

Orchestration de conteneurs et CI/CD

 ESGI Lyon - 2022/2023



- Présentation disponible à l'adresse: <https://dduportal.github.io/esgi-containers-ci-cd/2023>
- Version PDF de la présentation :  Cliquez ici
- Contenu sous licence Creative Commons Attribution 4.0 International License
 - Une partie du contenu provient de <https://github.com/cicd-lectures/slides> (auteurs: Julien Levesy et moi-même)
- Code source de la présentation:  <https://github.com/dduportal/esgi-containers-ci-cd>

Comment utiliser cette présentation ?

- Pour naviguer, utilisez les flèches en bas à droite (ou celles de votre clavier)
 - Gauche/Droite: changer de chapitre
 - Haut/Bas: naviguer dans un chapitre
- Pour avoir une vue globale : utiliser la touche "o" (pour "**Overview**")

Bonjour !

La suite: vers le bas ↓

Damien DUPORTAL

- Staff Software Engineer chez CloudBees pour le projet Jenkins 2011
- Freelancer
- Me contacter :
 -  damien.duportal <chez> gmail.com
 -  dduortal
 -  Damien Duportal
 -  @DamienDuportal

Et vous ?



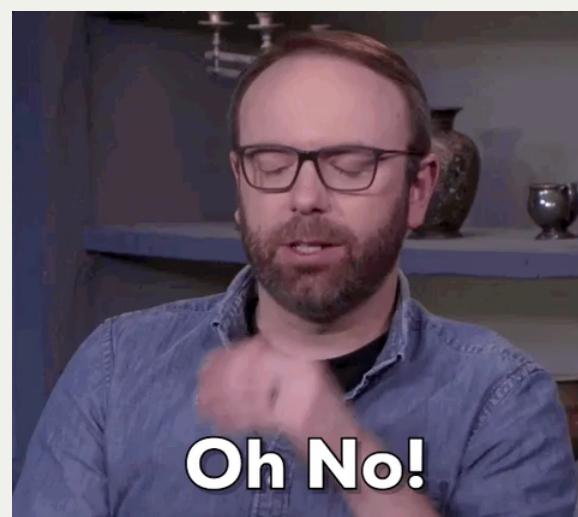
A propos du cours

- Alternance de théorie et de pratique pour être le plus interactif possible
- Compatible Distanciel / Présentiel
- Contenu entièrement libre et open-source
 - N'hésitez pas ouvrir des Pull Request si vous voyez des améliorations ou problèmes: sur cette page (😉 wink wink)

Calendrier

- **Distanciel**  Mercredi 01 février 2023 - 14h → 17h15
- **Présentiel**  Mardi 28 février 2023 - 14h → 17h15
- **Présentiel**  Mercredi 01 mars 2023 - 08h → 17h16
- **Présentiel**  Jeudi 02 mars 2023 - 08h → 13h
- **Distanciel**  Vendredi 28/04 - 09h45 → 13h00

Evaluation



- Pourquoi ? s'assurer que vous avez acquis un minimum de concepts
- Quoi ? Une note sur 20, basée sur une liste de critères exhaustifs
- Comment ? Un projet GitHub (public) à me rendre (timing à déterminer ensemble)

