

DevOps CI/CD sous GitLab



Brahim HAMDI
brahim.hamdi.consult@gmail.com



Janvier 2024

Plan de la formation

- ─ Gestion des versions avec Git
- ─ Git et son utilisation avec GitLab
- ─ Mise en œuvre et déploiement des conteneurs Docker
- ─ Kubernetes, mise en œuvre
- ─ Prometheus monitoring
- ─ Gitlab CI/CD - Fonctionnement
- ─ GitLab CI/CD - le fichier yaml
- ─ Les runners
- ─ Intégration Cypress dans GitLab
- ─ Plus loin avec Gitlab

Module1 : Gestion des versions avec Git

- Git et les solutions de gestion de versions
- Paramètres et niveaux de configuration de Git
- Utilisation de Git
- Retour en arrière
- Gestion des branches et résolution de conflits
- Utilisation avancée de Git
- Les plateformes Git
- Le travail collaboratif et les stratégies de branching

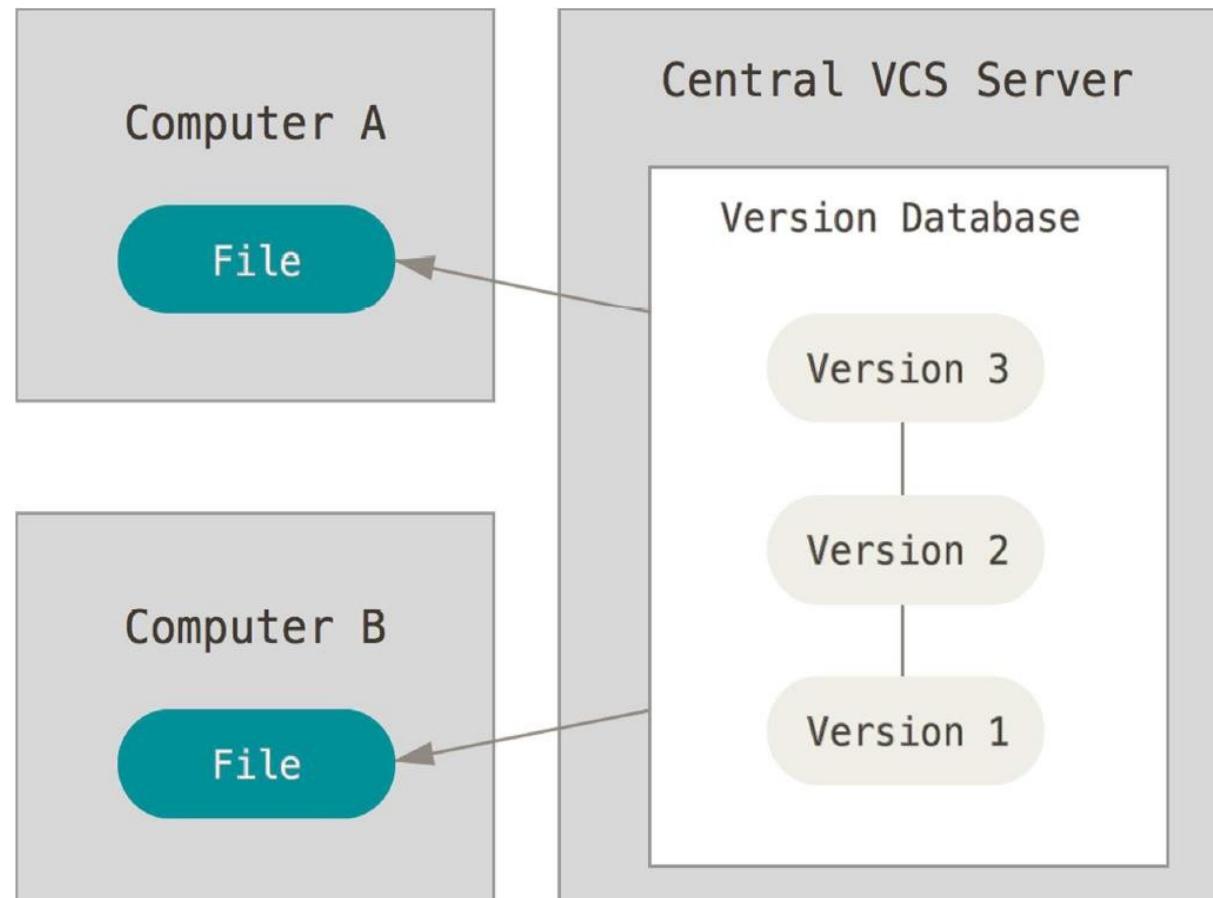
Git et les solutions de gestion de versions

Qu'est ce qu'un gestionnaire de version ?

- Un gestionnaire de version :
 - Enregistre les évolutions d'un fichier
 - Permet de revenir à une version précédente
 - Fonctionne avec tout type de fichier (.txt, .php, .java, .jpg, .exe, etc ...)
 - Permet de retrouver et de récupérer un fichier supprimé
 - Permet le travail collaboratif sur un projet
 - Déetecte les conflits ...

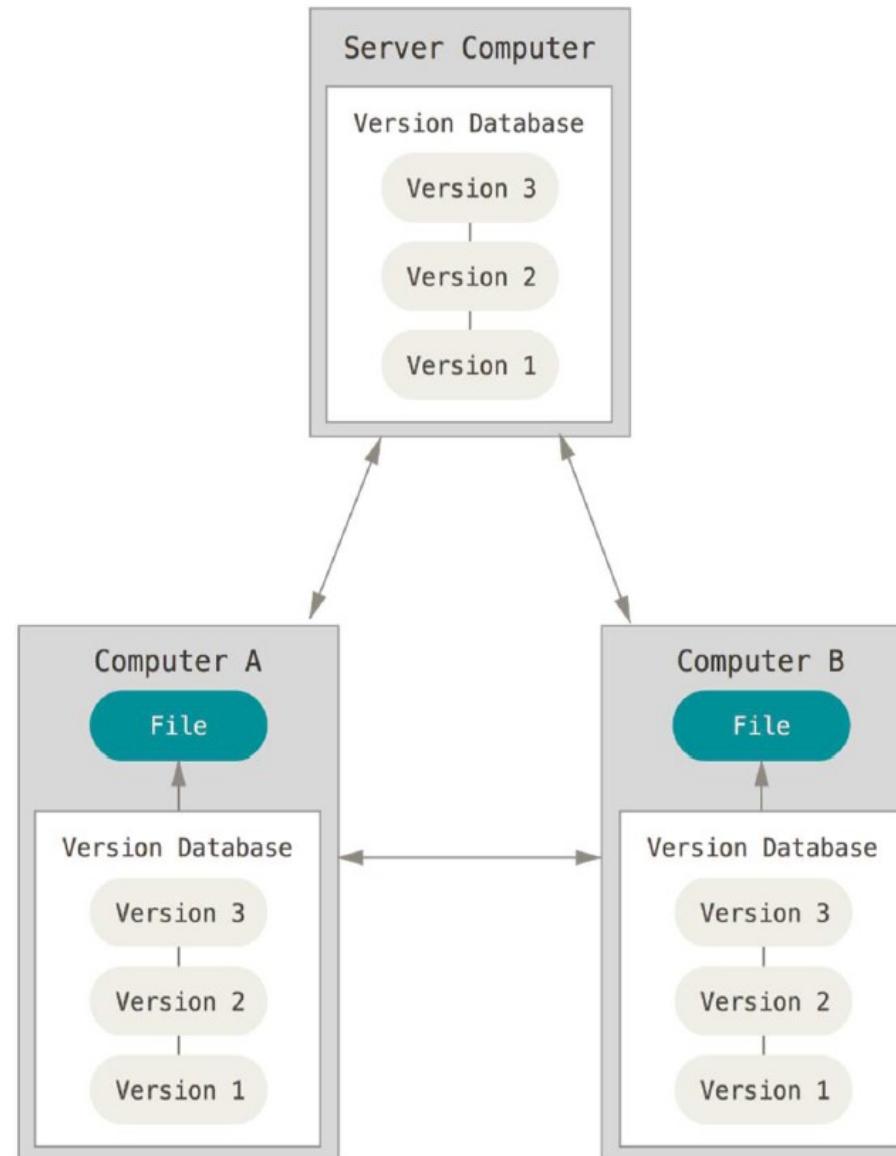
Les systèmes de gestion de version centralisés

- Exemples: CVS, SVN



Les systèmes de gestion de version distribués

■ Exemples: Git, Mercurial



- Logiciel de gestion de versions décentralisé.
- Logiciel libre et gratuit, créé en 2005
- Distribué sous licence GPLv2.
- Depuis les années 2010, il s'agit du logiciel de gestion de versions le plus populaire dans le développement logiciel
- Utilisé par des dizaines de millions de personnes, sur tous les environnements (Windows, Mac, Linux).
- C'est le système à la base du célèbre site web GitHub, le plus important hébergeur de code informatique.

Presque toutes les opérations sont locales

- La plupart des opérations de Git se font en local
- Même pour chercher dans l'historique du projet.
- L'intégralité de la base de données est en locale.

Paramètres et niveaux de configuration de Git

- Git contient un outils ‘git config’ pour modifier les paramètres de configuration
- Git a 3 niveaux de configuration
 - Global au système : /etc/gitconfig, modifiable avec ‘git config --system’
 - Global à l’utilisateur : ~/.gitconfig, modifiable avec ‘git config --global’
 - Global au projet : .git/config, modifiable avec ‘git config -local’
- Chaque niveau surcharge le précédent

- La première chose à faire après l'installation de Git est de renseigner votre nom et votre adresse email
 - \$ git config --global user.name "Brahim HAMDI"
 - \$ git config --global user.email "brahim.hamdi.consult@gmail.com"
- Vu que nous configurons ces options avec --global, nous n'aurons pas besoin de les reconfigurer à chaque projet.

- Nous faisons la même chose avec notre éditeur de texte.
 - \$ git config --global core.editor nano

- Il est possible de vérifier nos paramètres :
 - \$ git config --list
 - \$ git config --list --global

Utilisation de Git

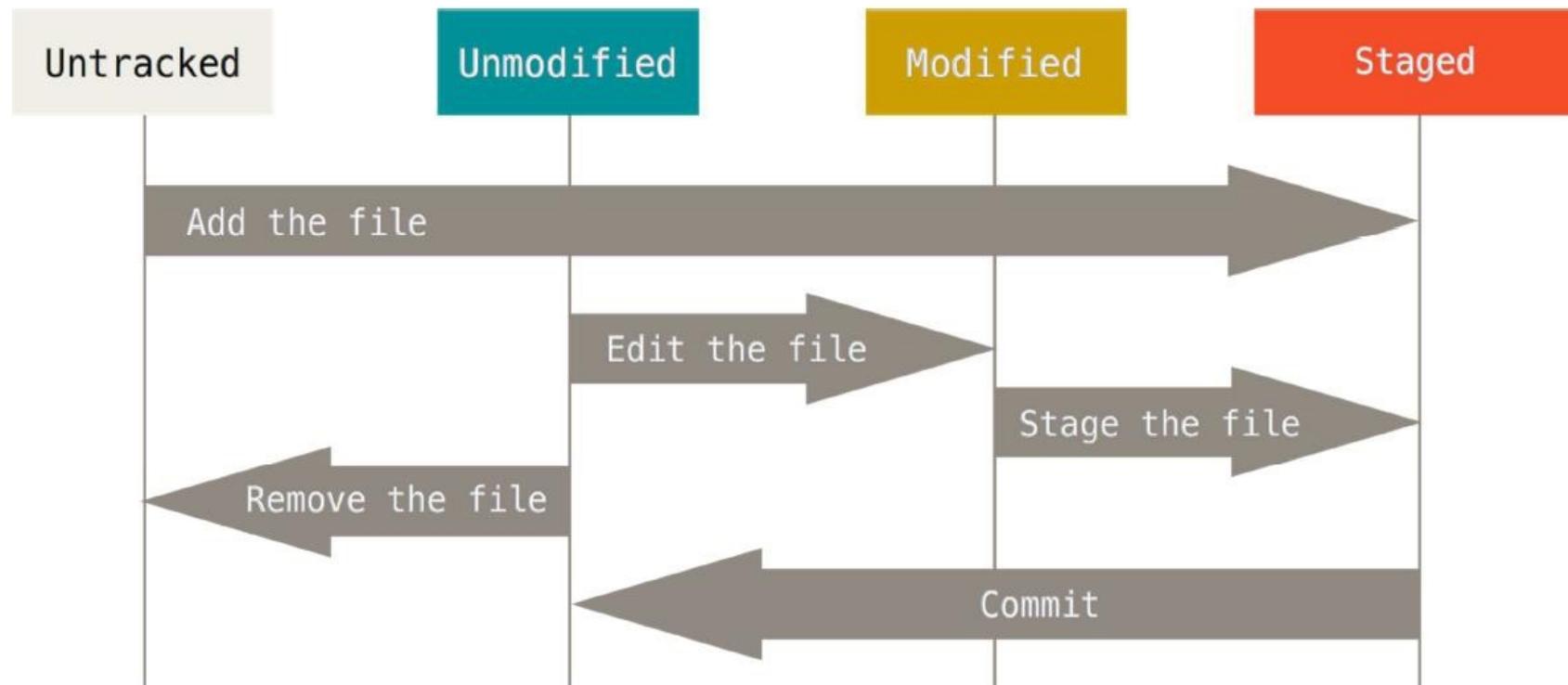
- Git traite des fichiers, des répertoires et leurs évolutions lors de la vie d'un projet.
- Sa conception est calquée sur ce principe :
 - Les fichiers sont représentés par des fichiers *blobs*
 - Les répertoires par des fichiers *trees*
 - Un instantané (*snapshot*)/une version du projet par un fichier *commit*.
- Tous les objets Git sont identifiés par SHA1

- Dans Git, tout est vérifié par une somme de contrôle avant d'être stocké et par la suite cette somme de contrôle, signature unique, sert de référence.
- Impossible de perdre un fichier dans Git sans que Git s'en aperçoive.
- Les sommes de contrôle sont en SHA-1.
 - 24b9da6552252987aa493b52f8696cd6d3b00373

- Dans votre projet, initialiser Git:
 - \$ git init
- Un dossier .git apparaît, l'initialisation est faite, mais aucun fichier n'est versionné !

- La commande pour récupérer un projet existant est '*git clone*'.
- Git reçoit une copie de quasiment toutes les données stockées dans le serveur.
- S'il y a un problème avec le disque du serveur, vous aurez une copie complète du projet (fichier et historique).
- Git gère le protocole `http://` ou `https://`
- Exemple :
 - `$ git clone https://github.com/brahimhamdi/projet1`

Le fonctionnement



- Pour connaître l'état des fichiers de votre projet :
 - \$ git status

```
$ git status
Sur la branche master
Votre branche est à jour avec 'origin/master'.
rien à valider, la copie de travail est propre
```

- Si on ajoute un nouveau fichier dans le projet, git status nous donne son nouvel état.
- Pour avoir un affichage plus concis :
 - \$ git status -s

Placer de nouveaux fichiers sous suivi de version

- Pour commencer à suivre un nouveau fichier, il faut utiliser la commande :
 - \$ git add <fichier>
- En faisant un ‘git status’, nous voyons que le fichier a changé d’état, le fichier est indexé dans Git.
- La version du fichier à l’instant où vous l’enregister avec ‘git add’ sera celle qui sera dans l’historique des instantanés
- On peut ajouter des répertoires, git ajoutera à l’index tout le répertoire récursivement.

- Maintenant, modifions un fichier qui est déjà sous suivi de version.
- Puis observons le status avec ‘git status’.
- Le fichier modifié a le status “Modifications qui ne seront pas validées”, le fichier n'est pas indexé.
- Pour indexer le fichier :
 - \$ git add <fichier>

- Dans les projets, nous avons souvent des fichiers que nous ne voulons pas ajouter à Git.
- Nous avons la possibilité de ne jamais les indexer avec 'git add'.
- Sinon, nous pouvons indiquer à Git de les ignorer.
- Il faut rentrer la liste des fichiers dans le fichier .gitignore
- Les fichiers n'apparaîtront plus dans 'git status'.
 - \$ cat .gitignore
 - *.[oa]
 - *~

Inspecter les modifications indexées et non indexées

- Pour avoir une vision plus précise des changements faits sur les fichiers, on utilise ‘git diff’.
- Cette commande répond aux questions :
 - Qu'est-ce qui a été modifié, mais pas encore indexé ?
 - Quelle modification a été indexée et est prête pour la validation ?
- git diff montre les changements ligne par ligne
- Pour voir les modifications qui font partie de la prochaine validation, vous pouvez utiliser ‘git diff --cached’

- Une fois que votre index est dans l'état désiré, nous pouvons valider les modifications.
- Rappel, ce qui n'est pas passé dans 'git add' ne fera pas partie de la prochaine validation. Ils resteront en modifié sur votre disque.
- La commande pour valider est 'git commit'
- Cette action lance votre éditeur de texte par défaut.
- Les options :
 - -v : ajoute le diff dans le commentaire
 - -m 'my comment' : Pour ajouter directement un commentaire sans passer par l'éditeur de texte.
- Une fois votre validation faite, votre commit a un id (SHA-1)

- Pour effacer un fichier de Git, vous devez l'éliminer des fichiers en suivi de version (plus précisément, l'effacer dans la zone d'index) puis valider.
- Si vous effacez le fichier avec ‘rm <fichier>’, il apparaît sous la section « Modifications qui ne seront pas validées » (c'est-à-dire, non indexé).
- Avec un ‘git rm <fichier>’, l'effacement du fichier est indexé.

- À la différence des autres VCS, Git ne suit pas explicitement les mouvements des fichiers, même s'il sera capable de voir qu'un fichier a été déplacé.
- Il existe la commande ‘git mv <source> <destination>’ qui revient exactement à :
 - git mv <source> <destination>

- À l'instar de la plupart des VCS, Git donne la possibilité d'étiqueter un certain état dans l'historique comme important.
- Généralement, on utilise cette fonctionnalité pour marquer les versions d'un projet (v1.0.0, v1.0.42, v2.0.1, ...)
- Nous verrons comment :
 - Lister les différentes étiquettes
 - Comment créer de nouvelles étiquettes

- Pour lister les étiquettes :
 - \$> git tag
- Les étiquettes sont listées par ordre alphabétique
- Pour faire une recherche :
 - \$> git tag -l 'v1.0.*'

- Créer des étiquettes est simple avec Git :
 - \$> git tag -a v1.1.0 -m 'Ma version 1.1.0'
- Puis pour afficher en détails votre nouveau tag :
 - \$>git show v1.1.0

- Il est possible d'étiqueter des anciens commit :
 - \$ git tag -a v1.0.0 9b92f3b

Retour en arrière

- Pour voir l'historique, il existe la commande ‘git log’
- Par défaut, git log invoqué sans argument énumère en ordre chronologique inversé les commits réalisés.

- git log dispose de plusieurs options permettant de paramétrier exactement ce que l'on cherche à voir :
 - `--oneline` : Afficher un commit par ligne
 - `--graph` : Affiche en caractères le graphe de branches et fusions en vis-à-vis de l'historique

Désindexer un fichier déjà indexé

- Git version 2.25.0 a introduit une nouvelle commande : ***git restore***.
 - Une alternative à *git reset* et *git checkout* pour quelques actions
- Pour désindexer :
 - \$> git restore --staged <file>
 - ou
 - \$> git reset HEAD <file>

Réinitialiser un fichier modifié

- Pour faire revenir un fichier au niveau du précédent checkout :
 - \$> git checkout -- <fichier>
 - ou
 - \$> git restore --source <fichier>
- Attention, vous perdrez définitivement les modifications apportées au fichier depuis le dernier checkout.

Réinitialiser le HEAD actuel à l'état spécifié

- Le HEAD point par défaut sur le dernier commit
- Git reset Amène le HEAD de la branche actuelle à un commit spécifique
 - \$> git reset <commit>
- Attention, vous risquerez de perdre des commits

Annuler les changements dans l'historique des commits

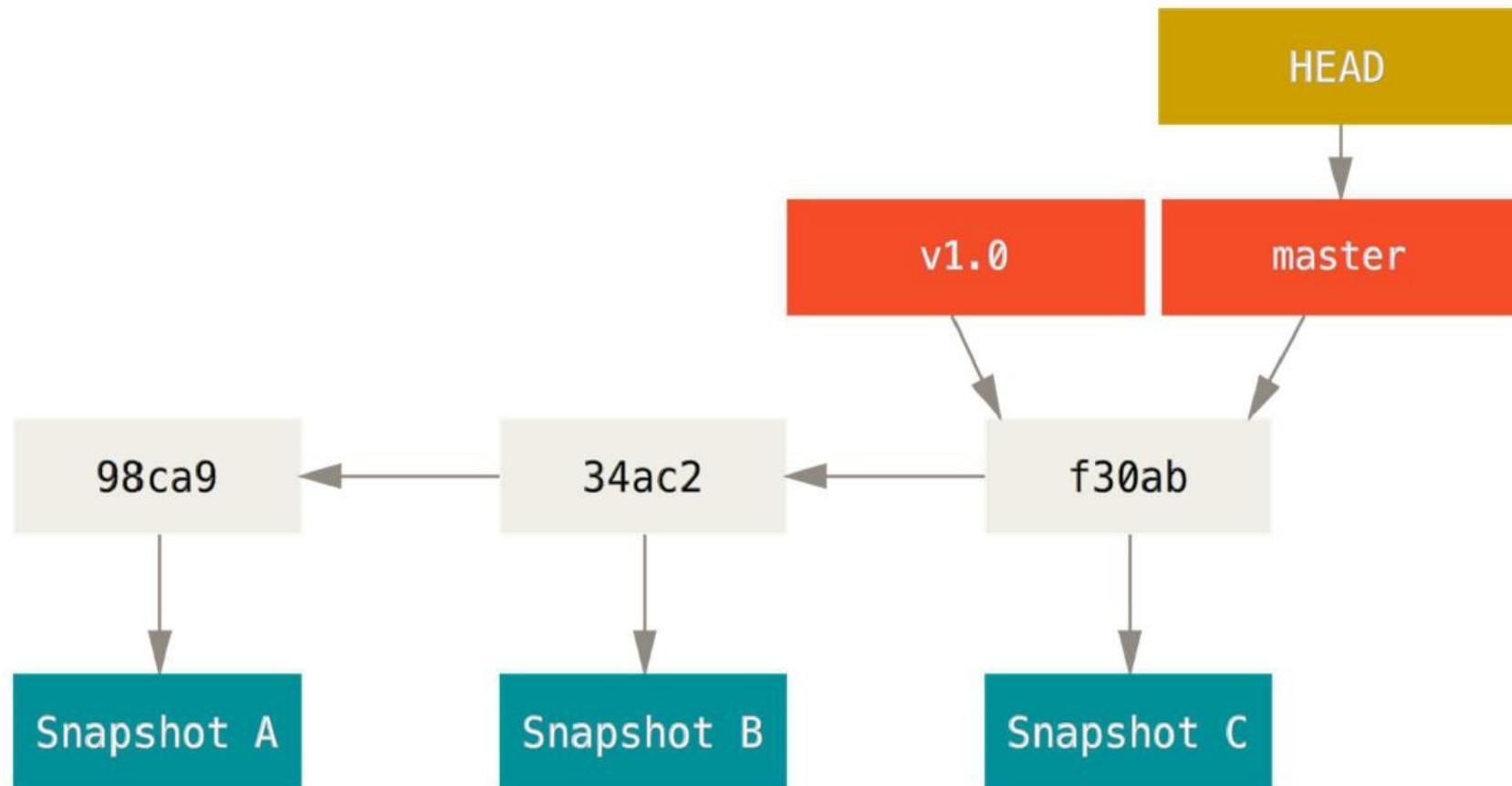
- git revert peut être considérée comme une commande de type « undo »
 - Il crée un nouveau commit qui annule un ancien
 - Mais ne supprime pas le commit annulée de l'historique du projet
- Utilisée lorsque vous souhaitez appliquer l'inverse d'un commit à partir de votre historique de projet.
- - \$> git revert <commit>

Gestion des branches et résolution des conflits

- Presque tous les VCS proposent une certaine forme de gestion de branches.
- Créer une branche signifie diverger de la ligne principale de développement et continuer à travailler sans impacter cette ligne.
- Permet le développement de différentes versions/fonctionnalités en parallèle

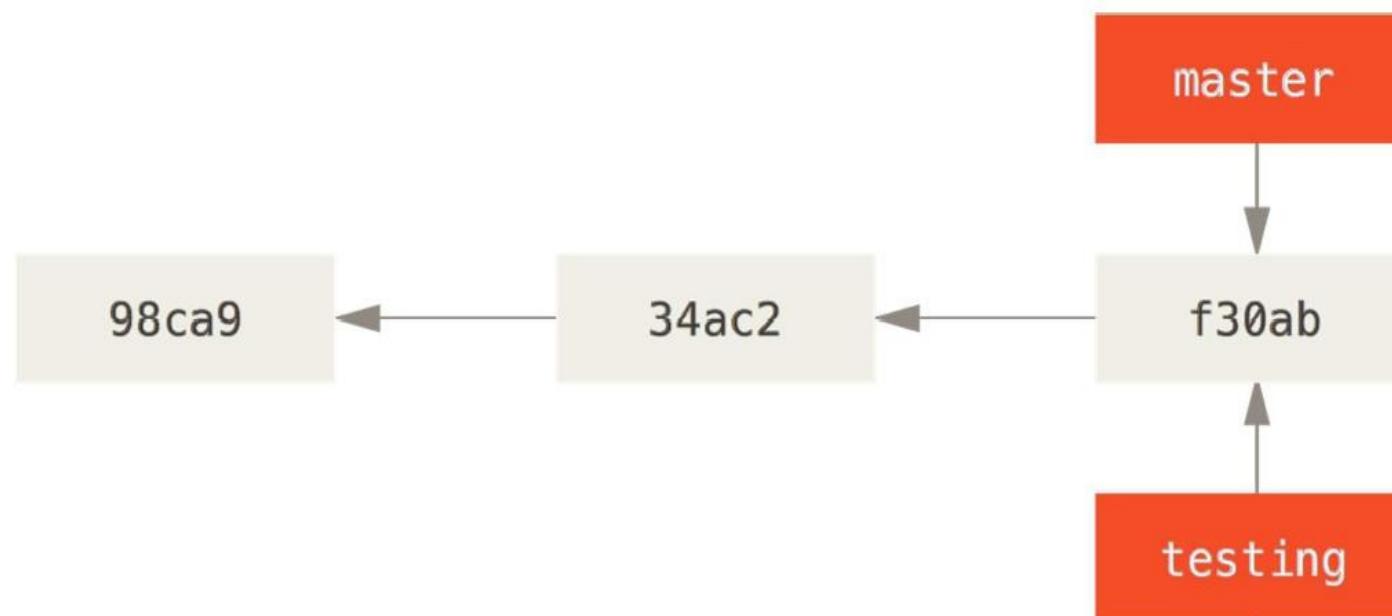
- La branche par défaut dans Git s'appelle *master*
- Cette branche master va se déplacer automatiquement à chaque nouveau commit pour pointer sur le dernier commit effectué tant qu'on reste sur cette branche.
- Elle n'est pas une branche spéciale pour Git : elle est traitée de la même façon que les autres branches.
- L'idée est que lorsqu'on tape une commande git init, une branche est automatiquement créée et que le nom donné à cette branche par Git par défaut est "master".
 - On pourrait très bien la renommer ensuite mais ça ne présente généralement aucun intérêt.

Créer une nouvelle branche



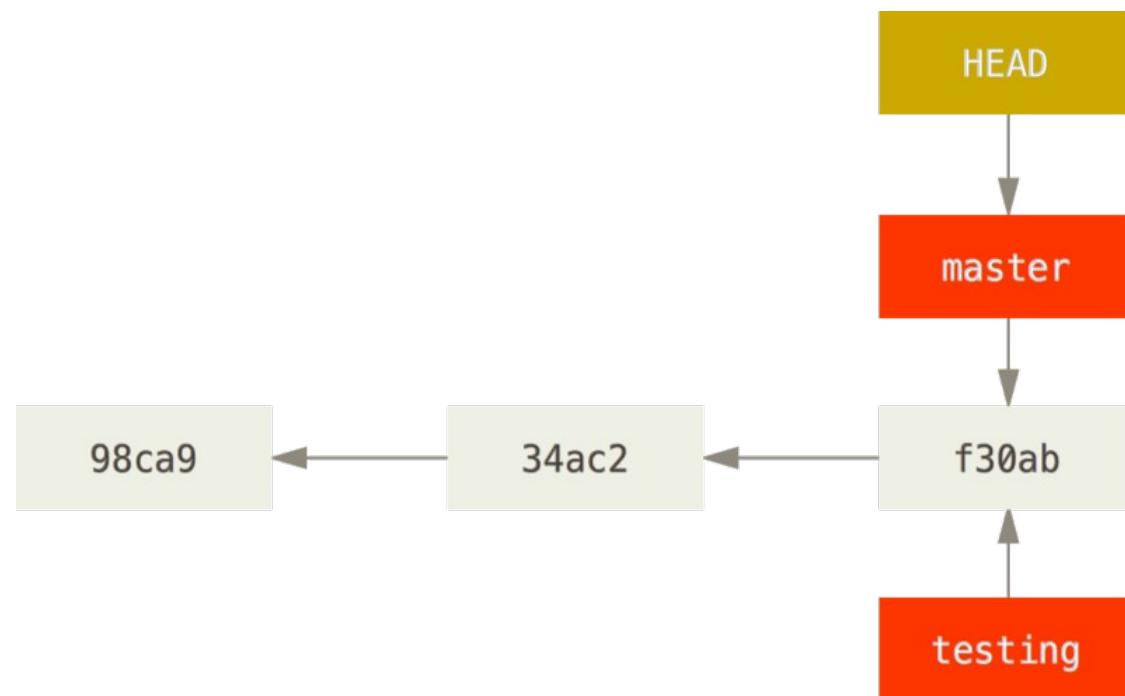
Créer une nouvelle branche

- Pour créer une nouvelle branche ‘testing’ par exemple :
 - \$> git branch testing



Créer une nouvelle branche

- Pour savoir sur quelle branche git se trouve actuellement, il ajoute un pointeur spécial.



