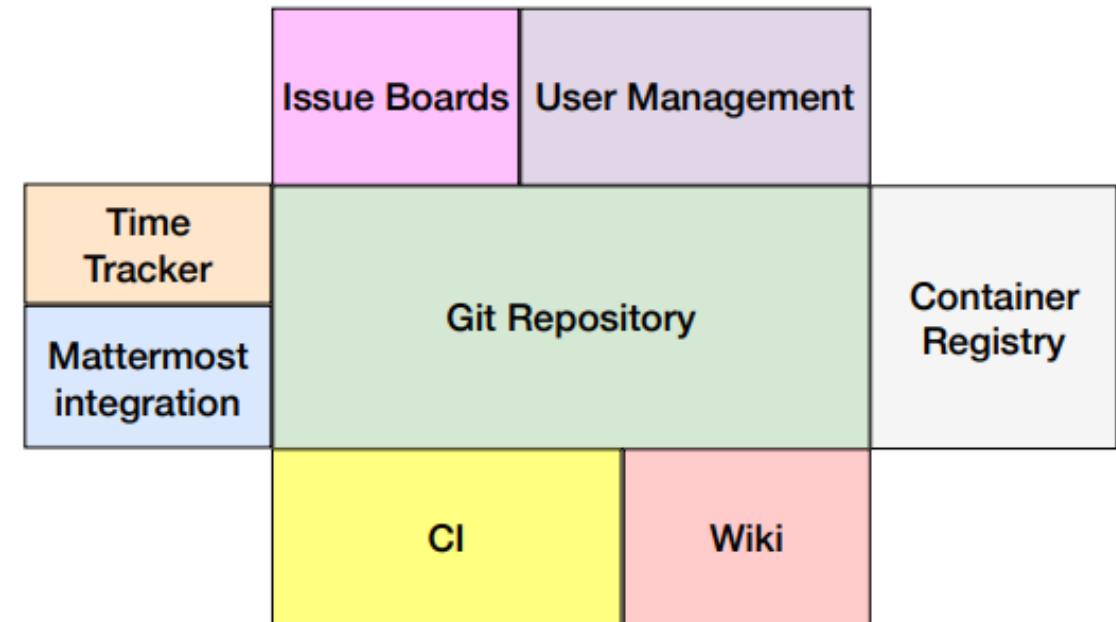
GitOps

Mini-Projet

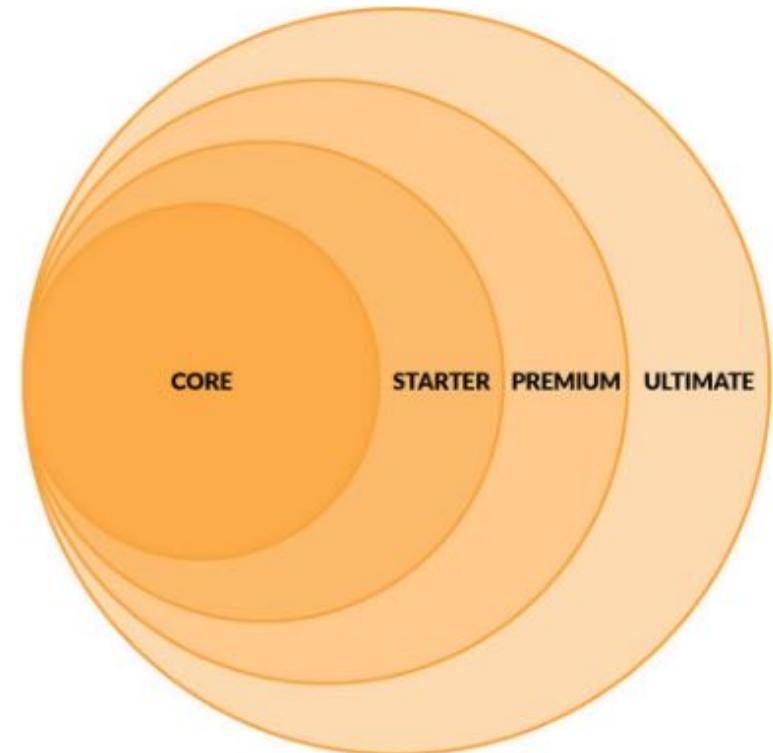
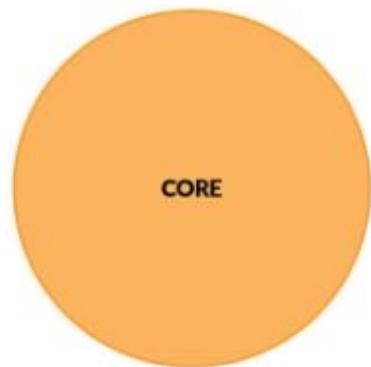


GITLAB (1/6): PRÉSENTATION

- Serveur Git
- Collaboration
- Open-Source, freemium
- Hosted (free) ou on premise
- Communauté



GITLAB (2/6): PRICING



GITLAB (2/6): PRICING

Get the DevOps platform

The image displays three pricing plans for GitLab: Free, Premium, and Ultimate. Each plan is represented by a vertical card with a decorative starburst icon at the top.

- Free:** \$0 per user/month (No credit card required). Includes a 'Get started' button and a list of features: Spans the DevOps lifecycle, Open Source - MIT License, Bring your own GitLab CI runners, Includes free static websites, and 400 CI/CD minutes per month [1].
- Premium:** \$19 per user/month (Billed annually at \$228 USD). Includes a 'Buy GitLab Premium' button and a list of features: Everything from Free, Faster code reviews, Advanced CI/CD, Enterprise agile planning, Release controls, Self-managed reliability, and 10,000 CI/CD minutes per month.
- Ultimate:** \$99 per user/month (Billed annually at \$1,188 USD). Includes a 'Buy GitLab Ultimate' button and a list of features: Everything from Premium, Advanced security testing, Security risk mitigation, Compliance, Portfolio management, Value stream management, Free guest users, and 50,000 CI/CD minutes per month.

All plans have unlimited (private) repositories. Use it SaaS or Self-Managed.



GitLab

INSTALLATION DE GITLAB CE

Installation GitLab CE

- Installation à partir de Docker
- Création du docker-compose
- Site officiel: <https://docs.gitlab.com/ee/install/docker.html#install-gitlab-using-docker-compose>
- Repo à utiliser : <https://github.com/sameersbn/docker-gitlab.git>
- Faire le docker-compose up et ouvrir le port 10080

[GitLab CE Docker image ↗](#)

To install the Community Edition, replace `ee` with `ce` in the commands on this page.

```
web:  
  image: 'gitlab/gitlab-ce:latest'  
  restart: always  
  hostname: 'gitlab.example.com'  
  environment:  
    GITLAB_OMNIBUS_CONFIG: |  
      external_url 'https://gitlab.example.com'  
      # Add any other gitlab.rb configuration here, each on its own line  
  ports:  
    - '80:80'  
    - '443:443'  
    - '22:22'  
  volumes:  
    - '$GITLAB_HOME/config:/etc/gitlab'  
    - '$GITLAB_HOME/logs:/var/log/gitlab'  
    - '$GITLAB_HOME/data:/var/opt/gitlab'
```



GitLab

PRISE EN MAIN DE GITLAB CE

GitLab Menu Search GitLab

Projects

New project

Your projects 1 Starred projects 0 Explore projects

Filter by name... Name

All Personal

M	GitLab Instance / Monitoring	Owner	This project is automatically generated and helps monitor this GitLab instance. Learn more.	★ 0	¥ 0	↑ 0	□ 0	Updated 16 minutes ago

Prise en main GitLab CE

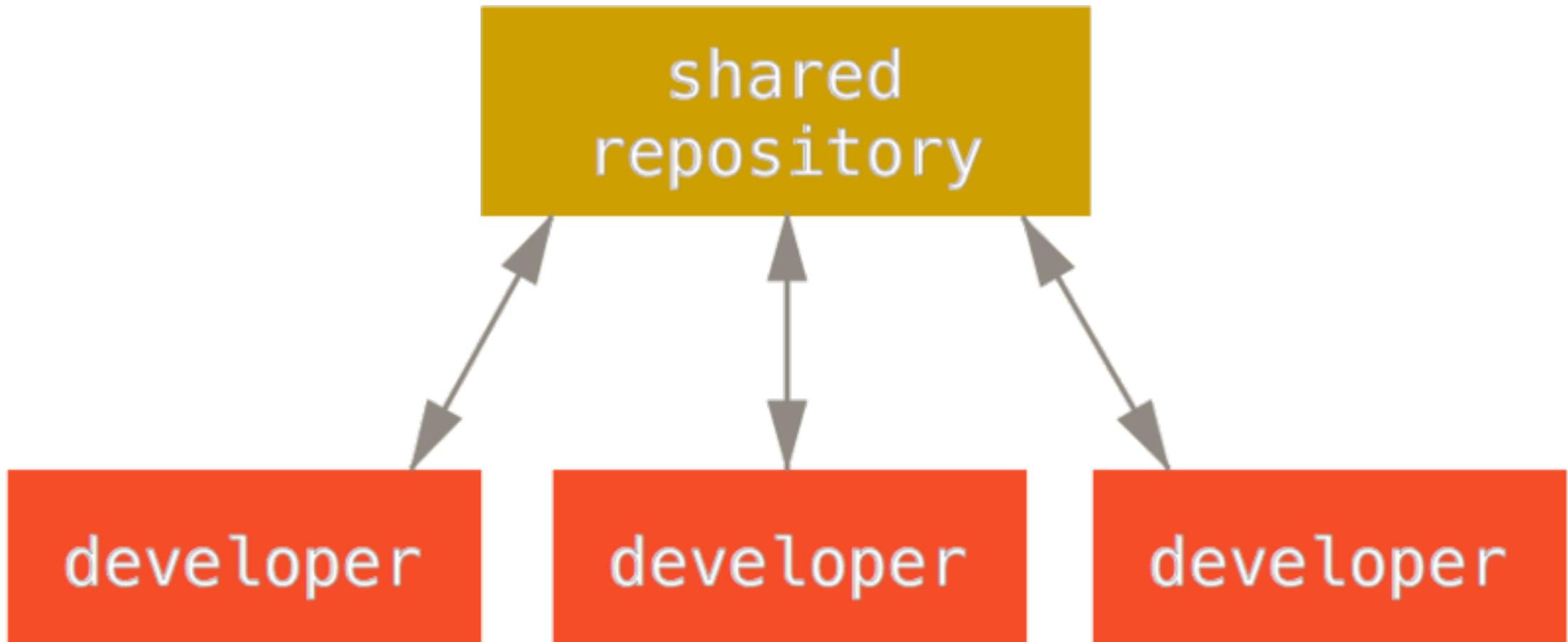
- Réinitialisation du mot de passe du compte root
- Connectez vous à l'aide du compte root

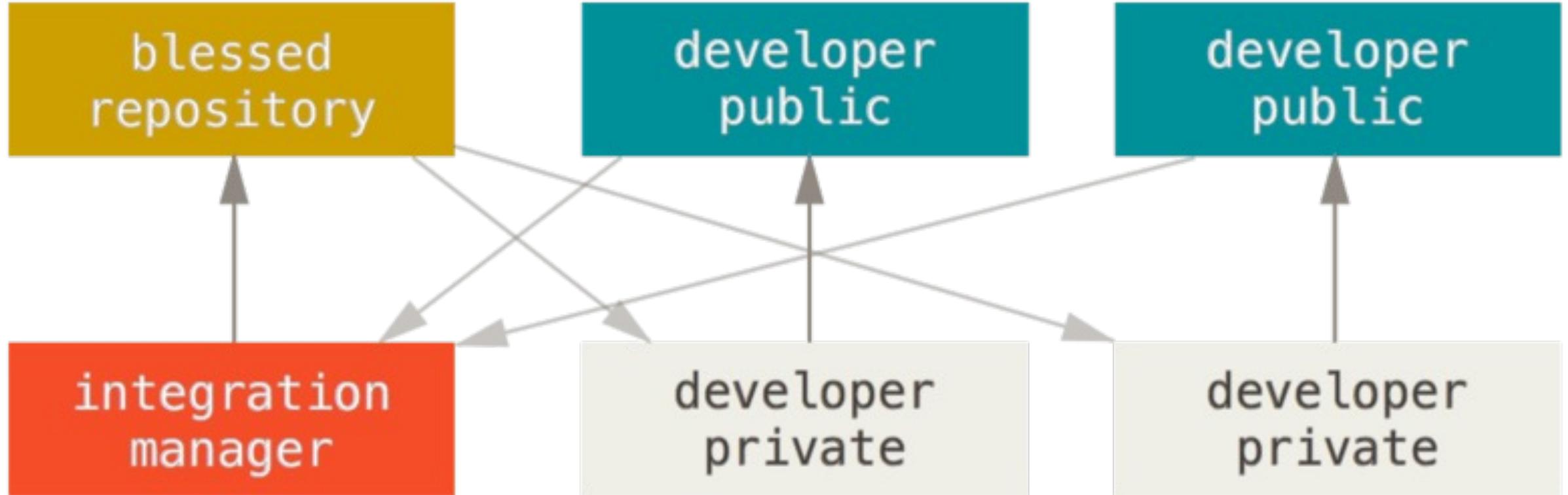
- Protégez l'accès à votre serveur GitLab (SignUp)
- Prise en main de l'interface GitLab-ce
- Création de nouveaux users avec limitation des droits
- Création d'un projet

TP-1: Mise en place de gitlab

- Accédez à la plateforme de labs eazytraining docker.labs.eazytraining.fr
- Créez une instance
- Git clonez le projet suivant: <https://github.com/sameersbn/docker-gitlab.git>
- Rendez-vous dans le dossier docker-gitlab et lancez la commande docker-compose up –d
- Connectez-vous en ouvrant le port 10080
- Reinitialisez le mot de passe du compte root
- Découvrez l'interface de gitlab