Plan

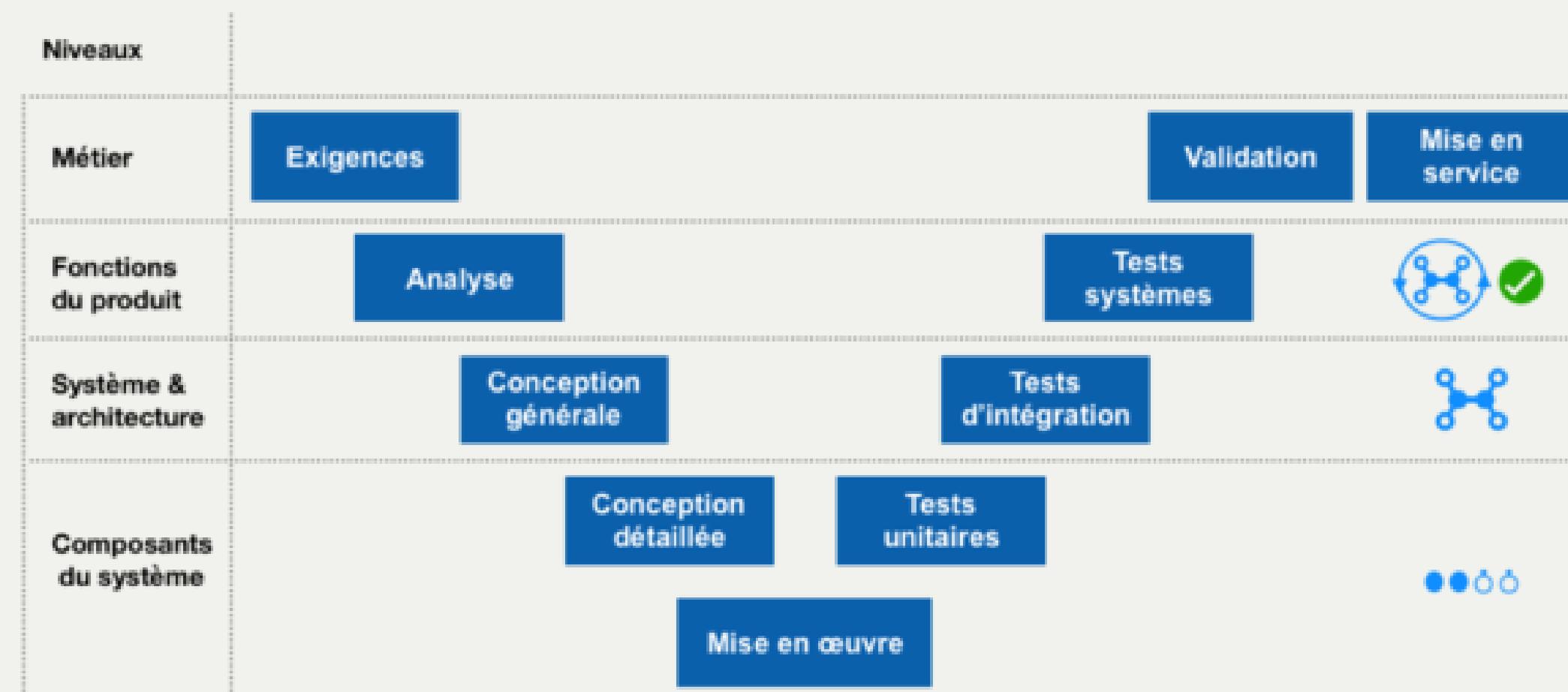


- Introduction
- Rappels (Ligne de commande, Git)
- Intégration Continue (CI)
- Rappels (Docker)
- Docker Avancé
- Orchestration de Containers
- Déploiement Continu (CD)