- Pour vérifier cela :
 - \$> git log --oneline --decorate

Basculer entre les branches

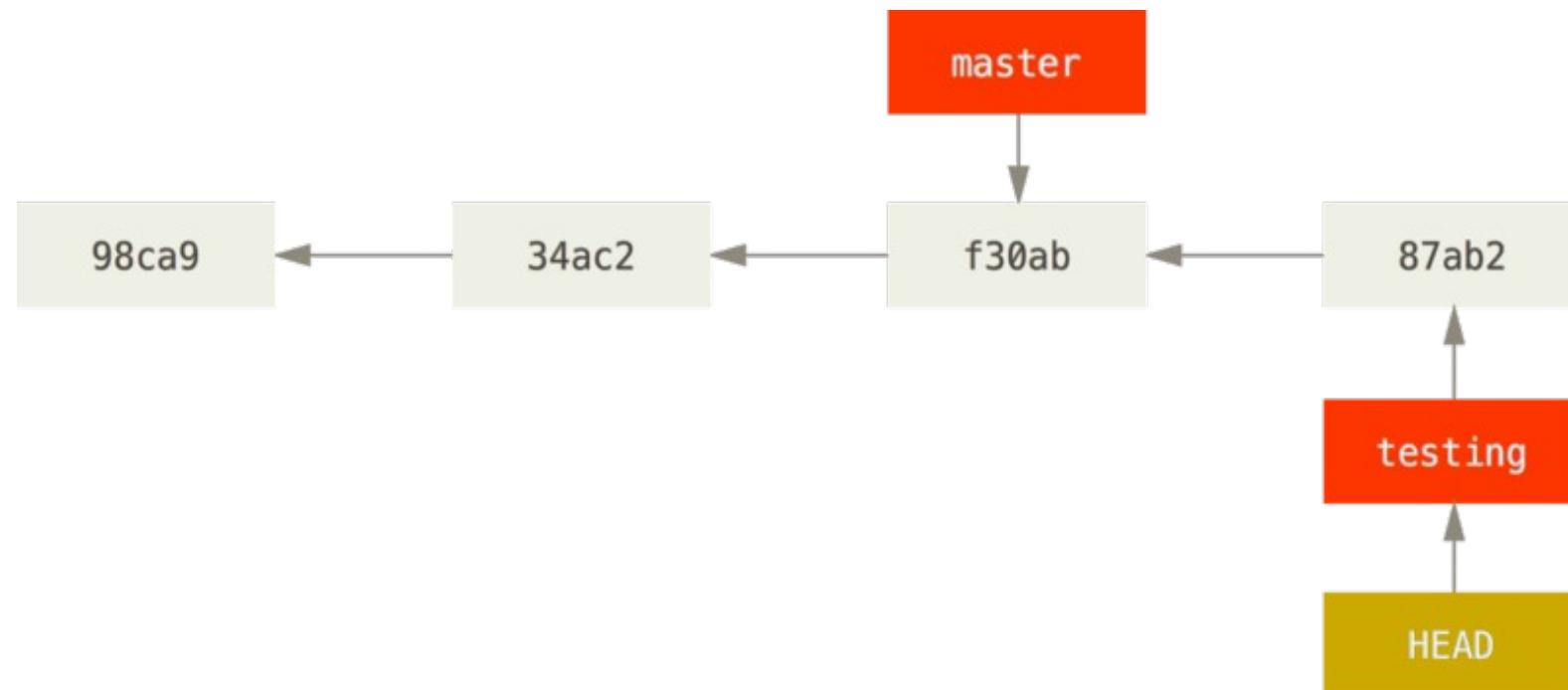
- \$> git checkout testing
- Ou
- \$> git switch testing



- Cette commande a provoqué deux changements :
 - Mettre le pointeur HEAD sur la branche testing
 - Elle a remplacé les fichiers de votre répertoire dans l'état du snapshot pointé par testing

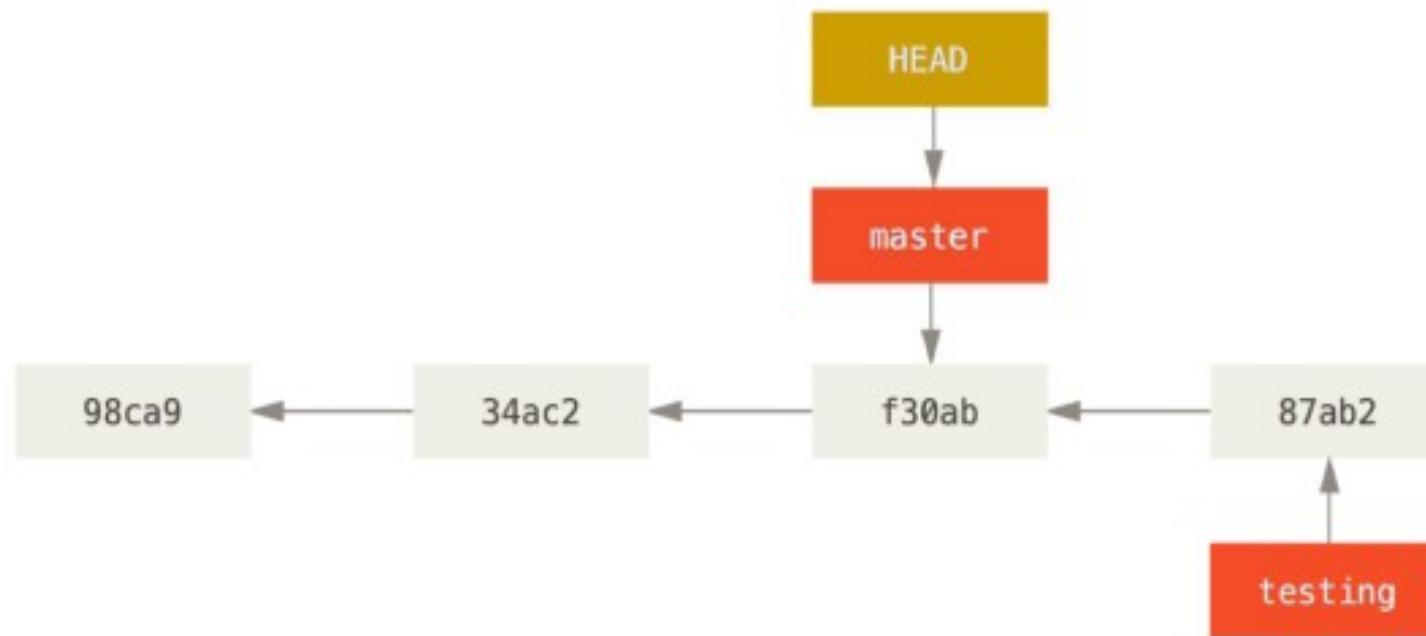
Basculer entre les branches

- On édite un fichier et on crée un commit :
 - \$> git commit -a -m 'nouvelle modification'



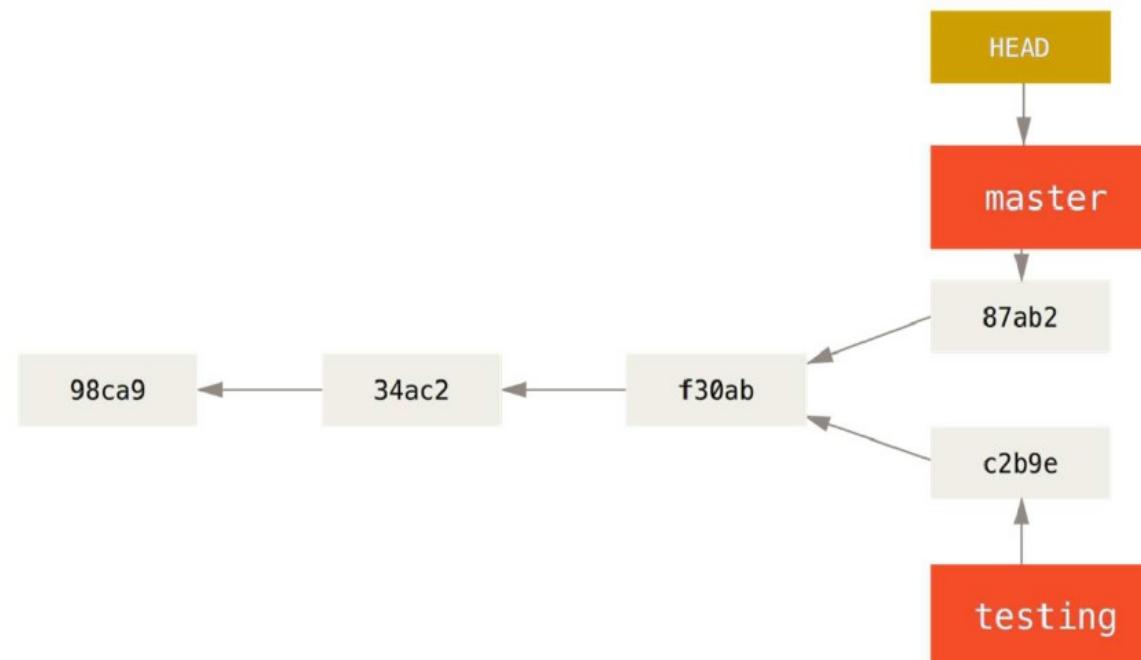
- Et si on revient sur master ?

- \$> git checkout master
- Ou
- \$> git switch master



- Cette commande a provoqué deux changements :
 - Mettre le pointeur HEAD sur master
 - Elle a remplacé les fichiers de votre répertoire dans l'état du snapshot pointé par master

- On continue nos tests.
- Faisons encore une modification dans un fichier et créons un nouveau commit.
- À quoi ressemble notre arbre ?



- On peut retrouver cet arbre avec :
 - \$> git log --oneline --decorate --graph -all
 - \$> gitk --all

- Maintenant que vous avez créé, fusionné et supprimé des branches, regardons de plus près les outils.
- ‘git branch’, sans argument, permet de lister les branches en local.
 - \$> git branch
- Pour avoir un peu plus d’informations
 - \$> git branch -v

- Maintenant, nous fusionnons la branche *testing* avec la branche *master* (nous sommes sur *master* maintenant)
 - \$> git merge testing
- On peut supprimer la branche '*testing*'
- \$> git branch -d testing

- Si nous modifions le même fichier dans deux branches qui seront fusionnées, il est possible que Git n'arrive plus à faire une fusion.
- La procédure est la suivante :
 - \$> git checkout master
 - \$> git merge future_conflit
- Un conflit apparaît, on rentre dans le(s) fichier(s) incriminé(s) et corrige le conflit.
 - \$> git add <les fichiers>
 - \$> git commit -m 'message'

Utilisation avancée de Git

- Pour modifier le commit :
 - \$> git commit --amend
- Cette commande va reprendre l'index du commit précédent
- L'éditeur s'ouvre avec le message.
- Il est possible de modifier le message du commit.

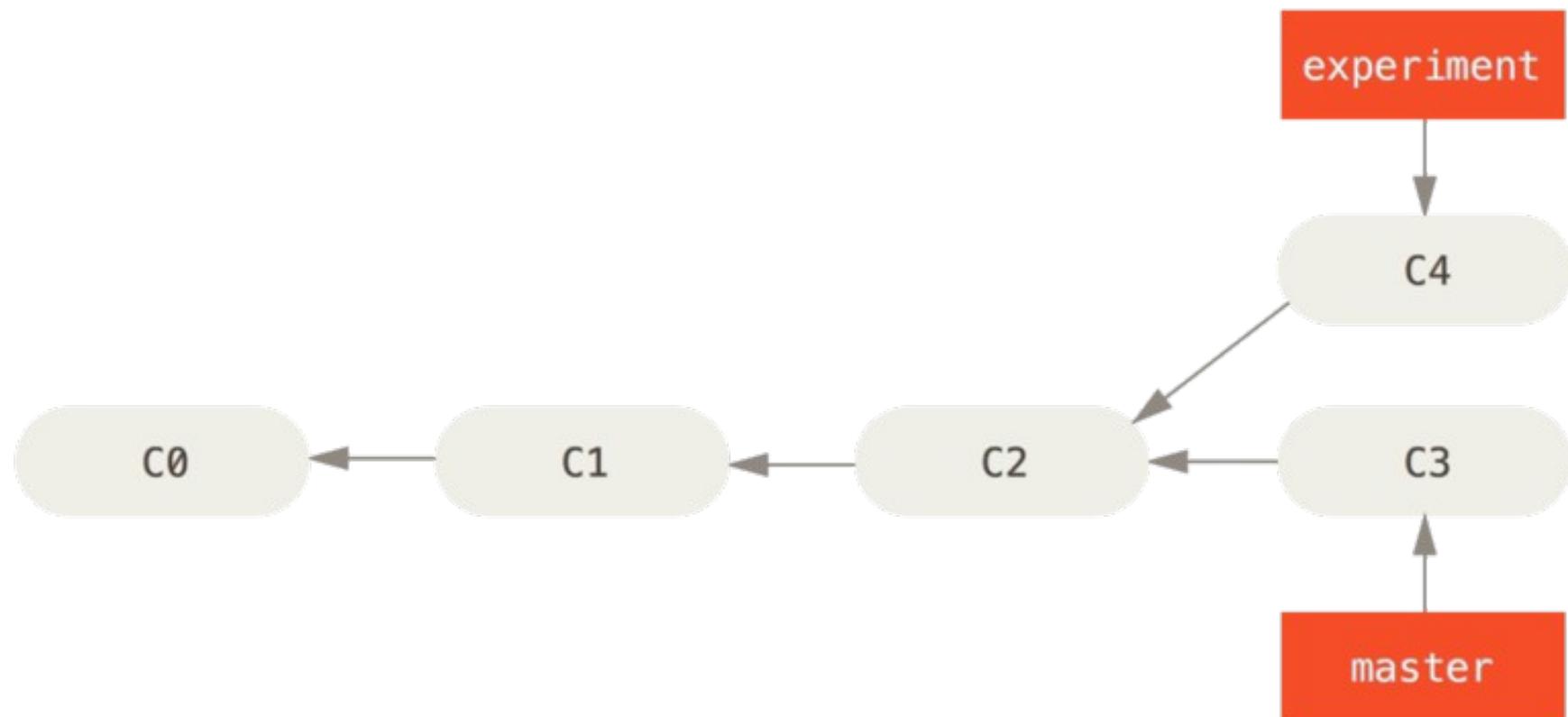
- Pour modifier un commit plus loin dans votre historique, vous devez utiliser **rebase**
- Par exemple, si vous voulez modifier les 3 derniers messages de commits ou n'importe lequel des messages dans ce groupe
 - \$> git rebase -i HEAD~3
- Remplacer le mot « **pick** » par le mot « **edit** » pour chaque commit à modifier
- Si, à la place de « **pick** » ou « **edit** », vous spécifiez « **squash** », Git applique cette modification et la modification juste précédente et fusionne les messages de validation.

Rebaser (rebasing)

- Il y a deux façons d'intégrer des modifications dans une branche :
 - En fusionnant (**merge**)
 - En rebasant (**rebase**)
- **Rebase** : prendre toutes les modifications qui ont été validées sur une branche et les rejouer sur une autre.

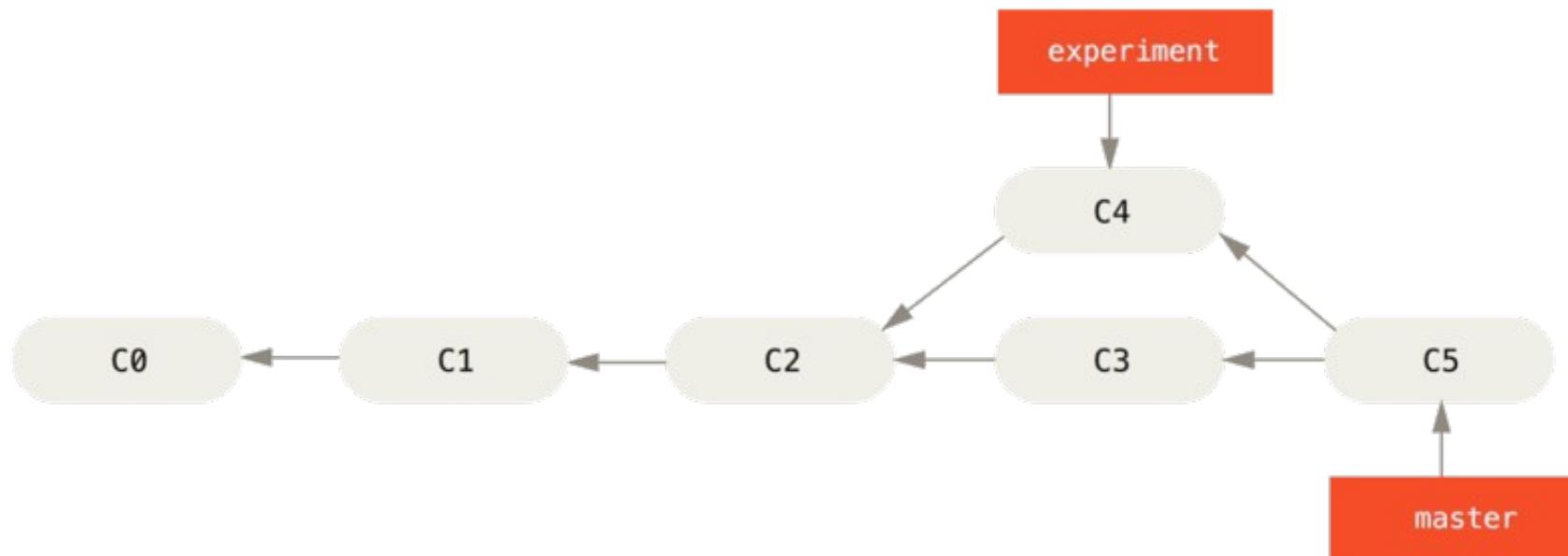
Rebaser (rebasing)

- Exemple :



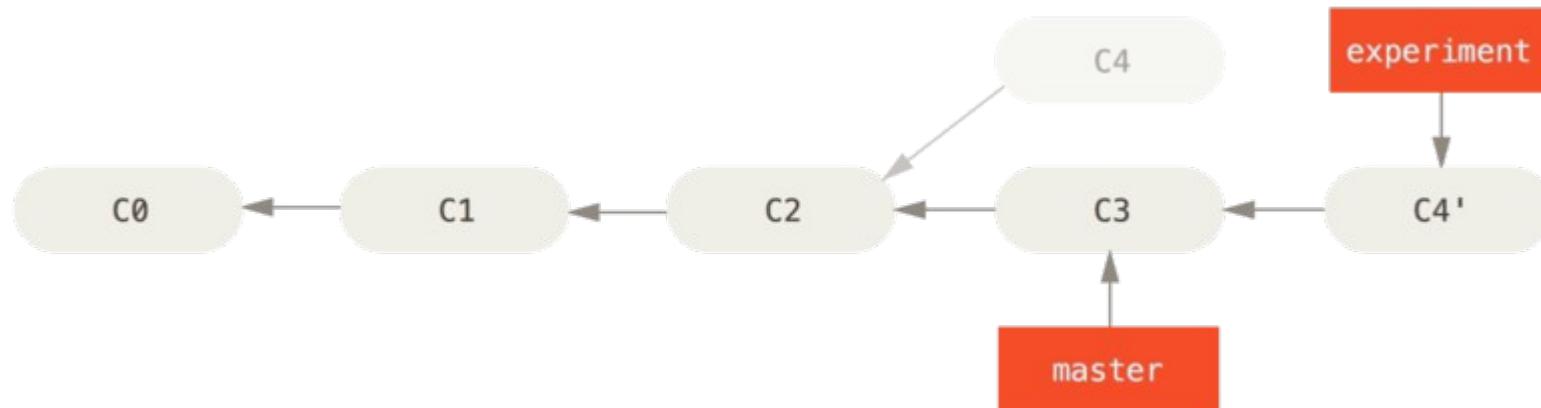
Rebaser (rebasing)

- Si on fait une merge :
 - \$> git checkout master
 - \$> git merge experiment



Rebaser (rebasing)

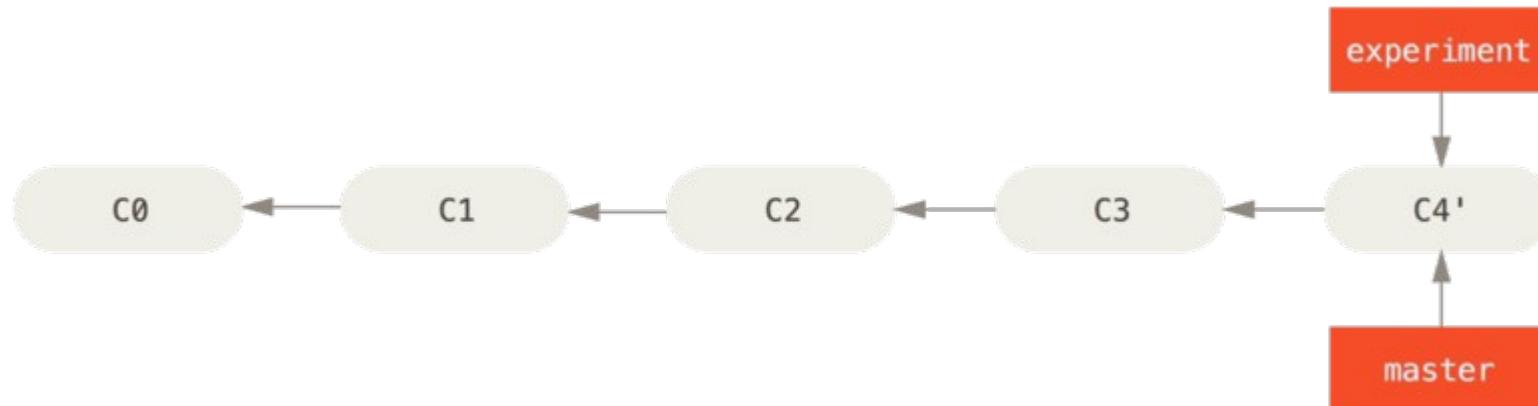
- Si on fait un rebasing :
 - \$> git checkout experiment
 - \$> git rebase master



- Prendre le patch de la modification introduite en C4 et le réappliquer sur C3. Dans Git, cette action est appelée "rebaser" (rebasing).

Rebaser (rebasing)

- Si on fait un rebasing :
 - \$> git checkout master
 - \$> git merge experiment master



- Vous pouvez retourner sur la branche master et réaliser une fusion en avance rapide (fast-forward merge).
- L'instantané pointé par C4' est exactement le même que celui pointé par C5 dans l'exemple de fusion.

- Commande puissante qui permet de choisir des commits Git et de les ajouter au HEAD actuel.
- Sélectionner un commit d'une branche et de l'appliquer à une autre.
- Peut être utile pour annuler des changements.
- Par exemple, Si un commit soit accidentellement intégré à la mauvaise branche. Vous pouvez passer à la bonne branche, sélectionner le commit et l'insérer à l'endroit où il devrait se trouver.
 - \$> git cherry-pick <commit>

Les plateformes Git

- GitHub est un service web d'hébergement et de gestion de développement de logiciels
- Il utilise le logiciel de gestion de versions Git.
- Propose des comptes professionnels payants, ainsi que des comptes gratuits pour les projets de logiciels libres.
- Assure un contrôle d'accès et des fonctionnalités destinées à la collaboration
- Le plus important dépôt de code au monde, utilisé comme dépôt public de projets libres ou dépôt privé d'entreprises.
- Acquis par Microsoft en Novembre 2018

- Logiciel libre basé sur git
- Propose des fonctionnalités de wiki, un système de suivi des bugs, l'intégration continue et la livraison continue.
- GitLab est scindé en deux versions
 - Une version libre, sous licence MIT nommé GitLab CE,
 - Une autre version contenant quelques modifications propriétaires, sous licence GitLab EE nommé GitLab EE
- GitLab.com : le service de forge en ligne, basé sur GitLab EE.
- Il possible d'installer GitLab sur un serveur privé de l'entreprise

Le travail collaboratif et les stratégies de branching

- Pour pouvoir collaborer sur un projet Git, il est nécessaire de savoir comment gérer les dépôts distants.
- Les dépôts distants sont des versions de votre projet qui sont hébergées sur Internet ou le réseau d'entreprise.
- Vous pouvez en avoir plusieurs, pour lesquels vous pouvez avoir des droits soit en lecture seule, soit en lecture/écriture.
- Nous allons voir comment :
 - Ajouter des dépôts distants.
 - Effacer des dépôts distants.

- Pour visualiser les serveurs distants que vous avez enregistré, vous pouvez lancer la commande ‘git remote’.
- Si vous avez cloné notre projet, vous devez avoir au moins un dépôt distant : ‘origin’.
- Vous pouvez aussi spécifier ‘-v’, qui vous montre l’URL que Git a stocké pour chaque nom court.

```
$ git remote -v
bakkdoor  https://github.com/bakkdoor/grit (fetch)
bakkdoor  https://github.com/bakkdoor/grit (push)
cho45    https://github.com/cho45/grit (fetch)
cho45    https://github.com/cho45/grit (push)
defunkt  https://github.com/defunkt/grit (fetch)
defunkt  https://github.com/defunkt/grit (push)
koke     git://github.com/koke/grit.git (fetch)
koke     git://github.com/koke/grit.git (push)
origin   git@github.com:mojombo/grit.git (fetch)
origin   git@github.com:mojombo/grit.git (push)
```

Ajouter des dépôts distants

- Pour ajouter des dépôts distants :
 - \$> git remote add projet1 <https://github.com/brahimhamdi/projet1>

Retirer et renommer des dépôts distants

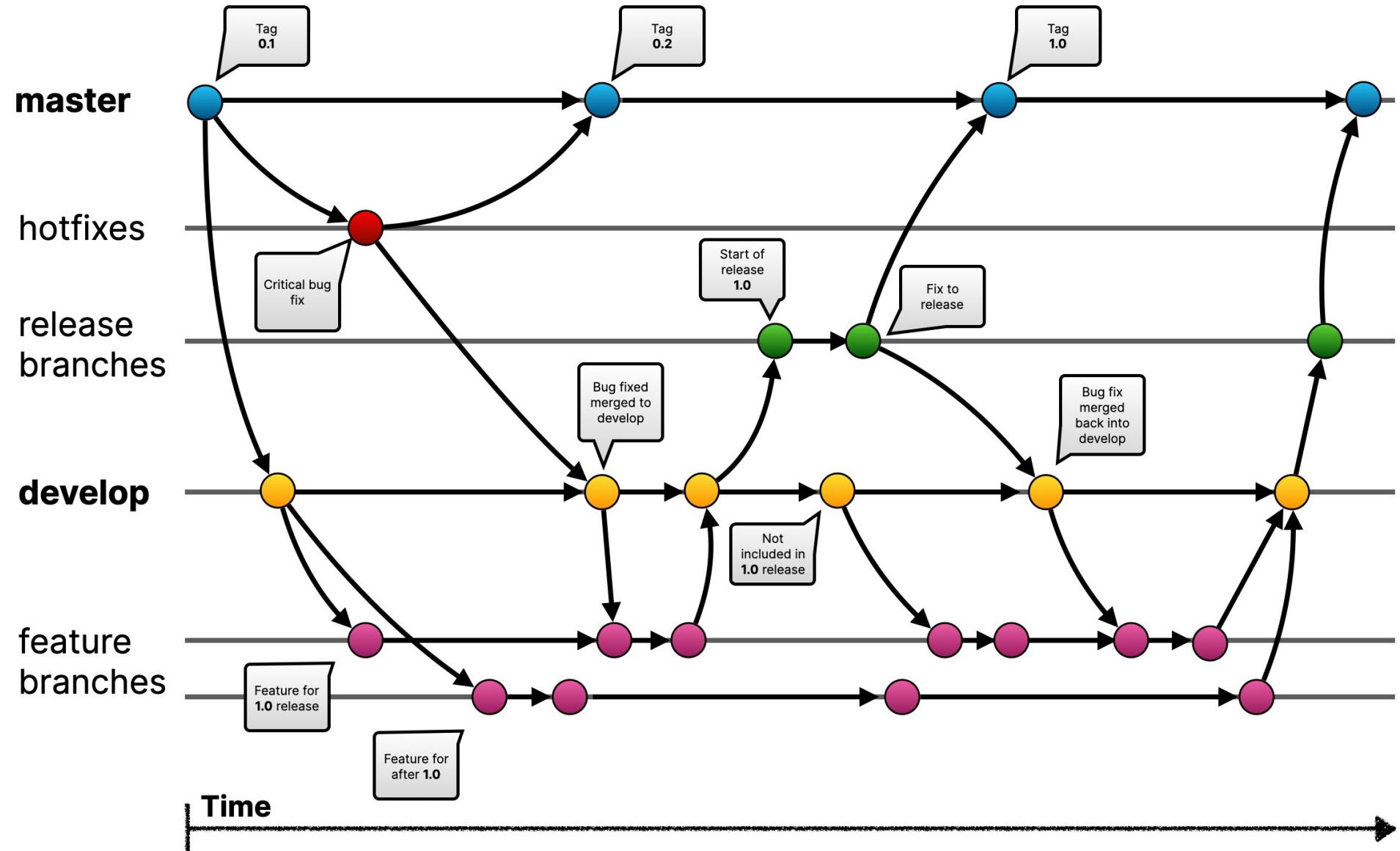
- Pour renommer une référence :
 - \$> git remote rename projet1 projet2
- Pour supprimer une référence :
 - \$> git remote rm projet1

- Vos branches locales ne sont pas automatiquement synchronisées sur les serveurs distants.
- Vous devez pousser explicitement les branches que vous souhaitez partager.
- Pour pusher une branche :
- \$> git push <distant> <branche>
- Exemple
 - \$> git push origin master
- La prochaine fois qu'un de vos collègues récupérera les données depuis le serveur, il récupérera la branche distante créée.

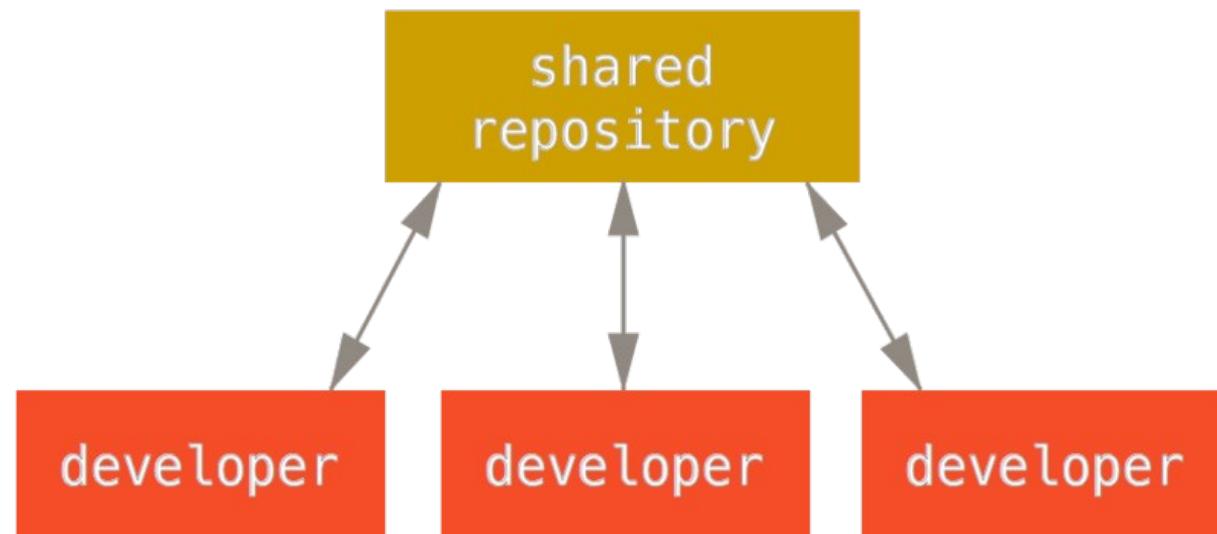
Récupérer et tirer depuis des dépôts distants

- Pour récupérer les données d'un dépôt distant :
 - \$> git fetch [nom du repos distant]
- Cette commande s'adresse au dépôt distant et récupère toutes les données de ce projet que vous ne possédez pas déjà.
- Si vous clonez un dépôt, le dépôt distant est automatiquement ajouté sous le nom « origin ».
- Si le dépôt est différent que celui ajouté par défaut, les branches seront ajoutées dans ‘<nom du dépôt distant>/master’.

- La commande ‘git fetch’ récupère l’ensemble des changements présents sur le serveur, mais elle ne modifie en rien votre répertoire de travail.
- Il faut faire un ‘git merge’ pour fusionner les nouveaux changements.
- La commande ‘git pull’ va faire un ‘git fetch’ et ‘git merge’ d’un coup.
 - \$> git pull

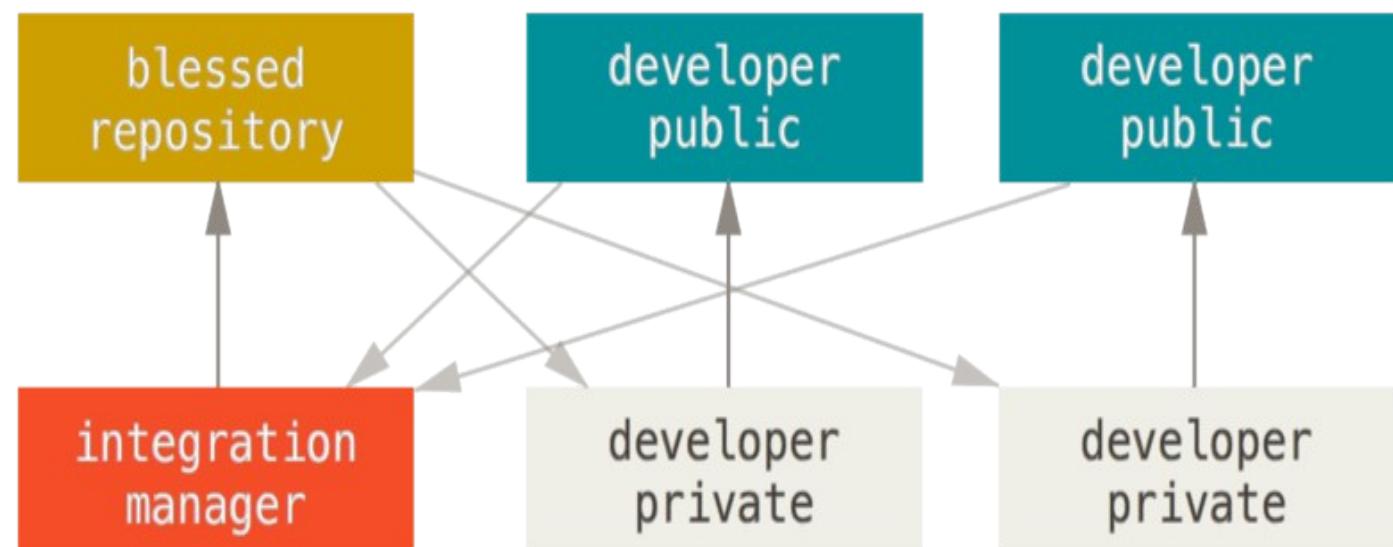


- Un seul modèle de collaboration.
- Tout le monde synchronise son travail avec un référentiel central.