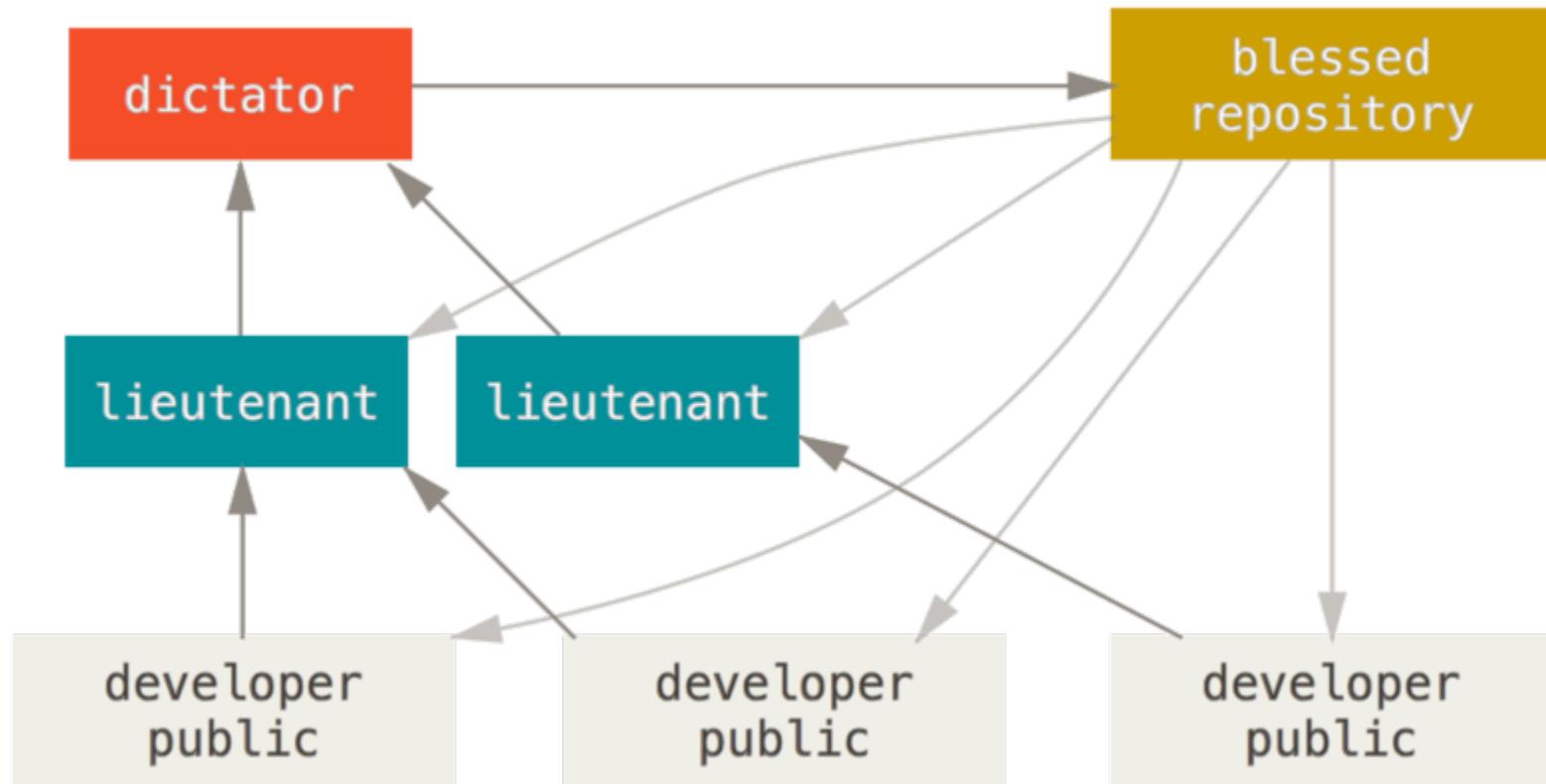
Travail collaboratif (1/4): Workflow centralisé



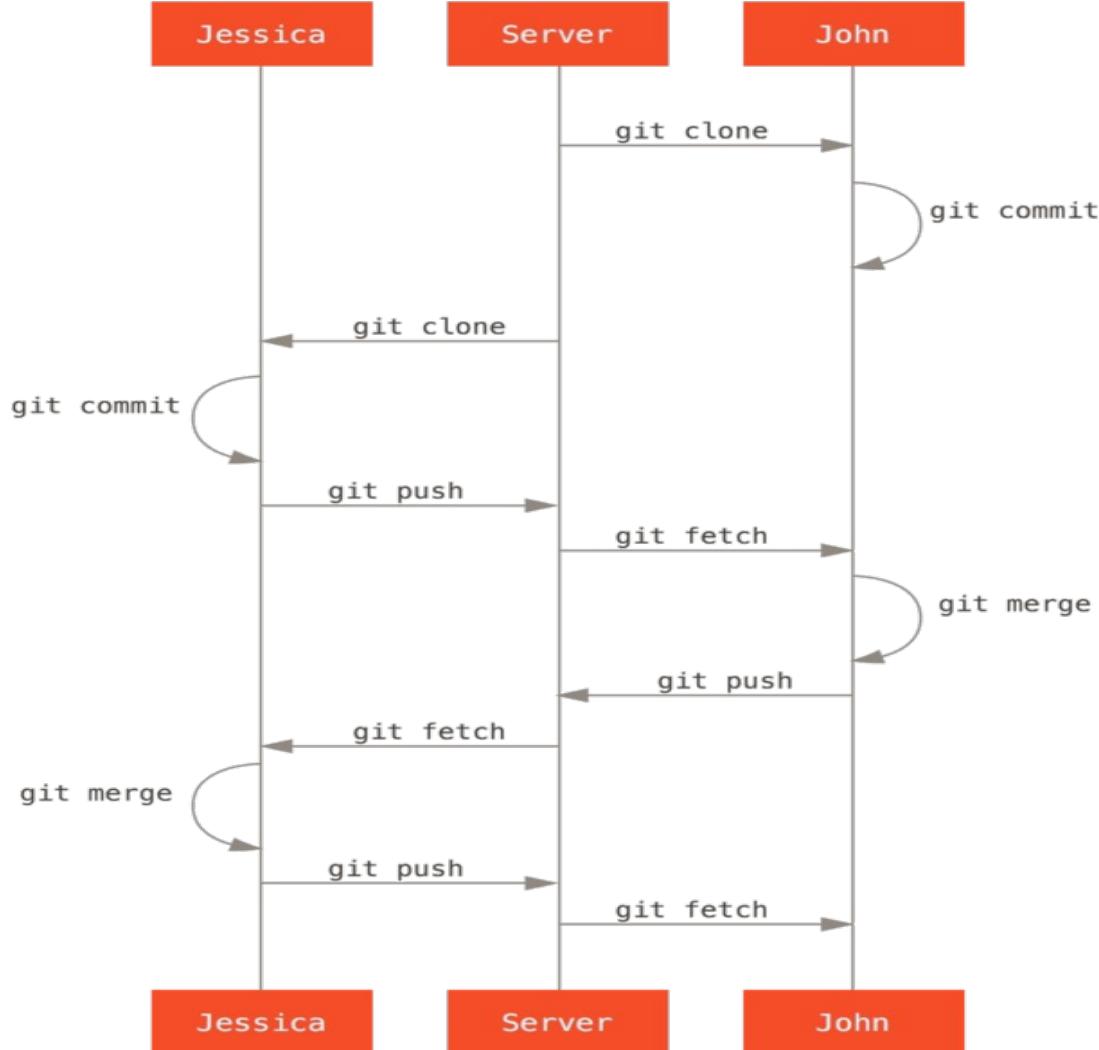


Travail collaboratif (2/4): Workflow Integration-Manager

Travail collaboratif (3/4): Workflow dictateur et lieutenants (Unix)



Travail collaboratif (4/4): Scénario de collaboration



TP-2: Travail collaboratif

- Sur votre gitlab-ce créé précédemment, créez 3 comptes (manager, dev1 et dev2)
- Créez un groupe de projet (nommé eazytraining)
- Rajoutez manager en tant que owner du groupe de projet, dev1 et dev2 en tant que contributeur
- Créez un repo deploy-users
- Poussez votre code sur gitlab en tant que manager et testez le scénario de collaboration suivant :
- Le dev 1 modifie le code (crée un fichier install.sh dans lequel il ajoute un script Bash d'installation de nginx pour Ubuntu) à travers une nouvelle branche feat1 et le pousse vers le repo distant, puis crée une merge request qui devra être validé par le manager
- Le dev2 à son tour effectue une modification (Crée le même fichier install.sh dans lequel il installe également nginx mais sous centos) à travers la branche feat2 puis pousse sur le repo distant et demande également une merge request au manager
- Le manager vérifie le travail et valide les merges request en gérant le conflit et privilégiant le travail fait l'environnement Ubuntu

TP-3: Accès SSH au Repo

Configurez l'accès SSH à l'utilisateur Manager

Testez le bon fonctionnement en réalisant une modification sur le repo deploy-users en poussant la modif autravers d'une branche "feat3" sans besoin d'utilisation du login et pass



GitLab

LA NOTION DE LABELS

Concepts et Création de label

- Utilisés pour catégoriser les issues et les merge requests
- Exemple Crédit d'un label « bug » qui regroupe tous les issues ou problèmes liés aux erreurs dans l'environnement de production
- Permet de réaliser des filtres sur les types de issues
- Il existe 02 types de Labels, (les group Labels et les projects Labels)

Labels	
Bug	Issues on production environment
default	minor error on production env
development	develop or implement new functionality in prod env
warning	warning in production env



GitLab

LA NOTION DE ISSUES

Concepts et Création de Issue

- Actions ou tâches à réaliser afin de corriger les problèmes rencontrés dans un projet ou dans le but de l'améliorer
- Un Issue est lié à un projet et peut être rattaché à un ou plusieurs labels
- Les issues sont de deux types : Issue ou Incident
- Possibilité de créer des templates afin de faciliter la création des issues pour certains projets

The screenshot shows a list of issues in a GitLab project. At the top, there are three buttons: 'Open' (3), 'Closed' (2), and 'All' (5). Below the buttons is a search bar with 'Recent searches' and a placeholder 'Search or filter results...'. The first issue listed is '#5 · created 8 minutes ago by Frazer · Bug · test'. The second issue is '#2 · created 1 hour ago by Frazer · Nov 30, 2021 · Bug · incident · fix issue on deploy job in prod env'. The third issue is '#1 · created 1 hour ago by Frazer · Nov 30, 2021 · development · add flask module'. Each issue card includes a date, author, labels, and a brief description.

Issue #	Created	Author	Labels	Description
#5	8 minutes ago	Frazer	Bug, test	test
#2	1 hour ago	Frazer	Bug, incident	fix issue on deploy job in prod env
#1	1 hour ago	Frazer	development	add flask module



GitLab

LA NOTION DE MILESTONE

Concepts et Création de Milestone

- Permet de séquencer les phases d'un projet (Release) en regroupant les issues
- Regrouper les issues devant impérativement se terminer dans une plage de temps déterminée entre eux
- Permet le suivi groupé de ces issues en vue de l'atteinte de l'objectif qui est la production d'une release ou étape de notre projet

A screenshot of a GitLab interface showing the creation of a new milestone. The milestone is titled "version 1.0" and is set to start on "Nov 30, 2021–Dec 31, 2021". It contains three issues:

- test**: #5 · created 41 minutes ago by Frazer · version 1.0 · Bug
- fix issue on deploy job in prod env**: #2 · created 1 hour ago by Frazer · version 1.0 · Nov 30, 2021 · Bug · incident
- add flask module**: #1 · created 2 hours ago by Frazer · version 1.0 · Nov 30, 2021 · development

At the bottom, there are buttons for "Edit", "Promote", "Close milestone", and "Delete". A progress bar indicates "0% complete". The "Start date" is listed as "Nov 30, 2021".

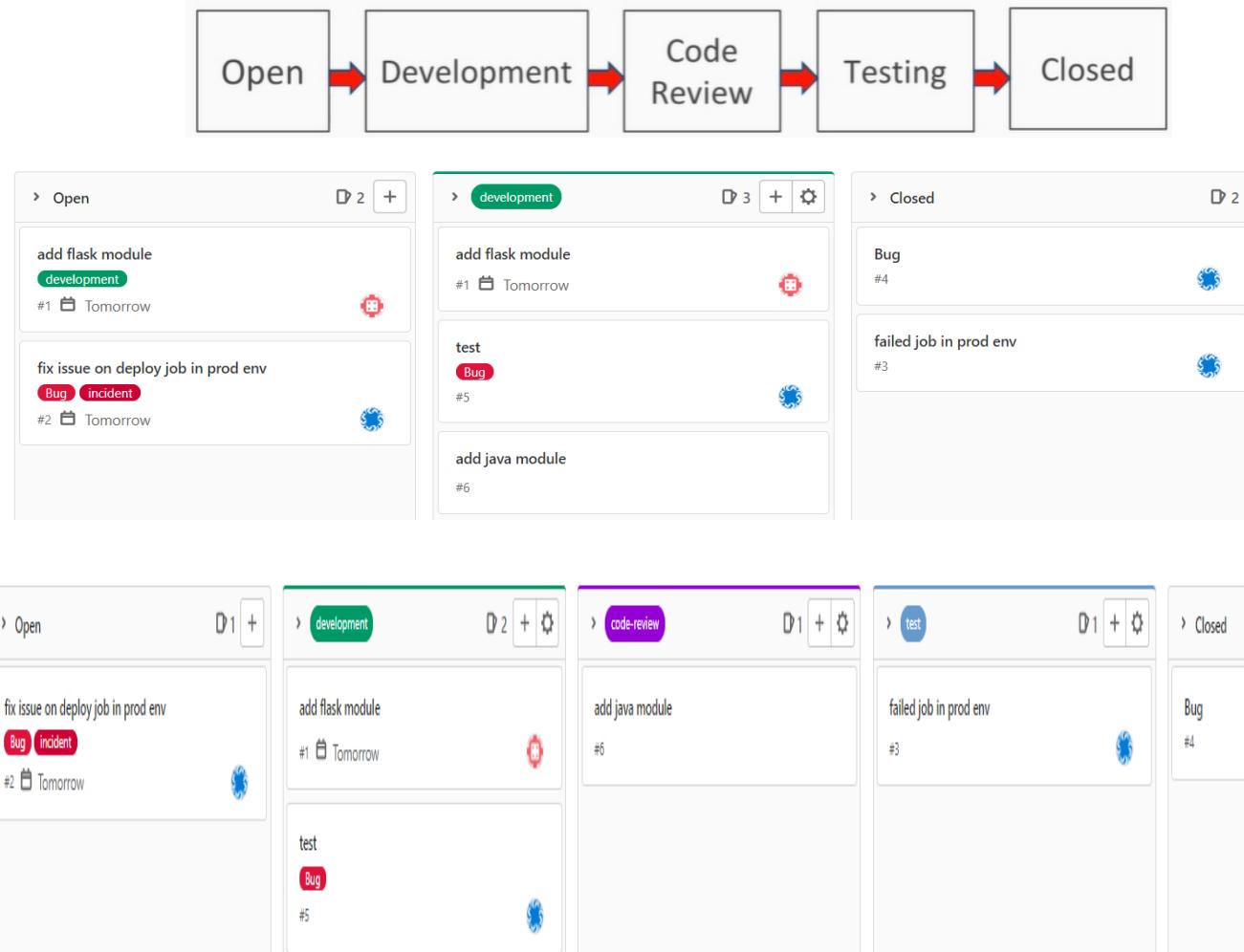


GitLab

LA NOTION DE ISSUE BOARD

Concepts et Création de Issue Board

- Créer des tableau de bord permettant d'augmenter la visibilité sur le statut du projet (état d'avancement)
- Permet de classer les différents issues sur un tableau de bord afin de faciliter leur suivi
- Accessible à tous les membres de l'équipe projet
- Dashboard pouvant classer et regrouper les issues par colonne ou étape (To Do, Doing, Done ou À faire, En cours, fait)
- Possibilité de créer des étapes dans notre issue board, les étapes par défaut et obligatoire sont (Open et closed, pour les issues Ouverts et fermés). Entre l'état ouvert et fermé on peut créer d'autres dépendamment de notre cycle de vie interne



TP-4: Travail collaboratif (Label, Issue ...)