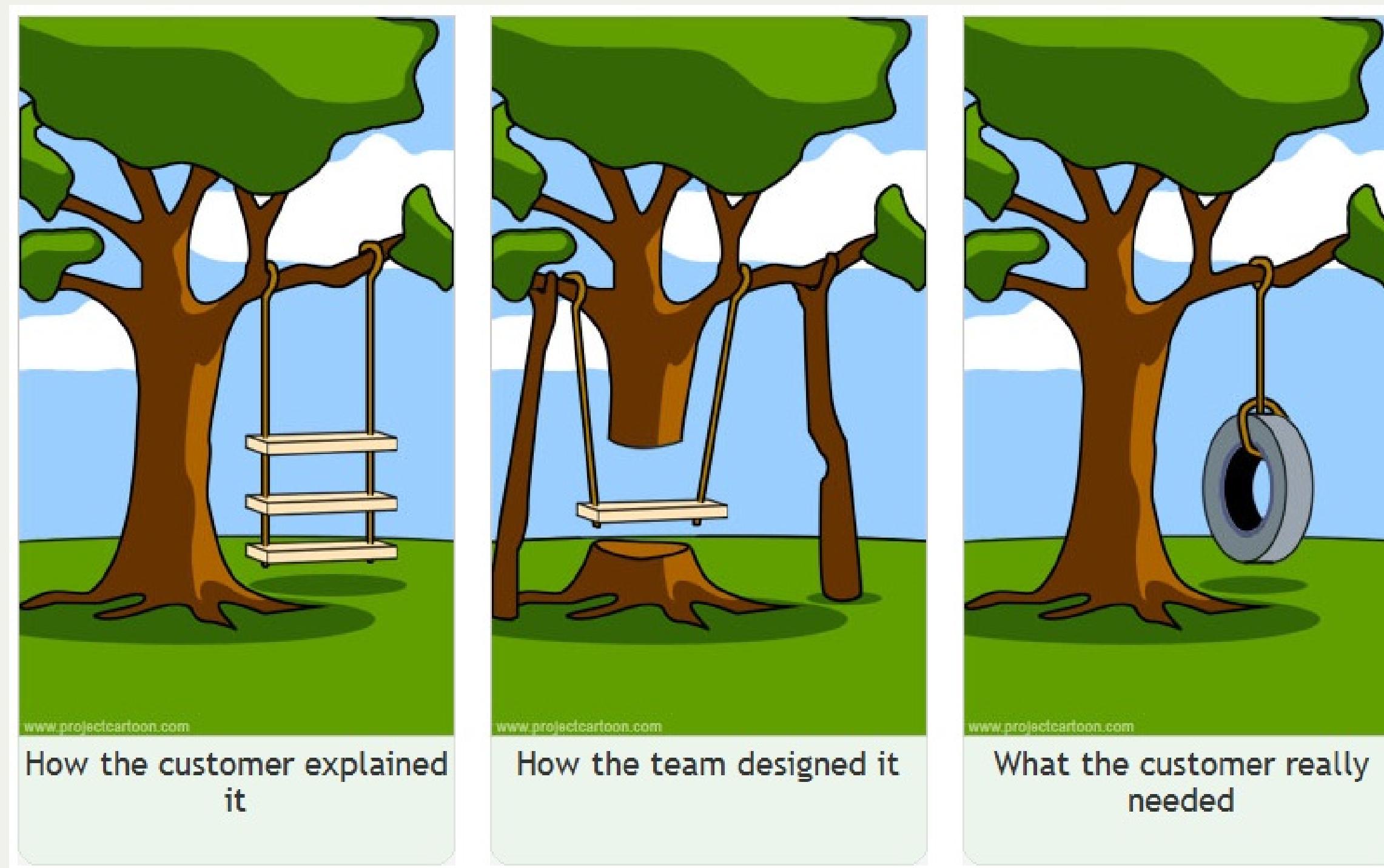
La suite: vers la droite ➔

Vous avez dit "DevOps" ?

Avant : le cycle en V



What could go right?



Copyright © <https://www.projectcartoon.com> under the Creative Commons Attribution 3.0 Unported License

Et dans la vraie vie c'est pire !



Copyright © <https://www.projectcartoon.com> under the Creative Commons Attribution 3.0 Unported License

Problématique

Travail basé sur les hypothèses de départ + Temps écoulé entre la capture du besoin et la mise à disposition en production

- Le besoin a forcément évolué dans ce laps de temps
- Erreur de capture du besoin == coût de l'erreur décuplé avec le temps

Agile

- Travailler par petites itérations **complètes**
 - Commencer petit
 - Confronter le logiciel au plus tôt aux utilisateurs.
 - Refaire des hypothèses basées sur ce que l'on à appris, et recommencer !
- "Embrasser" le changement : Votre logiciel va changer en **continu**