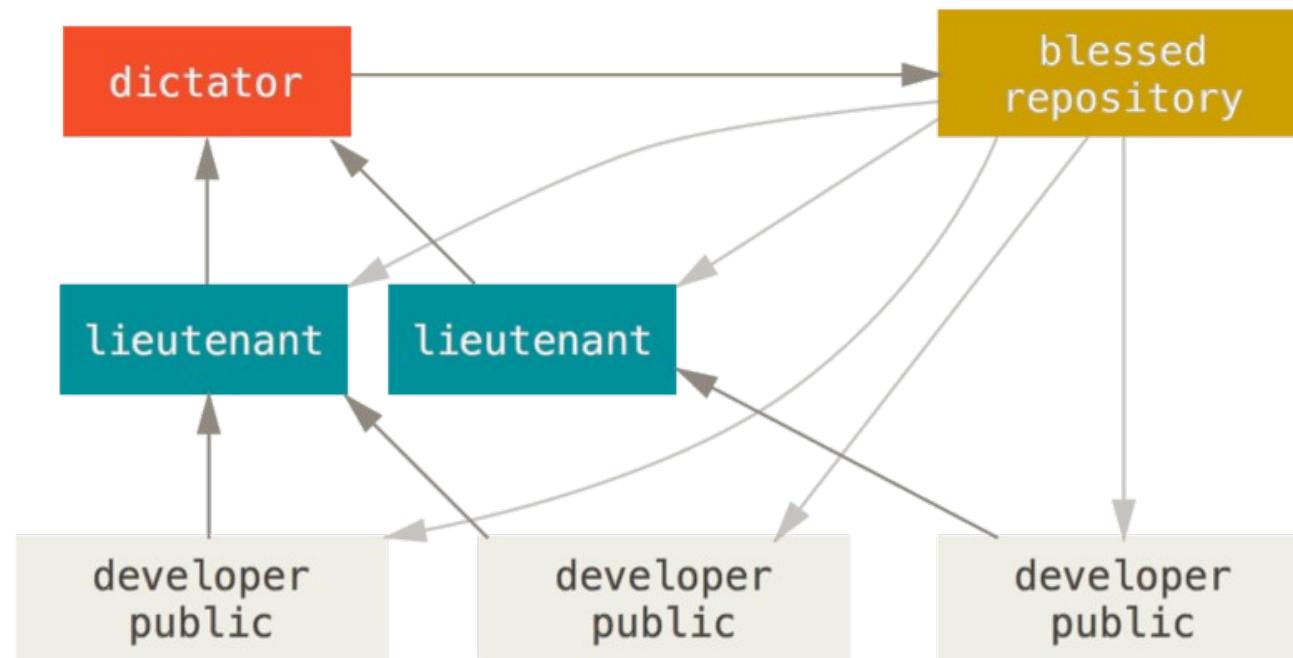
Workflow Integration-Manager

- 1 Le manager pousse vers son référentiel public
- 2 Un contributeur clone ce référentiel et apporte des modifications
- 3 Le contributeur pousse vers sa propre copie publique
- 4 Le contributeur envoie une demande au manager pour ajouter ses modifications
- 5 Le manager ajoute (s'il accepte) les modifications du contributeur au référentiel principal.



Workflow Dictator and Lieutenants

- 1 Les développeurs réguliers travaillent sur leur branche thématique et rebasent leur travail sur master.
- 2 Les lieutenants fusionnent les branches thématiques des développeurs dans leur branche master.
- 3 Le dictateur fusionne les branches master des lieutenants dans la branche master du dictateur.
- 4 Enfin, le dictateur pousse cette branche master vers le dépôt de référence afin que les autres développeurs puissent s'y baser.



- Qu'est ce que GitLab ?
- Les différentes distributions
- Les fonctionnalités de GitLab
- GitLab vs GitHub
- Variété des jargon Git sous GitLab
- Inscription sur GitLab.com
- Configurer l'accès via SSH



Qu'est-ce que Gitlab ?

- 📖 Utlil **open source** de gestion de projets **git** (licence MIT)
- 📖 Application Web développé en langage **Ruby** par GitLab Inc
- 📖 Dépôt : gitlab.com/gitlab-org/gitlab
- 📖 Dernière version : **16.7** (21 décembre 2023)



Fonctionnalités de GitLab

📘 Les principales fonctionnalités de GitLab sont de :

- 📘 gérer le cycle de vie de projets git.
- 📘 gérer les participants aux projets et leurs droits (rôles, groupes, etc.).
- 📘 déposer des Issues pour lister les bugs
- 📘 gérer la communication entre ces participants.
- 📘 proposer des Merge Requests pour fusionner les branches
- 📘 lancer des pipelines d'intégration et de déploiement continus via le module GitLab CI/CD
- 📘 fournir un Wiki pour la documentation

Distributions Gitlab : CE vs EE

Recommended

Enterprise Edition

- ✓ Take advantage of all of GitLab's features
- ✓ Free tier available*
- ✓ Support**
- ✓ Easily upgrade to paid tier in the future

[Install GitLab Enterprise edition >](#)

*See our [pricing page](#) for features available on each tier

**Support available on paid tiers only

Community Edition

- ✓ Open source
- ✓ Free tier only
- ✗ No support
- ✗ Will need to migrate to Enterprise Edition to upgrade to a paid tier in the future

[Install GitLab Community edition >](#)

If you're an organization using GitLab Community Edition looking for paid support, please [contact Sales](#) for options.

GitHub vs GitLab : Fonctionnalités

 GitHub	 GitLab
Les issues peuvent être suivies dans plusieurs repositories	Les issues ne peuvent pas être suivies dans plusieurs repositories
Repositories privés payants	Repositories privés gratuits
Pas d'hébergement gratuit sur un serveur privé	Hébergement gratuit possible sur un serveur privé
Intégration continue uniquement avec des outils tiers (Travis CI, CircleCI, etc.)	Intégration continue gratuite incluse
Aucune plateforme de déploiement intégrée	Déploiement logiciel avec Kubernetes
Suivi détaillé des commentaires	Pas de suivi des commentaires
Impossible d'exporter les issues au format CSV	Exportation possible des issues au format CSV par e-mail
Tableau de bord personnel pour suivre les issues et pull requests	Tableau de bord analytique pour planifier et surveiller le projet

GitHub vs GitLab : jargons Git

GitHub	GitLab	So, what does it mean?
Pull Request	Merge Request	In GitLab a request to merge a feature branch into the official master is called a Merge Request.
Gist	Snippet	Share snippets of code. Can be public, internal or private.
Repository	Project	In GitLab a Project is a container including the Git repository, discussions, attachments, project-specific settings, etc.
Organizations	Groups	In GitLab, you add projects to groups to allow for group-level management. Users can be added to groups and can manage group-wide notifications.



Inscription sur gitlab.com

Créer un compte [Gitlab](#) et authentifiez-vous avec votre login.

The screenshot shows the GitLab homepage with a purple header. On the left, there's a large 'Software. Faster.' heading and a 'Get free trial' button. In the center, a 'Vulnerability Report' card displays statistics: 49 Critical, 35 High, 56 Medium, and 2 Low vulnerabilities. Below the report are three recent findings: '23-01-21 ReDoS vulnerability', '23-01-19 Private key detected', and '23-01-17 Possible SQL injection'. To the right, there are two smaller cards: one for 'Approvals' showing 'Requires 2 approvals' and another for 'Review Security Scans' with a note about a security scan. On the far right, a snippet of Python code is shown:

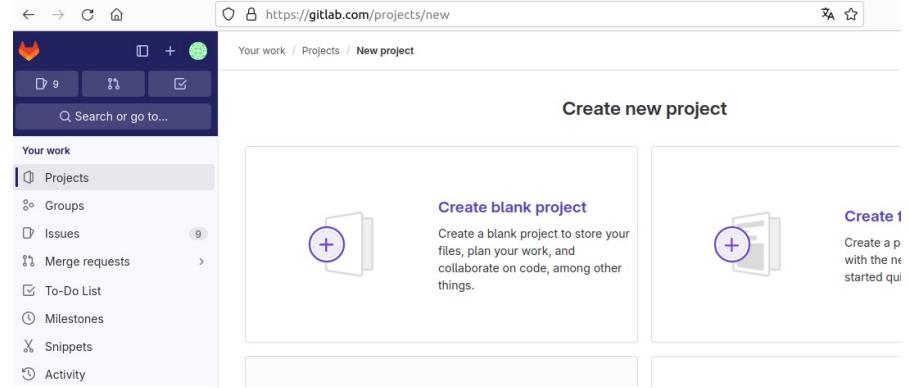
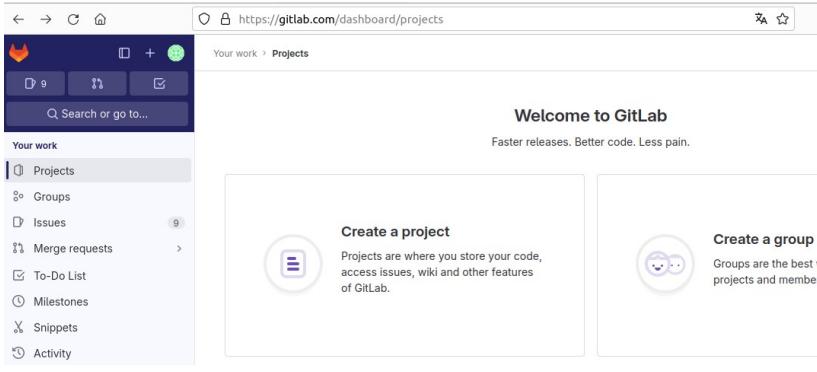
```
Translating languages = 'en':{ 'hello': 'He', 'goodbye': ' ', 'es':{
```

. At the top, a banner mentions Gartner naming GitLab as a Leader in the Magic Quadrant for DevOps Platforms, with a link to 'Read the report'.



Créer un nouveau projet

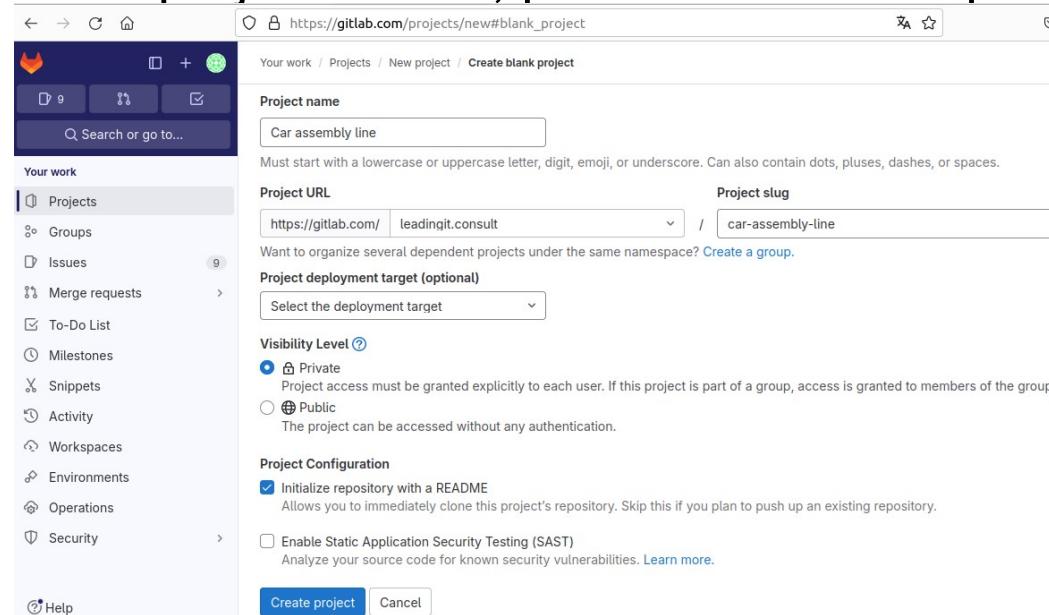
📘 Cliquez sur Project puis sur Create blank project/repository



The image shows two screenshots of the GitLab interface. The left screenshot is the 'Welcome to GitLab' dashboard with a sidebar containing 'Your work' sections like Projects, Groups, Issues, Merge requests, To-Do List, Milestones, Snippets, and Activity. A blue arrow points from the 'Create a project' button in the center of the dashboard to the right screenshot. The right screenshot is the 'Create new project' page, which has a similar sidebar. It features a large 'Create blank project' section with a plus icon and a brief description: 'Create a blank project to store your files, plan your work, and collaborate on code, among other things.' There is also a 'Create 1' section with a plus icon.

📘 Indiquez un Project name : **Car assembly line**

📘 Cochez qu'il s'agit d'un projet **Private**, puis validez en cliquant sur **Create project**.



The image shows the 'Create blank project' form on the GitLab website. The 'Project name' field contains 'Car assembly line'. The 'Project URL' field shows 'https://gitlab.com/ leadingit.consult' and the 'Project slug' field shows 'car-assembly-line'. Under 'Visibility Level', the 'Private' radio button is selected. In the 'Project Configuration' section, the 'Initialize repository with a README' checkbox is checked, with a note explaining it allows immediate cloning. The 'Create project' button is at the bottom.



Page d'accueil

The screenshot shows the GitLab homepage for the project "Car assembly line". The interface includes a header with a search bar and user profile, a sidebar with navigation links, and a main content area displaying project details, commit history, and file lists.

- Accès rapide aux merges requests, issues et tasks (Red box around the top navigation bar)
- Accès à la page d'accueil (projets, groupes, ...) (Red box around the top navigation bar)
- Gestion des paramètres du compte (Red box around the top navigation bar)
- Infos générales du projet (Red box around the project summary card)
- La branche en cours de consultation (Red box around the selected branch dropdown)
- Dernier commit sur la branche sélectionnée (Red box around the latest commit card)
- Liste des fichiers de la branche sélectionnée (Red box around the file list table)
- Clone du dépôt (Red box around the "Clone" button)

Project ID: 52099193

1 Commit 1 Branch 0 Tags 3 KiB Project Storage

Initial commit leadingit consult authored just now

main car-assembly-line / +

Name	Last commit	Last update
README.md	Initial commit	just now

README.md

Car assembly line



Les outils du projet

Le code source, les commits, les branches, ...

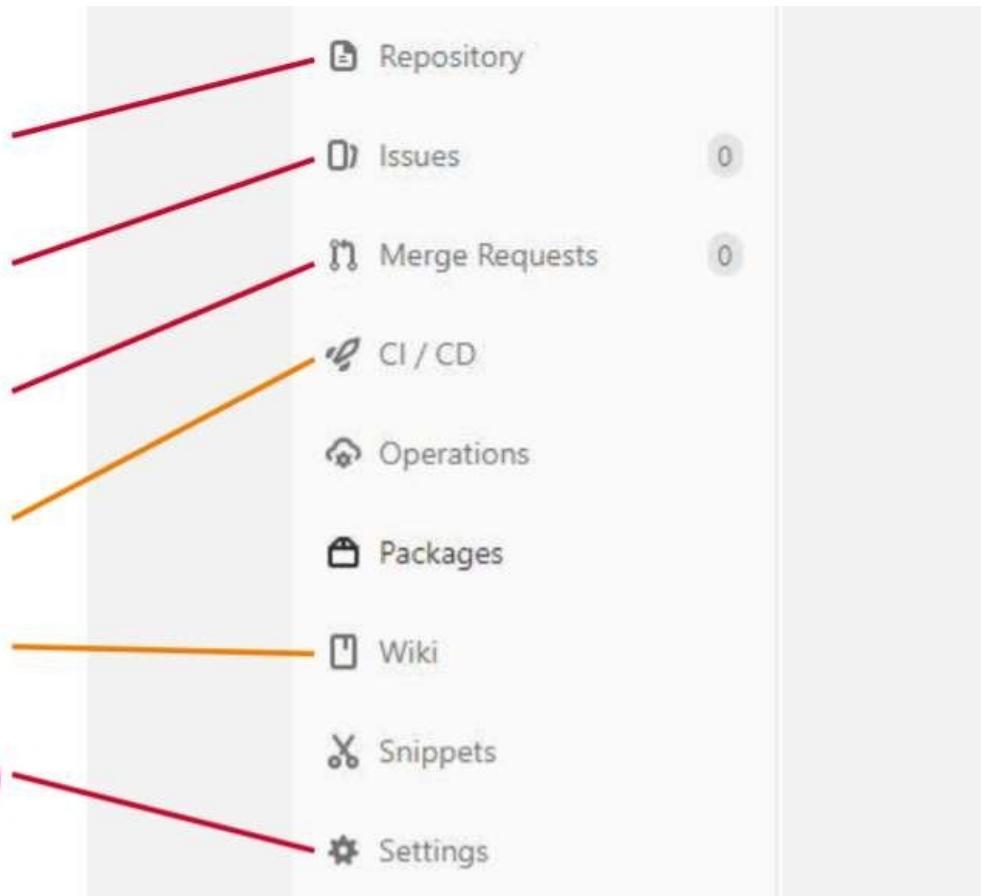
Board des développements en cours, le backlog, ...

Gestion des fusions de branches

Outils d'intégration et de déploiement continu

Documentation sur tout le projet

Paramétrage de tout le projet (membres, branches, ...)





Configurer l'accès via SSH

- 📘 Nous devons configurer par la suite l'accès à notre dépôt via ssh (uniquement par la paire clé privée/publique, pas par login et mot de passe)
- 📘 Il faut donc avoir créé sur la machine ce couple de clé en utilisant la commande : **ssh-keygen**
- 📘 Ensuite, vous devez définir un mot de passe pour protéger votre clé privée.

- 📘 Votre paire de clés publique/privée a été générée :
- 📘 **id_rsa** : clé **PRIVÉE**.
- 📘 **id_rsa.pub** : clé **PUBLIC**.

```
valentin@MintLinux:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/valentin/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/valentin/.ssh/id_rsa
Your public key has been saved in /home/valentin/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:U1c9F7CWC1apJn+t16z3XDKDkqN9uIuFVeYYDKY9X1M valentin@MintLinux
The key's randomart image is:
+---[RSA 3072]----+
| o   oEo.| 
| + o  .+o.o| 
| . o +oB+ o| 
| =.@o..| 
| S B ...| 
| + o o .| 
| . =.o =.o| 
| =.o.. *=| 
| o ++ oo+| 
+---[SHA256]----+
valentin@MintLinux:~$
```

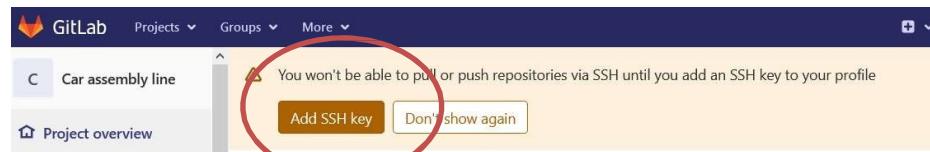


Configurer l'accès via SSH

Copier le contenu de votre clé public

```
valentin@MintLinux:~$ cat /home/valentin/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQgQCzck/SbzKNTx5bY4Im+dGv26henS/bZXV04lEgMjFL
hQXhsg//Sab6fQXl215jVT5QJHNiITZrPl4kGCqYegU7UDXD66Nmm7IE4hqwneEc+CqhSQj+o7icgGnG
8HU0or5PnsaSJZxNt4ms8BGTSR63WU4Knv358zgUgGxpRWBoRFLXikr0QyEqJP0qcTPQV/XYh2MrJ54v
p3UC0j5lgHdP4YIz8Fc32F6JzU5KfiImrXN7l8SZ4K68bTirkCL7oHgqqEvRZbLjEfwrQ+dAQj+dwdQd
T45KCkiRrmtv9MmcA2Vg2HAEHFi5WC9H+YolSuBrD/sSl5F85DSuf73DvkpIOfISudqnj7+++Hu9Ypo5
bp1IRfpILY3GjrIuaNZdUWm2n218Iz9VeEyRAh7ai4kJC0UJiGxR63xlq2SnkNdD8vrIgLTUk7fqhhtL
XQ2FDhuqiTw+559/pDHJgQ6mgYViSrEEasIEkQfd3zXlkqlU2ajHoG5i371XTkT1UGp42BM= valenti
n@MintLinux
```

Pour ajouter votre clé SSH à GitLab, cliquer sur le bouton **Add SSH key**



Collez votre clé publique dans la grande zone de texte et cliquez sur le bouton **Add Key**

Add an SSH key

To add an SSH key you need to generate one or use an existing key.

Key

Paste your public SSH key, which is usually contained in the file '`~/.ssh/id_ed25519.pub`' or '`~/.ssh/id_rsa.pub`' and begins with '`ssh-ed25519`' or '`ssh-rsa`'. Do not paste your private SSH key, as that can compromise your identity.

```
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQgQCzck/SbzKNTx5bY4Im+dGv26henS/bZXV04lEgMjFL
hQXhsg//Sab6fQXl215jVT5QJHNiITZrPl4kGCqYegU7UDXD66Nmm7IE4hqwneEc+CqhSQj+o7icgGnG
8HU0or5PnsaSJZxNt4ms8BGTSR63WU4Knv358zgUgGxpRWBoRFLXikr0QyEqJP0qcTPQV/XYh2MrJ54v
p3UC0j5lgHdP4YIz8Fc32F6JzU5KfiImrXN7l8SZ4K68bTirkCL7oHgqqEvRZbLjEfwrQ+dAQj+dwdQd
T45KCkiRrmtv9MmcA2Vg2HAEHFi5WC9H+YolSuBrD/sSl5F85DSuf73DvkpIOfISudqnj7+++Hu9Ypo5
bp1IRfpILY3GjrIuaNZdUWm2n218Iz9VeEyRAh7ai4kJC0UJiGxR63xlq2SnkNdD8vrIgLTUk7fqhhtL
XQ2FDhuqiTw+559/pDHJgQ6mgYViSrEEasIEkQfd3zXlkqlU2ajHoG5i371XTkT1UGp42BM= valenti
n@MintLinux
```

Title Expires at

Give your individual key a title.

Add key

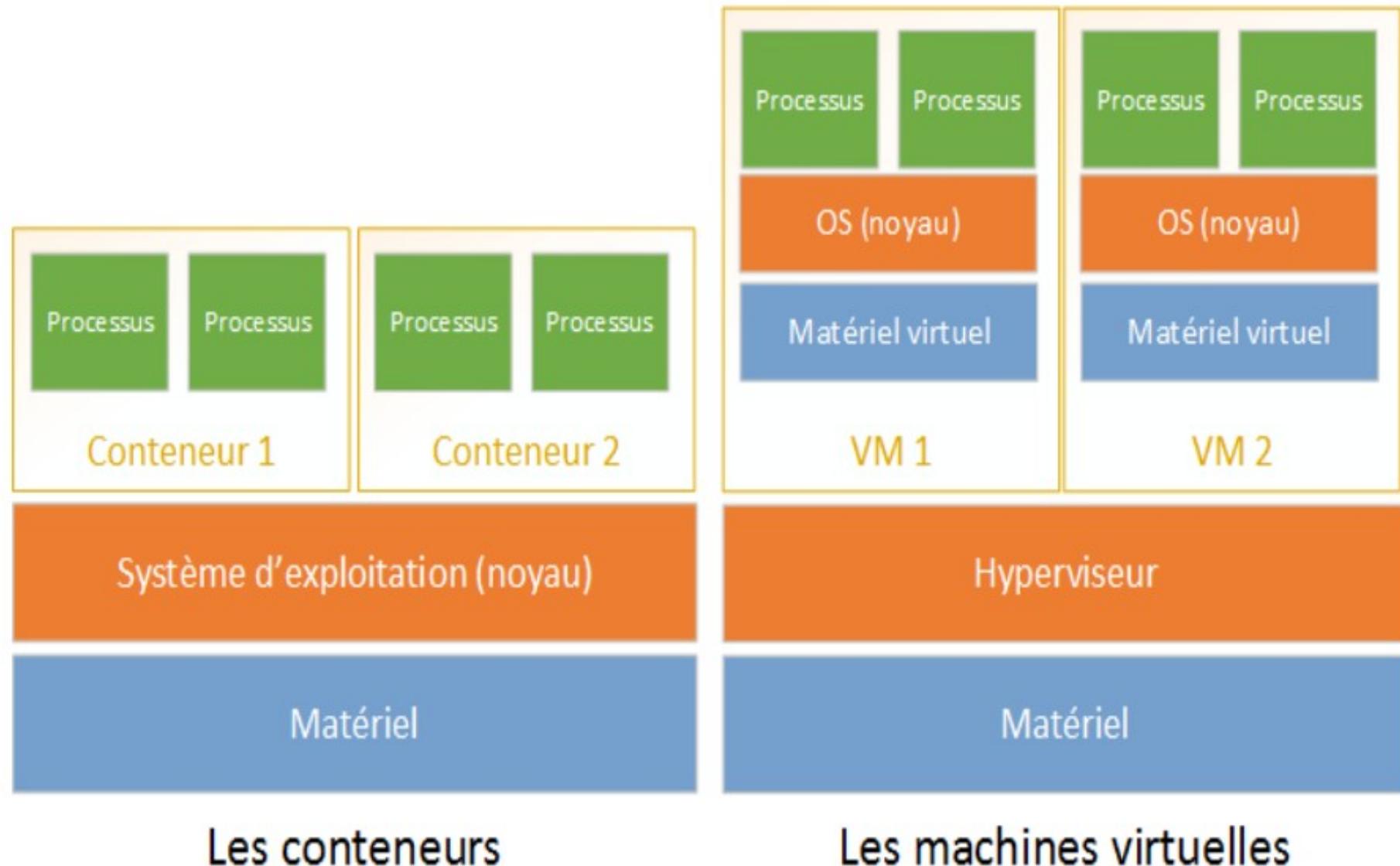
Module 3 : Gestion des conteneurs Docker

- Comprendre les conteneurs
- Docker
- Gestion des conteneurs
- Gestion des images
- Création des images
- Gestion des volumes
- Docker Compose

Comprendre les conteneurs

- Les mêmes idées que la virtualisation, mais sans virtualisation
 - Isolation et automatisation
 - Principe d'infrastructure consistante et répétable
 - Peu d'overhead par rapport à une VM !
 - Un super chroot
-
- Certains parlent de virtualisation "niveau OS" ou "légère", isolation applicative

Différences entre VM et conteneur



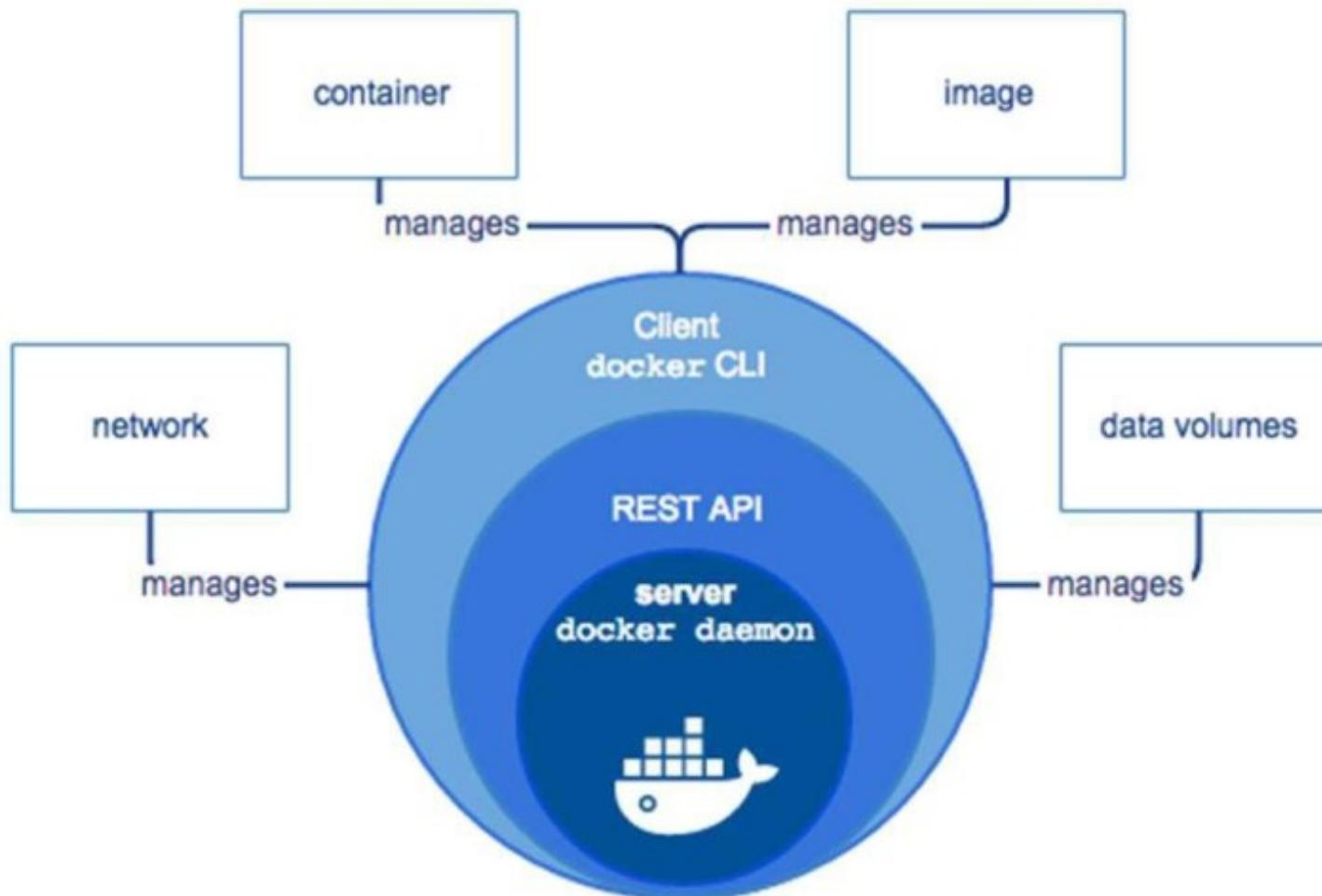
Les conteneurs

Les machines virtuelles

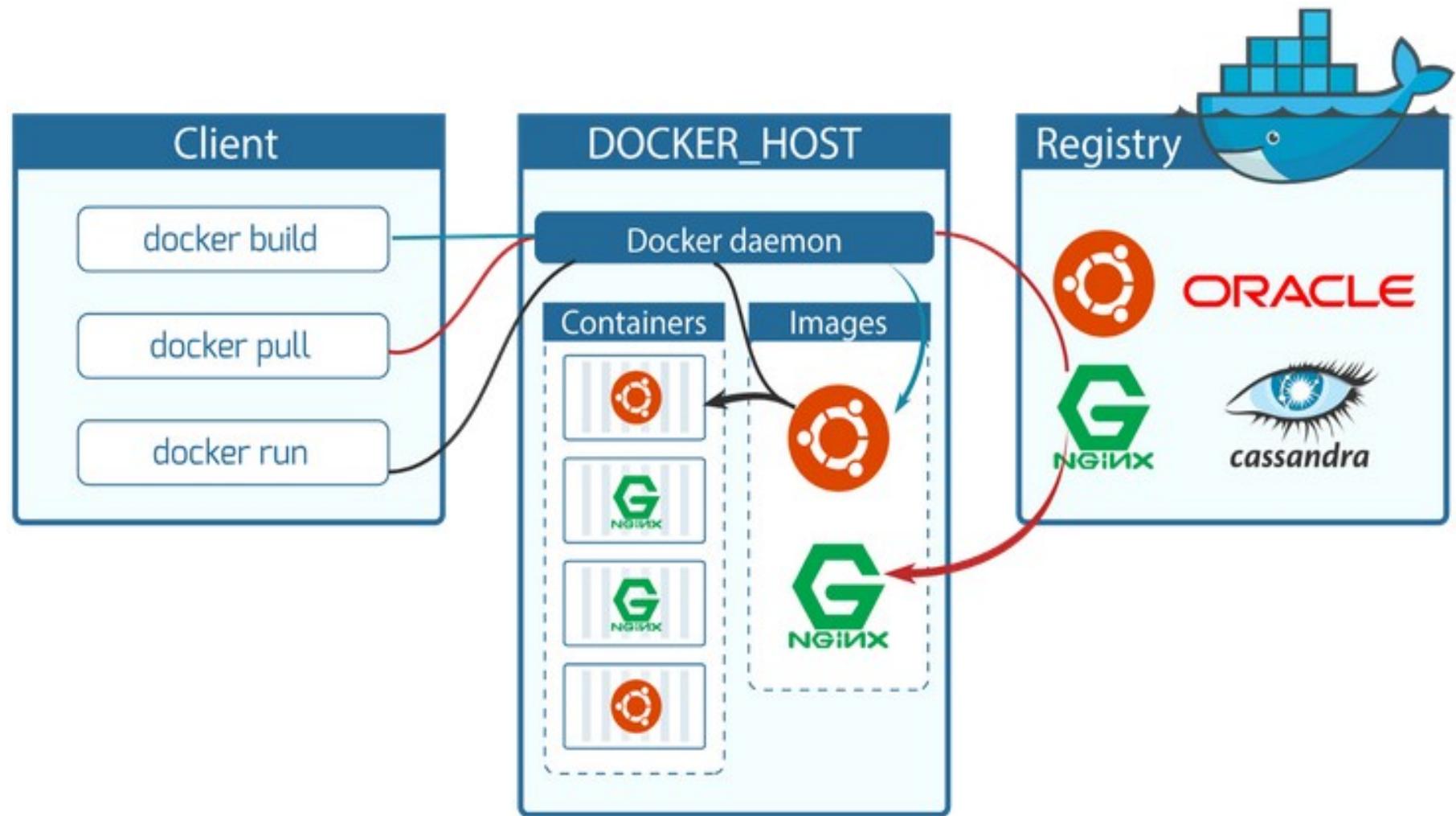
Docker

- Remplace les machines virtuels (VM).
- Permet de prototyper les applications.
- Permet le packaging d'applications.
- Ouverture vers les microservices.
- Modélisation d'un réseau informatique avec un budget réduit.
- Permet une certaine productivité mais avec des machines déconnectées.
- Réduire le temps de recherche des bugs.
- Renforce la documentation dès le début du cycle de vie d'une mise en production.
- Permet la mise en place du Continuous Delivery (CD).