- Vous êtes le manager d'une équipe de développeurs (équipe de 03 personnes, Manager, dev1 et dev2)
- Le manager met sur pied le serveur Gitlab-ce et crée les utilisateurs nécessaires pour le manager et les deux dev et partage les identifiants et l'adresse du serveur dans le fil de discussion
- Le manager crée un groupe (orsys-gitlab) et crée un projet (first-project) et bien entendu ajoute les utilisateurs comme membre du groupe en tant que développeurs
- Puis créez 03 labels (bug, warning, development) et un Template pour le label warning qui devra automatiquement lié les issue au label en question, notifier le manager et les assigner automatiquement au dev2
- Puis créer un issue lié à chacun des label en utilisant bien sur le Template précédemment crée lors de la création de l'issue lié au label warning
- Crée un milestone qui débute ce jour et se termine dans un mois auquel vous rattacherez tous vos issues
- Configurez les issues board et ajouter les étapes code; code-review et test.
- Le manager devra gérer les different issues depuis le issue board et déplacer un issue à l'étape code, le second à l'étape code-review

PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-Cl

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Monitoring

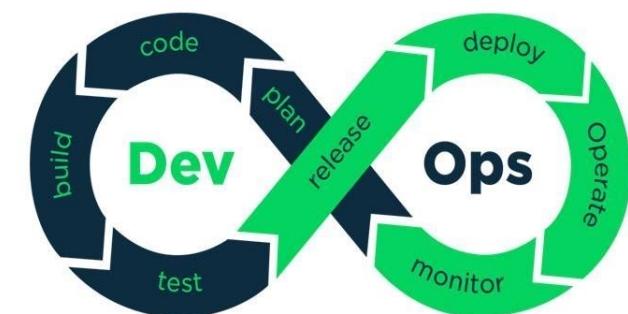
GitOps

Mini-Projet

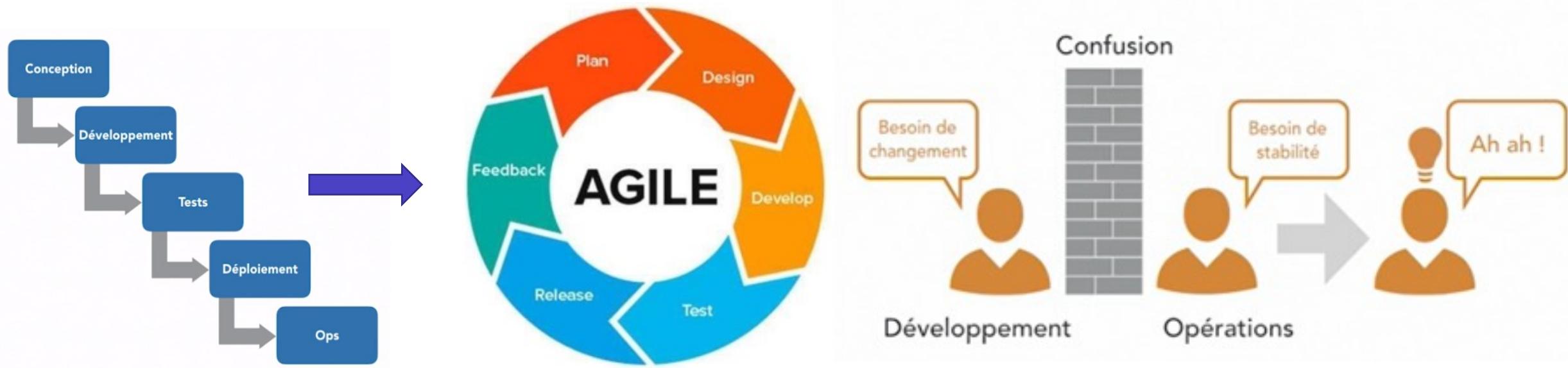
INTRODUCTION AU DEVOPS ET AU CI/CD : LE DEVOPS

- Agile: méthode de développement
- DevOps: agilité dans le Dev et l'Ops = CI + CD

Agile vs. DevOps



Méthodes traditionnelles vs Méthodes Agiles



Méthodes traditionnelles:

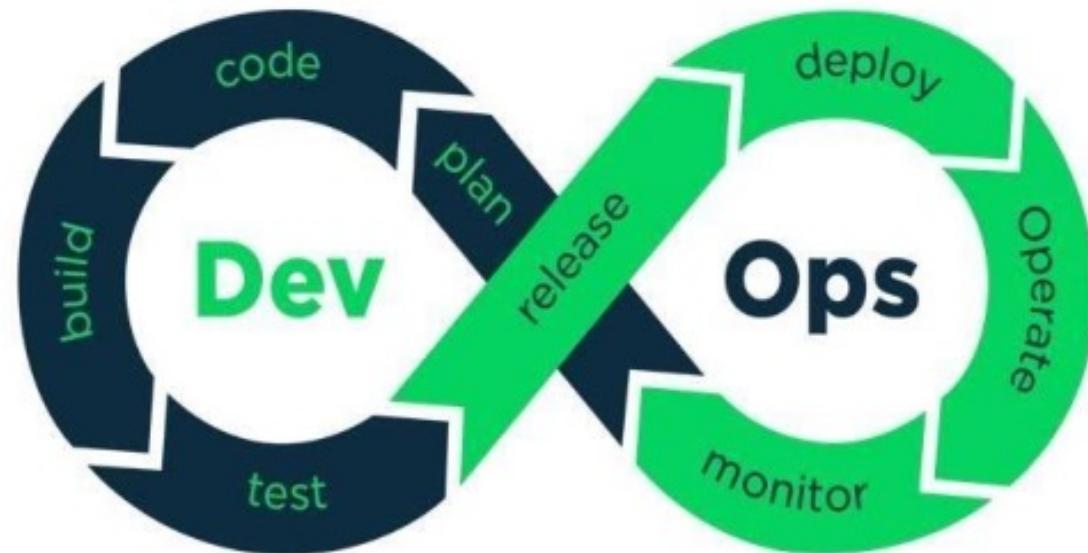
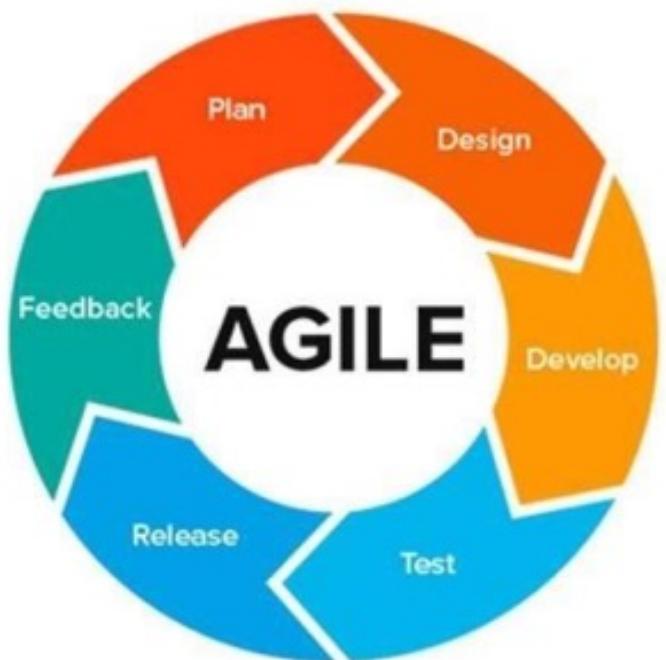
- Taches traitées séparément et séquentielles
- Grande lenteur dans tout le processus
- Mur de la confusion existant

Méthodes agiles:

- Agilité au niveau du développement
- Grande lenteur dans le déploiement
- Mur de la confusion existant

Méthodes Agiles Vs Devops

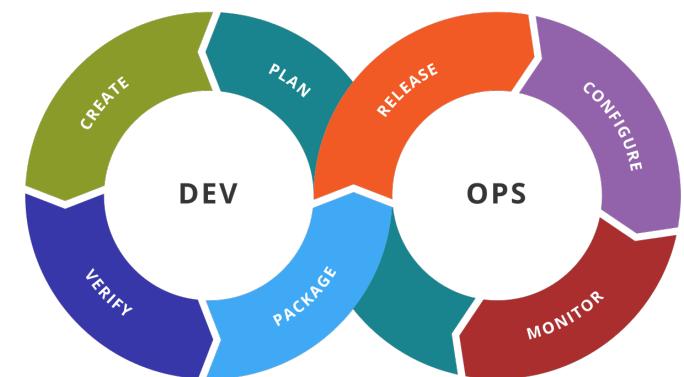
Agile vs. DevOps



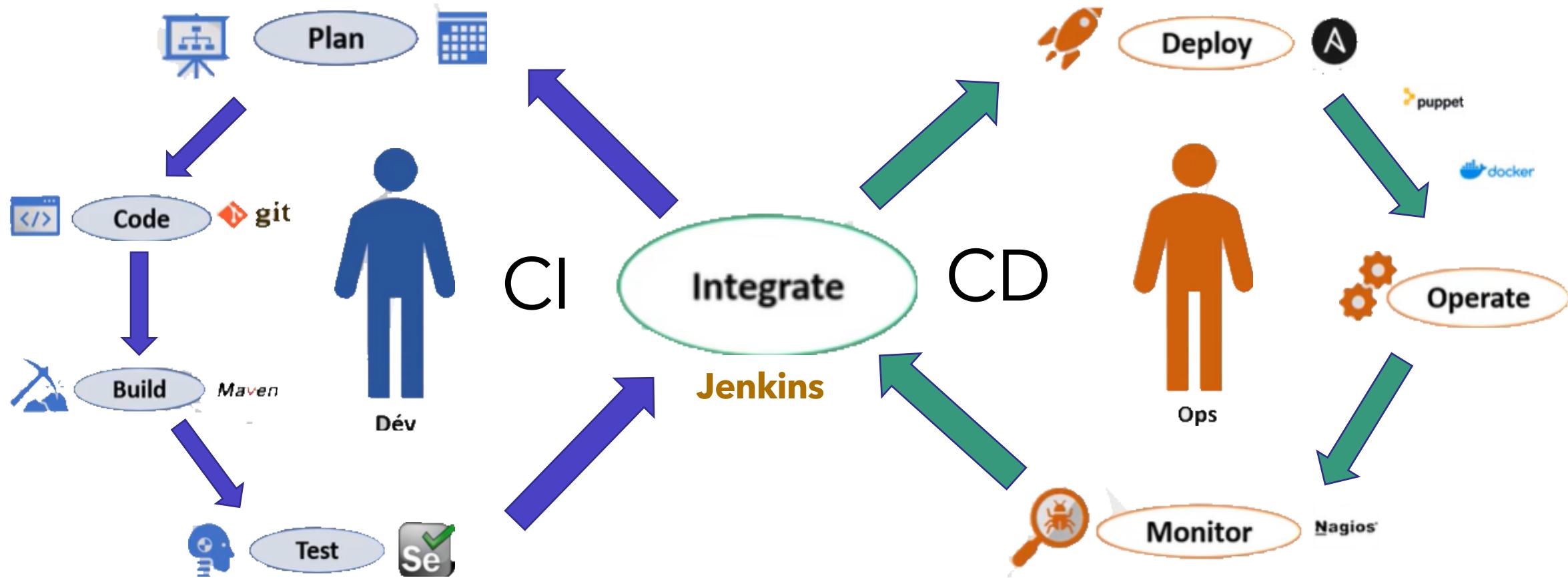


DevOps : qu'est-ce que c'est ?

- C'est un mouvement, une pratique technique visant à l'alignement des développeurs (**Dev**) et des opérationnels (**Ops**) vers des objectifs communs.
- Les **Ops** désignent de façon générale les administrateurs système, base de données, réseaux, stockage, les exploitants, etc ...
- Le DevOps est une **manière de travailler**, et les outils disponibles viennent promouvoir cette manière de travailler.
- Parlant de DevOps, les vocabulaires suivants sont incontournables : l'automatisation à travers l'**Infrastructure as Code**, les chaînes d'intégration continue et de déploiement continu (**CI/CD**), les **tests** unitaires, les méthodes **agiles**, etc ...