Source

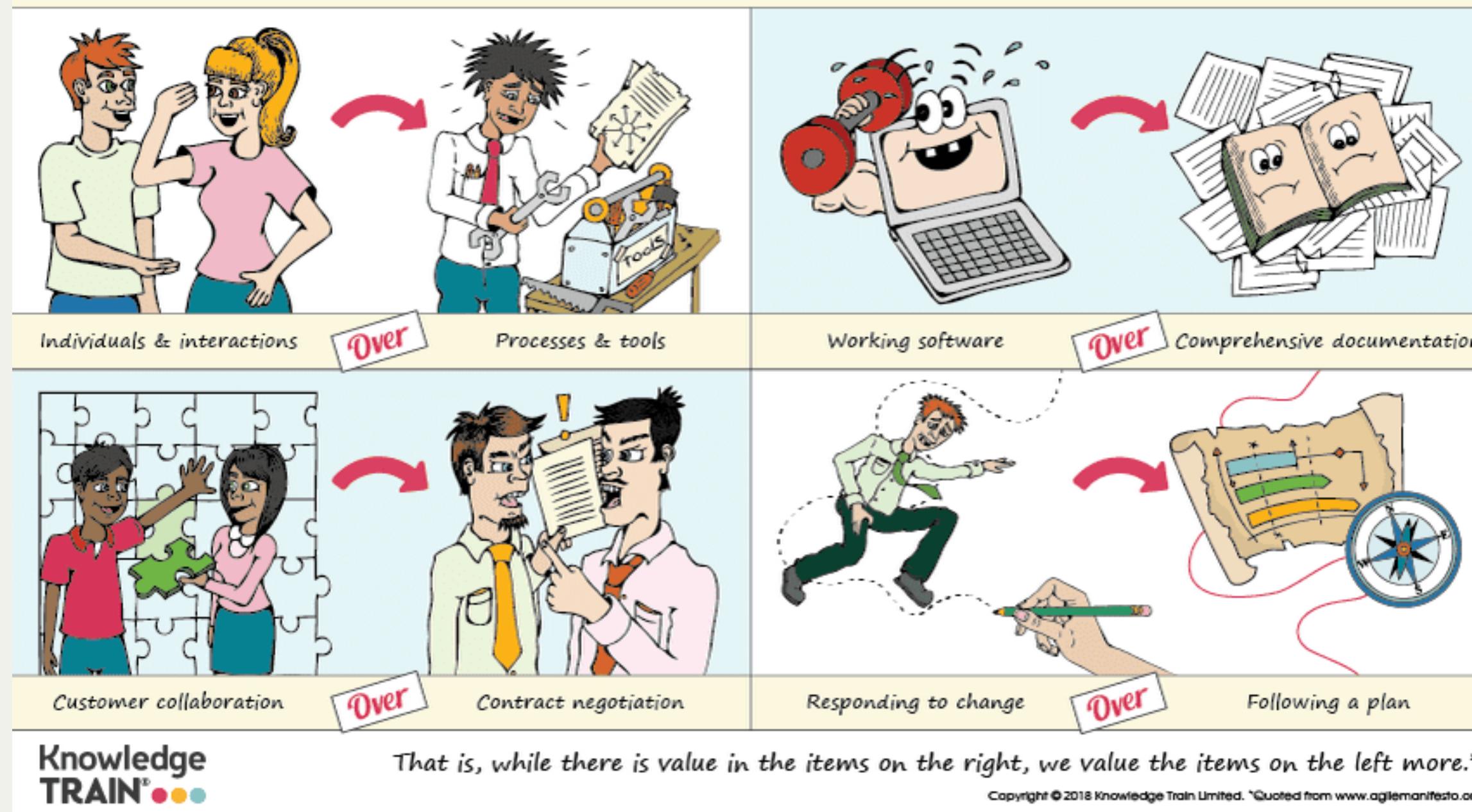
Manifest Agile

- <https://agilemanifesto.org/iso/fr/manifesto.html>

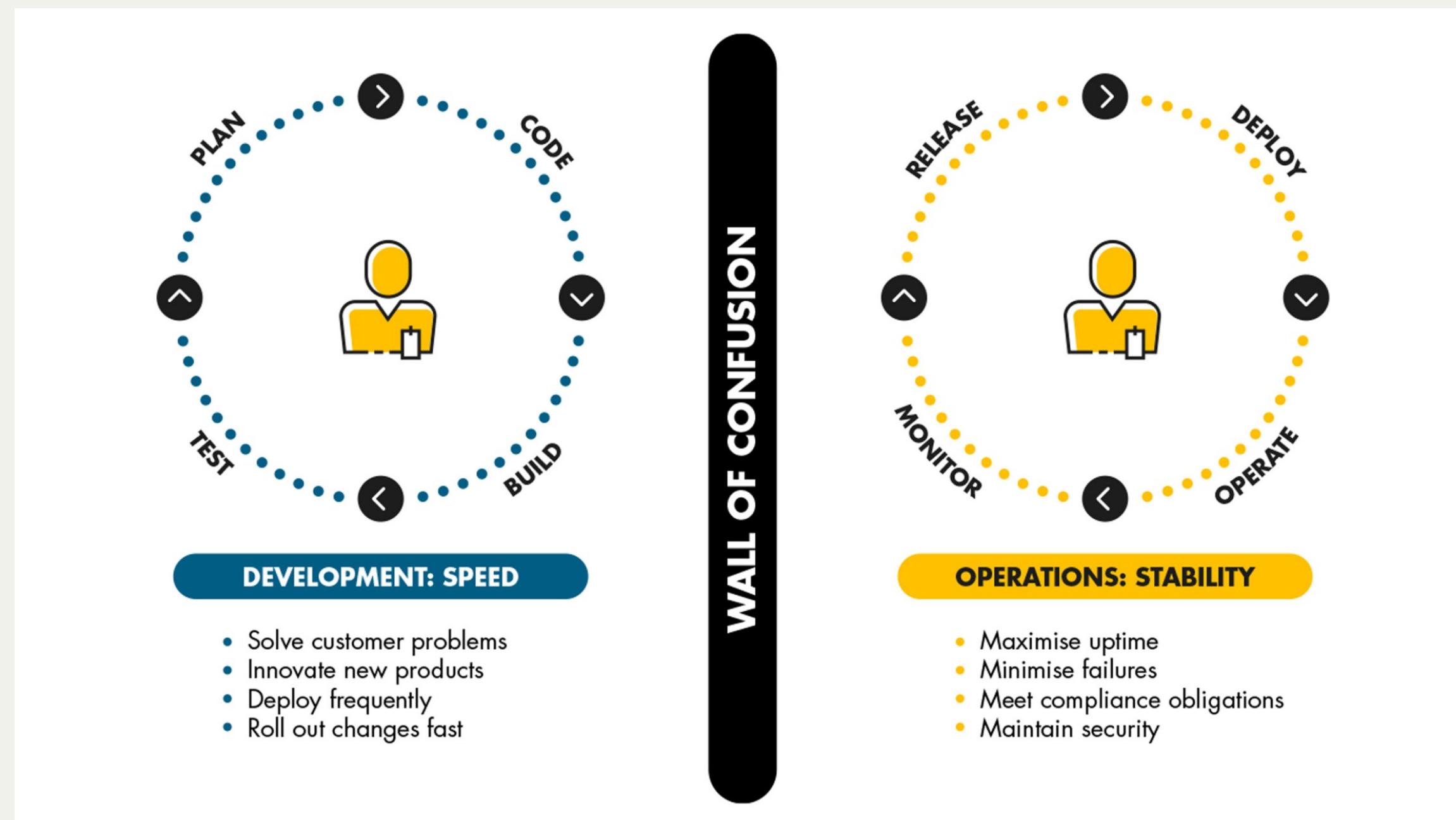
Manifesto for Agile Software Development*

"We are uncovering better ways of developing software by doing it and helping others do it.

Through this work we have come to value:



Le mur de la confusion



Source

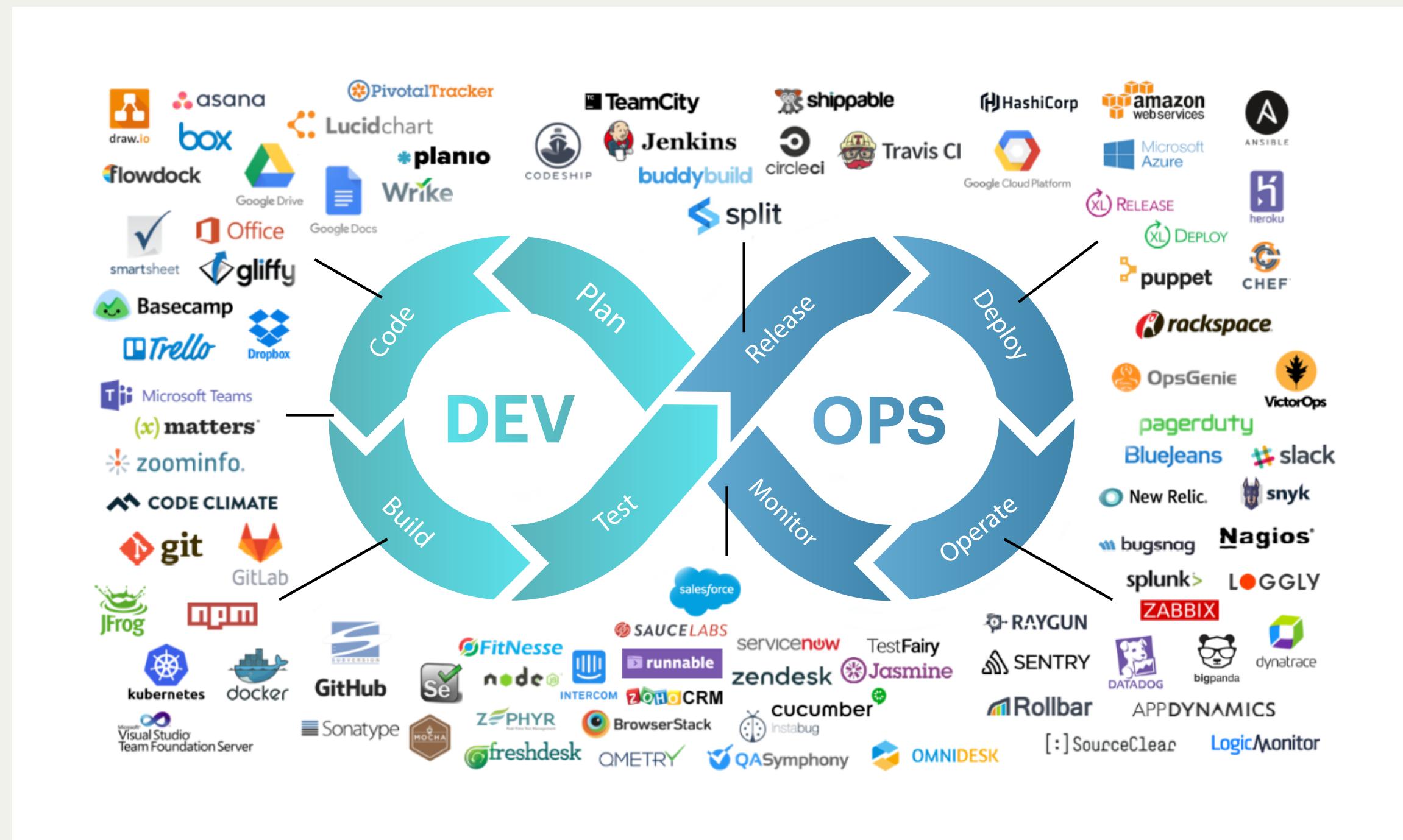
La clé : gérer le changement!

- Le changement ne doit pas être un événement, ça doit être la **norme**
- Faire en sorte que changer quelque chose soit:
 - Facile
 - Rapide
 - Stable
 - Sûr

Say Hello to DevOps



Heureusement, vous avez des outils à disposition !



Préparer votre environnement de travail

Outils Nécessaires



- Un navigateur web récent (et décent)
- Un compte sur GitHub

GitPod

GitPod.io : Environnement de développement dans le  "nuage"

- **But:** reproductible
- Puissance de calcul sur un serveur distant
- Éditeur de code VSCode dans le navigateur
- Gratuit pour 50h par mois ()