Architecture (1)

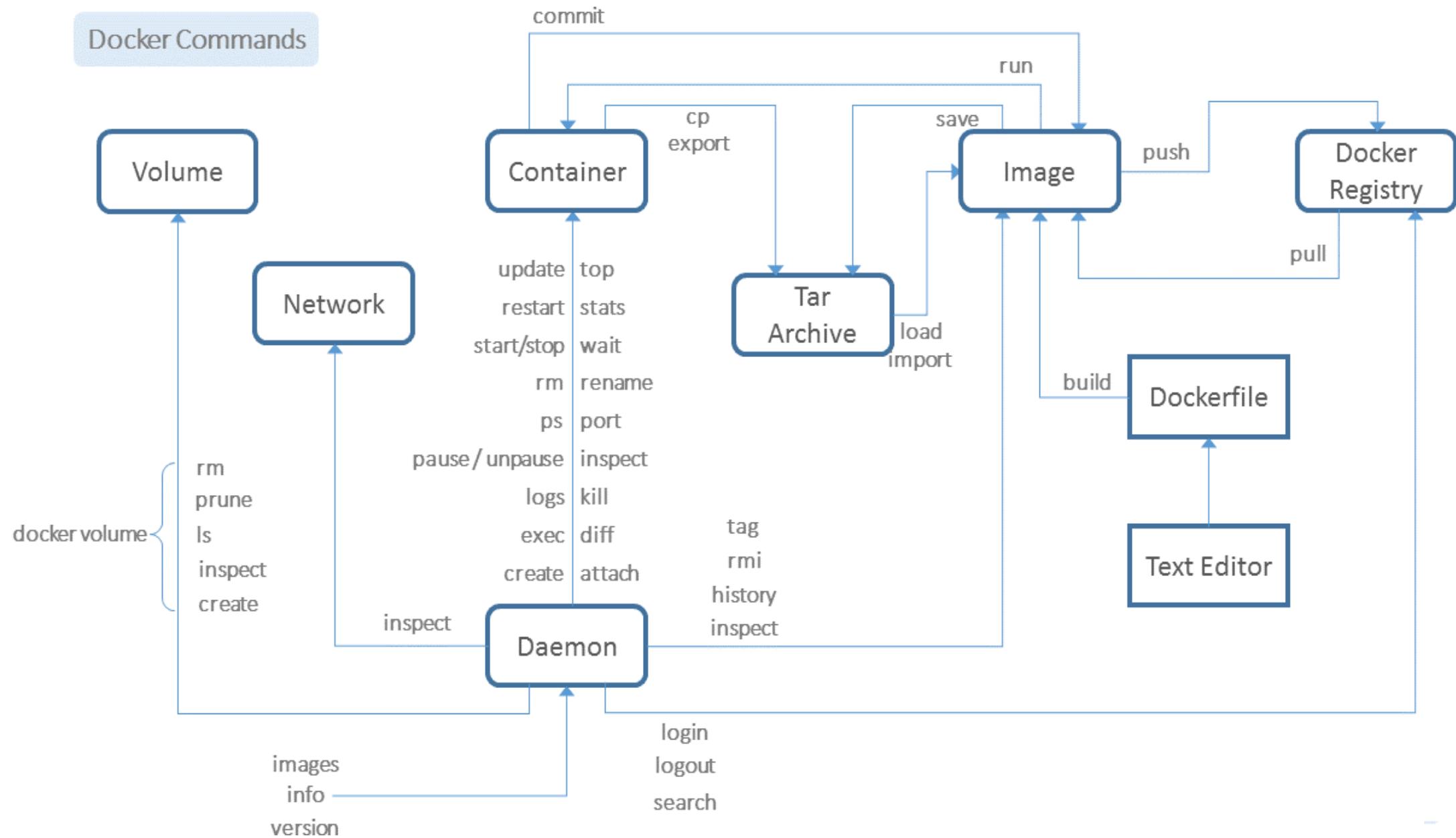


Architecture (2)



- Principe
 - Un serveur stockant des images docker
 - Possibilité de récupérer des images depuis ce serveur (pull)
 - Possibilité de publier de nouvelles images (push)
- Docker Hub
 - Dépôt publique d'images Docker

Commandes Docker



Gestion les conteneurs

Conteneur basique

```
$ docker run debian /bin/echo "Salut"  
Unable to find image 'debian:latest' locally  
latest: Pulling from library/debian  
bd8f6a7501cc: Pull complete  
Digest:  
sha256:ba4a437377a0c450ac9bb634c3754a17b1f814ce6fa3157c0dc9eef431b29d1f  
Status: Downloaded newer image for debian:latest  
Salut  
  
$ docker run debian /bin/echo "Coucou"  
Coucou
```

Commandes ps

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
1bc62c99a11b	debian	"/bin/echo Coucou"	About a minute ago	Exited (0) minute ago		blissful_ritchie
d23a8f3261da	debian	"/bin/echo Salut"	2 minutes ago	Exited (0) 2 minutes ago		crazy_kapitsa

```
$ docker logs crazy_kapitsa
```

Salut

```
$ docker rm crazy_kapitsa  
crazy_kapitsa
```

- rm uniquement sur un container arrêté!
- Sinon, il faut d'abord le stopper puis le détruire

Interrompre le conteneur et son processus

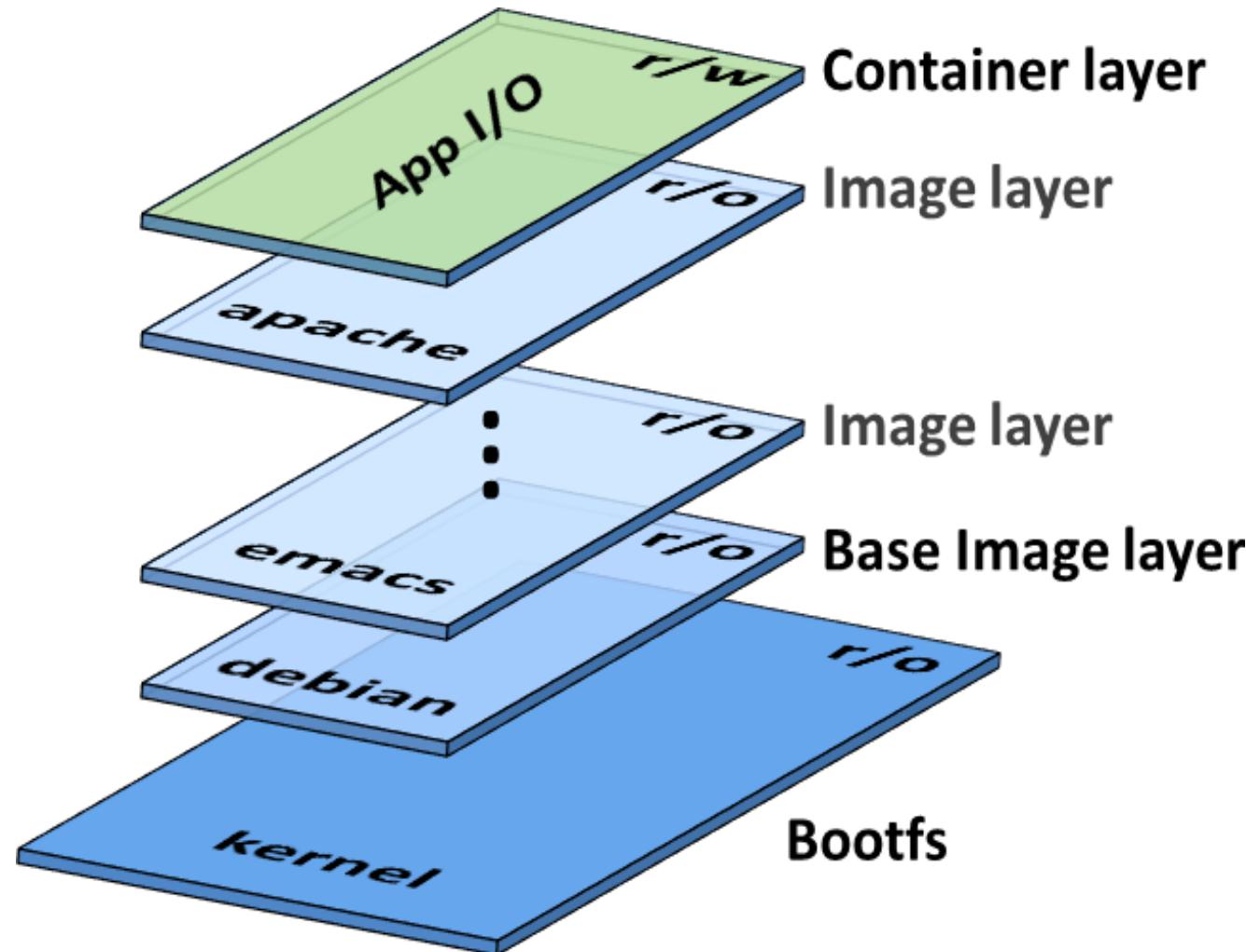
```
$ docker stop $conteneur  
$ docker kill $conteneur  
$ docker pause $conteneur #unpause
```

- Ou tout simplement arrêter le processus (exit pour bash)

Gestion des images

- Collection de fichiers + métadonnées
 - Ces fichiers forment le FS racine du conteneur
- Composées de couches, en théorie superposées
- Chaque couche peut rajouter, modifier, supprimer des fichiers
- Des images peuvent partager des couches pour optimiser
 - L'usage disque
 - Les temps de transfert

Notion de couches



- Ubuntu
- Mysql
- Wordpress
- Application JAVA
- nginx

...

- Service en ligne, officiel de Docker, pour distribuer les images:
<https://hub.docker.com/>
- Composants:
 - Un index: indexe toutes les méta-donnés des images hébergées pour recherche.
 - Un registre: stocke les couches des images pour récupération et upload.
 - Par défaut, les commandes de Docker liées aux images utilisent le Docker Hub.

- Il existe 3 manières de nommer des images
- Images officielles:
 - Ubuntu
 - debian
- Images utilisateur:
 - brahimhamdi/myapp
- Images auto-hébergées:
 - registry.example.com:5000/my-private/image

- Faire un pull pour récupérer l'image en local.
 - *\$ docker pull nginx*
 - *Using default tag: latest*
 - *latest: Pulling from library/nginx*
 - *f7ec5a41d630: Pull complete*
 - *aa1efa14b3bf: Pull complete*
 - *b78b95af9b17: Pull complete*
 - *c7d6bca2b8dc: Pull complete*
 - *cf16cd8e71e0: Pull complete*
 - *0241c68333ef: Pull complete*
 - *Digest: sha256:75a55d33ecc73c2a242450a9f1cc858499d468f077ea942867e662c247b5e412*
 - *Status: Downloaded newer image for nginx:latest*
 - *docker.io/library/nginx:latest*
- Elle sera ensuite utilisable pour créer un conteneur avec `docker run`
- NB: `docker run` fait un pull si l'image n'est pas disponible en local

- Images disponibles localement :

- `$ docker images`

- `REPOSITORY`

- `dockercoins_worker`

- `dockercoins_webui`

- `dockercoins_hasher`

- `dockercoins_rng`

- `python`

- `wiki_app`

- `wiki_db`

- `<none>`

- `app`

- `ruby`

- `nginx`

- `debian`

-

	<code>TAG</code>	<code>IMAGE ID</code>	<code>CREATED</code>	<code>SIZE</code>
• <code>dockercoins_worker</code>	<code>latest</code>	<code>1f49d8a6bd9d</code>	<code>2 days ago</code>	<code>55.2MB</code>
• <code>dockercoins_webui</code>	<code>latest</code>	<code>00662d83fad5</code>	<code>2 days ago</code>	<code>218MB</code>
• <code>dockercoins_hasher</code>	<code>latest</code>	<code>b1c27d7f9b33</code>	<code>2 days ago</code>	<code>263MB</code>
• <code>dockercoins_rng</code>	<code>latest</code>	<code>6f67397e4940</code>	<code>2 days ago</code>	<code>56.2MB</code>
• <code>python</code>	<code>alpine</code>	<code>bdb6a9278a0c</code>	<code>2 days ago</code>	<code>44.9MB</code>
• <code>wiki_app</code>	<code>1.0</code>	<code>0a925d75ecfe</code>	<code>12 days ago</code>	<code>510MB</code>
• <code>wiki_db</code>	<code>1.0</code>	<code>d7d5641ba218</code>	<code>12 days ago</code>	<code>387MB</code>
• <code><none></code>	<code><none></code>	<code>ed808816acc6</code>	<code>12 days ago</code>	<code>485MB</code>
• <code>app</code>	<code>1.0</code>	<code>979fc1fb7b99</code>	<code>13 days ago</code>	<code>845kB</code>
• <code>ruby</code>	<code>alpine</code>	<code>d46c4b238695</code>	<code>2 weeks ago</code>	<code>60MB</code>
• <code>nginx</code>	<code>latest</code>	<code>62d49f9bab67</code>	<code>2 weeks ago</code>	<code>133MB</code>
• <code>debian</code>	<code>latest</code>	<code>0d587dfbc4f4</code>	<code>2 weeks ago</code>	<code>114MB</code>

- OU `docker image ls`

- Les images peuvent avoir des tags
- Un tag définira une version, une variante différente d'une image
- par défaut le tag est latest:
- docker run ubuntu == docker run ubuntu:latest
- Un tag est juste un alias, un surnom pour un identifiant d'image
- plusieurs tags différents == une image

- `$ docker images prom/prometheus`
- *REPOSITORY TAG IMAGE ID CREATED SIZE*
- *prom/prometheus latest de242295e225 11 months ago 140MB*
- *prom/prometheus v2.1.0 c8ecf7c719c1 3 years ago 112MB*

```
$ docker tag debian brahimhamdi/debian:10
```

```
$ docker images |grep debian
```

brahimhamdi/debian	10	0d587dfbc4f4	2 weeks ago	114MB
debian	latest	0d587dfbc4f4	2 weeks ago	114MB

- On n'utilisera pas les tags
 - durant les tests et prototypage
 - Expérimentations
 - quand vous avez besoin de la dernière version

- On les utilisera
 - pour utiliser une image spécifique en production
 - pour créer une image qui évolue

Création des images

- Principe :
 - Recette automatisée de création d'images
 - Contient une suite d'instructions
 - La commande docker build utilise le Dockerfile pour créer l'image

- On travaille dans un dossier qui va contenir le Dockerfile propre à notre future image:

```
$ mkdir -p ~/docker/nginx
```

- On se place dans ce dossier et on ouvre un fichier Dockerfile
 - \$ cd ~/docker/nginx/
 - \$ edit Dockerfile
- Contenu du Dockerfile:

```
FROM debian
```

```
RUN apt-get update
```

```
RUN apt-get install -y nginx
```

- Un Dockerfile est composé d'instruction, une par ligne.
- FROM: image de base à utiliser pour notre future image
 - Un seul FROM par Dockerfile
- RUN: commandes shell à exécuter
 - seront exécutées durant le processus de build utilisable à volonté
 - non-interactive: aucun input possible durant le build

- Depuis le dossier contenant le Dockerfile :

```
$ docker build -t brahimhamdi/nginx:2.0 .
```

Sending build context to Docker daemon 2.048 kB

Step 1 : FROM debian

---> 93a2e30f1000

Step 2 : RUN apt-get update

---> Running in 20f50a8284f5

Get:1 http://security.debian.org jessie/updates InRelease [63.1 kB]

...

Processing triggers for sgml-base (1.26+nmu4) ...

---> 95f9e15fa8d2

Removing intermediate container e99f0c6f5bd2

Successfully built 95f9e15fa8d2

- L'image obtenue permet de démarrer un conteneur, de manière similaire à celle créée manuellement:

```
$ docker run -it brahimhamdi/nginx:2.0 nginx -v  
nginx version: nginx/1.14.2
```

- Avec l'instruction CMD, on peut définir une commande à exécuter par défaut lorsque l'on lance un conteneur.
- Par exemple:

```
FROM debian
RUN apt-get update
RUN apt-get install -y nginx
CMD nginx -v
```

Build et test de CMD

```
$ docker build -t brahimhamdi/nginx:3.0 .
$ docker run -it brahimhamdi/nginx:3.0
nginx version: nginx/1.14.2
```

Outrepasser CMD

```
$ docker run -it brahimhamdi/nginx:3.0 echo salut  
salut
```

- Définit une commande de base à exécuter par le conteneur,
- Les paramètres de la ligne commande sont ajoutés à ces paramètres.

Utilisation de ENTRYPPOINT

```
FROM debian
RUN apt-get update
RUN apt-get install -y nginx
ENTRYPOINT ["nginx", "-g"]
```

Build avec ENTRYPPOINT

```
$ docker build -t brahimhamdi/nginx:4.0 .
```

Exécution de ENTRYPPOINT

```
$ docker run -it brahimhamdi/nginx:4.0 "param bidon;"  
nginx: [emerg] unknown directive "param" in command line  
$ docker run -it brahimhamdi/nginx:4.0 "daemon off;"  
#nginx s'exécute en avant plan
```

- Depuis un autre terminal :

```
$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	
STATUS				
29e86a41e506	brahimhamdi/nginx4.0		About a minute ago	
About a minute ago				UP

CMD et ENTRYPOINT

```
FROM debian
RUN apt-get update
RUN apt-get install -y nginx
ENTRYPOINT ["nginx", "-g"]
CMD ["daemon off;"]
```

Build CMD et ENTRYPOINT

```
$ docker build -t brahimhamdi/nginx:5.0 .
```

Exécution CMD et ENTRYPOINT

- `$ docker run -d brahimhamdi/nginx:5.0`
- `10bb961dfe60fc4c92b45f6a1a390f62f7edc0f6d78fe2088a0cf08c6d6cb040`
- Nous avons un nginx qui tourne: comment y accéder ?

- L'instruction COPY permet de copier fichiers et dossiers depuis le contexte de génération, dans le conteneur.
- Imaginons que l'on veuille modifier la page d'accueil de notre serveur Nginx?

- `$ echo "Bienvenue sur mon image Nginx" > index.html`

- `FROM debian`
- `RUN apt-get update`
- `RUN apt-get install -y nginx`
- `EXPOSE 80 443`
- `COPY index.html /var/www/html/index.html`
- `ENTRYPOINT ["nginx", "-g"]`
- `CMD ["daemon off;"]`

- *\$ docker build -t brahimhamdi/nginx:7.0 .*

- `$ docker run -it -P -d brahimhamdi/nginx:7.0`
- `0e84e253eb42b025aa5d15debb0290a9944687c8587233bf787d9c21db1af64a`
- `$ docker port 0e84e253eb 80`
- `0.0.0.0:32771`



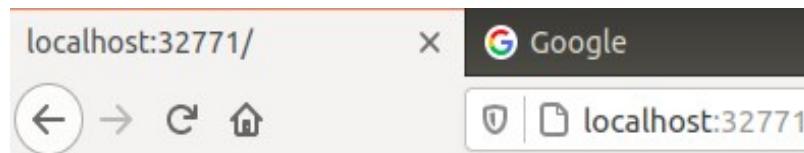
Gestion des volumes

- Si je veux modifier index.html, je dois regénérer une image
- Nginx génère des logs
- Ces modifications engendrent des données dans une couche
- Je voudrais les partager avec un autre serveur

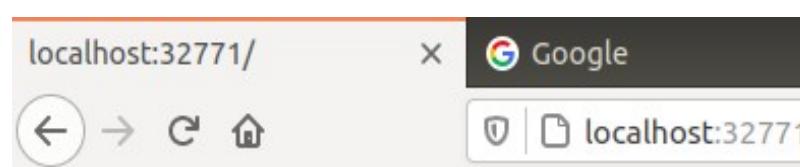
- Les volumes peuvent être partagés:
 - entre conteneurs
 - entre hôte et un conteneur
- Les accès au système de fichiers via un volume outrepassent le CoW:
 - Meilleures performances
 - Ne sont pas enregistrés dans une couche pour ne pas être enregistrés par un docker commit

Notre premier volume

```
$ docker run -d -v $(pwd):/var/www/html -P brahimhamdi/nginx:7.0  
3dea46a4744d62f6904eee8f200e902416624766137b999876e0114a4d76d527  
$ docker exec -it 3dea46a4744 ls /var/www/html  
Dockerfile index.html  
$ docker port 3dea46a4744 80  
0.0.0.0:32771
```



```
$ echo "Mise à jour du fichier index.html" > index.html
```



Création de volume nommé

```
$ docker volume create --name=logs  
logs
```

```
$ docker run -P -v logs:/var/log/nginx -d brahimhamdi/nginx:7.0  
5dd430f31331131319f6af1c4f1b3a4da53511d6d4bf5054d42af79b69a25c5a
```

```
$ docker run -it --volumes-from 5dd430f31 debian ls /var/log/nginx  
access.log  error.log
```

Notion de persistance

- Les volumes existent indépendamment des conteneurs.
- Si un conteneur est stoppé, ses volumes sont encore disponibles
- Vous êtes responsable de la gestion, de la sauvegarde des volumes

```
$ docker volume ls
```

DRIVER	VOLUME NAME
local	logs

- On peut monter ces volumes depuis un autre conteneur

Suppression des volumes

```
$ docker volume rm logs
```

Docker compose

- On veut coordonner des conteneurs
- On veut simplifier la gestion multi-conteneurs
- On ne veut pas utiliser de scripts shell complexes
- On veut une interface standardisée avec l'API Docker

Exemple : wordpress

```
$ docker run --name db -e MYSQL_ROOT_PASSWORD=secret -d mysql  
$ docker run --name wp -p 8080:80 --link db:mysql -d wordpress
```

- -e : permet le passage de paramètre
- -d expose un volume
- --link récupère le volume d'un autre container

Exemple : wordpress

The screenshot shows a web browser window for the WordPress setup configuration. The address bar displays "localhost:8080/wp-admin/setup-config.php". The main content area features the classic blue "W" WordPress logo. To the right, a vertical list of languages is presented in a dropdown menu. The "Français" option is highlighted with an orange rectangle, indicating it is selected. Other language options listed include Espanol de Puerto Rico, Espanol de Guatemala, Espanol de Uruguay, Espanol de Mexico, Espanol de Argentina, Espanol de Ecuador, Eesti, Euskara, فارسی, فارسی (افغانستان), Suomi, Francais de Belgique, Friulian, Gàidhlig, Galego, ગુજરાતી, هزاره گپ, עברית, हिन्दी, Hrvatski, Hornjoserbšćina, Magyar, Հայերեն, and Bahasa Indonesia. At the bottom right of the language list is a blue "Continuer" button.

WordPress › Setup Configuration

localhost:8080/wp-admin/setup-config.php

W

Espanol de Puerto Rico
Espanol de Guatemala
Espanol de Uruguay
Espanol de Mexico
Espanol de Argentina
Espanol de Ecuador
Eesti
Euskara
فارسی
فارسی (افغانستان)
Suomi
Francais de Belgique
Français
Francais du Canada
Friulian
Gàidhlig
Galego
ગુજરાતી
هزاره گپ
עברית
हिन्दी
Hrvatski
Hornjoserbšćina
Magyar
Հայերեն
Bahasa Indonesia

Continuer

Comment faire pour :

- Gérer les deux conteneurs à la volée?
- Gérer des volumes?
- Gérer des ports différents?
- Me souvenir de ces commandes?

- **Compose** permet d'éviter de gérer individuellement des conteneurs qui forment les différents services de votre application.
- Outil qui définit et exécute des applications multi-conteneurs
- Utilise un fichier de configuration dans lequel vous définissez les services de l'application
- A l'aide d'une simple commande, vous contrôlez le cycle de vie de tous les conteneurs qui exécutent les différents services de l'application.

Utilisation de docker compose

- Définir l'environnement de l'application pour qu'il soit possible de la générer de n'importe où.
 - à l'aide de Dockerfile
 - à l'aide d'image officielle
- Définir les services dans un fichier docker-compose.yml
- pour les exécuter et les isoler.
- Exécuter docker-compose qui se chargera d'exécuter l'ensemble de l'application

- Compose introduit une notion de service :
- Concrètement, un conteneur exécutant un processus
- Chaque conteneur exécute un service inter-dépendant
- Le service peut-être évolutif en lançant plus ou moins d'instances du conteneur avec Compose.
- Exemple Wordpress:
 - Service db
 - Service wordpress

Exemple de docker-compose.yml