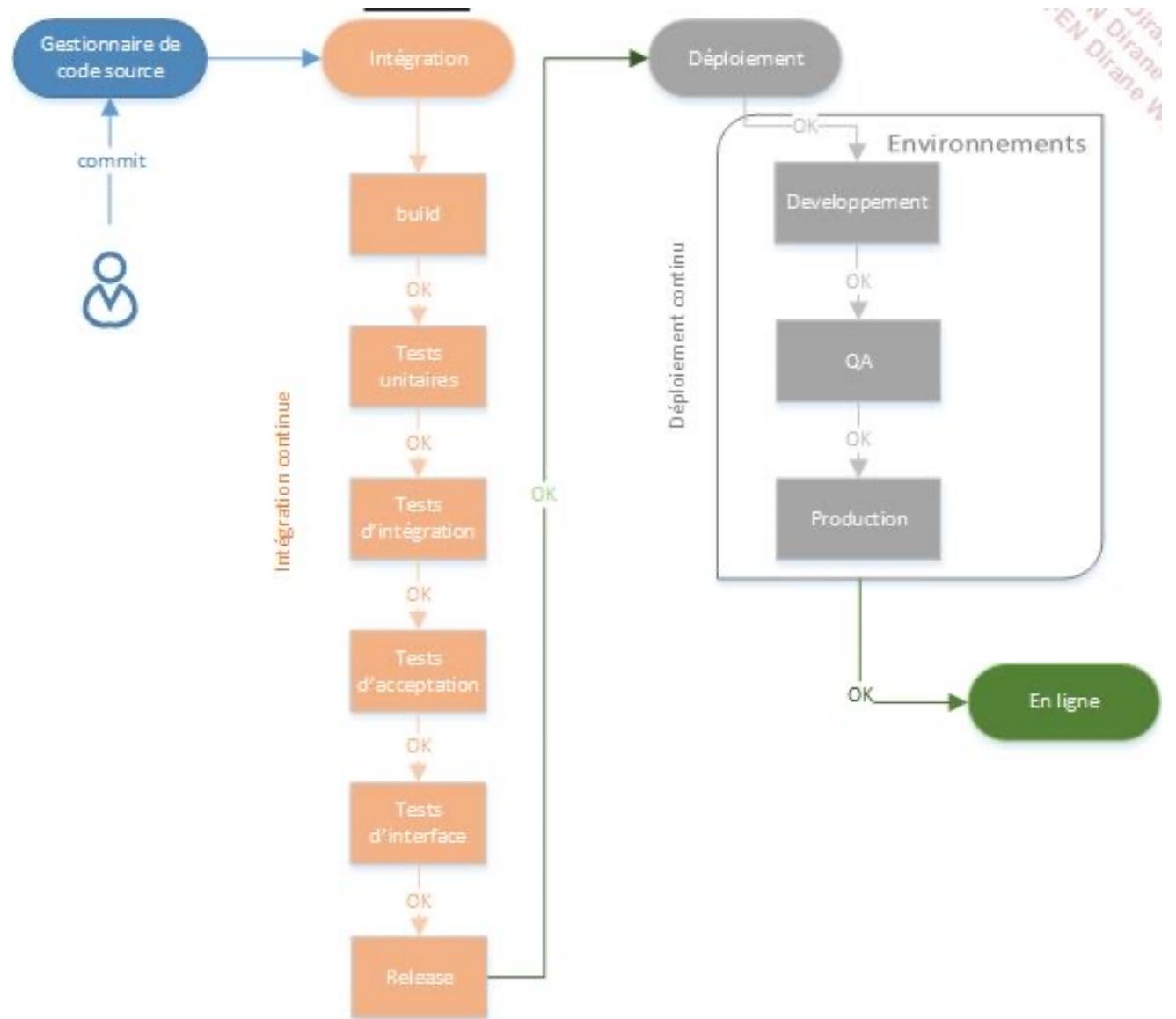


Process Devops ou Pipeline CI/CD (2/3): CI/CD



INTRODUCTION AU DEVOPS ET AU CI/CD : CI/CD

- Intégration en continu
- Test en continu
- Déploiement en continu
- Gitlab



PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-Cl

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Monitoring

GitOps

Mini-Projet



GITLAB-CI: CI/CD POUR DEVOPS



Par Frazer SADO

GITLAB (3/6):

GITLAB-CI

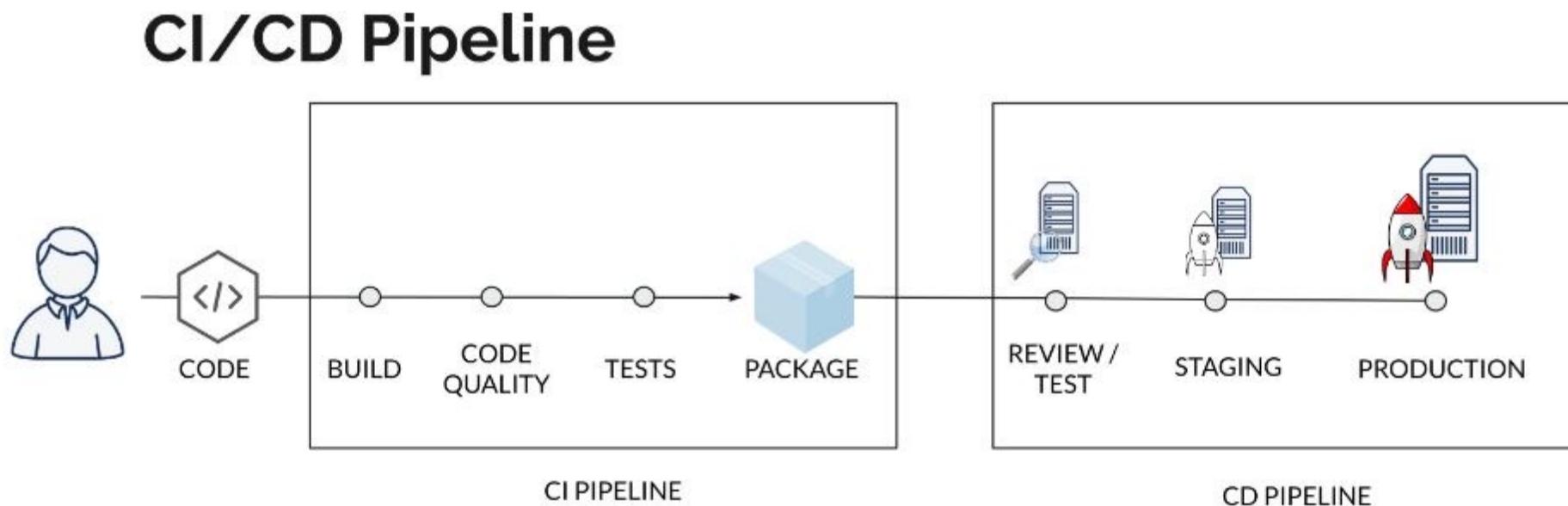
-
- C'est quoi ?
 - Intégré à gitlab depuis la v8.0
 - Git Hooks
 - Pipeline
 - Pourquoi ?
 - Code et build ensemble
 - Simple à lire et écrire
 - Scalable
 - Isolation