```
version: '3'
services:
  wordpress:
    image: wordpress:4.9.8
    restart: always
    volumes:
      - ./wp-content:/var/www/html/wp-content
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_NAME: wpdb
      WORDPRESS_DB_USER: user
      WORDPRESS_DB_PASSWORD: password
    ports:
      - 8080:80
  db:
    image: mysql:8
    command: "--default-authentication-plugin=mysql_native_password"
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: wpdb
      MYSQL_USER: user
      MYSQL_PASSWORD: password
```

Démarrage de l'application

- *\$ docker-compose up -d*
- *Creating wordpress ... done*
- *Creating mysql ... done*
- Les différents services qui composent mon application ont été démarrés, avec la configuration et l'environnement qui va bien.

Informations sur l'application

- On utilise la commande ps deCompose:
- ```
$ docker-compose ps
```

| Name      | Command                        | State | Ports               |
|-----------|--------------------------------|-------|---------------------|
| mysql     | docker-entrypoint.sh --def ... | Up    | 3306/tcp, 33060/tcp |
| wordpress | docker-entrypoint.sh apach ... | Up    | 0.0.0.0:8080→80/tcp |
- ```
$ docker-compose ps wordpress
```

Name	Command	State	Ports
wordpress	docker-entrypoint.sh apach ...	Up	0.0.0.0:8080->80/tcp

Conteneurs classiques

- Les services s'exécutent via des conteneurs sur l'hôte. Les commandes docker classiques sont toujours fonctionnelles.
- `$ docker ps`
- | <i>CONTAINER ID</i> | <i>IMAGE NAMES</i> | <i>COMMAND</i> | <i>CREATED</i> | <i>STATUS</i> | <i>PORTS</i> |
|---------------------|------------------------|---------------------------------|----------------------|---------------------|--------------------------------|
| <i>70f081415426</i> | <i>wordpress:4.9.8</i> | <i>"docker-entrypoint.s..."</i> | <i>5 minutes ago</i> | <i>Up 3 minutes</i> | <i>0.0.0.0:8080->80/tcp</i> |
| <i>21cf410942c9</i> | <i>mysql:8</i> | <i>"docker-entrypoint.s..."</i> | <i>5 minutes ago</i> | <i>Up 4 minutes</i> | <i>3306/tcp, 33060/tcp</i> |

- Logs d'une application:
 - *\$ docker-compose logs*
 - *Attaching to wordpress, mysql*
 - ...
- Logs d'un service :
 - *\$ docker-compose logs db*
 - *Attaching to mysql*
 - ...

- On peut passer à l'échelle un service.
- Autrement dit, on peut augmenter/diminuer le nombre de conteneurs exécutant un service
- Par défaut, Compose exécute chaque service avec un conteneur.

- On utilise la commande scale pour changer le nombre de réplicas d'un service:
- ```
$ docker-compose scale db=2
```
- *Creating test\_db\_2 ... done*
- ```
$ docker-compose ps db
```
- | Name | Command | State | Ports |
|----------------|--------------------------------|-------|---------------------|
| wordpress_db_1 | docker-entrypoint.sh --def ... | Up | 3306/tcp, 33060/tcp |
| wordpress_db_2 | docker-entrypoint.sh --def ... | Up | 3306/tcp, 33060/tcp |

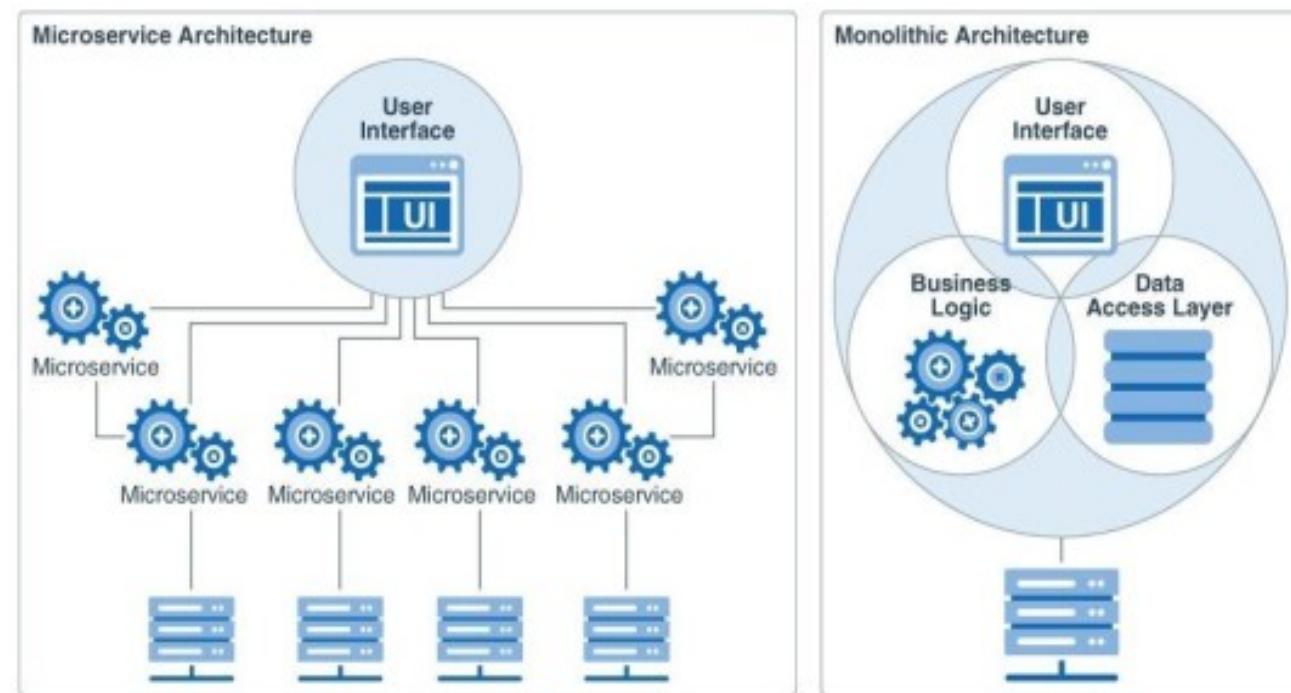
Module 4 : Kubernetes, mise en œuvre

- Introduction à Kubernetes
- Cluster Kubernetes
- Exploiter Kubernetes
- Kubernetes en production

Introduction à Kubernetes

Architecture Microservices

- Découpage de l'application en processus (services) indépendants
 - Chacun a sa propre responsabilité métier
 - Equipe dédiée pour chaque service
 - Plus de liberté de choix dans le langage
 - Mise à jour et scaling horizontale
 - Containers très adaptés pour
 - les microservices



- Si l'un des composants tombe en panne, cela n'impacte pas l'ensemble de l'application.
- Les microservices individuels et isolés sont beaucoup plus faciles et rapides à déployer.

Qu'est-ce que Kubernetes?

- Un système robuste d'orchestration de conteneurs.
- 100% Open source, écrit en Go.
- Crée par Google, basé sur Borg et Omega, les systèmes qui fonctionnent aujourd'hui chez Google et dont la robustesse est éprouvée depuis plus de 10 ans. Google exécute 2 milliards de conteneurs par semaine par semaine avec ces systèmes.
- Crée par trois employés de Google initialement pendant l'été 2014.
- Il a grandi de façon exponentielle et est devenu le premier projet à être donné à la CNCF.
- La première version de production: v1.0.1 en Juillet 2015.
- Une nouvelle version mineure tous les trois mois depuis la v1.2.0 en Mars 2016.

Quelques fonctionnalités de Kubernetes

- **Horizontal scaling and autoscaling:**

- Faites passer votre application à l'échelle à l'aide d'une simple commande, d'une interface graphique, ou automatiquement en fonction de l'utilisation du processeur ou de métriques personnalisées.

- **Service Discovery and Load Balancing:**

- Inutile de modifier votre application pour utiliser un mécanisme de découverte de service tiers.
- Kubernetes donne aux conteneurs leurs propres adresses IP et un seul nom DNS pour un ensemble de conteneurs, et peut équilibrer la charge entre eux.

- **Secret and configuration management:**

- Déployez et mettez à jour les secrets et la configuration de l'application sans reconstruire votre image et sans exposer les secrets les fichiers de configuration de votre déploiement.

Quelques fonctionnalités de Kubernetes

- **Automated rollouts and rollbacks:**

- Kubernetes déploie progressivement les modifications apportées à votre application ou à sa configuration, tout en surveillant l'intégrité de l'application afin de s'assurer qu'elle ne tue pas toutes vos instances en même temps. En cas de problème, Kubernetes annulera les changements pour vous.

- **Storage Orchestration:**

- monte automatiquement le système de stockage de votre choix, qu'il s'agisse du stockage local, d'un fournisseur de cloud public tel que GCP ou AWS, ou d'un système de stockage réseau tel que NFS, iSCSI, Gluster, Ceph, Cinder ou Flocker

Cluster Kubernetes

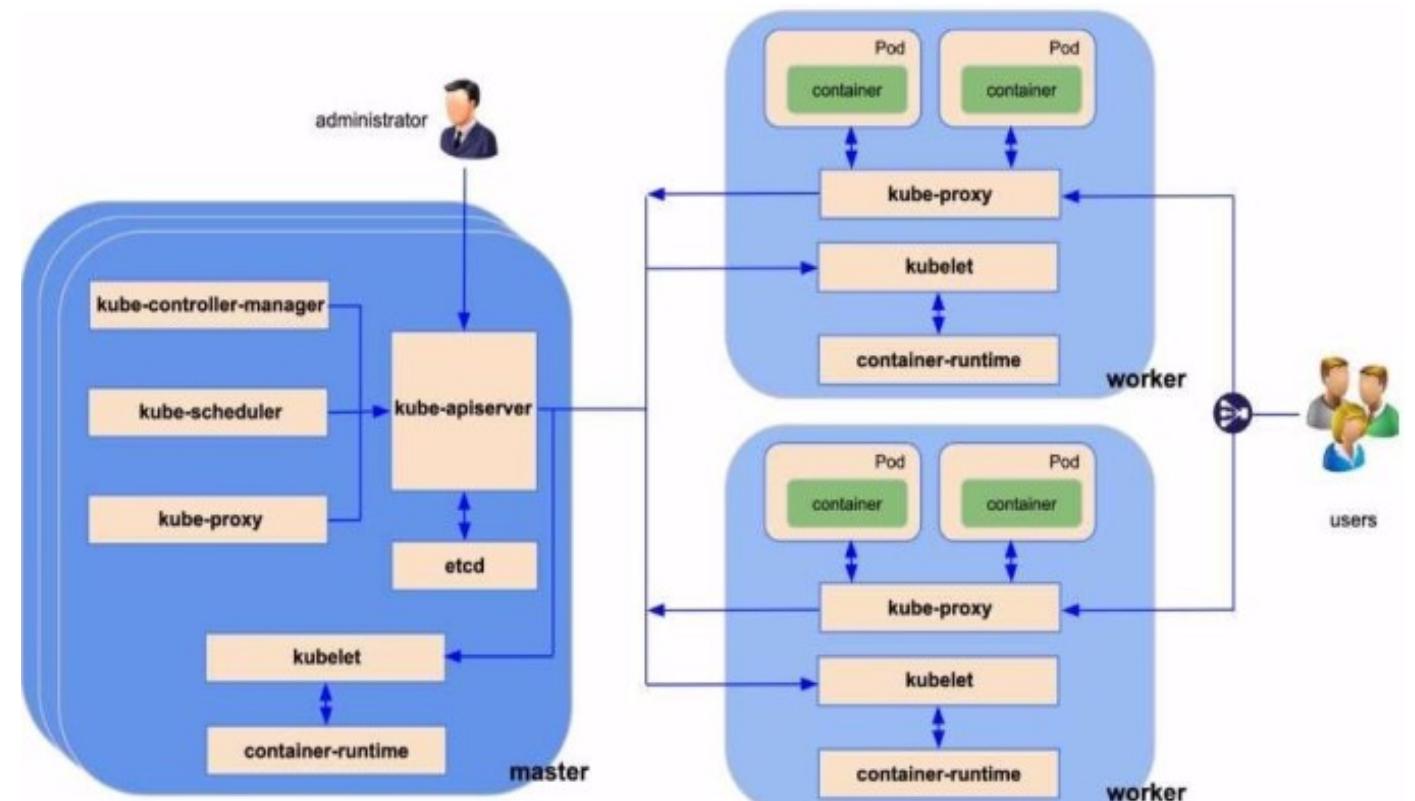
Les différents types de nodes

■ Master:

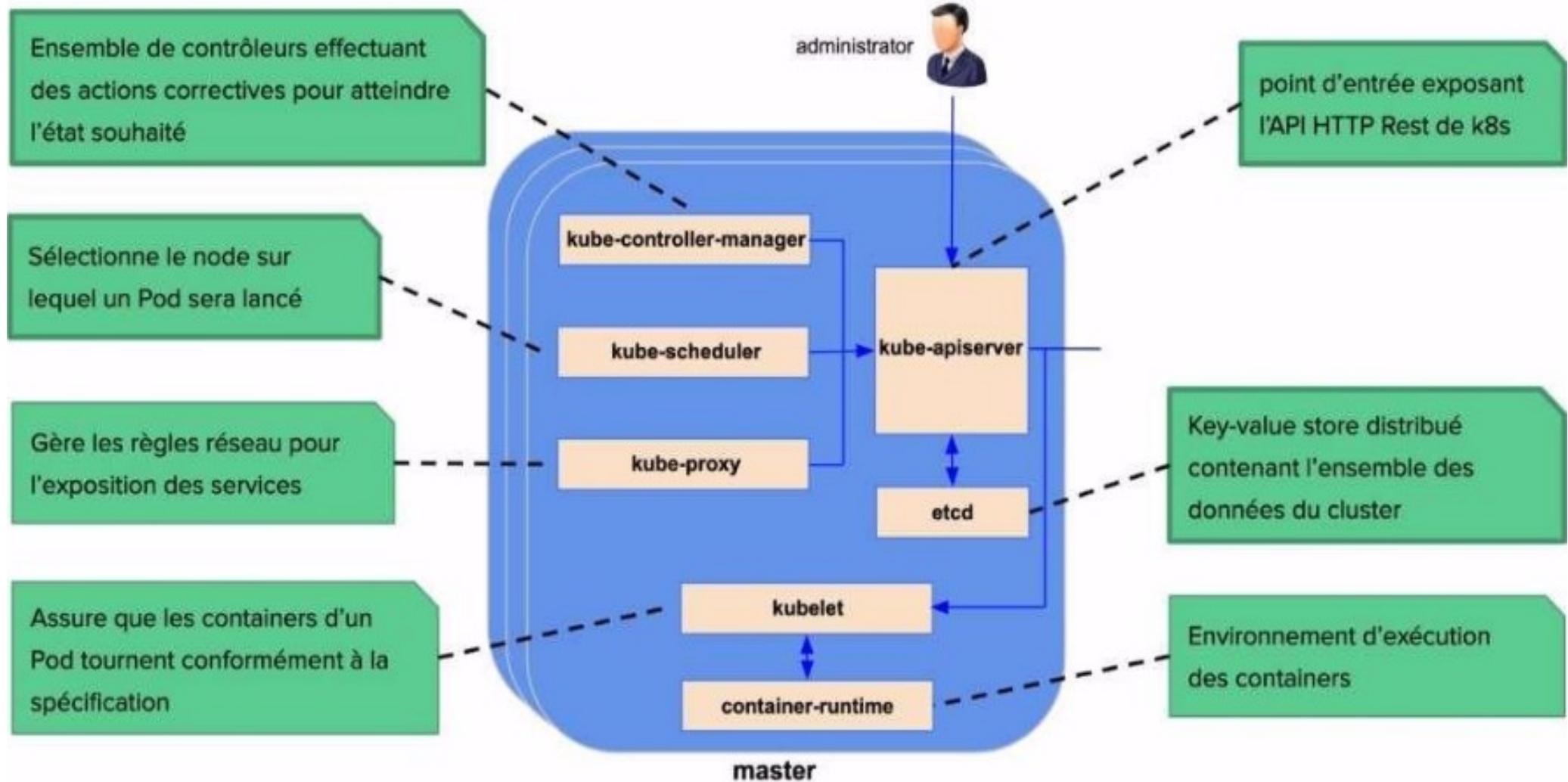
- Responsable de la gestion du cluster ("control plane")
- Expose l'API server
- Schedule les Pods sur les nodes du cluster

■ Worker/Node :

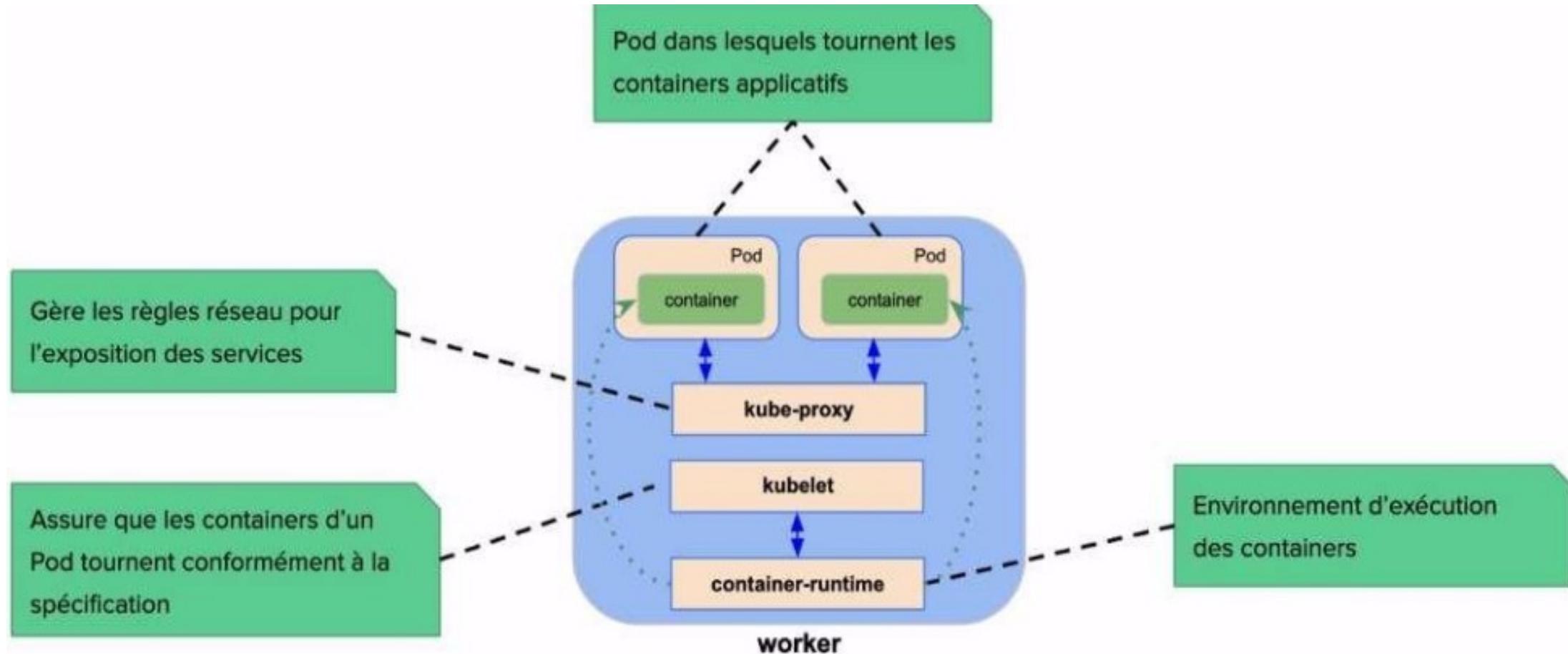
- Node sur lequel sont lancés les Pods applicatifs
- Communique avec le Master
- Fournit les ressources
- aux Pods



Composants du master



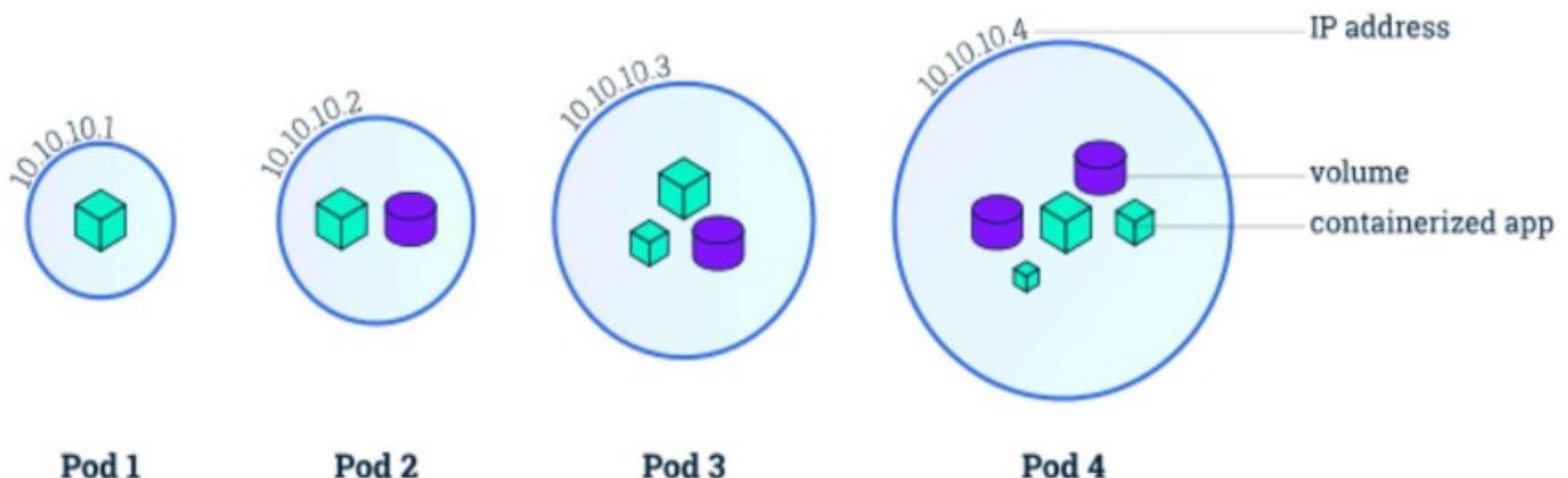
Composants du worker/node



Exploiter Kubernetes

- Kubernetes contient un certain nombre d'abstractions représentant l'état de votre système: applications et processus conteneurisés déployés, leurs ressources réseau et disque associées, ainsi que d'autres informations sur les activités de votre cluster.
- Les contrôleurs s'appuient sur les objets de base et fournissent des fonctionnalités supplémentaires. Ces objets de base de Kubernetes incluent: Pod, Service, Volume et Namespace.
- Autres objets de Kubernetes:
 - ReplicaSet
 - Deployment
 - StatefulSet
 - DaemonSet
 - Job
 - Cron job

- Le Pod représente la plus petite unité déployable et exécutable au sein d'un cluster.
- C'est la brique de base qui encapsule un ou plusieurs conteneurs ayant une seul et unique @IP.
- Il a son propre réseau et volume interne.



Pod 1

Pod 2

Pod 3

Pod 4

Pod avec plusieurs containers

```
$ cat wp-pod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: wp
spec:
  containers:
    - image: wordpress:4.9-apache
      name: wordpress
      env:
        - name: WORDPRESS_DB_PASSWORD
          value: mysqlpwd
        - name: WORDPRESS_DB_HOST
          value: 127.0.0.1
    - image: mysql:5.7
      name: mysql
      env:
        - name: MYSQL_ROOT_PASSWORD
          value: mysqlpwd
  volumeMounts:
    - name: data
      mountPath: /var/lib/mysql
  volumes:
    - name: data
      emptyDir: {}
```

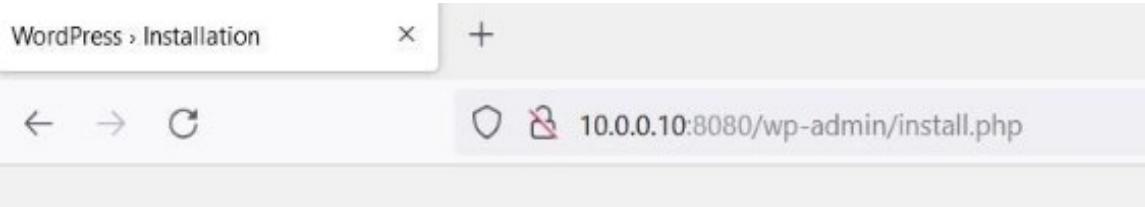
```
$ kubectl apply -f wp-pod.yaml
```

Pod avec plusieurs containers

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
wp	2/2	Running	0	18s

```
$ kubectl port-forward --address 0.0.0.0 wp 8080:80
```

A screenshot of the WordPress installation welcome screen. It features the classic blue "W" logo at the top. Below it, the word "Welcome" is centered. A paragraph of text explains the purpose of the installation process. Under the heading "Information needed", there are three input fields: "Site Title" (empty), "Username" (empty), and "Password" containing the value "uJp3The4UUNiclate#*". A green button labeled "Strong" indicates the password is secure. A note at the bottom states: "Important: You will need this password to log in. Please store it in a secure location."

- Un Deployment contrôle et pilote des ReplicaSets et des pods.
- Il ajoute des fonctionnalités aux ReplicaSet comme :
 - Déployer (rollout) des ReplicaSet : les ReplicaSets vont créer des pods en tâche de fond et le Deployment va s'assurer que le déploiement (rollout) se passe bien
 - Déclarer un nouvel état des pods : le Deployment va piloter la mise à jour des pods, des anciens ReplicaSet vers les nouveaux. Ce déploiement va créer une nouvelle révision du Deployment
 - Revenir à une ancienne version du Deployment (rollback) : si le déploiement engendre des instabilités, le Deployment nous permet de revenir à une précédente révision.
 - Mettre à l'échelle le Deployment pour gérer les montées en charge par exemple
 - Mettre en pause un déploiement : pour fixer des éventuelles erreurs et reprendre le rollout

Le Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deploy
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: guestbook
  tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/g
```

Deployment

ReplicaSet

Pod

frontend:v3

Le Deployment

```
# Lancement du Deployment  
$ kubectl create -f vote-deployment.yaml  
# Liste des Deployments  
$ kubectl get deploy
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
vote-deploy	3/3	3	3	11m

```
# Liste des ReplicaSet  
$ kubectl get rs
```

NAME	DESIRED	CURRENT	READY	AGE
vote-deploy-7c4d84d659	3	3	3	15m

```
$ kubectl get pod
```

NAME	READY	STATUS	RESTARTS	AGE
vote-deploy-7c4d84d659-4jblj	1/1	Running	0	18m
vote-deploy-7c4d84d659-95nwp	1/1	Running	0	18m
vote-deploy-7c4d84d659-mcd8x	1/1	Running	0	18m

Un ReplicaSet est créé et associé au Deployment



Le ReplicaSet gère les 3 Pods (replicas) définis dans la spécification du Deployment



Le Deployment

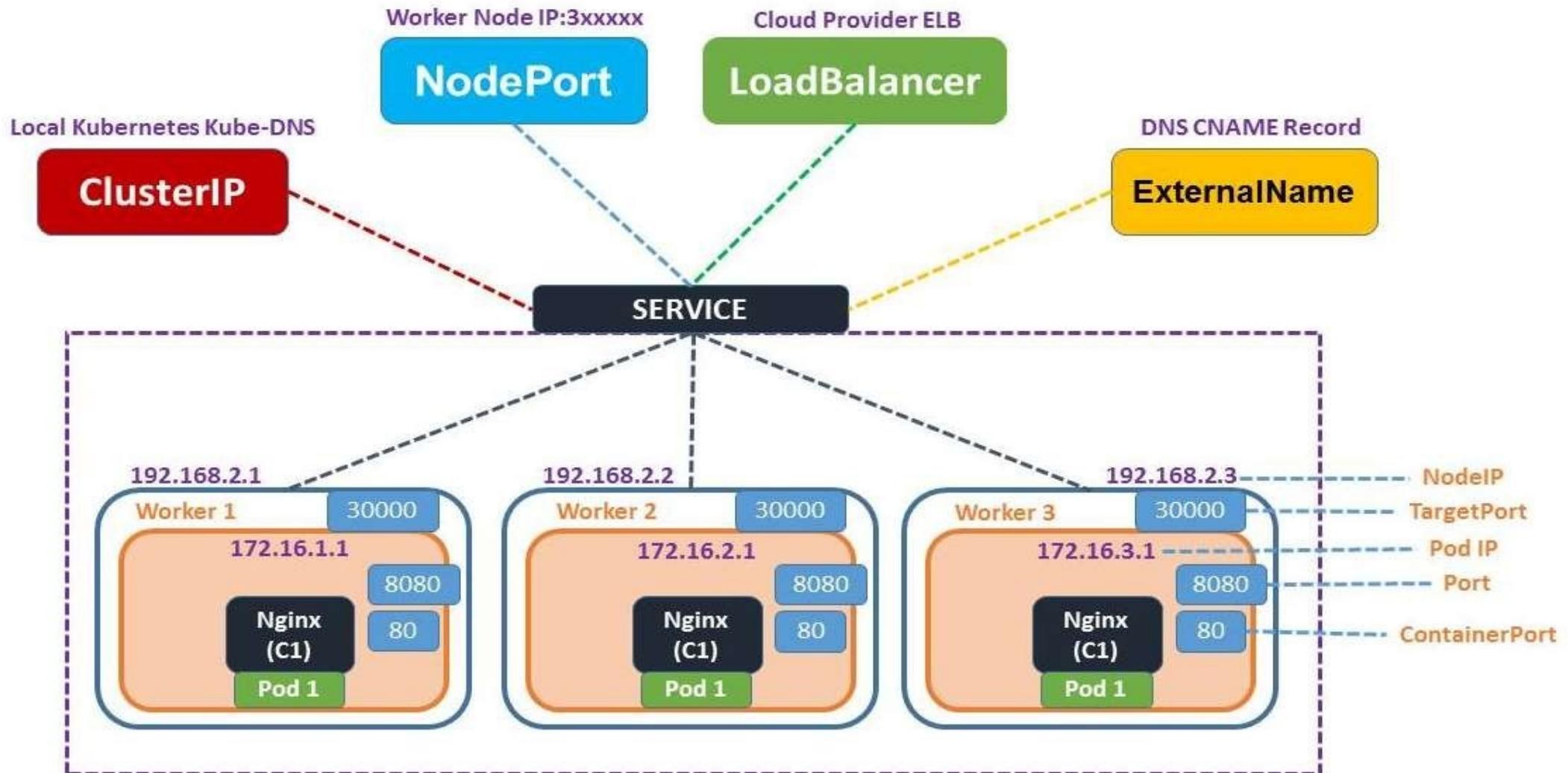
```
#Scaler les répliques de vos Pods  
$ kubectl scale --replicas=4 deploy vote-deploy  
# Détails du Deploy  
$ kubectl describe deploy vote-deploy
```

```
Name:          vote-deploy  
...  
NewReplicaSet: vote-deploy-7c4d84d659 (4/4 replicas  
Events:        created)  
Type  Reason           Age    From            Message  
----  ----           ----  --            -----  
Normal ScalingReplicaSet 2m49s deployment-controller Scaled up replica set vote-deploy-7c4d84d659 to 3  
Normal ScalingReplicaSet 70s  deployment-controller Scaled up replica set vote-deploy-7c4d84d659 to 4
```

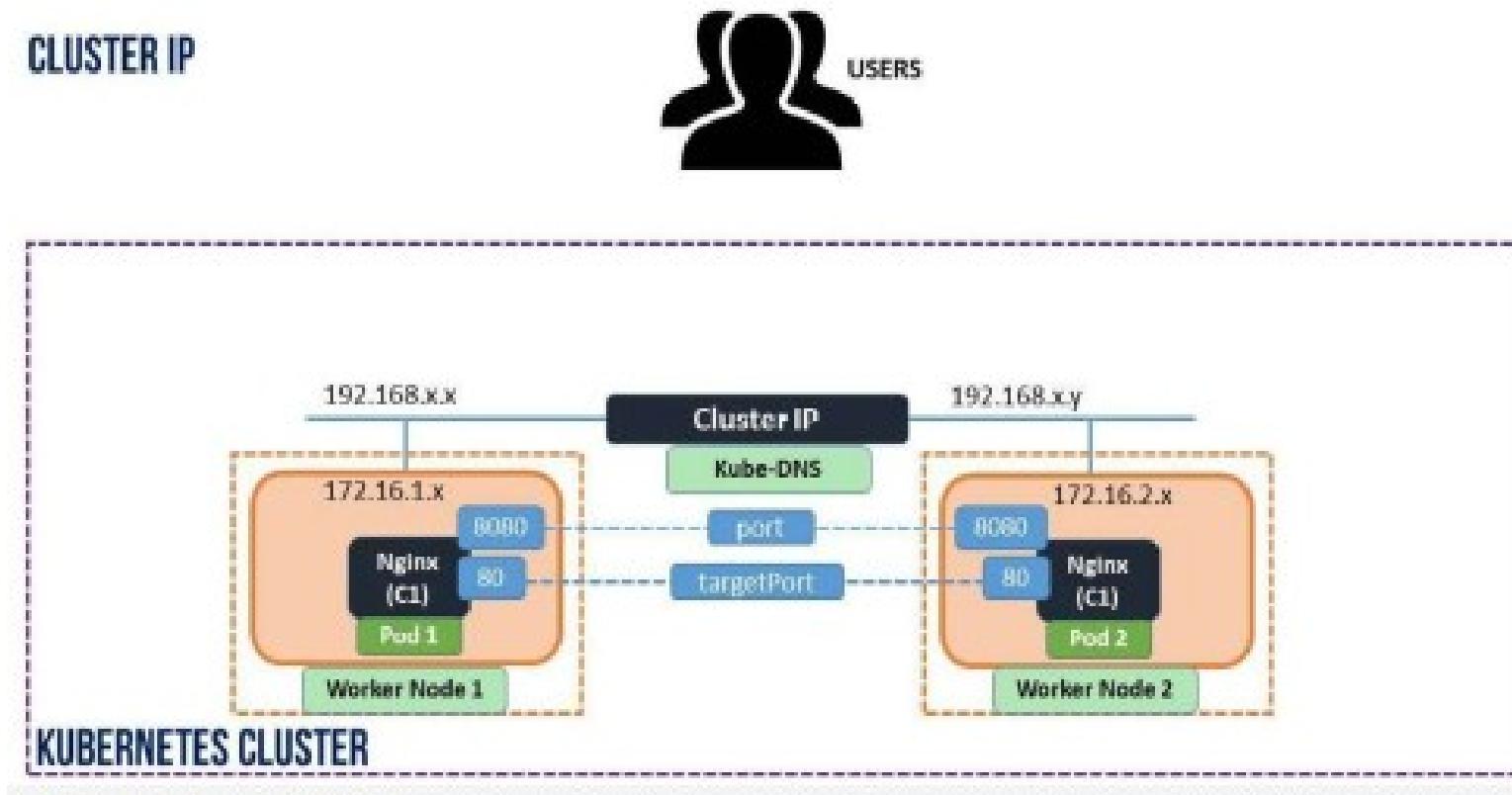
```
# Changer la valeur de replicas par 2  
$ kubectl apply -f vote-deployment.yaml
```

```
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: frontend-deploy  
spec:  
  replicas: 2  
...
```

Les services



- Expose le service sur une IP interne au cluster.
- Le choix de cette valeur rend le service uniquement accessible à partir du cluster.
- Il s'agit du ServiceType par défaut.