GitLab CI

Open Source Continuous Integration made easy

GITLAB (4/6): GITLAB-CI PIPELINE



GITLAB (5/6): GITLAB-CI CONCURRENT

 TeamCity

cloud
 bees

urban
 {code}

 BuildMaster

 Octopus Deploy

 SEMAPHORE

 Buddy

 Bamboo

 Buildkite



 AWS CodePipeline

 AppVeyor

 Travis CI

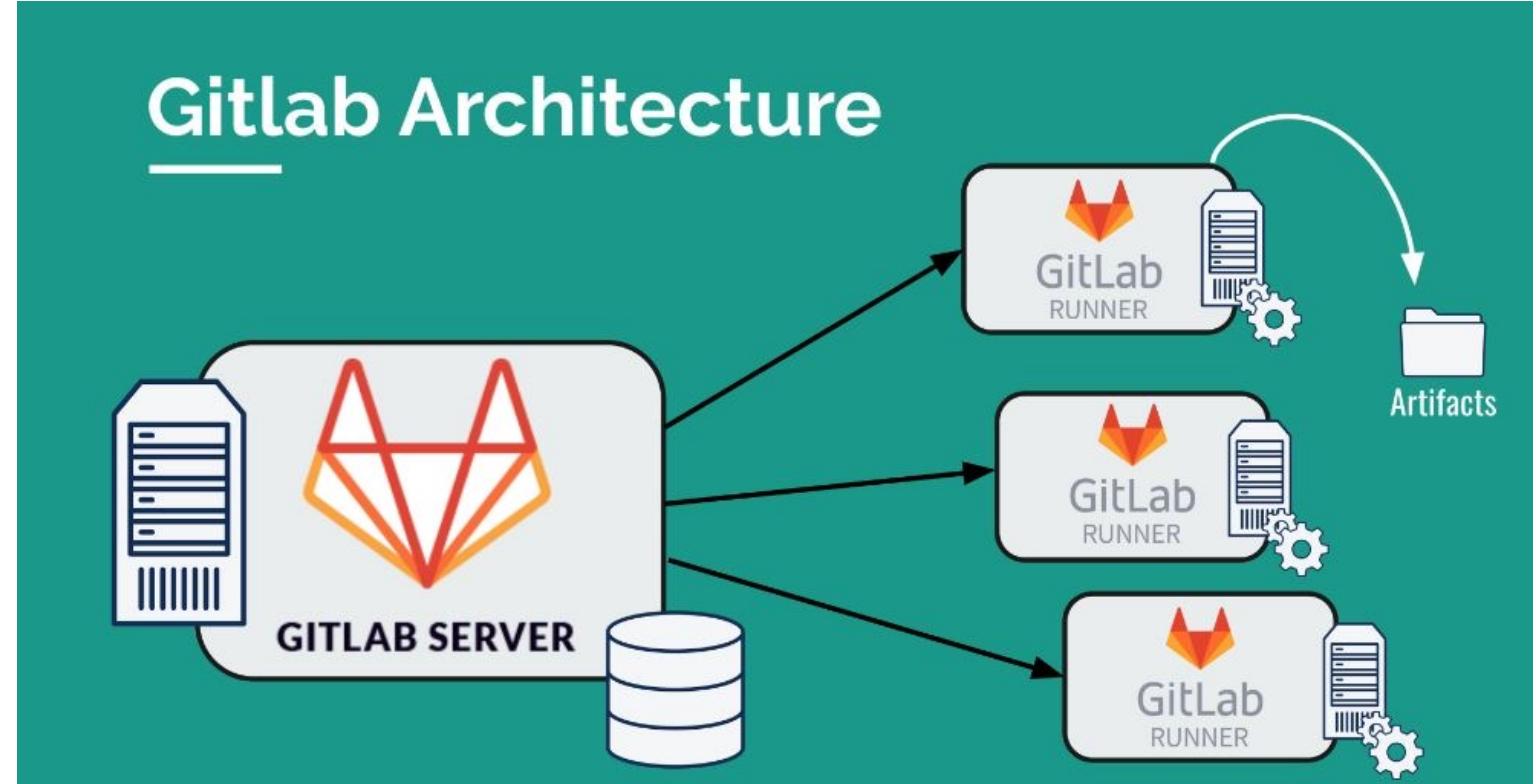
 DRONE

 circleci

 go

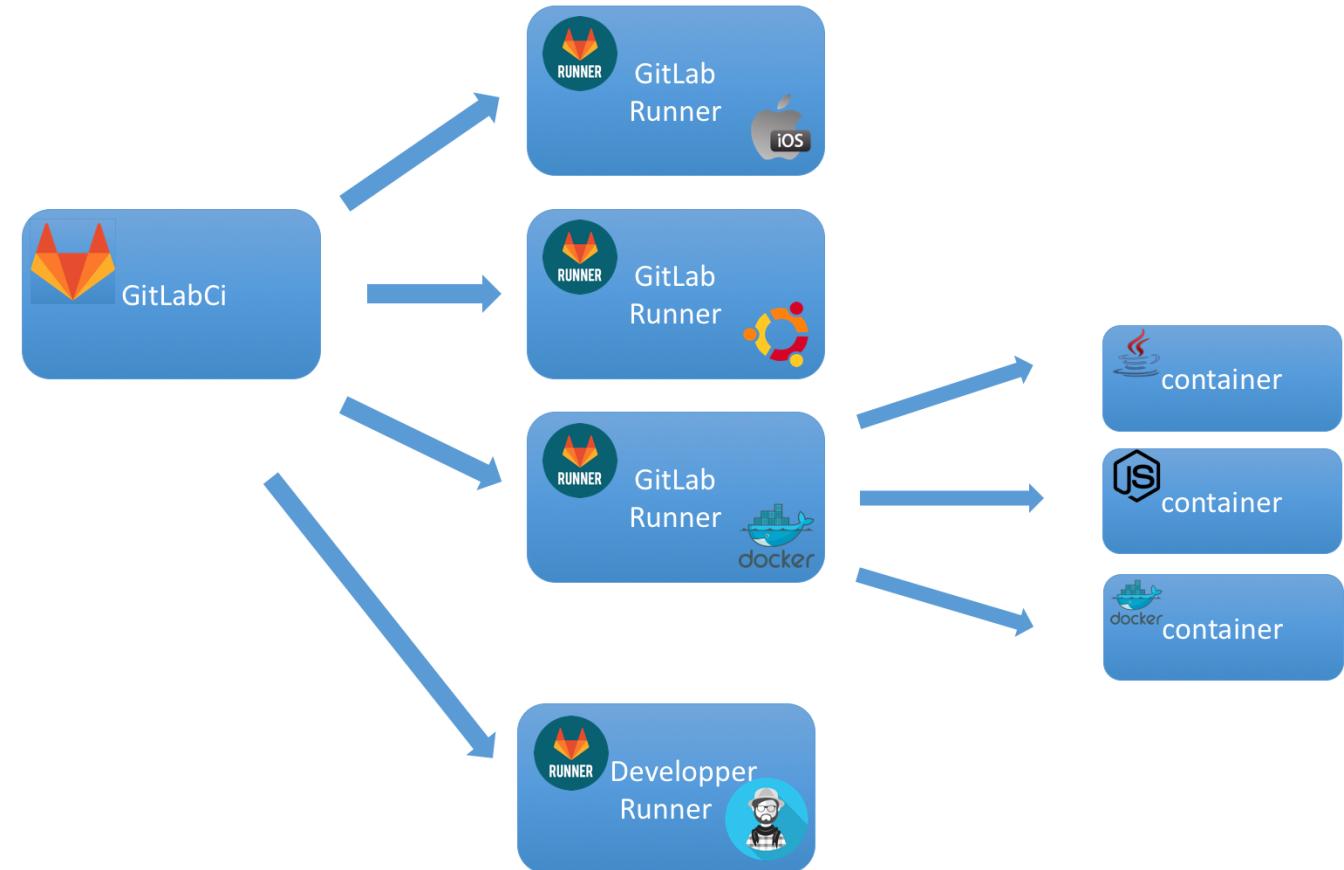
 GitLab

GITLAB (6/6): GITLAB-CI ARCHITECTURE





GITLAB-CI: RUNNERS CONCEPTS

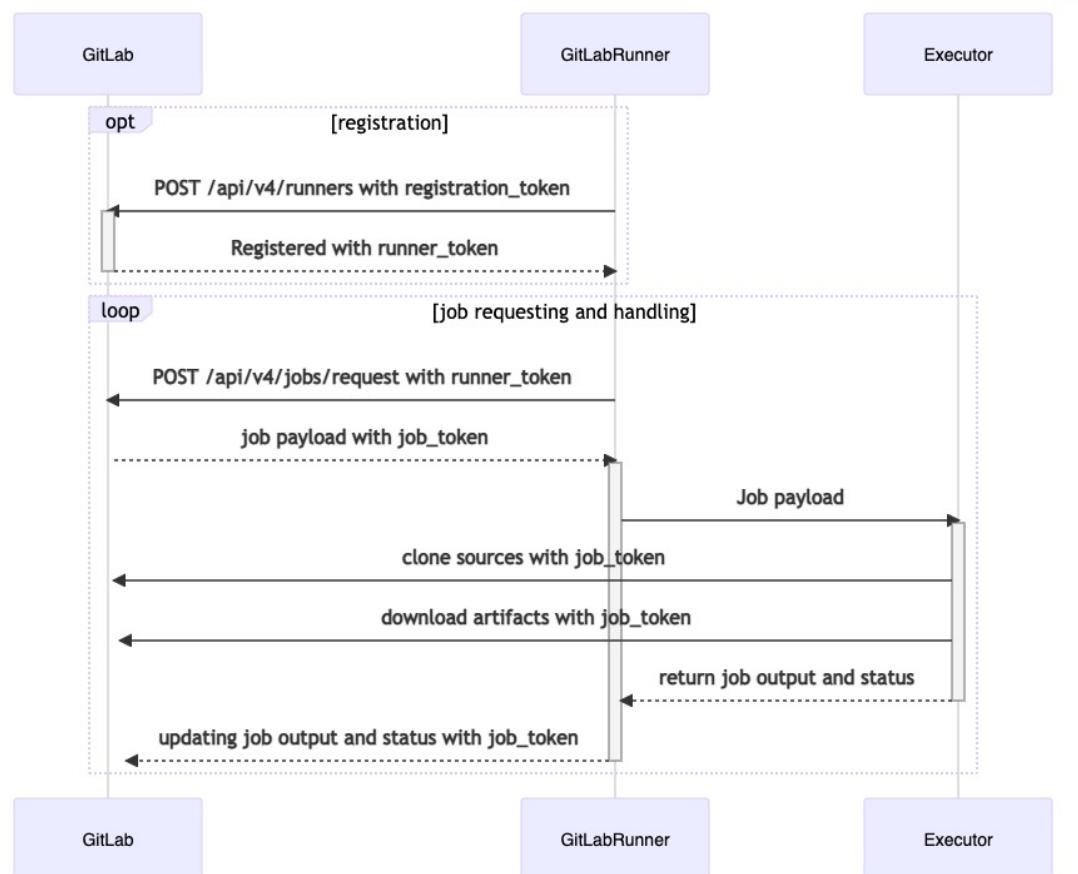


Concepts

- Compatibles avec plusieurs OS (Linux, Windows, macOS, FreeBSD)
- Peut être installé dans un containers, ou en dur sur une machine en téléchargeant les fichiers binaires
- Il existe 03 types de Runners :
 - Les Shared Runners
 - Les Groups Runners
 - Project Runners
- Il existe une limite de compilation avec les shared Runners de Gitlab
- Pas de limite de compilation avec les specific Runners



GITLAB-CI: RUNNERS CONCEPTS



Concepts

- La machine Gitlab-Runner est en effet un service déployé sur une machine tierce capable de communiquer avec Gitlab Serveur
- Le Gitlab Runner a besoin d'un executor pour pouvoir exécuter les ordres reçus du Gitlab server
- Les executors sont des sous-processus qui exécuteront en effet les actions de notre pipeline
- Il existe plusieurs types de executor (ssh, shell, docker, Kubernetes ...)
- La configuration d'un GitLab Runner passe donc par le choix du type de Runner à utiliser.



GITLAB-CI: RUNNERS CONCEPTS

Executor	SSH	Shell	VirtualBox	Parallels	Docker	Kubernetes	Custom
Clean build environment for every build	✗	✗	✓	✓	✓	✓	conditional (4)
Reuse previous clone if it exists	✓	✓	✗	✗	✓	✗	conditional (4)
Runner file system access protected (5)	✓	✗	✓	✓	✓	✓	conditional
Migrate runner machine	✗	✗	partial	partial	✓	✓	✓
Zero-configuration support for concurrent builds	✗	✗ (1)	✓	✓	✓	✓	conditional (4)
Complicated build environments	✗	✗ (2)	✓ (3)	✓ (3)	✓	✓	✓
Debugging build problems	easy	easy	hard	hard	medium	medium	medium



GITLAB-CI: RUNNERS INSTALLATION

Installation à l'aide de containers docker

L'installaion se fait en 02 étapes:

- Enregistrement du Runner à l'aide du token
- Activation du Runner en lançant son container

Etape 1: Récupération du Token, Enregistrement et création du fichier de config

Admin Area > Runners

Set up a group Runner for a project

Set up a shared runner manually

1. Install GitLab Runner and ensure it's running.
2. Register the runner with this URL:
<http://3.238.108.61:10080/>
3. And this registration token:
[EyXaoPHY9fxBhzxC61sU](#)

And this registration token:
[XndFhPWViaz8yaYSLfy_](#)

Set up a specific Runner for a project

1. Install GitLab Runner and ensure it's running.
2. Register the runner with this URL:
<http://3.238.108.61:10080/>

And this registration token:
[x3-HBisnrH144V-BdyzA](#)

```
docker run --rm -it -v $(pwd)/config:/etc/gitlab-runner gitlab/gitlab-runner register
```

```
Enter the GitLab instance URL (for example, https://gitlab.com/):  
http://3.238.108.61:10080  
Enter the registration token:  
EyXaoPHY9fxBhzxC61sU  
Enter a description for the runner:  
[f25797f04d69]: frazer-shared-runner  
Enter tags for the runner (comma-separated):  
Name=frazer-shared-runner  
Registering runner... succeeded  
runner=EyXaoPHY  
Enter an executor: docker-ssh+machine, kubernetes, custom, parallels, ssh,  
docker  
Enter the default Docker image (for example, ruby:2.6):  
ubuntu  
Runner registered successfully. Feel free to start it, but if it's running
```

Type/State	Runner
shared	#1 (WbwVq2my) frazer-shared-runner

Installation avec les Binaires Linux : <https://docs.gitlab.com/runner/install/linux-manually.html>



GITLAB-CI: RUNNERS INSTALLATION

Installation à l'aide de containers docker

L'installaion se fait en 02 étapes:

- Enregistrement du Runner à l'aide du token
- Activation du Runner en lançant son container

Etape 2: Lancement du container et activation du Runner

```
config.toml
ubuntu@ip-172-31-3-15:~/gitlab-runner/config$ sudo cat config.toml
concurrent = 1
check_interval = 0

[session_server]
  session_timeout = 1800

[[runners]]
  name = "frazer-shared-runner"
  url = "http://3.238.108.61:10080"
  token = "WbwVq2myc4_BnxUeQxWw"
  executor = "docker"
  [runners.custom_build_dir]
  [runners.cache]
    [runners.cache.s3]
    [runners.cache.gcs]
    [runners.cache.azure]
  [runners.docker]
    tls_verify = false
    image = "ubuntu"
    privileged = false
    disable_entrypoint_overwrite = false
    oom_kill_disable = false
    disable_cache = false
    volumes = ["/cache"]
    shm_size = 0
```

```
docker run -d --name gitlab-runner --restart always \
  -v $(pwd)/config:/etc/gitlab-runner \
  -v /var/run/docker.sock:/var/run/docker.sock \
  gitlab/gitlab-runner:latest
```

Type/State	Runner	Version	IP Address
shared	#1 (WbwVq2my) frazer-shared-runner	14.5.0	3.238.108.61

Installation avec les Binaires Linux : <https://docs.gitlab.com/runner/install/linux-manually.html>



GITLAB-CI: RUNNERS INSTALLATION

Installation et configuration de Runners

Notre infrastructure contient donc 01 Gitbab Serveurs

et 03 Runners dont 01 shared runners, 01 group

Runners et 01 project Runners

Specific runners

These runners are specific to this project.

Set up a specific Runner for a project

1. Install GitLab Runner and ensure it's running.

2. Register the runner with this URL:

<http://3.238.108.61:10080/>

And this registration token:

x3-HBisnrH144V-BdyzA

[Reset registration token](#)

[Show Runner installation instructions](#)

Shared runners

These runners are shared across this GitLab instance.

The same shared runner executes code from multiple projects, unless you configure autoscaling with [MaxBuilds](#) set to 1 (which it is on GitLab.com).

Enable shared runners for this project



Available shared runners: 1

#1 (WbWVq2my)

frazer-shared-runner

[Name=frazer-shared-runner](#)

Group runners

These runners are shared across projects in this group.

Group runners can be managed with the [Runner API](#).

[Disable group runners](#) for this project

Available specific runners

#3 (P4WY6sKj)

frazer-project-runner1

[Name=frazer-project-runner1](#)

Remove runner

Available group runners: 1

#2 (UL-ux1j5)

frazer-group-runner

[Name=frazer-group-runner](#)

7107f43cd32d	gitlab/gitlab-runner:latest	/usr/bin/dumb-init ...	2 minutes ago	Up 2 minutes	frazer-project-runner1
74236f5615ab	gitlab/gitlab-runner:latest	/usr/bin/dumb-init ...	9 minutes ago	Up 9 minutes	gitlab-runner



TP-5: SPECIFIC RUNNER

- Créez un compte sur gitlab.com
- Créez un groupe, puis un repo « alpinehelloworld »
- Récupérez le code de l'application et poussez le sur votre repo <https://github.com/eazytrainingfr/alpinehelloworld.git>
- Vérifiez bien qu'il existe des shared runners dans votre infrastructure Gitlab
- Créez un group Runner lié à votre groupe avec comme executor shell et activez le
- Créez également Runner lié à votre projet avec comme executor docker et activez le
- Vérifiez votre infrastructure et rassurez vous d'avoir de disponible pour votre projet en plus des shared runners, les 02 runners précédemment créés.

PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-Cl

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Monitoring

GitOps

Mini-Projet



LE YAML

YAML: Langage de structuration de données

Généralement utilisé pour représenter les données

XML

```
<Servers>
  <Server>
    <name>Server1</name>
    <owner>John</owner>
    <created>12232012</created>
    <status>active</status>
  </Server>
</Servers>
```

JSON

```
{
  Servers: [
    {
      name: Server1,
      owner: John,
      created: 12232012,
      status: active,
    }
  ]
}
```

YAML

```
Servers:
  - name: Server1
    owner: John
    created: 12232012
    status: active
```



LE YAML

LE YAML

- Langage de type bloc où chaque bloc représente une données sous un format et type spécifique
- Les données sont généralement définies sous forme de clé/valeur, où la clé et la valeur sont séparé par « : »
- Toujours respecter l'espace après le « : » permettant de séparer la clé de sa valeur
- Respecter l'indentation et l'aligement des champs



yaml icon



yml file yaml



yaml example

YAML Example Code

```
1 # Employee records
2
3 - Employee one:
4   name: Alex
5   job: Developer
6   skills:
7     - python
8     - C/C++
9     - pascal
10
11 - Employee two:
12   name: Bob
13   job: Tester
14   skills:
15     - lisp
16     - fortran
17     - REST APIs
```

CSEstack.org

Block 1

Block 2



LE YAML - LES CHAINES DE CARACTÈRES OU NOMBRES

Valid variable names	Not valid
<code>foo</code>	* <code>foo</code> , Python keywords such as <code>async</code> and <code>lambda</code>
<code>foo_env</code>	playbook keywords such as <code>environment</code>
<code>foo_port</code>	<code>foo-port</code> , <code>foo port</code> , <code>foo.port</code>
<code>foo5</code> , <code>_foo</code>	<code>5foo</code> , <code>12</code>

```
---
```

```
# Un fichier yaml démarre par les 3 tirets ci-dessus
# Déclaration simple
chaine_simple: "Une chaîne simple"
# Ici _42 va contenir un entier :
_42: 42
# _33 va contenir un chiffre à virgule :
_33: 33.333
```



LE YAML - LES TABLEAUX / LISTES

```
a:  
  - 1  
  - 2  
  - "trois"
```

```
# Version compacte  
a: [ 1, 2, "trois" ]
```

```
# Version très compacte  
a: [1,2,"trois"]
```

```
# Version invalide (manque l'espace après  
a:[1,2,"trois"]
```



LE YAML - STRUCTURES CLÉ/VALEUR (DICTIONNAIRE)

```
utilisateur1:  
  nom: perre  
  prenom: yannig  
  date_de_naissance:  
    jour: 7  
    mois: 11  
    annee: 1977
```

```
utilisateur1:  
  nom: perre  
  prenom: yannig  
  
utilisateur2:  
  nom: perre  
  prenom: sarah
```



LE YAML - LISTE DE DICTIONNAIRES

```
liste_utilisateurs:  
  - nom: perre  
    prenom: yannig  
  - nom: perre  
    prenom: sarah
```

```
# Version un peu plus compacte  
users:  
  - { nom: perre, prenom: yannig }  
  - { nom: perre, prenom: sarah }  
# Version plus compacte, mais moins lisible  
users: [{nom: perre, prenom: yannig},{nom: perre, prenom: sarah}]
```



LE YAML

LE YAML

- Les Dictionnaires sont des structures de données non ordonnées
- Les listes sont des structures de données ordonnées

Dictionary – Unordered
List – Ordered

Dictionary/Map

Banana:
Calories: 105
Fat: 0.4 g
Carbs: 27 g

Banana:
Calories: 105
Carbs: 27 g
Fat: 0.4 g



Array/List

Fruits:
- Orange
- Apple
- Banana



Fruits:
- Orange
- Banana
- Apple

PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-Cl

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Monitoring

GitOps

Mini-Projet

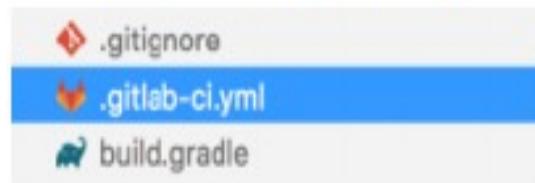


BUILD (1/4): JOBS

PIPELINE - BUILD

- Le code étant disponible, nous commencerons notre pipeline par le Build de notre application
- Toute action que l'on souhaite exécuter dans un pipeline se fait par le biais d'un Job
- Créer le fichier « `.gitlab-ci.yml` » à la racine du repo
- Modifier ce fichier et définir le bloc « `job` » (script yaml)
- Le job basique est constitué d'un stage (étape)
- D'un bloc script qui représente la liste des actions à exécuter dans cette étape
- Possibilité de préciser l'environnement dans lequel s'exécutera le job

`.gitlab-ci.yml` in root



`job1:`

`stage: build`

`script:`

`- execute-script-for-job1`

`Define environment`

`image: node:8.3`



BUILD (2/4): STAGES

- Ordre d'exécution
- Les grandes lignes (analyse rapide)

stages:

- **build**
- **test**
- **deploy**

job1:

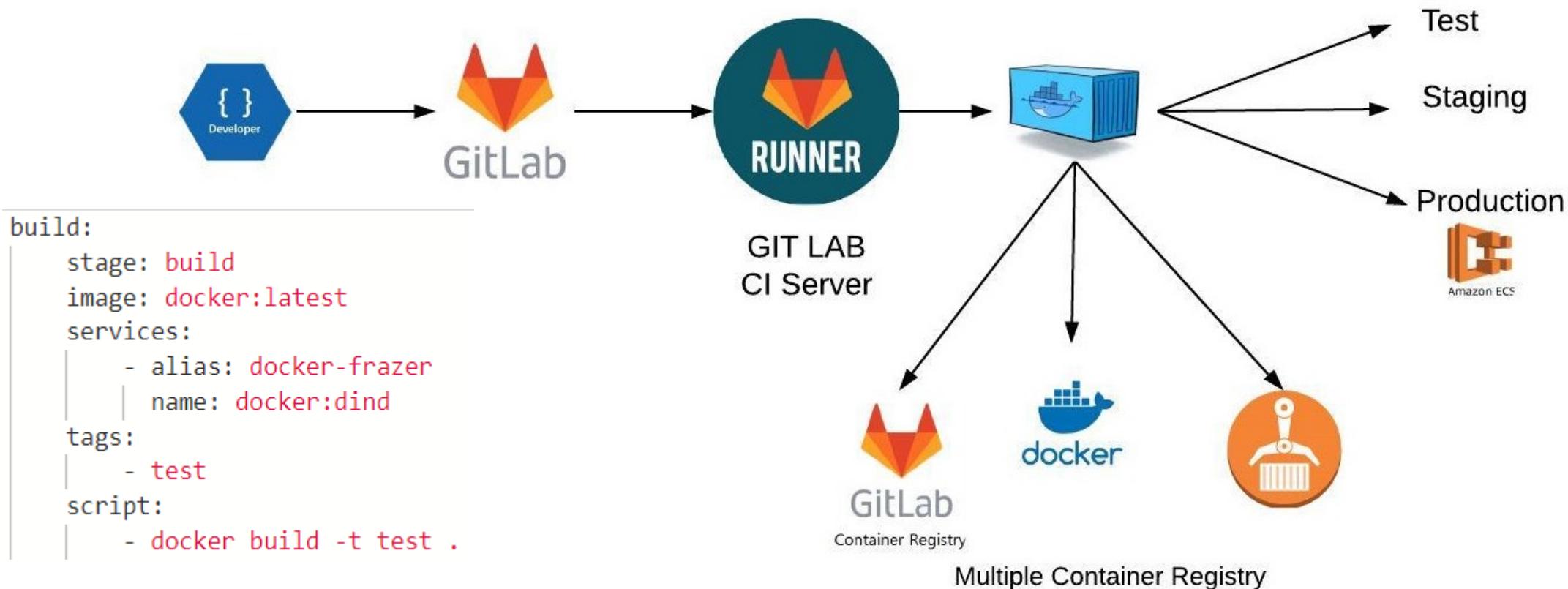
stage: build

script:

- **execute-script-for-job1**



BUILD (3/4): DOCKER



Build Docker Container using GITLab CI / CD



BUILD (4/4): PIPELINE SCHEDULE

- Pipeline triggers
- Relance d'un pipeline en échec
- Pipeline schedules

All 1,000+	Pending 88	Running 683	Finished 1,000+	Branches	Tags	Run Pipeline	Clear Runner Caches	CI Lint
Status	Pipeline	Triggerer	Commit	Stages				
<input type="text" value="Filter pipelines"/>								
running	#146411330		I\`31649 -o dacc7ea3 Merge branch 'nicolasdular/sto...					
failed	#146410995		I\`32306 -o 9a5d2aa1 Merge branch '12-10-stable-e...					
passed	#146410705		I\`31801 -o 42738af2 Merge branch '210018-remove...					
passed	#146410223		master -o d635c709 Merge branch '22691-externali...					



TP-6: BUILD

- Sur votre compte Gitlab contenant le projet précédemment crée au TP4,
- Créez votre fichier .gitlab-ci.yml
- Créez un job build1 qui va permettre de builder votre application sous forme d'image docker à partir du Dockerfile fourni dans le dossier de l'application (ce job build1 devra utiliser un shared runner fourni par Gitlab)
- Lancez votre pipeline et validez que l'image est bien buildée



TP-7: BUILD

- Créez un deuxième job Build2 job qui va permettre également de Builder votre application sous forme d'image docker à partir du Dockerfile fourni dans le dossier de l'application (specific Runner à utiliser)
- Lancez votre pipeline et validez que l'image est bien buildée
- Que constatez vous?

PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-Cl

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Monitoring

GitOps

Mini-Projet



TEST D'ACCEPTANCE (1/4): JOBS EN PARALLÈLE

```
test artifact:  
  - image: alpine  
  - stage: test  
  - script:  
    - grep -q "Gatsby" ./public/index.html  
  
test website:  
  - stage: test  
  - script:
```

```
pdf:  
  script: xelatex mycv.tex  
artifacts:  
  paths:  
    - mycv.pdf  
  expire_in: 1 week
```

```
docker-build:  
  stage: build  
  script:  
    - docker build -t youtube-webapp .  
    - docker save youtube-webapp > youtube-webapp.tar  
artifacts:  
  paths:  
    - youtube-webapp.tar  
  
test-image:  
  stage: test  
  script:  
    - docker load < youtube-webapp.tar  
    - docker run -d --name webapp -p 80:80 youtube-webapp  
    - sleep 5  
    - apk --no-cache add curl  
    - curl "http://docker-frazer:80" | grep -q "Youtube"
```

TEST D'ACCEPTANCE (2/4): ARTEFACTS



TEST D'ACCEPTANCE (3/4): CACHE

```
image: node:latest

# Cache modules in between jobs
cache:
  key: ${CI_COMMIT_REF_SLUG}
  paths:
    - .npm/

before_script:
  - npm ci --cache .npm --prefer-offline

test_async:
  script:
    - node ./specs/start.js ./specs/async.spec.js
```

```
stages:
  - build
  - test

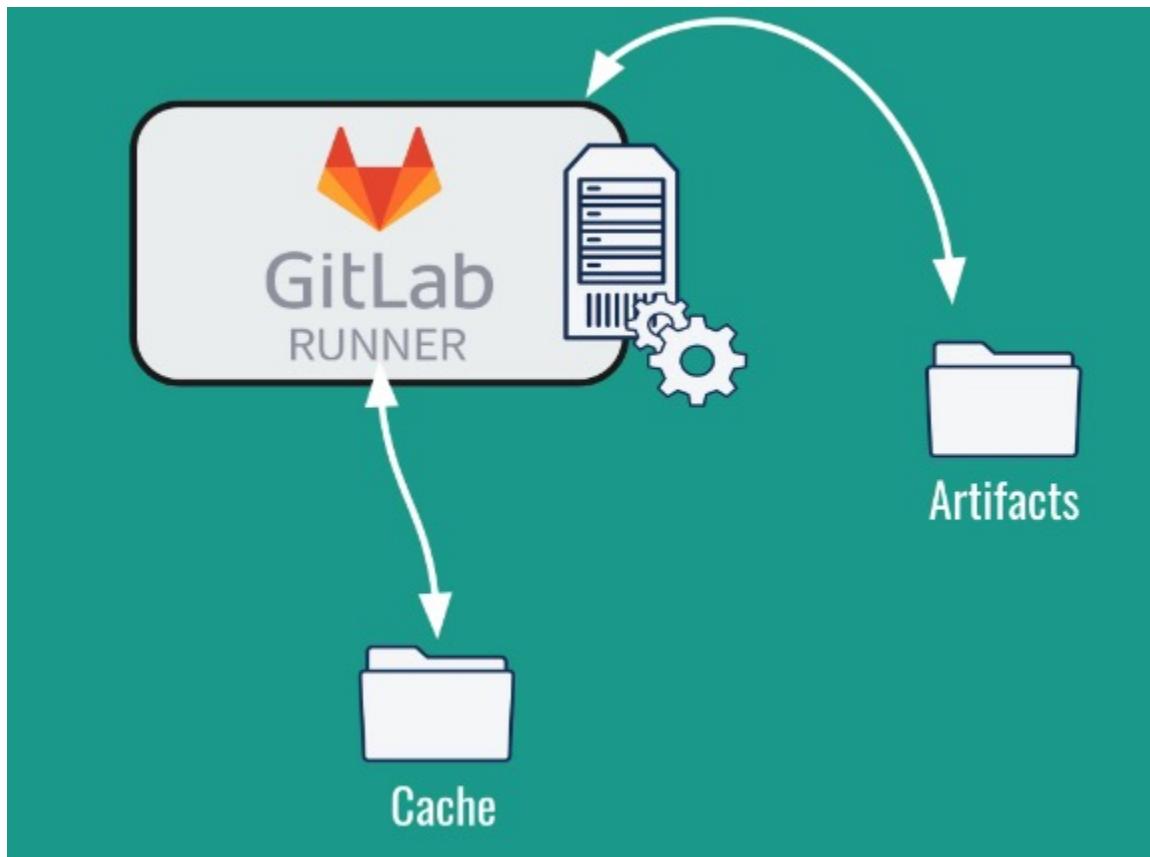
before_script:
  - echo "Hello"

job A:
  stage: build
  script:
    - mkdir vendor/
    - echo "build" > vendor/hello.txt
  cache:
    key: build-cache
    paths:
      - vendor/
  after_script:
    - echo "World"

job B:
  stage: test
  script:
    - cat vendor/hello.txt
  cache:
    key: build-cache
```



TEST D'ACCEPTANCE (4/4): ARTEFACT VS CACHE





TP-8: TEST D'ACCEPTANCE

- Modifiez le pipeline afin de rajouter un job qui lance l'image buildée et la teste à l'aide de curl
- Le curl doit tester si le site contient bien « Hello world! »
- Vous aurez un soucis car l'image n'existera pas au moment où vous allez lancer le conteneur (la variable \$PORT pourra être mise à 80)
- Vous allez utiliser la notion d'artefact pour permettre à l'image d'être vue par le job suivant (vous pouvez utiliser la notion d'import et export d'image docker)
- Si le test avec curl passe, alors vous êtes prêt pour la prochaine étape

PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-Cl

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Monitoring

GitOps

Mini-Projet

RELEASE (1/2): VARIABLES D'ENVIRONNEMENT PRÉDÉFINIES

```
export CI_JOB_ID="50"
export CI_COMMIT_SHA="1ecfd275763eff1d6b4844ea3168962458c9f27a"
export CI_COMMIT_SHORT_SHA="1ecfd275"
export CI_COMMIT_REF_NAME="master"
export CI_REPOSITORY_URL="https://gitlab-ci-token:abcde-1234ABCD5678ef@example.com/gitlab-org/
export CI_COMMIT_TAG="1.0.0"
export CI_JOB_NAME="spec:other"
export CI_JOB_STAGE="test"
export CI_JOB_MANUAL="true"
export CI_JOB_TRIGGERED="true"
export CI_JOB_TOKEN="abcde-1234ABCD5678ef"
export CI_PIPELINE_ID="1000"
export CI_PIPELINE_IID="10"
export CI_PAGES_DOMAIN="gitlab.io"
export CI_PAGES_URL="https://gitlab-org.gitlab.io/gitlab-foss"
export CI_PROJECT_ID="34"
```



RELEASE (2/2): VARIABLES D'ENVIRONNEMENT UTILISATEUR

Variables ?