```
$ cat vote-pod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: web
  labels:
    app: web
spec:
  containers:
    - name: web
      image: nginx
      ports:
        - containerPort: 80
```

```
$ cat vote-service-clusterIP.yaml
```

```
apiVersion: v1
kind: Service
metadata:
  name: web-service
spec:
  selector:
    app: web
  type: ClusterIP
  ports:
    - port: 80
      targetPort: 80
```

💡 Chaque requête reçue par le service est envoyée sur l'un des Pods ayant le label spécifié

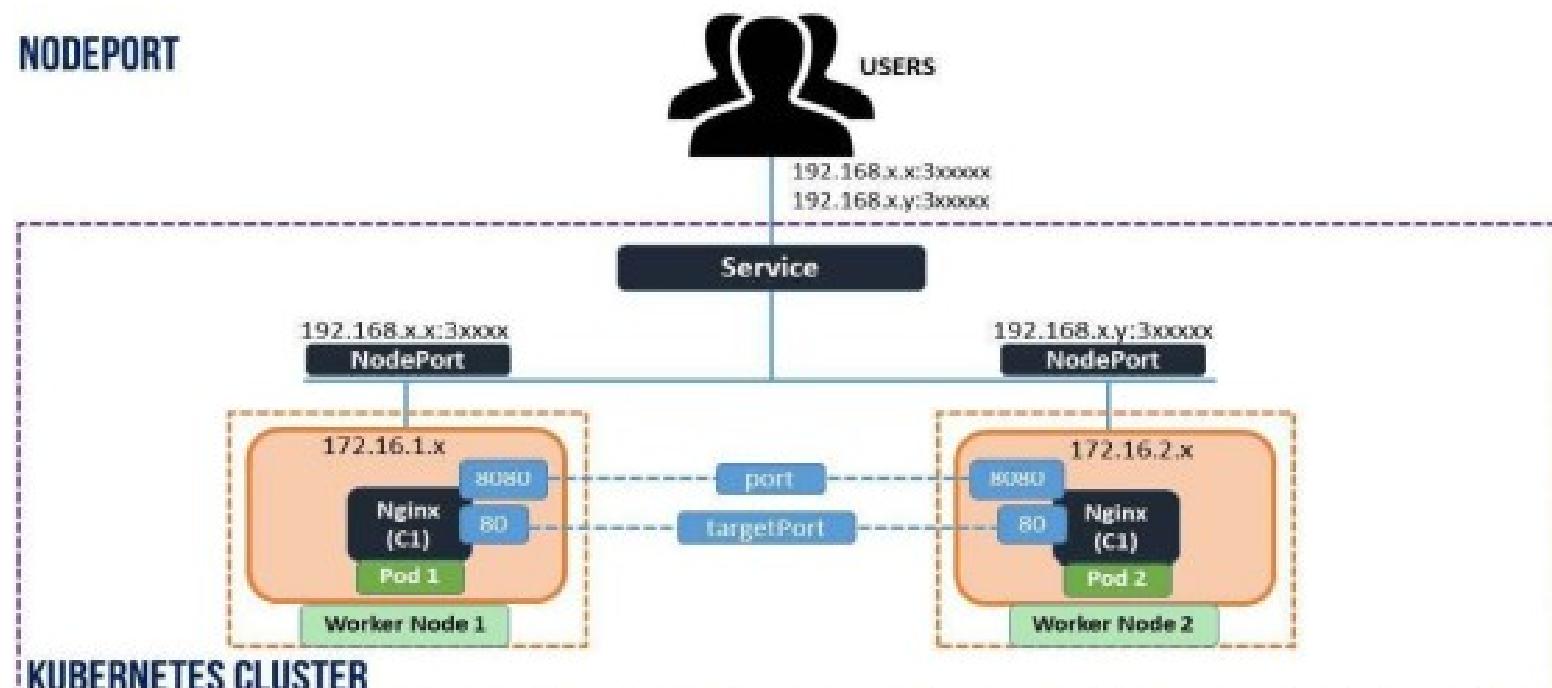
Les services : ClusterIP

```
# Lancement du Pod vote
$ kubectl create -f vote-pod.yaml
# Lancement du Service de type ClusterIP
$ kubectl create -f vote-service-clusterIP.yaml
# Lancement d'un Pod utilisé pour le debug
$ kubectl create -f pod-debug.yaml
# Accès au Service vote depuis le Pod debug
$ kubectl exec debug -- sh
/ # apk update && apk add curl
/ # curl http://web-service
(code html de l'interface vote)
...
```

```
$ cat pod-debug.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
  name: debug
spec:
  containers:
    - name: debug
      image: rancher/curl
      command:
        - "sleep"
        - "10000"
```

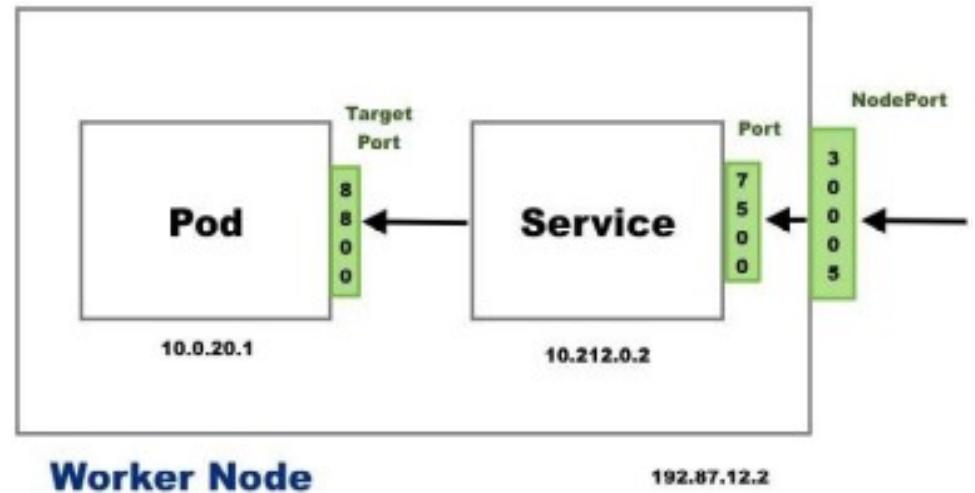
- Chaque nœud du cluster **ouvre un port statique** et redirige le trafic vers le port indiqué via le contrôleur Kube-proxy.
- Le nœud maître du Kubernetes alloue un port statique à partir d'une plage, et chaque nœud va proxy ce port (le même numéro de port sur chaque nœud) pour notre service. Cela se fait via iptables.



Les services : NodePort

- Spécification d'un Service nommé vote-service de type NodePort.
 - Kubernetes alloue un port à partir d'une plage spécifiée: 30000-32767
- ```
$ cat vote-service-nodePort.yaml
```

```
apiVersion: v1
kind: Service
metadata:
 name: web-srv
spec:
 selector:
 app: web
 type: NodePort
 ports:
 - port: 80
 targetPort: 80
 nodePort: 31000
```



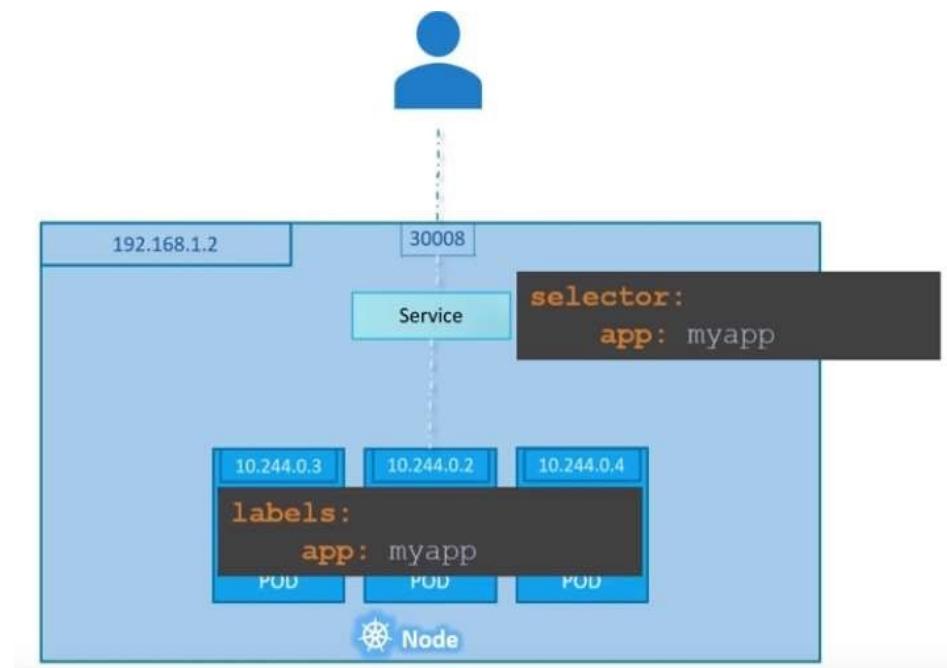
- Il est possible d'exposer le déploiement vote-deploy avec le service vote-service de type NodePort.

# Lancement du Service

```
$ kubectl create -f vote-service-nodePort.yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: web-deploy
spec:
 replicas: 3
 selector:
 matchLabels:
 app: web
 template:
 metadata:
 labels:
 app: web
...
...

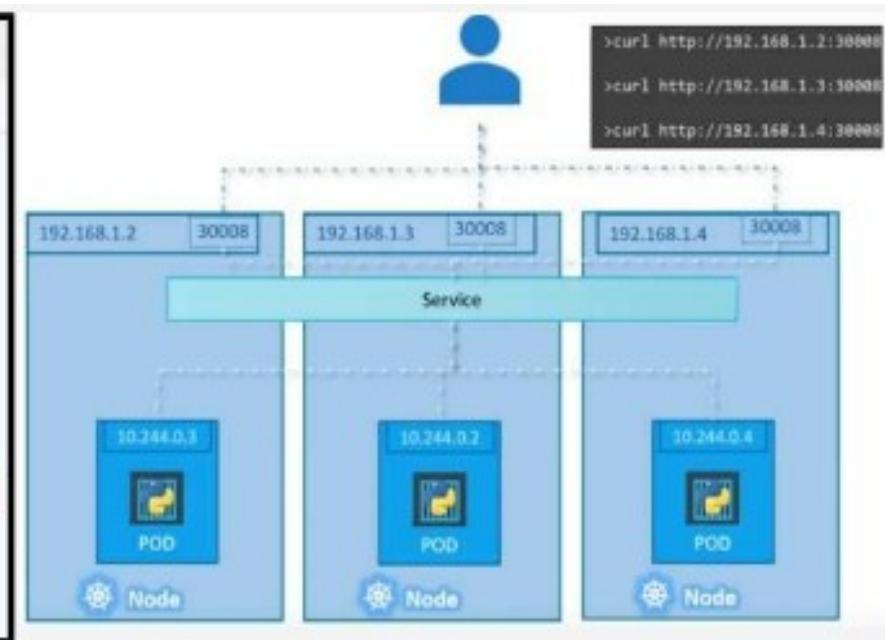
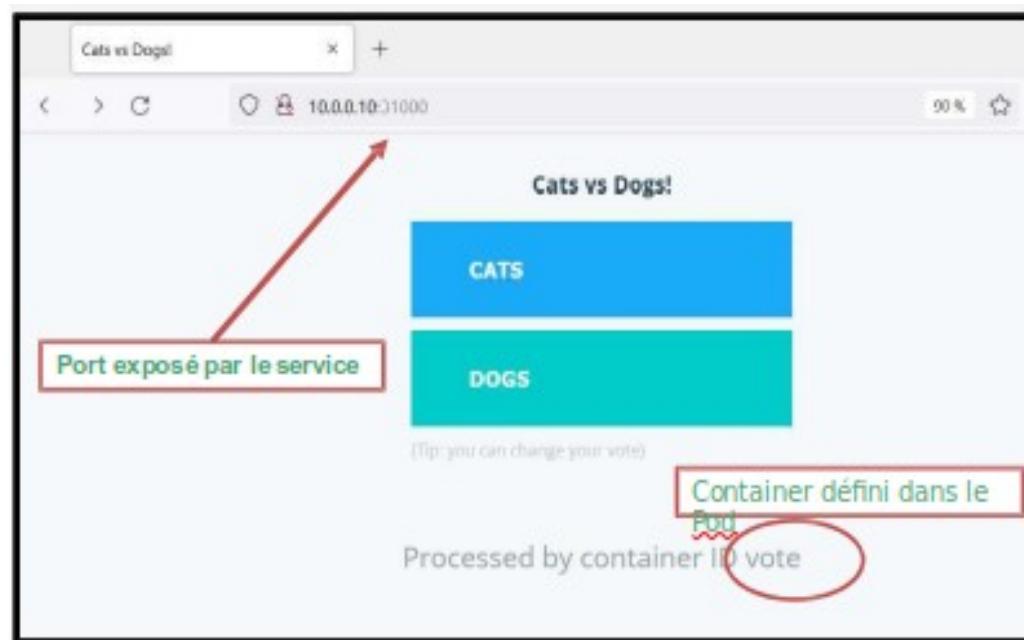
apiVersion: v1
kind: Service
metadata:
 name: web-srv
spec:
 selector:
 app: web
 type: NodePort
 ports:
 - port: 80
 targetPort: 80
 nodePort: 31000
```



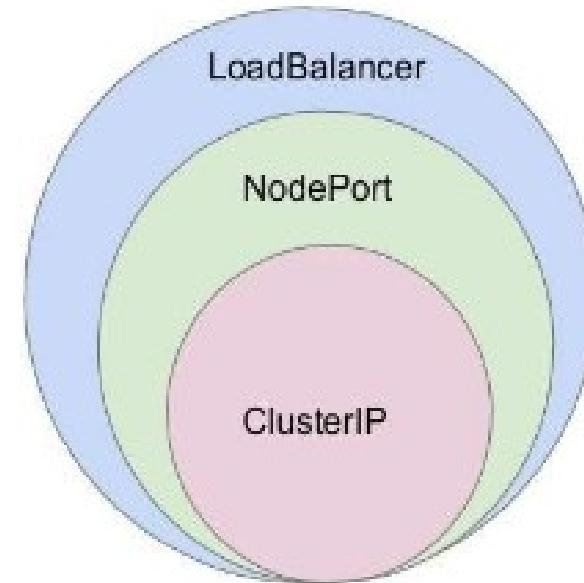
- Il est possible d'exposer le déploiement vote-deploy avec le service vote-service de type NodePort.

# Lancement du Service

```
$ kubectl create -f vote-service-nodePort.yaml
```



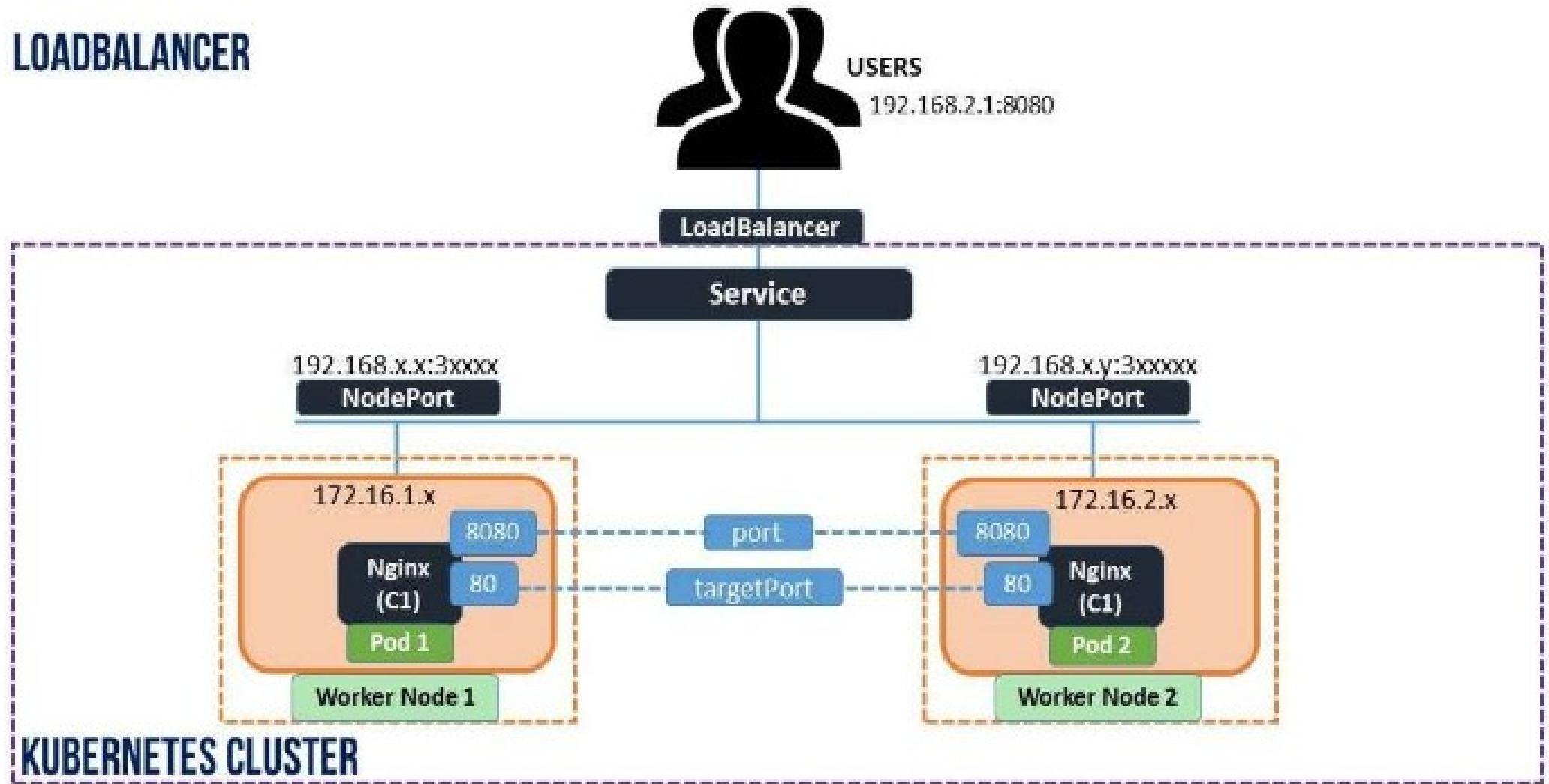
- Expose le service en externe à l'aide de l'équilibrEUR de charge d'un fournisseur de cloud.
- Ainsi, les services NodePort et ClusterIP sont créés automatiquement et sont acheminés par l'équilibrEUR de charge externe.
- Exemples de cloud provider:
  - AWS ELB/ALB/NLB
  - GCP LoadBalancer
  - Azure Balancer
  - OpenStack Octavia



<https://kubernetes.io/fr/docs/concepts/services-networking/service/#loadbalancer>

# Les services : LoadBalancer

LOADBALANCER



# Les services : LoadBalancer

- Il est possible d'exposer le déploiement vote-deploy avec le service vote-service de type LoadBalancer

```
$ cat vote-service-LoadBalancer.yaml
```

```
apiVersion: v1
kind: Service
metadata:
 name: web-srv
spec:
 selector:
 app: web
 type: LoadBalancer
 ports:
 - port: 80
 targetPort: 80
 nodePort: 32000
```

```
Lancement du Deploy vote
$ kubectl create -f vote-deployment.yaml

Lancement du Service de type LoadBalancer
$ kubectl create -f vote-service-LoadBalancer.yaml

Affichage de service de type LoadBalancer
$ kubectl get svc
```

| NAME         | TYPE         | CLUSTER-IP  | EXTERNAL-IP |
|--------------|--------------|-------------|-------------|
| kubernetes   | ClusterIP    | 10.96.0.1   | <none>      |
| vote-service | LoadBalancer | 10.97.46.31 | <pending>   |

- Notez le <pending> : cela veut dire que notre service n'a pas d'IP externe. (En gros : il ne sert à rien. Il lui faut une IP pour que nous puissions l'utiliser.)

---

# Kubernetes en production

---

# Les stratégies de déploiement

- Il existe plusieurs stratégies de déploiement dans Kubernetes, les plus connues:
  - Recreate: native
  - RollingUpdate (Ramped): native
  - Blue/Green: étape supplémentaire nécessaire
  - Canary: étape supplémentaire nécessaire

| Strategy                                                                                                  | ZERO DOWNTIME | REAL TRAFFIC TESTING | TARGETED USERS | CLOUD COST | ROLLBACK DURATION | NEGATIVE IMPACT ON USER | COMPLEXITY OF SETUP |
|-----------------------------------------------------------------------------------------------------------|---------------|----------------------|----------------|------------|-------------------|-------------------------|---------------------|
| <b>RECREATE</b><br>version A is terminated then version B is rolled out                                   | ✗             | ✗                    | ✗              | ■□□        | ■■■               | ■■■                     | □□□                 |
| <b>RAMPED</b><br>version B is slowly rolled out and replacing version A                                   | ✓             | ✗                    | ✗              | ■□□        | ■■■               | ■□□                     | ■□□                 |
| <b>BLUE/GREEN</b><br>version B is released alongside version A, then the traffic is switched to version B | ✓             | ✗                    | ✗              | ■■■        | □□□               | ■□□                     | ■□□                 |
| <b>CANARY</b><br>version B is released to a subset of users, then proceed to a full rollout               | ✓             | ✓                    | ✗              | ■□□        | ■□□               | ■□□                     | ■□□                 |

<https://github.com/ContainerSolutions/k8s-deployment-strategies>

- Un namespace est un moyen de séparer logiquement un cluster physique en plusieurs clusters virtuels
- On peut ainsi faire un découpage par projet et procéder à une gestion des droits plus fine
- Les utilisateurs Kubernetes ne peuvent avoir accès qu'à un seul namespace , avec des quotas définis en terme de nombre de pods , de CPU, de mémoire...
- Un namespace par défaut existe et est utilisé quand on ne renseigne pas le namespace utilisé, il s'appelle « default »

# Les namespaces : création

```
Création du namespaces development (option 1)
$ kubectl create namespace development
namespace "development" created

Liste des Namespaces
$ kubectl get namespace
Suppression du namespace
$ kubectl delete namespace/development
namespace "development" deleted

Création du namespace development (option 2)
$ cat production-ns.yaml
```

\$ kubectl create -f production-ns.yaml

```
namespace "development" created
```



```
apiVersion: v1
kind: Namespace
metadata:
 name: production
 labels:
 name: production
```

# Les namespaces : utilisation

```
$ cat nginx-pod-prod.yaml
```

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx
 namespace: production
spec:
 containers:
 - name: www
 image: nginx:1.12.2
```

```
Lancement d'un Pod dans le namespace development
$ kubectl create -f nginx-pod-prod.yaml
pod "nginx" created
```

# Les namespaces : utilisation

```
Liste des Pods dans le namespace default
$ kubectl get po
No resources found in default namespace.

Liste des Pods dans le namespace production
$ kubectl get po --namespace=production

NAME READ STATUS RESTART AGE
nginx 1/1 Running 0 2m
x 4s es namespaces

$ kubectl get po --all-namespaces
```

| NAMESPACE   | NAME                       | READ | STATUS  | RESTARTS | AGE   |
|-------------|----------------------------|------|---------|----------|-------|
| kube-system | calico-node-chjst          | 1/1  | Running | 1        | 31h   |
| kube-system | coredns-74ff55c5b-flgk8    | 1/1  | Running | 1        | 31h   |
| kube-system | etcd-master-node           | 1/1  | Running | 1        | 31h   |
| kube-system | kube-apiserver-master-node | 1/1  | Running | 1        | 31h   |
| kube-system | kube-proxy-dtr6t           | 1/1  | Running | 1        | 31h   |
| kube-system | kube-scheduler-master-node | 1/1  | Running | 1        | 31h   |
| production  | nginx                      | 1/1  | Running | 0        | 3m37s |

# Les namespaces : utilisation

```
Création d'un Deployment dans le namespace production
$ kubectl create deployment www --image nginx:1.12.2 --namespace production --replicas=2
deployment.apps/www created
Liste des Deployments dans le namespace production
$ kubectl get deploy --namespace production
```

|   | NAM<br>E | READ<br>Y | UP-TO-<br>DATE | AVAILABL<br>E | AG<br>E          |
|---|----------|-----------|----------------|---------------|------------------|
| # | www      | 2/2       | 2              | 2             | 82<br>s oduction |

```
$ kubectl get po --namespace production
```

|    | NAME                     | READ<br>Y | STATUS      | RESTART<br>S | AGE   |
|----|--------------------------|-----------|-------------|--------------|-------|
|    | nginx                    | 1/1       | Runnin<br>g | 0            | 22m   |
| #  | www-744cd6f665-<br>9fl7l | 1/1       | Runnin<br>g | 0            | 2m57s |
| \$ | www-744cd6f665-<br>gxgln | 1/1       | Runnin<br>g | 0            | 2m57s |

# Module 5 : Prometheus monitoring

---

- La supervision
- Prometheus
- Les Exporters et les Clients library
- Dashboarding avec Grafana

# La supervision

# La surveillance informatique

---

- La disponibilité des serveurs, réseaux, logiciels, processus, switchs, routeurs, parefeux, onduleurs, connexions internet, bornes wifi...
- La disponibilité des ressources d'un système, telles que l'espace disque, la mémoire, le CPU,
- La santé des équipements (postes de travail, téléphonie, ...),
- La performance, par exemple les temps de réponse d'une application, le débit réseau, la température d'une salle,
- La fiabilité / qualité, par analyse de la disponibilité sur une période,
- La sécurité, pour prévenir des attaques, (vol de données confidentielles clients ou internes, ransomware, virus...)

# La surveillance informatique

---

- La détection au plus tôt d'un dysfonctionnement, ou d'un risque, conduit à une résolution plus rapide, et limite les impacts pour l'entreprise et ses utilisateurs.
- Les mesures sont prises en amont, l'environnement est sécurisé et l'efficacité est améliorée.
- La supervision est le moyen de surveiller infrastructure et réseaux pour se prémunir efficacement des pannes.
- Ces données mettent en lumière les risques potentiels pour lesquels l'anticipation doit être l'assurance d'une exploitation optimale du SI.

# Prometheurs

# Qu'est ce que Prometheus ?

---

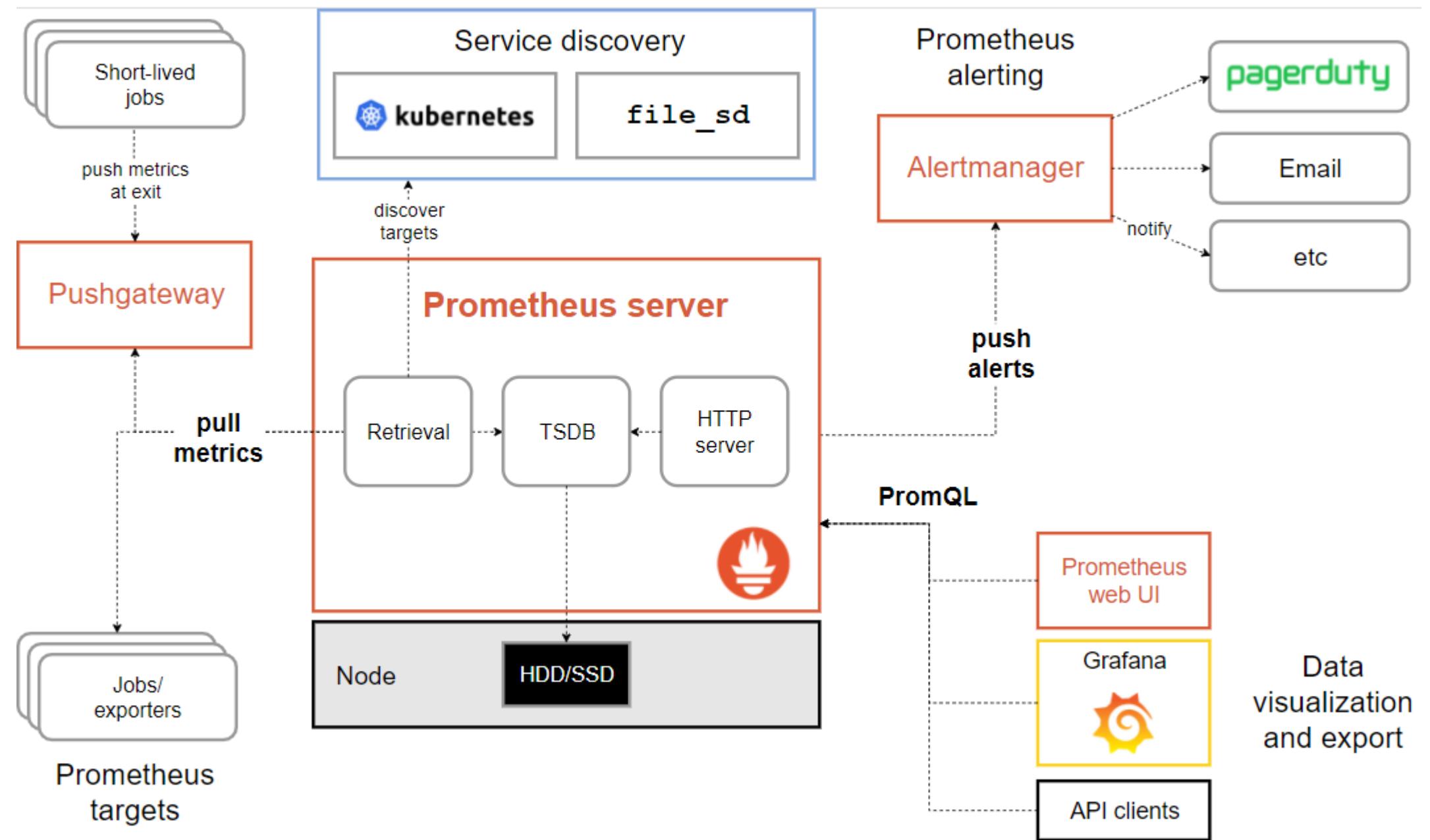
- Prometheus est une boîte à outils de surveillance et d'alerte de systèmes open source créée à l'origine chez SoundCloud.
- Prometheus doit son inspiration au Borgmon de Google.
- Il a été développé à l'origine par Matt T.Proud, un ancien SRE de Google
- Accepté en 2016 comme deuxième projet de la CNCF (Cloud Native Computing Foundation)

# Qu'est ce que Prometheus ?

---

- Depuis sa création en 2012, de nombreuses entreprises et organisations ont adopté Prometheus, et le projet a une communauté de développeurs et d'utilisateurs très active.
- Il s'agit désormais d'un projet open source autonome et géré indépendamment de toute entreprise.

# Architecture Prometheus



# Les Exporters et les Client Library

# Client Librairies

---

- Pour les applications propriétaires ou Open Source
- Quelqu'un doit introduire le code source qui produit leurs métriques
- Le projet Prometheus fournit des bibliothèques clientes officielles pour les langages Go, Python, Java et Ruby
- Il existe également une variété de bibliothèques clientes sous forme de tierce partie, pour d'autres langages tels que C#/.Net, Node.js, Haskell, Erlang et Rust.

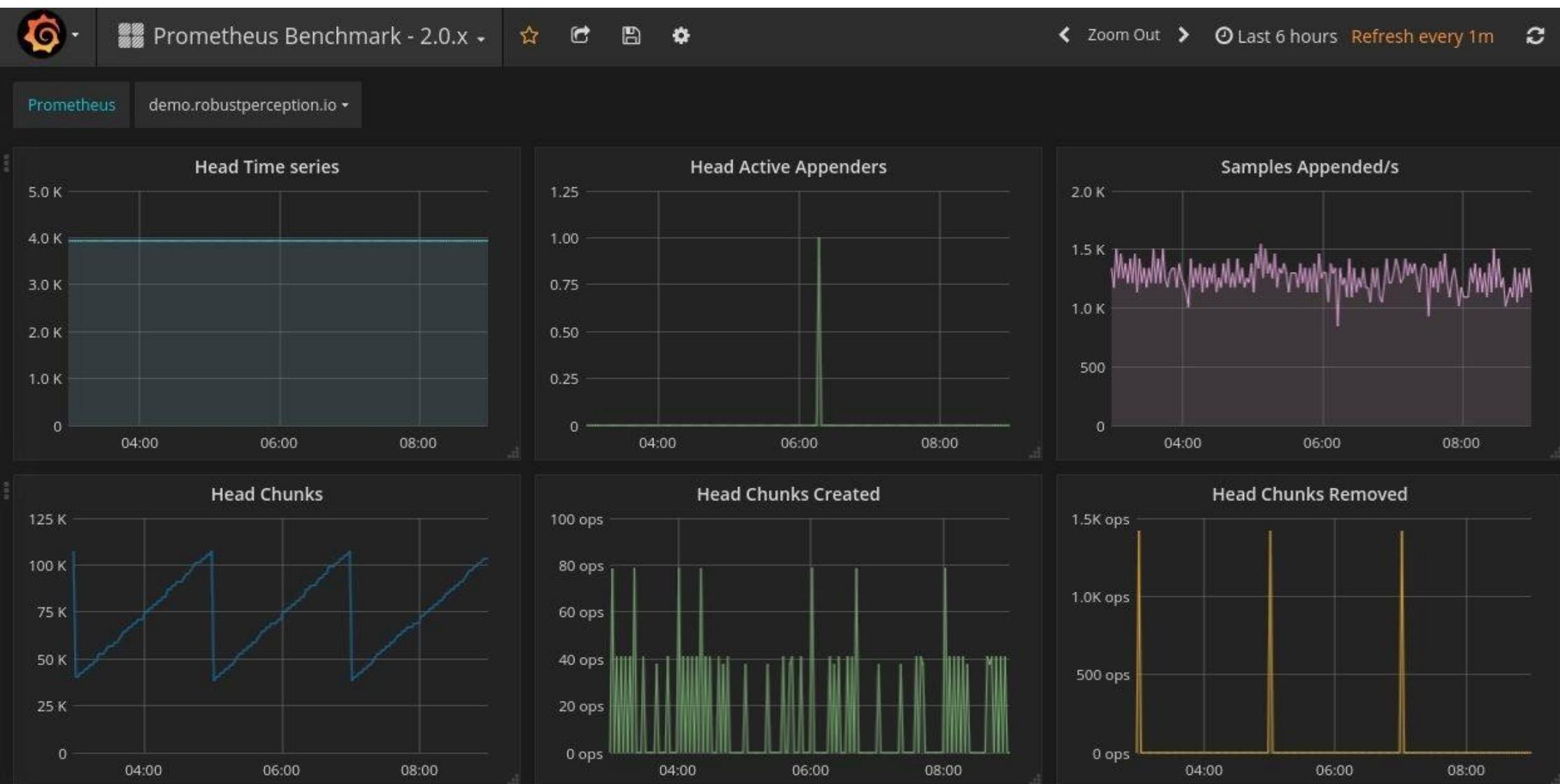
- Une liste statique de ressources fournie par l'utilisateur.
- Découverte basée sur des fichiers
  - par exemple, en utilisant un outil de gestion de configuration pour générer une liste de ressources qui sont automatiquement mises à jour dans Prometheus.
- Découverte automatisée
  - par exemple, interroger un repo de données comme Consul, exécuter instances dans Amazon ou Google, ou en utilisant des enregistrements DNS SRV pour générer une liste de ressources.

- La découverte et le réétiquetage des services nous donnent une liste de cibles à surveiller.
- Maintenant, Prometheus doit récupérer les métriques.
- Prometheus le fait en envoyant une requête HTTP appelée scrape.
- La réponse au scrape est analysée et ingérée dans le stockage.
- Plusieurs mesures utiles sont également ajoutées, comme si le scraping a réussi et combien de temps cela a pris.
- Les time-series se produisent régulièrement; en général, vous le configurez pour qu'il se produise toutes les 10 à 60 secondes pour chaque cible.

# Dashboarding avec Grafana

- API HTTP qui fournit des données brutes et évalue des requêtes PromQL.
- PromQL est utilisé pour produire des graphiques et des tableaux de bord.
- Expression Browser pour l'interrogation ad-hoc et à l'exploration de données.
- Il est recommandé d'utiliser Grafana pour les tableaux de bord.