Collapse

Environment variables are applied to environments via the runner. They can be protected by only exposing them to protected branches or tags. Additionally, they will be masked by default so they are hidden in job logs, though they must match certain regexp requirements to do so. You can use environment variables for passwords, secret keys, or whatever you want. You may also add variables that are made available to the running application by prepending the variable key with `K8S_SECRET_`. [More information](#)

Type	Key	Value	State	Masked	
Variable	TEST	HELLO WORLD	Protected <input checked="" type="checkbox"/>	Masked <input checked="" type="checkbox"/>	<input type="button" value="–"/>
File	GREETING	HELLO WORLD	Protected <input checked="" type="checkbox"/>	Masked <input checked="" type="checkbox"/>	<input type="button" value="–"/>
Variable	Input variable ke	Input variable	Protected <input checked="" type="checkbox"/>	Masked <input checked="" type="checkbox"/>	

Save variablesHide values



TP-9: RELEASE

- Créez un job qui va tagguer l'image avec deux tag, un qui représentera la branche et l'autre qui représentera le SHA court (voir variable d'environnement prédéfinis)
- Pour donner le nom de votre image vous allez définir ce dernier à l'aide du variable d'environnement que vous appellerez lors du tag
- Poussez l'image tagguée sur votre private registry gitlab
- Vérifiez que l'image a bien été poussée avec les bons tags
- Si c'est le cas, votre release est terminée, vous êtes prêt pour suite 😊

PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-Cl

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Déploiement via SSH

Monitoring

Mini-Projet



DEPLOIEMENT (1/6): ENVIRONNEMENT

```
stages:
  - test
  - build
  - deploy

test:
  stage: test
  script: echo "Running tests"

build:
  stage: build
  script: echo "Building the app"

deploy_staging:
  stage: deploy
  script:
    - echo "Deploy to staging server"
environment:
  name: staging
  url: https://staging.example.com
only:
  - master
```

Available 2 Stopped 0 [New environment](#)

Environment	Deployment	Job	Commit	Updated
production			No deployments yet	
staging	#2 by 	deploy_staging #350350...	 ec75f5bf  Use busybox	27 minutes ago

   Re-deploy

 Deploy prod



DEPLOIEMENT (2/6): VARIABLE

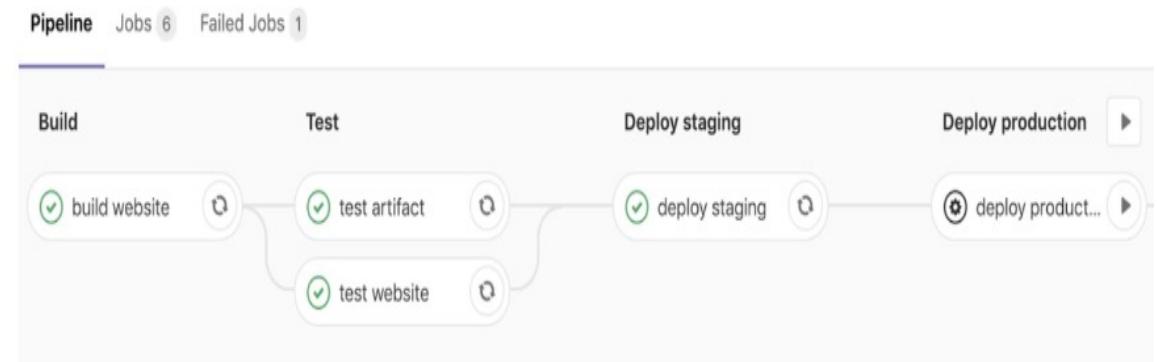
```
variables:  
  STAGING_DOMAIN: instazone-staging.surge.sh  
  PRODUCTION_DOMAIN: instazone.surge.sh
```

```
deploy staging:  
  stage: deploy staging  
  environment:  
    name: staging  
    url: http://$STAGING_DOMAIN  
  script:  
    - npm install --global surge  
    - surge --project ./public --domain $STAGING_DOMAIN
```



DEPLOIEMENT (3/6): MANUAL DEPLOYMENT

```
deploy production:  
  stage: deploy production  
  environment:  
    name: production  
    url: $PRODUCTION_DOMAIN  
  when: manual  
  script:  
    - npm install --global surge  
    - surge --project ./public --domain $PRODUCTION_DOMAIN
```

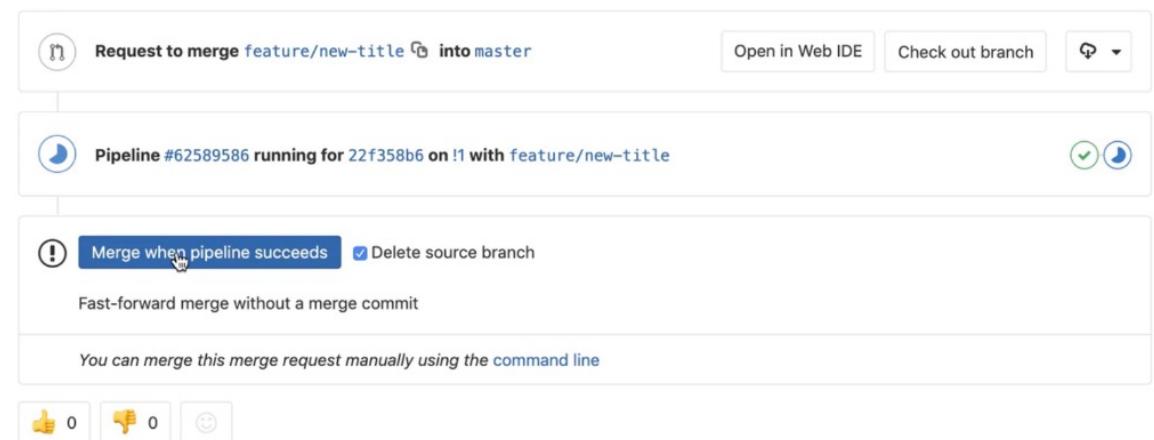




DEPLOYMENT (4/6): MERGE REQUEST

```
18
19 build website:
20   stage: build
21   only:
22     - master
23     - merge_requests
24   script:
25     - echo $CI_COMMIT_SHORT_SHA
26     - npm install
27     - npm install -g gatsby-cli
28     - gatsby build
29     - sed -i "s/%VERSION%/$CI_COMMIT_SHORT_SHA/" ./public/index.html
30   artifacts:
31     paths:
32       - ./public
33
```

Feature/new title



The screenshot shows a merge request interface on a GitLab-like platform. At the top, there's a header with the title 'Feature/new title'. Below it, a button says 'Request to merge feature/new-title into master'. To the right are buttons for 'Open in Web IDE', 'Check out branch', and a dropdown. The main area shows a pipeline status: 'Pipeline #62589586 running for 22f358b6 on !1 with feature/new-title' with two green checkmarks. Below that, there are merge options: 'Merge when pipeline succeeds' (with a checked checkbox) and 'Delete source branch'. A note says 'Fast-forward merge without a merge commit'. At the bottom, there are like/dislike buttons and a smiley face icon.



DEPLOIEMENT (5/6): ENVIRONNEMENT DYNAMIQUE

```
deploy review:  
  stage: deploy review  
  only:  
    - merge_requests  
  environment:  
    name: review/$CI_COMMIT_REF_NAME  
    url: https://instazone-$CI_ENVIRONMENT_SLUG.surge.sh  
  script:  
    - npm install --global surge  
    - surge --project ./public --domain instazone-$CI_ENVIRONMENT_SLUG.surge.sh
```

Request to merge feature/dynamic-environments into master

Pipeline #62596415 passed for d5d8f87e on !3 with feature/dynamic-env...

Deployed to review/feature/dynamic-... 6 minutes ago

[View app](#)



DEPLOIEMENT (6/6): CLEANUP

```
stop review:  
  stage: deploy review  
  only:  
    - merge_requests  
variables:  
  GIT_STRATEGY: none  
script:  
  - npm install --global surge  
  - surge teardown instazone-$CI_ENVIRONMENT_SLUG.surge.sh  
when: manual  
environment:  
  name: review/$CI_COMMIT_REF_NAME  
  action: stop
```

```
deploy review:  
  stage: deploy review  
  only:  
    - merge_requests  
  environment:  
    name: review/$CI_COMMIT_REF_NAME  
    url: http://instazone-$CI_ENVIRONMENT_SLUG.surge.sh  
    on_stop: stop review  
  script:  
    - npm install --global surge  
    - surge --project ./public --domain instazone-$CI_ENVIR  
  
stop review:  
  stage: deploy review  
variables:  
  GIT_STRATEGY: none
```



TP-10 : DEPLOIEMENT EN PROD ET STAGING

- Créez un compte sur heroku
- Créez un token pour l'api
- Créez une variable d'environnement nommée HEROKU_API_KEY avec le contenu de la clé créée
- Rajouter deux jobs qui permettront de déployer le conteneur en prod et un staging (deux environnements seront nécessaires avec 2 projets sur heroku)
- Le déploiement ne doit se faire que sur la branche master
- Vous pourrez vous inspirer cette documentation <https://devcenter.heroku.com/articles/container-registry-and-runtime>
- Pour avoir la ligne de commande vous pourrez utiliser l'utilitaire npm pour l'installer directement dans votre pipeline



TP-11: ENVIRONNEMENT DYNAMIQUE

- Créez un job qui permettra de réaliser une review uniquement lors de merge request
- Bien évidemment ce job sera dans un environnement dédié au review à part entière
- Créez un job dont le but sera de supprimer l'environnement de review après la validation de la Merge Request (dont de la suppression de la branche qui était en review)
- Vérifiez que tout cela marche, si c'est le cas, vous êtes prêt pour la suite ☺

PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-Cl

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Déploiement via SSH

Monitoring

Mini-Projet

TEST DE VALIDATION (1/4): BEFORE_SCRIPT ET AFTER_SCRIPT

```
default:  
  before_script:  
    - global before script  
  
job:  
  before_script:  
    - execute this instead of global before script  
  script:  
    - my command  
  after_script:  
    - execute this after my script
```



TEST DE VALIDATION (2/4): JOB TEMPLATE

```
.deploy_template: &deploy
only:
  - master
script:
  - npm install --global surge
  - surge --project ./public --domain $DOMAIN
environment:
  url: http://$DOMAIN
```

```
deploy_staging:
<<: *deploy
stage: deploy_staging
variables:
  DOMAIN: $STAGING_DOMAIN
environment:
  name: staging
```

TEST DE VALIDATION (3/4): DÉSACTIVER DES JOBS

```
.build · website:  
  · stage: build  
  · script:  
    · - npm · install  
    · - npm · install · -g · gatsby · cli  
    · - gatsby · build  
  · artifacts:  
    · paths:  
      · - ./public
```



TEST DE VALIDATION (4/4): ANCHORS

```
.job_template: &job_definition
  image: ruby:2.6
  services:
    - postgres
    - redis

test1:
  <<: *job_definition
  script:
    - test1 project

test2:
  <<: *job_definition
  script:
    - test2 project
```

```
.job_template:
  image: ruby:2.6
  services:
    - postgres
    - redis

test1:
  image: ruby:2.6
  services:
    - postgres
    - redis
  script:
    - test1 project

test2:
  image: ruby:2.6
  services:
    - postgres
    - redis
  script:
    - test2 project
```



TP-12: TEST DE VALIDATION

- Créez deux jobs qui vont tester le fonctionnement du site internet en staging et en prod à l'aide de curl et uniquement sur la branche master
- Créez un job template permettant de réaliser le curl en fonction de la valeur d'une variable \$DOMAIN qui vous définirez
- Insérez ensuite le template dans la définition des jobs réalisés précédemment

PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-Cl

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Déploiement via SSH

Monitoring (Prometheus)

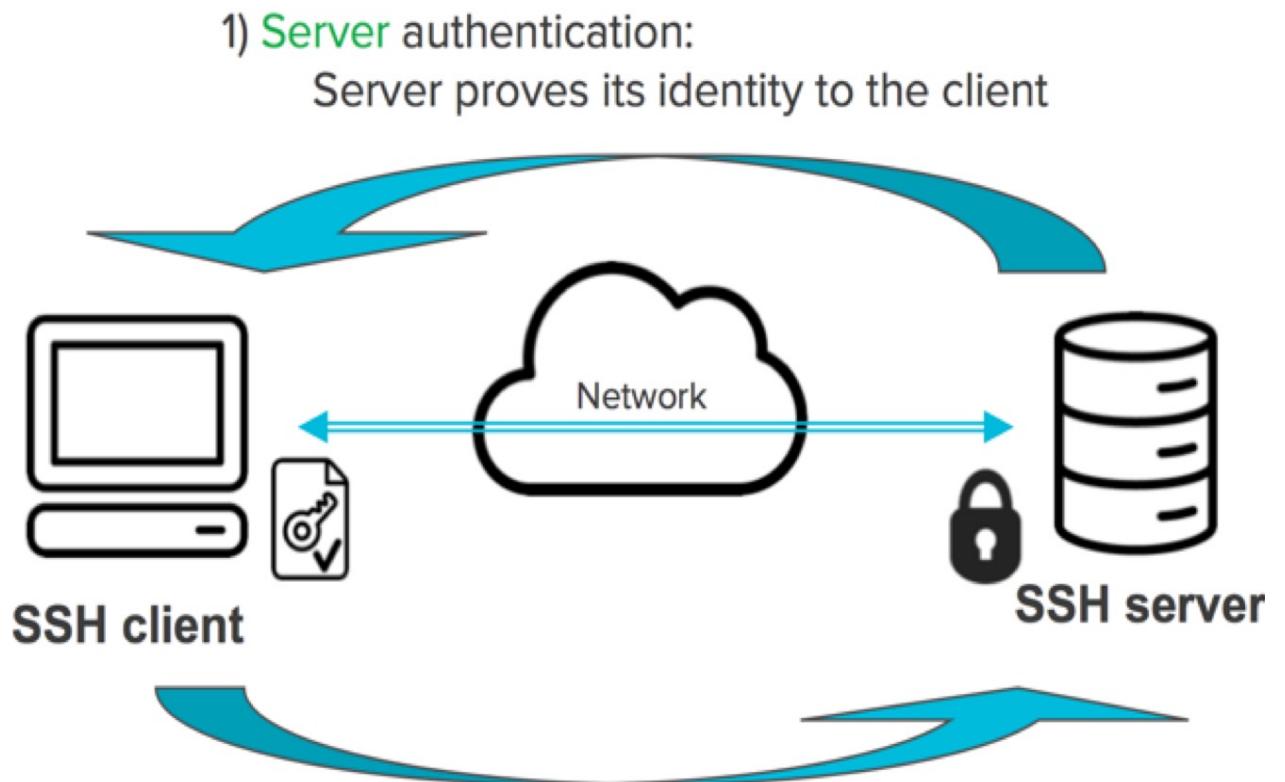
Mini-Projet



GITLAB-CI : DEPLOIEMENT VIA SSH

PIPELINE - DEPLOY SSH

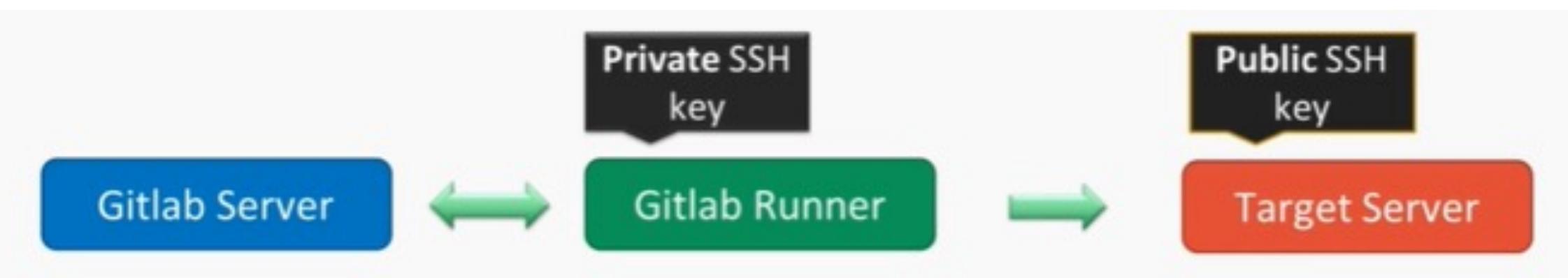
- ssh-keygen
- authorized_keys
- -o StrictHostKeyChecking=no



2) **User authentication:**
Client proves user's identity to the server



GITLAB-CI : DEPLOIEMENT VIA SSH



Héberge :

1. Le code applicatif
2. Le fichier .gitlab-ci.yml
3. Les variables

1. Exécute les actions de build
2. Génère les artefacts (images, fichier war ou autres)
3. Copie les artefacts vers le serveurs
4. Générer la clé SSH privée et publique
5. Copie les fichiers applicatifs vers le serveur de production

1. Installer les dépendances nécessaires
2. L'utilisateur doit avoir la clé publique qui va bien
3. Lancer les commandes pour déployer l'application



GITLAB-CI : DEPLOIEMENT VIA SSH

EXAMPLE

```
deploy:
  image: alpine
  stage: deploy
  before_script:
    # Get ssh keys from target server and add them to known_hosts
    - apk update
    - 'which ssh-agent || ( apk add openssh-client )'
    - eval $(ssh-agent -s)
    - mkdir -p ~/.ssh
    - chmod 700 ~/.ssh
    - ssh-keyscan -p 22 -4 $PRODUCTION_HOST >> ~/.ssh/known_hosts
    - chmod 644 ~/.ssh/known_hosts
    # Add the private key defined in PRIVATE_KEY variable
    - echo "$PRIVATE_KEY" | tr -d '\r' | ssh-add - > /dev/null
  script:
    - echo "Deploying the system"
    - ssh root@$PRODUCTION_HOST -p 22 -4 "mkdir -p /opt/tomcat/webapps"
    - scp build/libs/deploy-*.war $PRODUCTION_HOST:/opt/tomcat/webapps/deploy.war
    - scp docker/docker-compose-prod.yml $PRODUCTION_HOST:/opt/tomcat/
    - ssh root@$PRODUCTION_HOST -p 22 -4 "docker-compose -f /opt/tomcat/docker-compose-prod.yml up -d --force-recreate"
  environment: production
  only:
    - master
```



TP-13: DEPLOIEMENT VIA SSH

- Un nouvel environnement de production a été mis sur pied. Il s'agit en fait d'une instance EC2 de Ubuntu hébergée dans un cloud AWS
- Ajoutez un job à votre projet actuel un job permettant de déployer votre application dans cet autre environnement de production.

NB: Les paramètres liés à cet nouvel environnement vous seront fournis.

PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-CI

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Déploiement via SSH

Git Hooks et Time tracking

Monitoring (Prometheus)

Déploiement vers K8s (Gitops)

Mini-Projet



GIT HOOKS : CONCEPTS

Type de hooks supportés par Gitlab

- `pre-receive`
- `post-receive`
- `update`

Sur Gitlab, on peut créer des hooks à portée :

- Globale (Global server hooks) : scripts devant s'exécuter pour tous les repo de notre environnement
- Liée à un projet (Project hooks) : portée limité au dit projet

Besoins :

- Serveur Gitlab On Premise (local)
- Accès au serveurs avec les droits administrateurs
- Accéder au répertoire du projet sur Gitlab et créer son Hook et le rendre exécutable

global server hook for all repositories

The default directory:

- For an installation from source is usually `/home/git/gitlab-shell/hooks`.
- For Omnibus GitLab installs is usually `/opt/gitlab/embedded/service/gitlab-shell/hooks`.

`pre-receive.d`, `post-receive.d` `update.d`

Create a server hook for a repository

- For an installation from source, the path is usually `/home/git/repositories/<group>/<project>.git`.
- For Omnibus GitLab installs, the path is usually `/var/opt/gitlab/git-data/repositories/<group>/<project>.git`.

`custom_hooks`, `pre-receive` `post-receive`

NB : les scripts ou hooks peuvent être écrits en plusieurs langages, il faut juste disposer des dépendances nécessaires sur l'hôte permettant d'exécuter ou de comprendre ledit langage.

Veiller également à préciser à Git le langage utilisé à l'aide du shabang (`#!/...`) en début de votre script



GIT HOOKS : EXEMPLE DE MISE EN PLACE

Enoncé: Mise en place d'un project Hook à l'aide de python qui oblige les contributeurs ou collaborateur sur ledit projet de respecter une certaine nomenclature lors de la création de nouvelle branche.

NB: Le but ici étant d'utiliser une nomenclature liée aux issues <numéro-issue>-<nom-de-la-branche>

Exemple : pour l'issue #1, la branche à créer qui sera lié à cette issue devra s'appeler 1-feature1

```
base commit ref\n,
```

```
6f359ed2c711672dfe7bd0a7799177e5a3b57694
```

```
01724c7c0277b951b201606921e200f1f4397c0d refs/heads/my-branch-name
```

```
#!/usr/bin/env python
# coding: utf-8

import sys
import re

data = sys.stdin.read()
data = data.strip()
base, commit, ref = data.split(" ")

branch_name = ref.split("/")[-1]

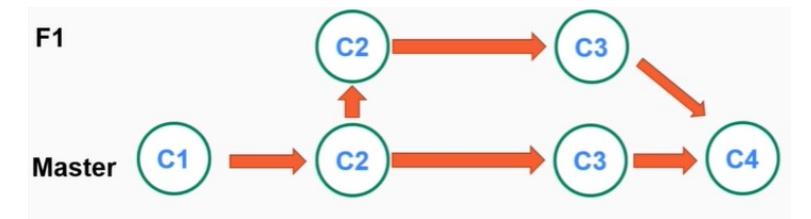
if not re.match(r"^(master|develop|hotfixes|[0-9]+-[a-z0-9-]+)$", branch_name):
    print("#" * 50)
    print("Invalid branch name %s" % branch_name)
    print("#" * 50)
    sys.exit(1)

cd /var/opt/gitlab/git-data/repositories/<group>/<repository-name>.git/
mkdir custom_hooks

chmod +x custom_hooks/pre-receive
```



TIME TRACKING & NOTIFICATIONS



⌚ error of one service in prod env

#1 · created 1 hour ago by sadofrazer

bug

Commit message

Update Readme.md
#1

Target Branch

feature1

⌚ Set by 🏹 sadofrazer to be merged automatically when the pipeline succeeds

Cancel auto-merge

Deletes the source branch

Closes issue #1

Assign yourself to this issue

Test-project > first-project > Value Stream Analytics

Value Stream Analytics

Issue -

Plan -

Code 4m

Test 4m

Review 4m

Staging 4m



Filter results

New Issue

1.0

Commits

28.0

Deploys

6.0

Deployment Frequency

0.2 per day

PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-Cl

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Déploiement via SSH

Monitoring (Prometheus)

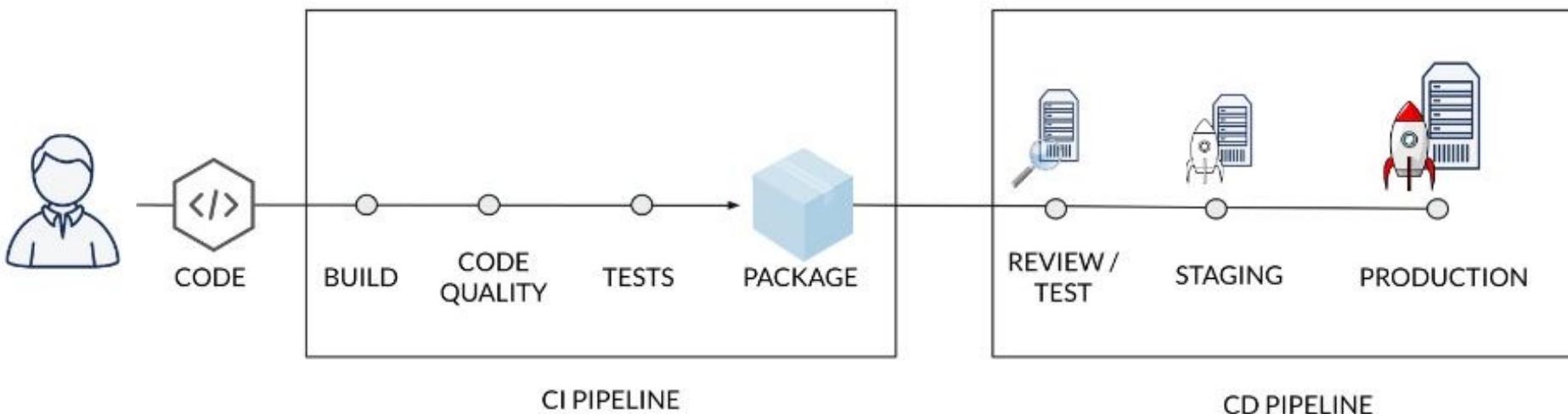
Mini-Projet



MINI-PROJET: STATIC WEBSITE

- <https://github.com/bennzhang/docker-demo-with-simple-python-app>

CI/CD Pipeline



NB:

- REVIEW/TEST : HEROKU
- STAGING : HEROKU
- PRODUCTION : EC2 INSTANCE (AWS), DÉPLOIEMENT MANUEL SI STAGING OK

PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-CI

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Déploiement via SSH

Git Hooks et Time tracking

Monitoring (Prometheus)

Déploiement vers K8s (Gitops)

Mini-Projet



DEPLOIEMENT VERS K8S: GITOPS

<https://eazytraining.fr/pipeline-ci-cd-gitops-avec-kubernetes-flux-gitlab-ce-et-aws/>

PLAN

Présentation du formateur

Prérequis

GIT

Gitlab

Introduction au DevOps et au CI/CD

Gitlab-CI

Introduction au YAML

Build

Test d'acceptance

Release

Deploiement

Test de validation

Déploiement via SSH

Git Hooks et Time tracking

Monitoring (Prometheus)

Déploiement vers K8s (Gitops)

Mini-Projet

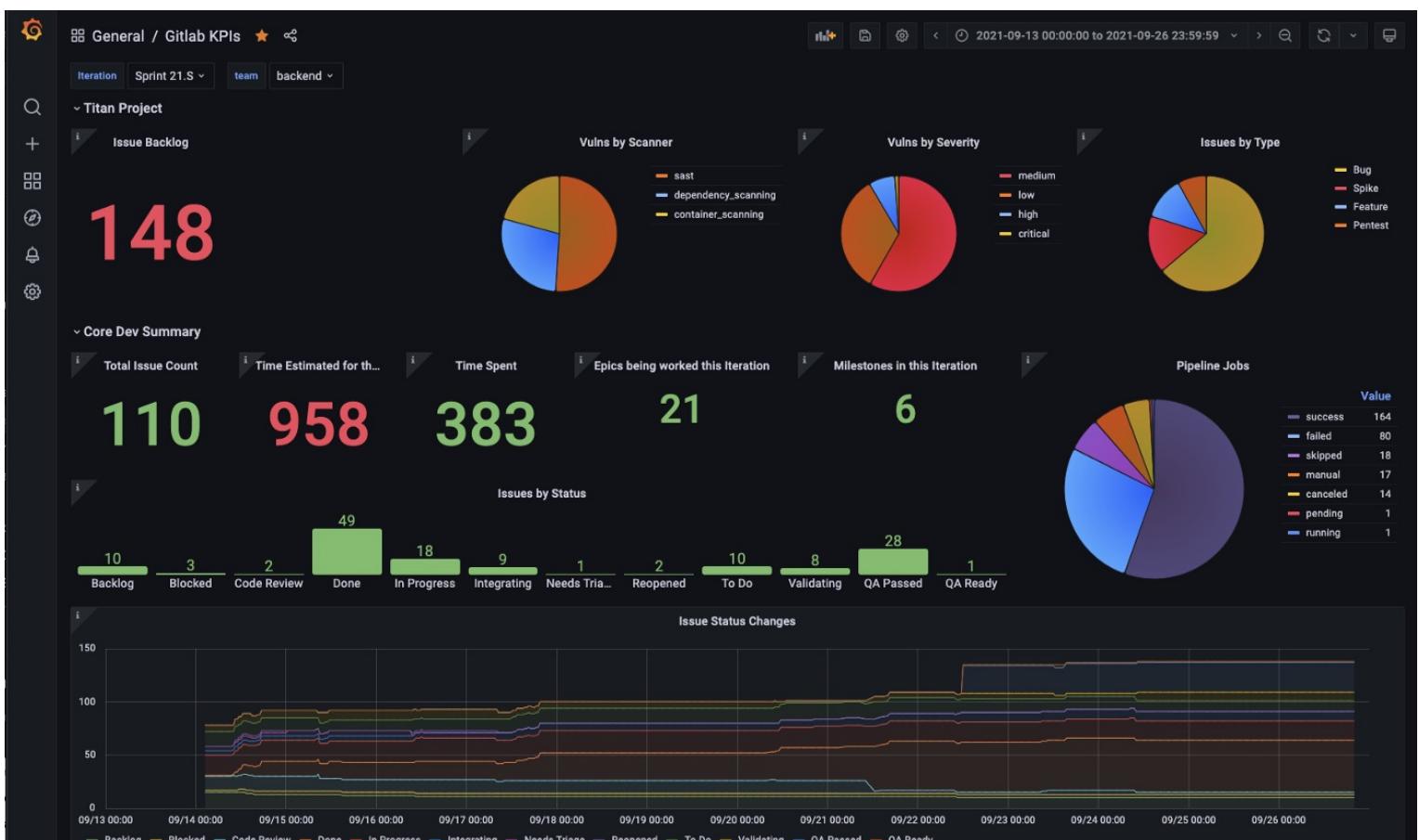


GITLAB-CI : MONITORING VIA PROMETHEUS

<https://medium.com/geekculture/monitoring-websites-using-grafana-and-prometheus-69ccf936310c>



GitLab



MERCI POUR VOTRE ATTENTION