# Dashboards



# Module 6 : Gitlab CI/CD - Fonctionnement

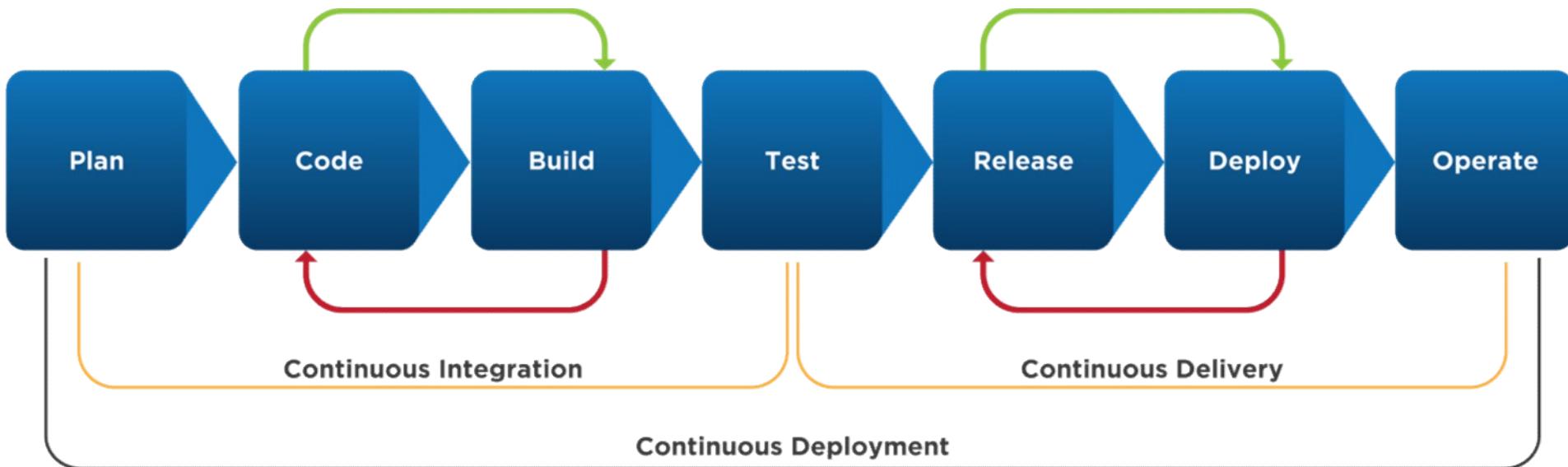
---

- Pipeline CI/CD
- Continuous Integration
- Continuous Delivery
- Continuous Deployment
- Gitlab Serveur et Runner
- Le Runner
- Fonctionnement de Gitlab CI/CD

# Pipeline CI/CD

# Pipeline

- La gestion de la pipeline CD/CD dans GitLab se fait simplement par l'ajout d'un fichier YAML « `.gitlabci.yml` » dans la racine de votre projet git concerné .
- YAML est complet, très largement utilisé et il fonctionne pour tout type d'environnement de développement (exemple Ansible Playbooks, Docker-compose ... )

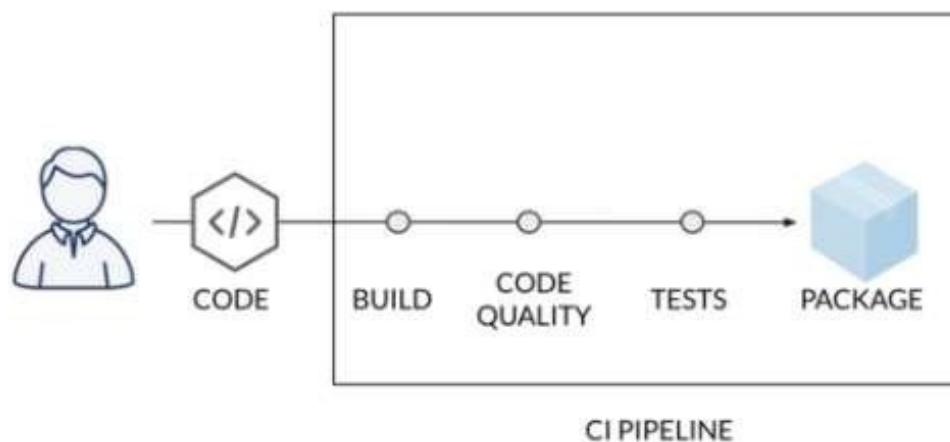


| Planning                                                                                                                                                                                           | Code                                                                                                                                                                                                | Build                                                                                                                                                                                                                                              | Test                                                                                                                                                                                            | Release                                                                                                                                    | Deploy                                                                                                                                                 | Operate                                                                                                                                                                |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>Requirement finalization</li><li>Updates &amp; new changes</li><li>Architecture &amp; design</li><li>Task assignment</li><li>Timeline finalization</li></ul> | <ul style="list-style-type: none"><li>Development</li><li>Configuration finalization</li><li>Check-in source code</li><li>Static-code analysis</li><li>Automated review &amp; peer review</li></ul> | <ul style="list-style-type: none"><li>Compile code</li><li>Unit testing</li><li>Code-metrics</li><li>Build container images or package</li><li>Preparation or update in deployment templated</li><li>Create or update monitor dashboards</li></ul> | <ul style="list-style-type: none"><li>Integration test with other component</li><li>Load &amp; stress test</li><li>UI testing</li><li>Penetration testing</li><li>Requirement testing</li></ul> | <ul style="list-style-type: none"><li>Preparing release notes</li><li>Version tagging</li><li>Code freeze</li><li>Feature freeze</li></ul> | <ul style="list-style-type: none"><li>Updating the infrastructure i.e staging, production</li><li>Verification on deployment i.e smoke tests</li></ul> | <ul style="list-style-type: none"><li>Monitor designed dashboard</li><li>Alarm triggers</li><li>Automatic critical events handler</li><li>Monitor error logs</li></ul> |

# Continuous Integration CI

## 📘 Intégration continue ( CI = Continuous Integration)

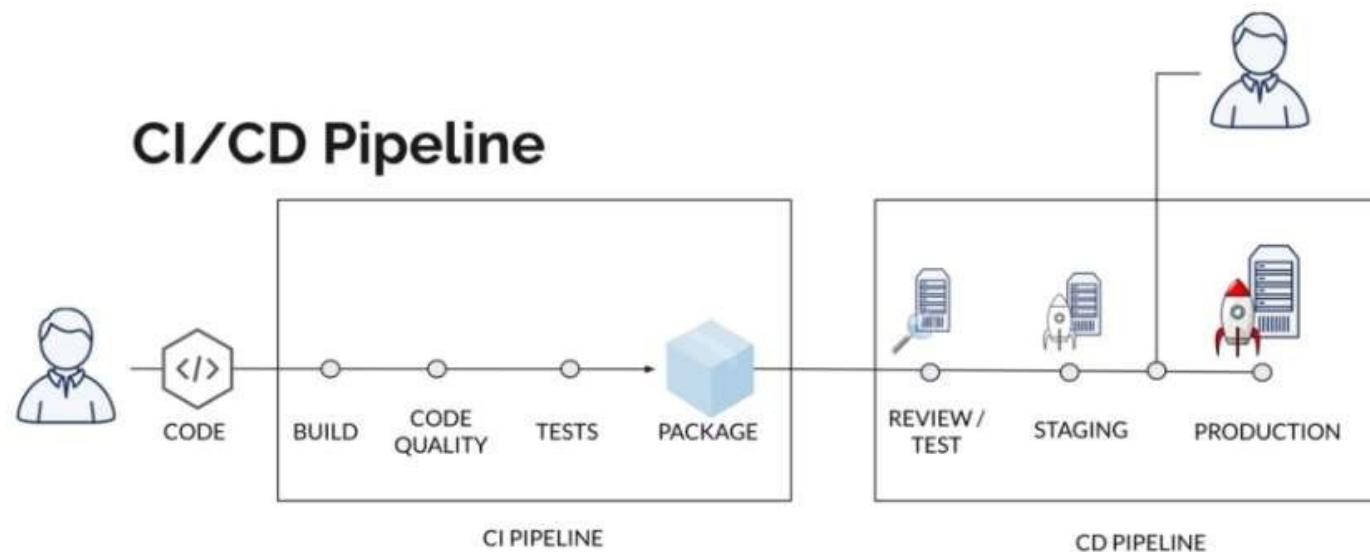
- 🌐 CI est une pratique qui permet d'intégrer le code avec d'autres développeurs
- 🌐 Le plus souvent, l'intégration du code est de vérifier si l'étape de construction (build stage) est toujours fonctionnelle
- 🌐 Une pratique courante consiste à vérifier également si les tests unitaires (Unit Testing stage) sont toujours fonctionnelles
- 🌐 L'objectif du pipeline CI est de construire un **package** qui sera déployé par la suite.



# Continuous Delivery CD

## Livraison continue (CD = Continuous Delivery)

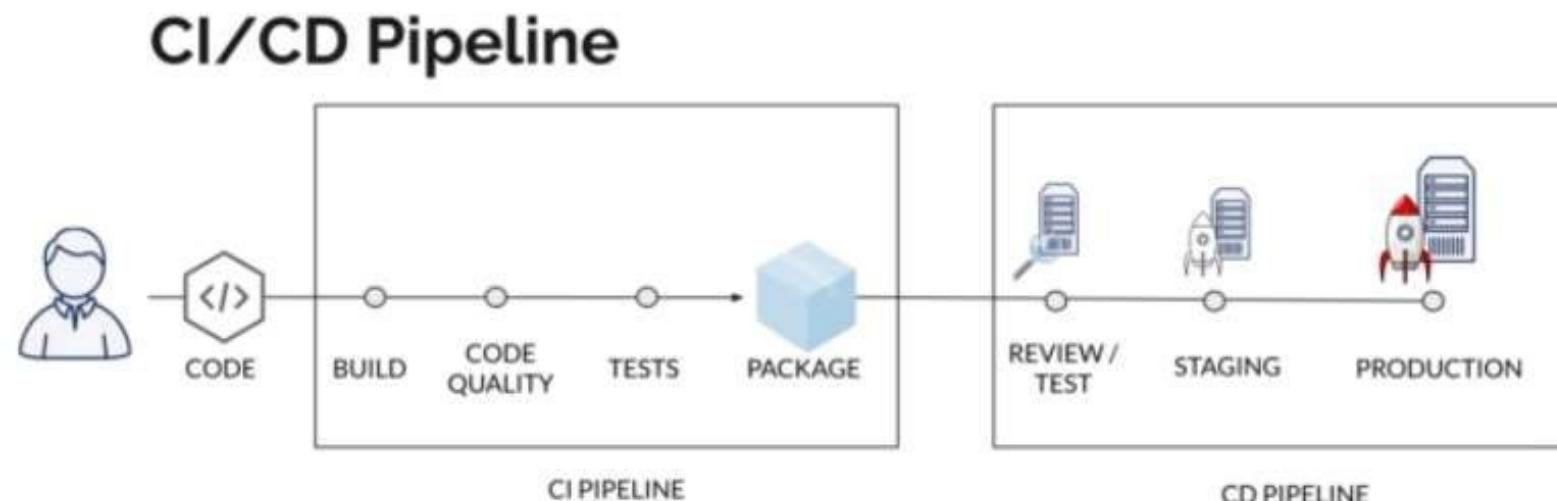
- ➊ C'est une extension de l'intégration continue
- ➋ L'objectif du CD est de prendre le package qui a été créé par le pipeline de CI et de tester son déploiement dans un **environnement de test (REVIEW et STAGING)**
- ➌ En ajoutant cette étape de pré-production et en exécutant certains tests, cela nous permet d'exécuter différents types de tests qui nécessitent la réponse de l'ensemble du système (généralement appelé tests d'acceptation)
- ➍ L'étape de déploiement en production (STAGE DEPLOY) est lancée **manuellement** si et seulement si le package a passé par toutes les étapes précédentes avec succès



# Continuous Deployment CD

## 📘 Déploiement continu (CD = Continuous Deployment)

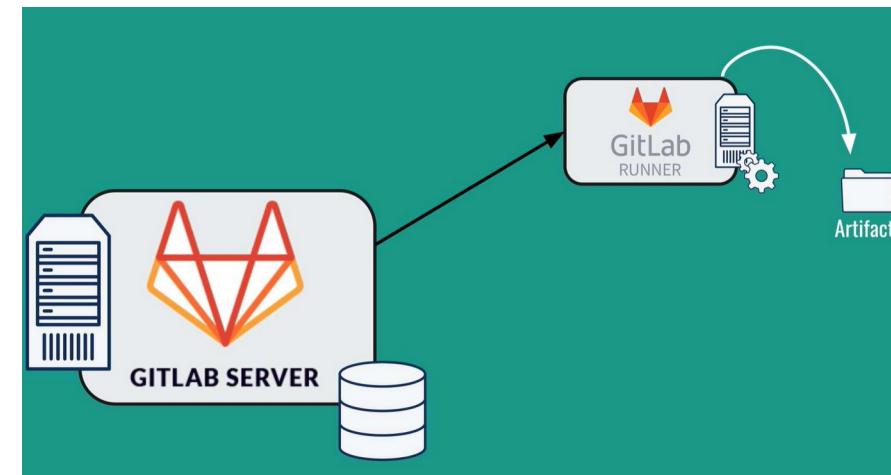
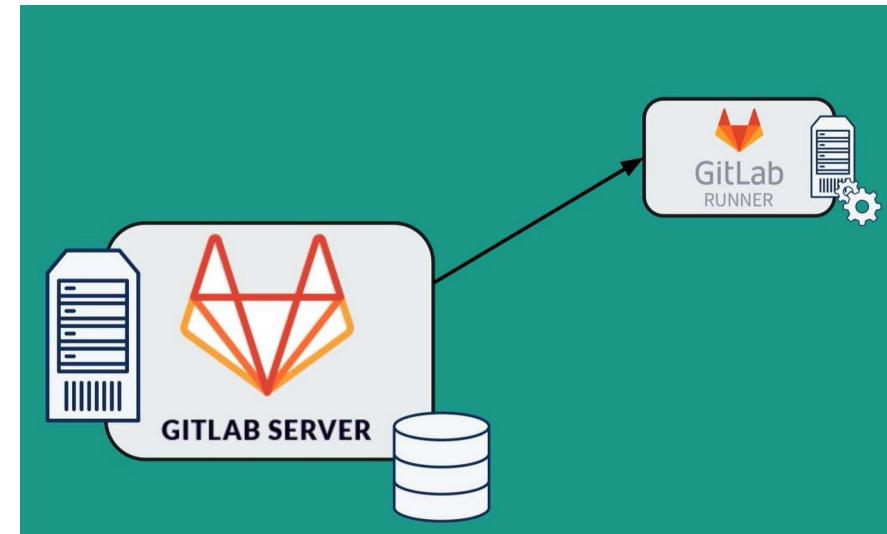
- 📘 Le déploiement continu est la pratique d'automatisation complète de l'ensemble des processus du pipeline CI/CD dans un environnement de **production**.
- 📘 Le package doit d'abord passer par toutes les étapes précédentes avec succès
- 📘 Aucune intervention manuelle n'est requise: **c'est automatiquement**



# Fonctionnement CI/CD

# Gitlab Serveur + Runner

- 📘 Vous avez besoin d'au moins d'un **serveur Gitlab** pour installer l'interface Web, vos dépôts Git (référentiels) et un **Runner**.
- 📘 Pour que le serveur Gitlab n'exécute pas les travaux (Jobs) et pour avoir une architecture **Évolutive**, facile à déployer et **scalable**, L'exécution du pipeline sera déléguée au **Runner**
- 📘 Si le Job a été exécuté avec succès, nous pouvons sauvegarder les résultats (les fichiers et/ou dossiers) dans des **Artifacts**. Ces derniers vont être stockés au sein des pipelines pour être utilisés par d'autres Jobs.

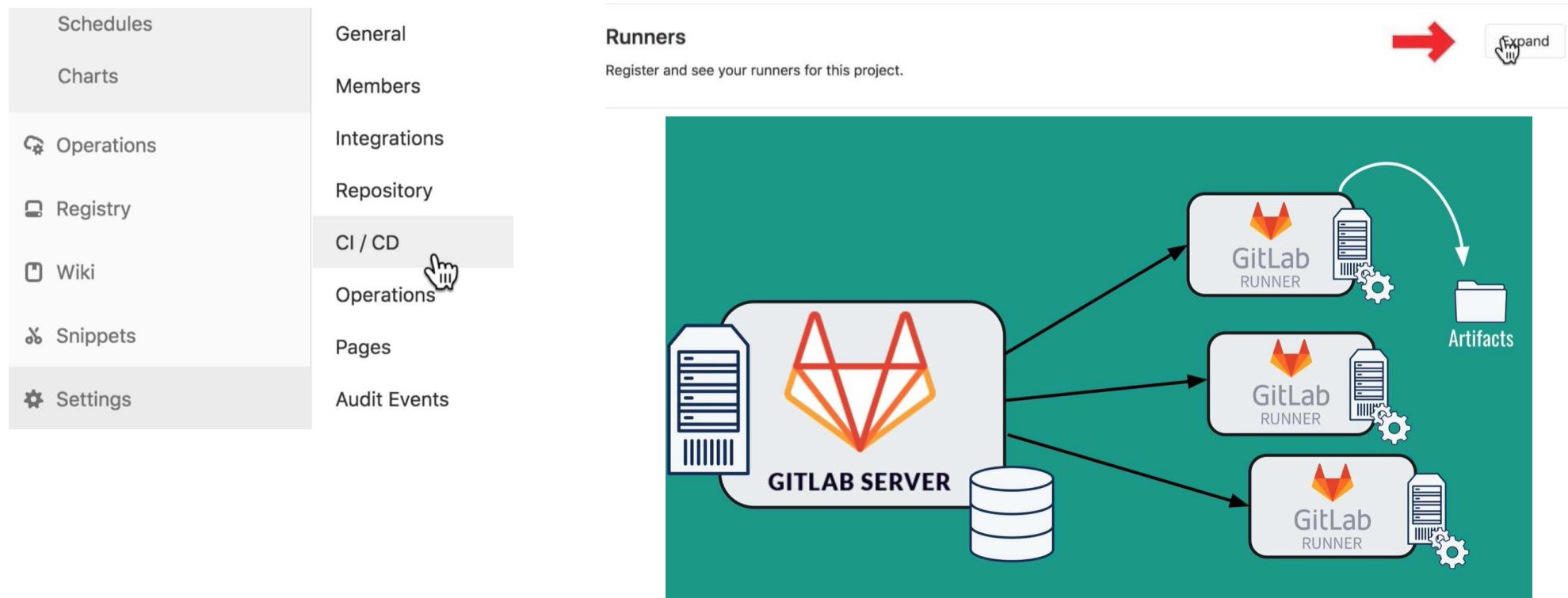


# Les Runners

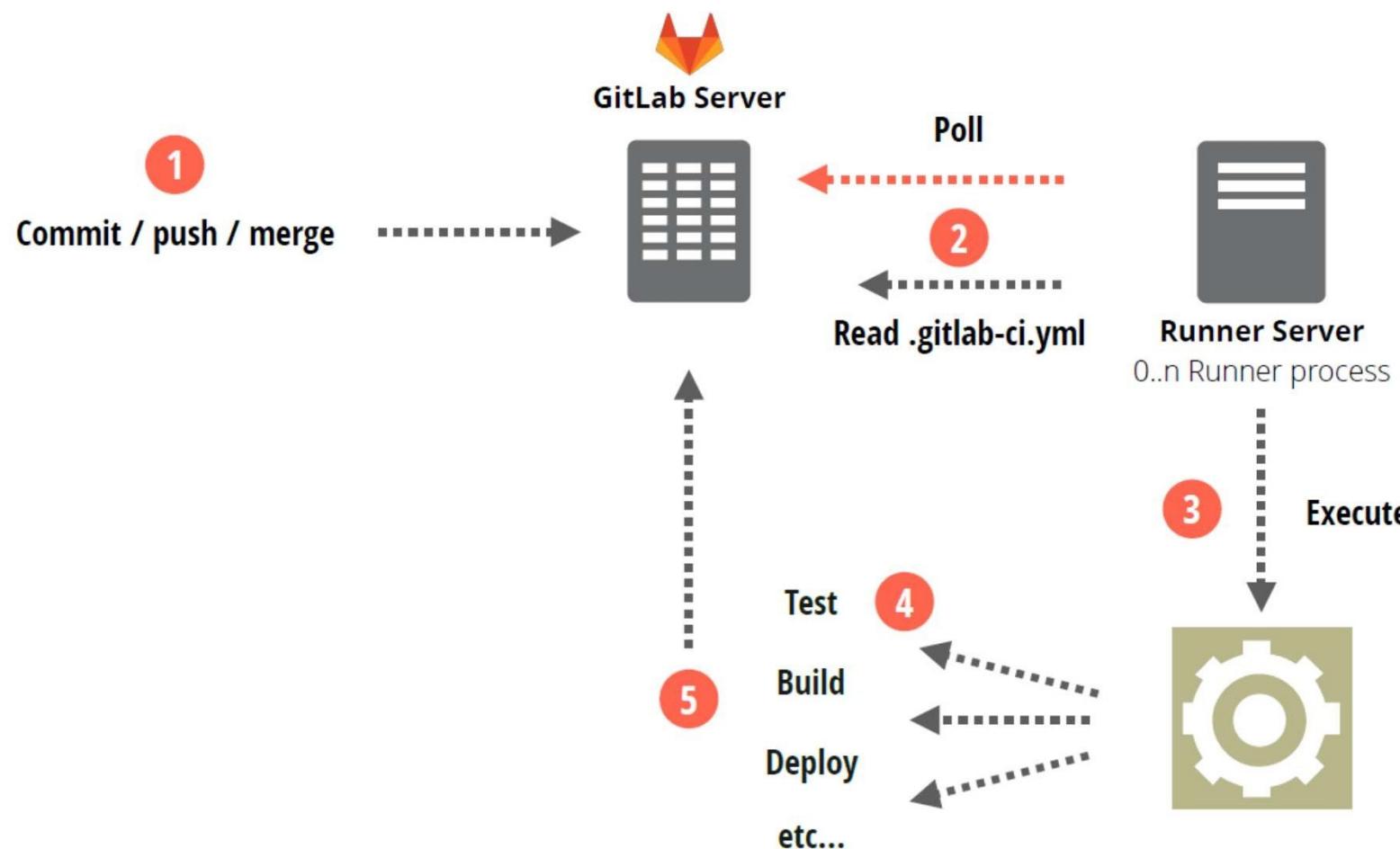
📘 Vérifier les Runner de votre projet dans : **Settings > CI/CD > Runners**

📘 Nous utilisons Shared Runners fournis par Gitlab.com

📘 vous pouvez créer vos propres Runners sur votre infrastructure et continuer à utiliser gitlab.com



# Fonctionnement de GitLab



# Module 7 : Gitlab CI/CD - le fichier YAML

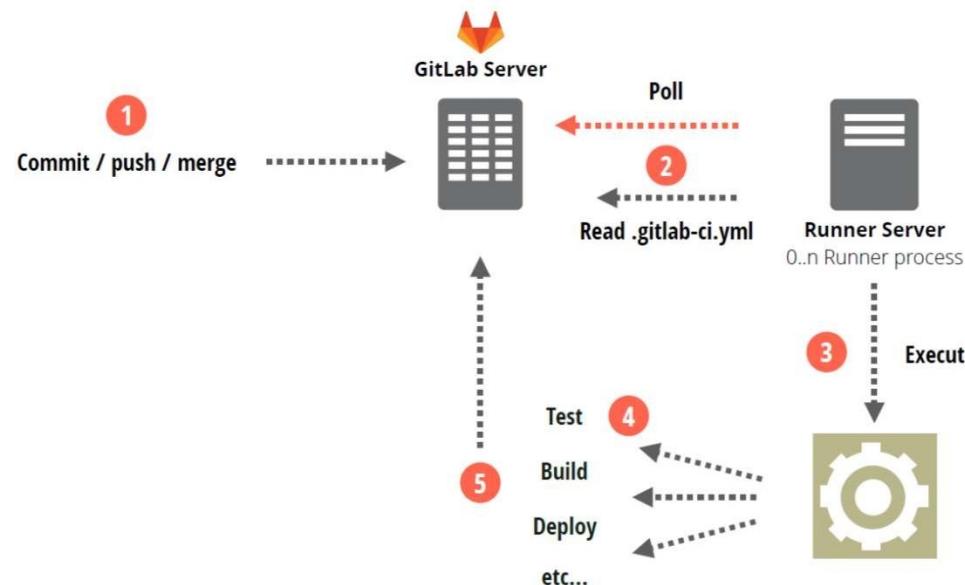
---

- Le manifest .gitlab-ci.yml
- Directives principales
- Autres directives

# Le manifest YAML

# Le manifeste .gitlab-ci.yml

- Pour que la CI/CD sur GitLab fonctionne il vous faut un **manifeste .gitlab-ci.yml** à la racine de votre projet
- Dans ce manifeste vous allez pouvoir définir des **stages**, des **jobs**, des **variables**, des **anchors**, etc.
- Le pipeline est déclenché à chaque **commit** ou **push** et s'exécute dans le Runner
- Et .gitlab-ci.yml explique au(x) Runner(s) ce qu'il faut faire



# Le manifeste .gitlab-ci.yml

- 📘 La gestion de la pipeline CD/CD dans GitLab se fait simplement par l'ajout d'un fichier **YAML** «**.gitlab-ci.yml**» dans la racine de votre projet git concerné .
- 📘 **YAML** est un langage de sérialisation de données utilisé, par exemple, pour le déploiement de configurations par **Ansible** ou pour la configuration d'applications multi-containers via **Docker Compose**.

```
1 ---
2 #Blog about YAML
3
4 title: YAML Ain't Markup Language
5 author:
6 first_name: Lauren
7 last_name: Malhoit
8 twitter: "@Malhoit"
9 learn:
10 - Basic Data Structures
11 - Commenting
12 - When and How
```

- 📘 Pour plus de détails sur YAML:

- <https://opensharing.fr/yaml-memo-bases>
- [https://docs.gitlab.com/ee/ci/yaml/gitlab\\_ci\\_yml.html](https://docs.gitlab.com/ee/ci/yaml/gitlab_ci_yml.html)

---

## Directives principales du manifest YAML

---

- 📖 Dans le manifeste de GitLab CI/CD vous pouvez définir un nombre **illimité** de jobs, avec des contraintes indiquant quand ils doivent être exécutés ou non.
- 📖 Voici comment déclarer un job le plus simplement possible :

```
job:1:
 script: echo 'my first job'

job:2:
 script: echo 'my second job'
```

- 📖 Les noms des jobs doivent être uniques et ne doivent pas faire parti des mots réservés: *image, services, stages, types, before\_script, after\_script, variables, cache...*
- 📖 Dans la définition d'un job seule la déclaration **script** est obligatoire.

- 📘 La déclaration script est donc la seule obligatoire dans un job. Cette déclaration est le cœur du job car c'est ici que vous indiquerez les actions à effectuer.
- 📘 Il peut appeler un ou plusieurs script(s) de votre projet, voire exécuter une ou plusieurs ligne(s) de commande.

```
job:script:
 script: ./bin/script/my-script.sh ## Appel d'un script de votre projet

job:scripts:
 script: ## Appel de deux scripts de votre projet
 - ./bin/script/my-script-1.sh
 - ./bin/script/my-script-2.sh

job:command:
 script: printenv # Exécution d'une commande

job:commands:
 script: # Exécution de deux commandes
 - printenv
 - echo $USER'
```

- » Cette déclaration est simplement l'image docker qui sera utilisée lors d'un job ou lors de tous les jobs

```
image: alpine # Image utilisée par tous les `jobs`, ce sera l'image par défaut
```

```
job:node: # Job utilisant l'image node
 image: node
 script: yarn install
```

```
job:alpine: # Job utilisant l'image par défaut
 script: echo $USER
```

# STAGES

- 📘 Cette déclaration permet de **grouper** des jobs en **étapes**.
- 📘 Par exemple on peut faire une étape de build, de test, de deployment, ....
- 📘 Si vous n'avez pas défini à quel un job appartient, il sera automatiquement attribué au stage **Test**.

stage

```
stages: # Ici on déclare toutes nos étapes
- build
- test
- deploy

job:build:
 stage: build # On déclare que ce `job` fait partie de l'étape
 build script: make build

job:test:unit:
 stage: test # On déclare que ce `job` fait partie de l'étape
 test script: make test-unit

job:test:functional:
 stage: test # On déclare que ce `job` fait partie de l'étape
 test script: make test-functional

job:deploy:
 stage: deploy # On déclare que ce `job` fait partie de l'étape
 deploy script: make deploy
```



## Autres directives

# BEFORE\_SCRIPT ET AFTER\_SCRIPT

- ☰ Ces déclarations permettront d'exécuter des actions avant et après votre script principal.
- ☰ Ceci peut être intéressant pour bien diviser les actions à faire lors des jobs, ou bien appeler ou exécuter une action avant et après chaque job

```
before_script: # Exécution d'une commande avant chaque `job`
 - echo 'start jobs'

after_script: # Exécution d'une commande après chaque `job`
 - echo 'end jobs'

job:no_overwrite: # Ici le job exécutera les actions du `before_script` et `after_script` par défaut
script:
 - echo 'script'

job:overwrite:before_script:
 before_script:
 -echo 'overwrite' # N'exécutera pas l'action définie dans le `before_script` par défaut
 script:
 - echo 'script'

job:overwrite:after_script:
 script:
 -echo 'script' after_script:
 - echo 'overwrite' # N'exécutera pas l'action définie dans le `after_script` par défaut
```

# ONLY ET EXCEPT

- 📘 Ces deux directives permettent de mettre en place des contraintes sur l'exécution d'une tâche.
- 📘 Vous pouvez dire qu'une tâche s'exécutera uniquement sur l'événement d'un push sur master ou s'exécutera sur chaque push d'une branche sauf master.

```
job:only:master:
 script: make deploy only:
 - master # Le job sera effectué uniquement lors d'un événement sur la branche master
```

```
job:except:master:
 script: make test except:master:
 - master # Le job sera effectué sur toutes les branches lors d'un événement sauf sur la branche master
```

# ONLY avec schedules

- Pour l'utilisation de **schedules** il faut dans un premier temps définir des règles dans l'interface web.
- On peut les configurer dans l'interface web de Gitlab : **CI/CD -> Schedules** et remplir le formulaire.

Schedule a new pipeline

Description  
Test schedule

Interval Pattern  
 Custom (Cron syntax)    Every day (at 4:00am)    Every week (Sundays at 4:00am)    Every month (on the 1st at 4:00am)

0 20 \* \* \*

Cron Timezone  
Paris

Target Branch  
master

Variables

|                    |                      |  |
|--------------------|----------------------|--|
| RELEASE            | staging              |  |
| Input variable key | Input variable value |  |

Activated  
 Active

Comme pour les directives `only` et `except`, la directive `when` est une contrainte sur l'exécution de la tâche. Il y a quatre modes possibles :

- 📘 **on\_success** : le job sera exécuté uniquement si tous les jobs du stage précédent sont passés
- 📘 **on\_failure** : le job sera exécuté uniquement si un job est en échec
- 📘 **always** : le job s'exécutera quoi qu'il se passe (même en cas d'échec)
- 📘 **manual** : le job s'exécutera uniquement par une action manuelle

```
stages:
- build
- test
- report

job:build:
 stage: build
 script:
 - make test
job:test:
 stage: test
 script:
 - make test
when: on_success # s'exécutera uniquement si le job `job:build` passe

job:report:
 stage: report
 script:
 - make report
when: on_success # s'exécutera si le job `job:build` ou `job:test` ne passe pas
```

# ALLOW\_FAILURE

- 📖 Cette directive permet d'accepter qu'un job échoue sans faire échouer la pipeline.

```
- stages:
 - build
 - test
 - report
 - clean

 ...

 stage: clean
 script:
 - make clean
 when: always
 allow_failure: true # Ne fera pas échouer la pipeline
 ...
```

- 📖 Avec GitLab Runner vous pouvez héberger vos propres runners sur un serveur ce qui peut être utile dans le cas de configuration spécifique.
- 📖 Chaque runner que vous définissez sur votre serveur à un nom, si vous mettez le nom du runner en tags, alors ce runner sera exécuté.

```
job:tag:
 script: yarn install
 tags:
 - shell # Le runner ayant le nom `shell` sera lancé
```

- 📖 Cette déclaration permet d'ajouter des services (container docker) de base pour vous aider dans vos jobs.
- 📖 Par exemple si vous voulez utiliser une base de données pour tester votre application c'est dans services que vous le demanderez.

```
test:functional:
 image: registry.gitlab.com/username/project/php:test
 services:
 - postgres # On appelle le service `postgres` comme base de données
 before_script :
 - composer install -n
 script :
 - codecept run functional
```

# ENVIRONMENT

---

- 📖 Cette déclaration permet de définir un environnement spécifique au déploiement
- 📖 Il est possible de spécifier :
  - 🌐 un **name**,
  - 🌐 une **url**,
  - 🌐 une condition **on\_stop**,
  - 🌐 une **action** en réponse de la condition précédente.

```
deploy:demo:
 stage: deploy
 environment: demo # Déclaration simple de l'environnement
 script:
 - make deploy

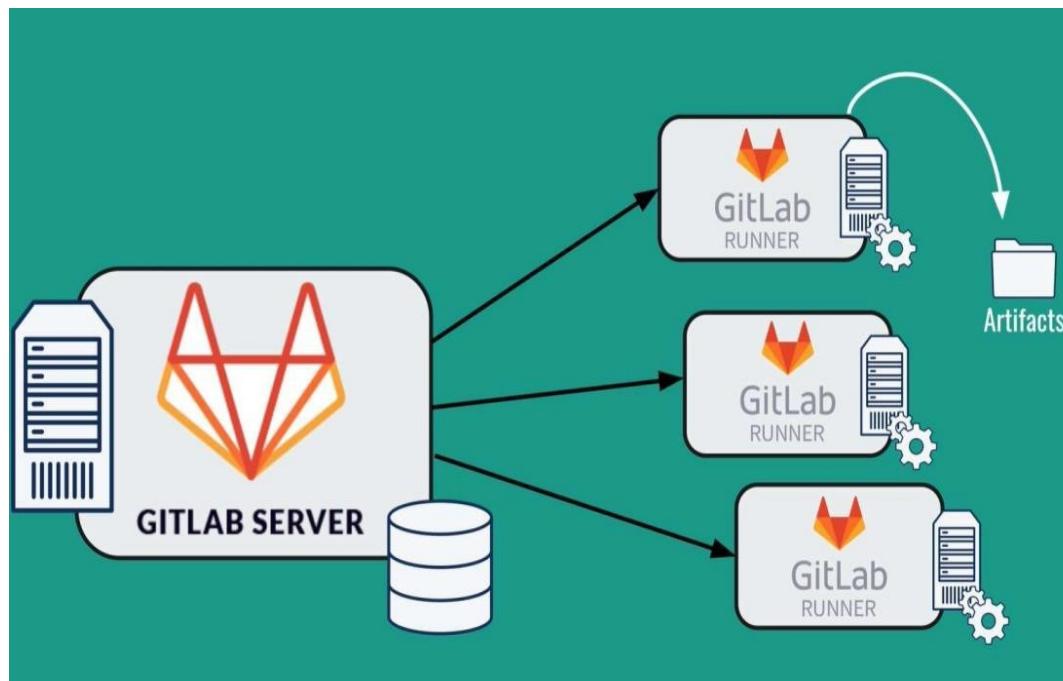
deploy:production:
 environment: # Déclaration étendue de l'environnement
 name: production
 url: 'https://blog.eleven-labs.fr/gitlab-ci/' # Url de l'application
 script:
 - make deploy
```

- Gitlab Runner
- Types de runners
- Runner executor
- Comment choisir un executor ?
- Docker executor

# Gitlab Runner

# Gitlab Runner

- Un Runner Gitlab-CI est un simple démon qui attend les Jobs comme vus dans le diagramme précédent.
- Dans Gitlab, vous pouvez créer vos propres Runners sur votre propre infrastructure en fonction de vos besoins.
- Selon le nombre de projets que vous avez ou de l'activité que vous avez sur un projet, il vous faudra davantage des Runners ou beaucoup de patience.



# Types des runners

# Les shared runners

---

- 📘 Sur GitLab.com, on retrouve notamment les Runners par défaut qui sont aussi couramment appelés Runners partagés (**Shared runners**).
- 📘 Ils ne nécessitent aucune installation, ni configuration et peuvent être amenés à exécuter vos tâches.
- 📘 Chaque Runner dispose d'un ensemble de tags permettant d'exécuter des jobs spécifiques en fonction de ceux mentionnés dans le fichier « `.gitlab-ci.yml` ».
- 📘 Assigner à un Runner des tâches particulières en fonction de plusieurs paramètres :
  - système d'exploitation,
  - environnement d'exécution,
  - capacités techniques (CPU et RAM) de la machine,
  - ...

# Les specific runners

---

- ☞ Il est possible d'installer et de créer des runners sur une infrastructure privée
- ☞ Plusieurs avantages :
  - Minimiser le temps d'exécution de la pipeline;
  - Avoir la main ou contrôler la configuration du Runner, mais aussi les couches plus basses comme le système d'exploitation;
  - Déployer au plus proche de l'environnement cible.
  - Héberger sa propre installation de GitLab. Dans cette situation, une installation d'au moins un Runner est requise car il n'y en a pas par défaut.

# Runner executor

# Qu'est ce qu'un executor ?

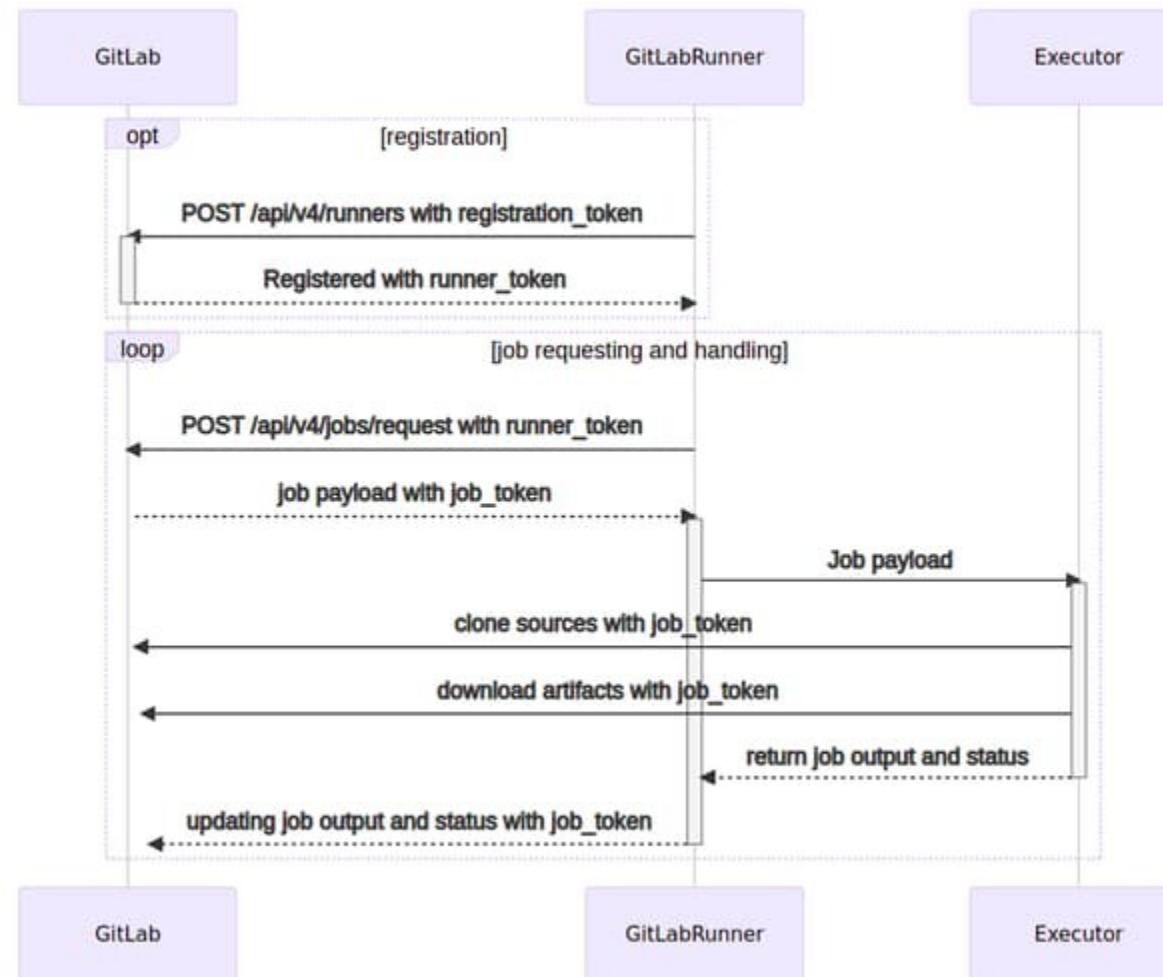
---

- 📘 Une fois un Job reçu celui-ci va demander à « un exécuteur » de traiter la demande.
- 📘 Les **exécuteurs** sont des sous-processus qui vont se charger de faire les commandes (scripts) que vous avez définies dans votre gitlab-ci.
- 📘 Gitlab-CI est capable de fonctionner de différente manière :
  - SSH,
  - Shell,
  - Parallels,
  - VirtualBox,
  - Docker,
  - Docker Machine (auto-scaling),
  - Kubernetes
  - Personnalisé (Custom)

# Fonctionnement

📘 L'avantage est double :

- Pas de limite en nombre de compilation.
- Accès à vos ressources locales pour le déploiement.



# Comparaison des executors

| Executor                                                         | SSH  | Shell | VirtualBox | Parallels | Docker | Kubernetes | Custom      |
|------------------------------------------------------------------|------|-------|------------|-----------|--------|------------|-------------|
| Nettoyer l'environnement de build (pour chaque build)            | X    | X     | ✓          | ✓         | ✓      | ✓          | conditional |
| Réutiliser le clone précédent s'il existe                        | ✓    | ✓     | X          | X         | ✓      | X          | conditional |
| Protéger l'accès au système de fichiers du Runner                | ✓    | X     | ✓          | ✓         | ✓      | ✓          | conditional |
| Migrer la machine de Runner                                      | X    | X     | partial    | partial   | ✓      | ✓          | ✓           |
| Prise en charge de zéro-configuration pour les builds simultanés | X    | X     | ✓          | ✓         | ✓      | ✓          | conditional |
| Environnements de construction très compliqués                   | X    | X     | ✓          | ✓         | ✓      | ✓          | ✓           |
| Débogage des problèmes de build                                  | easy | easy  | hard       | hard      | medium | medium     | medium      |

<https://docs.gitlab.com/runner/executors/>

# Comment choisir un executor ?

---

## **Shell**

- C'est le plus simple de tous.
- Vos scripts seront lancés sur la machine qui possède le Runner.

## **Parallels, VirtualBox**

- Le Runner va créer (ou utiliser) une machine virtuelle pour exécuter les scripts.
- Pratique pour avoir un environnement spécifique (exemple macOS)

## **Docker**

- Utilise Docker pour créer / exécuter vos scripts et traitement (en fonction de la configuration de votre .gitlab-ci.yml)
- Solution la plus simple et à privilégié

## **Docker Machine (auto-scaling)**

- Identique à docker, mais dans un environnement Docker multi-machine avec auto-scaling.

## **Kubernetes**

- Lance vos builds dans un cluster Kubernetes.
- Très similaire à Docker-Machine

## **SSH**

- À ne pas utiliser. Il existe, car il permet à Gitlab-CI de gérer l'ensemble des configurations possibles.

# Docker executor

---

- 📘 GitLab Runner utilise **Docker executor** pour exécuter les jobs sur des images Docker.
- 📘 **Docker executor** utilise Docker Engine pour exécuter chaque job dans un conteneur distinct et isolé.
- 📘 Pour se connecter à Docker Engine, l'exécutor utilise :
  - l'image et les services que vous définissez dans .gitlab-ci.yml.
  - Les configurations que vous définissez dans config.toml
- 📘 Docker executor divise le job en plusieurs étapes :
  - **1.** Prepare : crée et démarre les services.
  - **2.** Pre-job : clone, restaure le cache et télécharge les artefacts des étapes précédentes. S'exécute sur une image Docker spéciale.
  - **3.** Job : exécute votre build dans l'image Docker que vous configurez pour le runner.
  - **4.** Post-job : créez un cache, téléchargez des artefacts sur GitLab. Fonctionne sur une image Docker spéciale.

# Module 9 : Intégration Cypress dans GitLab

---

- Automatisation des tests
- Les tests end-to-end avec Cypress
- Intégration Cypress avec Gitlab

# Automatisation des tests

# Types de tests logiciels

---

- ☞ Les tests sont un processus d'exécution d'un programme logiciel pour **rechercher des erreurs** dans l'application en cours de développement
- ☞ Les tests sont **essentiels** pour déployer des logiciels sans erreur
- ☞ Les tests de logiciels sont généralement classés en deux types:
  - Tests fonctionnels
  - Tests non fonctionnels

# Tests fonctionnels

---

- ☞ Les tests fonctionnels d'un système impliquent des tests qui évaluent les **fonctionnalités** que le système devrait réaliser
- ☞ Les fonctionnalités sont ce que le système doit faire
- ☞ Parmi les types de tests fonctionnels:
  - Tests unitaires
  - Tests d'intégration
  - Tests de système
  - Tests d'interface

# Tests non-fonctionnels

---

- ☞ Les tests non-fonctionnels d'un système évaluent les caractéristiques des systèmes et des logiciels comme l'utilisabilité, la performance ou la sécurité
- ☞ Le test non-fonctionnel est le test de "comment" le système se comporte
- ☞ Parmi les types de tests non fonctionnels :
  - Tests de performances
  - Tests de charge
  - Tests de sécurité
  - Tests d'installation

# Pourquoi automatiser les tests ?

---

- ➥ Optimiser la couverture de test : Les tests automatisés permettent de couvrir la non régression pendant que les tests manuels peuvent se concentrer sur les nouvelles fonctionnalités du sprint, les fonctionnalités non automatisables ou des cas de tests bien spécifiques.
- ➥ Optimiser la qualité des développements : En testant plus souvent un périmètre plus important, la stabilité du code et des développements s'en retrouvent de meilleure qualité.
- ➥ Optimiser la durée des cycles de développement : Les tests automatisés étant effectués plus rapidement que les manuels, et pouvant être exécutés simultanément ou en horaire décalé.

# Quels sont les tests à automatiser ?

---

- ☞ Automatiser les tests les plus rentables.
- ☞ On peut donc se baser sur les critères suivants :
  - Le nombre d'exécution du test : plus un test sera joué, plus il sera rentable. Donc prioriser l'automatisation des tests de recevabilité et de non régression.
  - La faisabilité/complexité technique : plus un test est difficile à automatiser, plus il sera coûteux de le développer et de le maintenir.
  - La Maturité du code : plus un code est instable, plus il sera nécessaire de le maintenir, et par conséquent, coûteux.

# Les tests end-to-end avec Cypress

# L'assurance qualité

---

- ☞ L'assurance qualité a pris son importance dans le monde de l'informatique.
- ☞ Il est aujourd'hui impensable de laisser passer de nouvelle *features* ou amélioration de logiciel/site web, sans assurer que la qualité de la précédente version du système ciblé ne soit pas détériorée.
- ☞ Ces processus de qualité veillant à la satisfaction de l'utilisateur peuvent être coûteux pour une entreprise

- ☞ Jusqu'à présent, effectuer des tests bout à bout (ou tests end-to-end) n'était pas forcément chose facile.
- ☞ Il s'agit en effet de dérouler un scénario complet de tests à la manière d'une situation réelle de production.
- ☞ À travers cette scénarisation, l'objectif est bien de valider le logiciel développé et son association avec des tierces applications via API par exemple.
- ☞ C'est ici qu'entre en jeux Cypress.io
  - outil open source permettant de mettre facilement en place ces fameux tests d'applications utilisant des frameworks JavaScript modernes.
  - Il permettra de tester tout ce qui fonctionne dans le navigateur.

# Pourquoi Cypress ?

---

- ─ Cypress offre une toute nouvelle architecture et s'exécute dans le navigateur avec votre application.
- ─ Cypress fonctionne, quel que soit le framework JavaScript que vous utilisez : React Js, Angular, Vue Js,...
- ─ Vous écrivez vos tests en JavaScript (avec Mocha, Chai et Sinon). Vous restez donc dans une technologie commune.

# Intégration de Cypress avec Gitlab

# Intégration Gitlab

---

- ─ GitLab fournit une intégration avec Cypress via ses pipelines CI/CD.
- ─ On peut configurer GitLab pour exécuter des tests Cypress dans le pipeline et afficher les résultats des tests dans l'interface utilisateur de GitLab.
- ─ GitLab offre également la possibilité de stocker des artefacts de test tels que des captures d'écran et des vidéos.

# Intégration Gitlab

The screenshot shows the GitLab web interface for importing projects. The URL in the browser bar is [https://gitlab.com/projects/new#import\\_project](https://gitlab.com/projects/new#import_project). The left sidebar is titled 'Your work' and includes 'Projects' (selected), 'Groups', 'Issues', 'Merge requests', and 'To-Do List'. The main content area has a title 'Import project' with a sub-instruction 'Migrate your data from an external source like GitHub, Bitbucket, or another instance of GitLab.' Below this is a section 'Import project from' with a note about migrating GitLab projects via direct transfer. It lists several import options: 'GitLab export', 'GitHub', 'Bitbucket Cloud', 'Bitbucket Server', 'FogBugz', 'Gitea', and 'Repository by URL'. There is also a 'Manifest file' button.

# Intégration Gitlab

The screenshot shows the GitLab interface for importing repositories from GitHub. The left sidebar lists various project management options like Projects, Groups, Issues, Merge requests, To-Do List, Milestones, Snippets, Activity, Workspaces, Environments, Operations, and Security. The main content area is titled "Import repositories from GitHub" and shows a search bar with "cypress". Below the search bar, there are tabs for "Owned", "Collaborated", and "Organization", with "Owned" selected. A warning message states: "The more information you select, the longer it will take to import". There are several checkboxes for import settings: "Import issue and pull request events" (unchecked), "Use alternative comments import method" (unchecked), "Import Markdown attachments (links)" (unchecked), "Import collaborators" (checked), and "Import direct repository collaborators who are not outside collaborators. Imported collaborators who aren't members of the group you imported the project into consume seats on your GitLab instance." At the bottom, a table shows the import status for "brahimhamdi/gitlab\_cypress" to "gitlab\_cypress" by "brahimhamdi". The status is "Not started" and there is an "Import" button.

New project / Import project / GitHub import

## Import repositories from GitHub

Select the repositories you want to import

Owned   Collaborated   Organization

cypress

Import 1 repository

Advanced import settings

**⚠ The more information you select, the longer it will take to import**

Import issue and pull request events  
For example, opened or closed, renamed, and labeled or unlabeled. Time required to import these events depends on how many issues or pull requests your project has.

Use alternative comments import method  
The default method can skip some comments in large projects because of limitations of the GitHub API.

Import Markdown attachments (links)  
Import Markdown attachments (links) from repository comments, release posts, issue descriptions, and pull request descriptions. These can include images, text, or binary attachments. If not imported, links in Markdown to attachments break after you remove the attachments from GitHub.

Import collaborators  
Import direct repository collaborators who are not outside collaborators. Imported collaborators who aren't members of the group you imported the project into consume seats on your GitLab instance.

| From GitHub                | To GitLab                    | Status      |
|----------------------------|------------------------------|-------------|
| brahimhamdi/gitlab_cypress | brahimhamdi / gitlab_cypress | Not started |

Import

# Intégration Gitlab

The screenshot shows a web browser window displaying a GitLab project page. The URL in the address bar is [https://gitlab.com/brahimhamdi/gitlab\\_cypress](https://gitlab.com/brahimhamdi/gitlab_cypress). The page title is "gitlab\_cypress". The left sidebar contains navigation links for Project, Issues, Merge requests, Manage, Plan, Code, Build, Secure, Deploy, Operate, Monitor, Analyze, and Settings. The main content area shows a commit history for the 'master' branch. The first commit is titled "fix code smells" by Lucas Détré, authored 10 months ago. Below the commit history is a table listing various files and their last update times.

| Name            | Last commit                                                         | Last update   |
|-----------------|---------------------------------------------------------------------|---------------|
| .gitlab         | Upload New File                                                     | 2 years ago   |
| .scannerwork    | fix gitlab CI - sonar analyse                                       | 1 year ago    |
| cypress         | Merge branch 'review-accessibilit...' (from review-accessibilit...) | 11 months ago |
| docker          | maj conf                                                            | 1 year ago    |
| src             | fix code smells                                                     | 10 months ago |
| .browserslistrc | report correctifs prod                                              | 2 years ago   |
| .editorconfig   | dev connexion avec PE connect                                       | 3 years ago   |
| .eslintignore   | upgrade angular version to versio...                                | 2 years ago   |
| .eslintrc.js    | upgrade angular version to versio...                                | 2 years ago   |

**Project Information:**

- 1,779 Commits
- 1 Branch
- 0 Tags
- 15.1 MiB Project Storage

**Actions:**

- README
- GNU AGPLv3
- CI/CD configuration
- Add CHANGELOG
- Add CONTRIBUTING
- Add Kubernetes cluster
- Add Wiki
- Configure Integrations

# Intégration Gitlab

The screenshot shows the GitLab Pipeline Editor interface. On the left, there's a sidebar with project navigation (gitlab\_cypress), pinned items, and various management sections like Plan, Code, Build, Pipelines, Jobs, Pipeline editor (which is selected), Pipeline schedules, Artifacts, Secure, and Deploy. The main area is titled "Pipeline Editor" and shows a code editor with a YAML configuration file. The file defines stages, variables, cache, paths, and a build job for a frontend application.

```
1 stages:
2 - build-phase-1
3 - build-phase-2
4 - deploy-application-recette
5 - build-docker-configuration-production
6 - deploy-application-production
7 - dynamic-application-security-testing
8
9 variables:
10 | LAUNCH_CYPRESS_TESTS: "true"
11
12 cache: &global_cache
13 | key: ${CI_JOB_STAGE}-${CI_COMMIT_REF_SLUG}
14 | paths:
15 | - .npm/
16 | - cache/Cypress
17 | - node_modules/
18 | - .sonar/cache
19
20 build_frontend_application :
21 | stage: build-phase-1
22 | image:
23 | name: registry.beta.pole-emploi.fr/customize-docker-images/angular-ng
24 | entrypoint: []
25 | script:
26 | #lancement du build de l'application
```

Below the code editor, there's a "Commit message" field containing "Update .gitlab-ci.yml file". At the top of the browser window, the URL is https://gitlab.com/brahimhamdi/gitlab\_cypress/-/ci/editor?branch\_name=master, and the page is at 90% zoom.

# Module 10 : Plus loin avec Gitlab

---

- Optimisation des performances du pipeline
- Gestion des variables et des secrets
- Autoscaling des runners avec Docker Machine
- Répartition des jobs dans un cluster Kubernetes
- Intégration de l'outil de monitoring Prometheus

# Optimisation des performances du pipeline

# Optimisation du YAML

---

- 1. Paralléliser les gros jobs (directive parallel)**
- 2. Configurez la mise en cache, divisez le cache et définissez la politique (directive cache)**
3. Utilisez de petites distributions Linux (directive image)
4. Téléchargez uniquement les artefacts nécessaires (directive dependencies)
5. Utilisez des *rules* adaptées (directive rule)
6. Définir judicieusement les étapes et ajuster en fonction des besoins
7. Configurer les pipelines interruptibles (directive interruptible)
8. Ré-exécutez automatiquement les jobs qui ont échoué (directive retry)

# Paralléliser les gros jobs

- ─ En exécutant des suites de tests ou des jobs étendues, la parallélisation peut réduire considérablement la durée du pipeline.
- ─ On peut diviser le travail en plusieurs jobs exécutées simultanément, améliorant ainsi la vitesse globale du pipeline.
- ─ Faut définir le nombre de jobs concurrents dans *config.toml*
- ─ Exemple :

```
Test-in-parallel:
 Parallel : 3
 script:
 - bundle
 - bundle exec rspec_booster --job $CI_NODE_INDEX/$CI_NODE_TOTAL
```

# Utiliser le cache

- ☞ Le cache en CICD permet de conserver des dossiers ou fichiers à travers les pipelines
- ☞ Lorsqu'il existe plusieurs serveurs de Runners, nous devons configurer le cache partagé d'exécution lorsque nous visons la vitesse du pipeline.
- ☞ Une chose à considérer lors de la définition du cache est de la diviser en plusieurs types de caches, le cas échéant, en profitant du fait que le mot-clé *cache* peut prendre une liste de caches.
- ☞ Exemple :

```
Test-job:
script:
 - echo "This job uses a cache."
cache:
 key: binaries-cache
 paths:
 - binaries/*.apk
 - .config
```

# Optimisation de la configuration du projet

---

1. Désactivez le cache séparé pour les branches protégées :  
(Settings > CI/CD > General pipelines)
2. Évitez la reconstruction des images Docker avec une fusion *fast-forward*
3. Configurez les *push rules* pour éviter les pipelines pour les branches mal configurées (Settings > Repository > Branch rules)

# Optimisation de la configuration des Runners

---

1. Utiliser le cache pour Docker build et push
2. Utiliser le cache pour les images des jobs (pull\_policy = if-not-present)
3. Optimiser la compression des caches et des artefacts (FF\_USE\_FASTZIP=1)
4. Dimensionnez correctement les Runners et ajustez le maximum de jobs parallélisés (concurrent = 10)

# Gestion des variables et des secrets

# Utiliser les variables

- Il est possible de créer vos propres variables en utilisant variables dans vos jobs.
  - Ces variables sont de la forme *clé: valeur*
- Il est également possible de créer des variables dans les paramètres CI/CD de votre projet.
  - Ces variables sont soit de la forme de fichier (File) ou Variable et peuvent être cachées/protégées.
  - On peut également limiter la portée à un environnement précis.

Add variable

Key  
test2

Value  
Je suis un second test

Type  
Variable

Environment scope  
All (default)

Flags

Protect variable ?  
Export variable to pipelines running on protected branches and tags only.

Mask variable ?  
Variable will be masked in job logs. Requires values to meet regular expression requirements. More information

Cancel Add variable

# Gestion des secrets

---

- 📘 Lorsque vous travaillez avec GitLab CI/CD, il est essentiel de gérer les secrets et les informations sensibles, tels que :
  - les clés d'API
  - les mots de passe
  - Etc ...
- 📘 Utilisez GitLab CI/CD pour stocker ces informations de manière sécurisée en utilisant les variables d'environnement protégées ou les fichiers de variables.
- 📘 Évitez de les inclure directement dans votre fichier `.gitlab-ci.yml`
- 📘 Pensez à intégrer des outils de sécurité tels que SonarQube ou GitLab SAST (Static Application Security Testing) dans vos pipelines de CI/CD. Pour identifier et résoudre les vulnérabilités de sécurité dès le début du processus de développement.

# Autoscaling des runners

# Scalabilité automatique (autoscaling) des runners

---

- 📘 Le Runner standard, qui utilise l'exécuteur Docker, Shell, SSH, VirtualBox ou Parallels, exécute une seule instance directement sur le serveur GitLab Runner
- 📘 Le Runner autoscaler est différent du premier concept car ce n'est pas l'instance même qui exécutera les conteneurs et donc vos jobs GitLab CI, mais elle pilotera comme une tour de contrôle la création d'autres instances (agents):
  - D'autres machines virtuelles si l'exécuteur est Docker+Machine
  - D'autres conteneurs si l'exécuteur est Docker-autoscaler (Béta)
- 📘 Une fois les jobs terminés ou si le Runner n'est plus sollicité, les instances "agents" seront détruites.

# Autoscaling des runners avec Docker Machine

---

- ─ Docker machine existe depuis les débuts de Docker, et qui a pris son essor lorsque Docker Desktop n'existait pas encore.
- ─ Cet outil permet de gérer un parc de machines faisant tourner docker engine, et de configurer le client docker pour se connecter directement vers l'une des machines de ce parc.
- ─ Docker a obsolète Docker Machine et l'a remplacé par Docker Desktop.
- ─ GitLab maintient un fork Docker Machine pour les utilisateurs de GitLab Runner qui s'appuient sur l'exécuteur Docker+Machine.
- ─ GitLab Runner Autoscaler est le successeur de Docker Machine

```
curl -O "https://gitlab-docker-machine-downloads.s3.amazonaws.com/main/docker-machine-Linux-x86_64"
cp docker-machine-Linux-x86_64 /usr/local/bin/docker-machine
chmod +x /usr/local/bin/docker-machine
```

# Autoscaling des runners avec Docker Autoscaler

---

- ─ Docker Autoscaler crée des instances à la demande pour prendre en charge les jobs exécutés par le runner.
- ─ Il encapsule Docker executor afin que toutes les options et fonctionnalités de l'exécuteur Docker soient prises en charge.
- ─ Il utilise des plugins de Fleeting pour l'autoscaling. Fleeting est une abstraction pour un groupe d'instances autoscalé, qui en charge les fournisseurs de cloud, tels que GCP, AWS et Azure.
- ─ Il est en version Béta depuis GitLab Runner 16.6

---

## Répartition des jobs dans un cluster Kubernetes

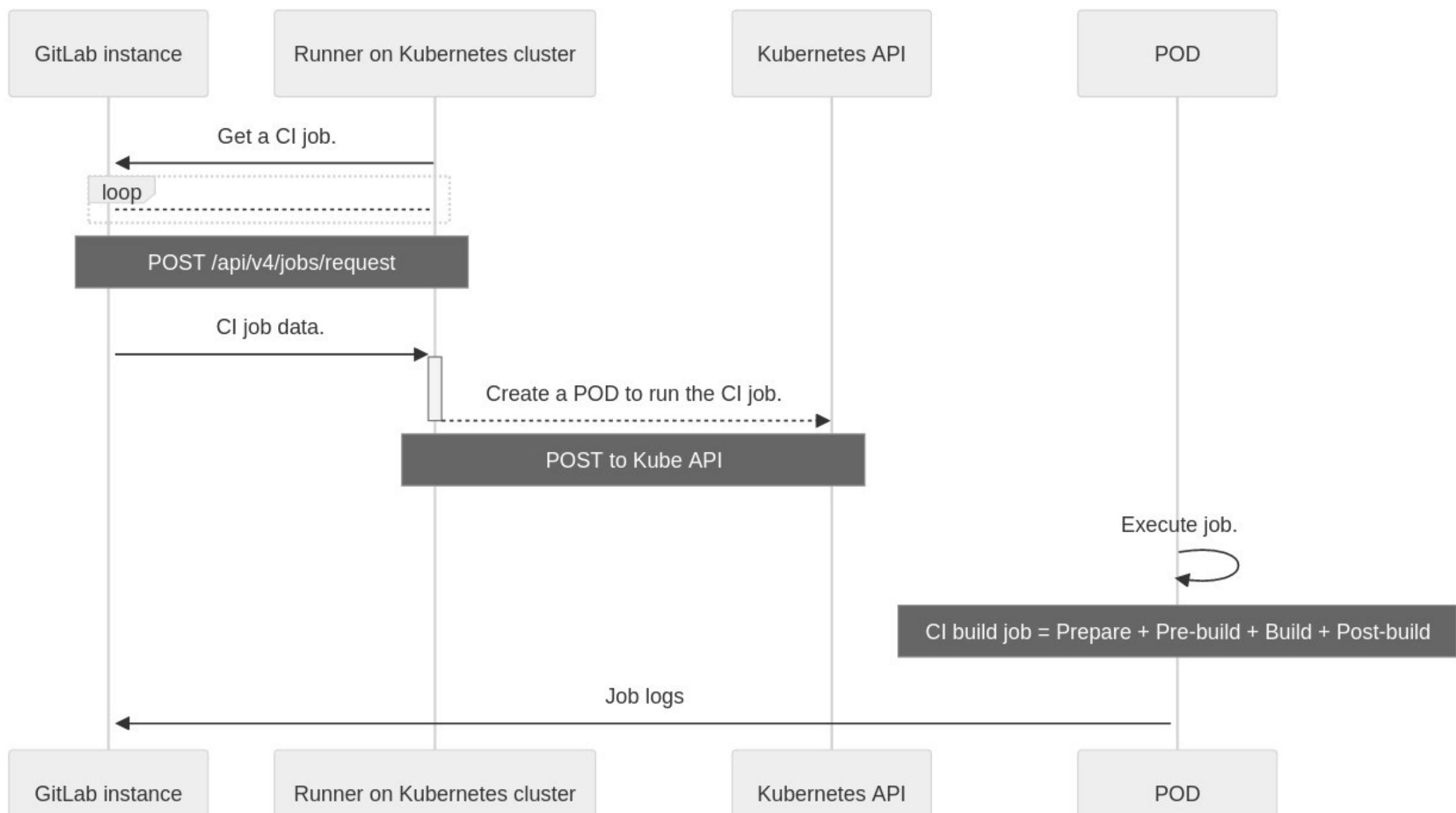
---

# Kubernetes executor

---

- ─ Kuberentes executor utilise les clusters kubernetes pour exécuter les jobs de GitLab ci.
- ─ Un cluster kubernetes est un ensemble de nœuds qui exécute des Pods
- ─ Un Pod est est un ensemble composé d'un ou plusieurs conteneurs (Docker ou autre)
- ─ Un nouveau Pod est crée pour l'exécution de chaque job

# Kubernetes executor



---

## Intégration de l'outil de monitoring Prometheus

---

# Monitoring de Gitlab

---

- ─ La solution utilisée pour la surveillance des instances GitLab et GitLab Runner est Prometheus
- ─ Prometheus permet le monitoring de n'importe quelle système et pas seulement GitLab
- ─ Les agents de Prometheus, appelés exporters, exportent les métriques sur des ports standards.
- ─ Il sauvegarde les métriques collectés régulièrement dans sa base de données de type time-series
- ─ Prometheus et ses exporters sont activés par défaut sur le serveur GitLab
- ─ Parmi les exporters utilisés par Gitlab :
  - Node exporter
  - Web exporter
  - Redis exporter
  - PostgreSQL exporter