Démarrer avec GitPod



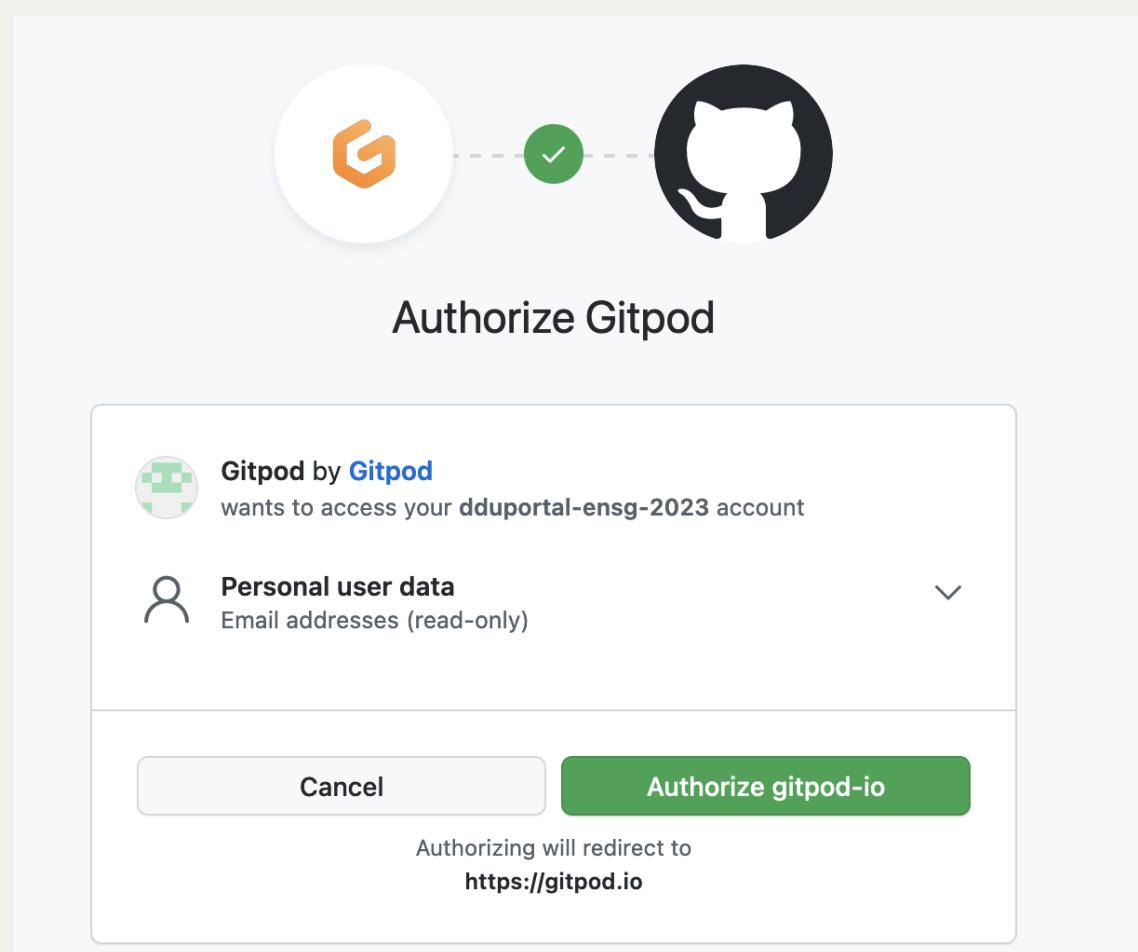
- Rendez vous sur <https://gitpod.io>
- Authentifiez vous en utilisant votre compte GitHub:
 - Bouton "Login" en haut à droite
 - Puis choisissez le lien " Continue with GitHub"

△ Pour les "autorisations", passez sur la slide suivante

Autorisations demandées par GitPod



Lors de votre première connexion, GitPod va vous demander l'accès (à accepter) à votre email public configuré dans GitHub :



▲ Passez à la slide suivante avant d'aller plus loin

Validation du Compte GitPod



GitPod vous demande votre numéro de téléphone mobile afin d'éviter les abus (service gratuit).
Saisissez un numéro de téléphone valide pour recevoir par SMS un code de déblocage :

User Validation Required

⚠ To use Gitpod you'll need to validate your account with your phone number. This is required to discourage and reduce abuse on Gitpod infrastructure.

Enter a mobile phone number you would like to use to verify your account. Having trouble? [Contact support](#)

Mobile Phone Number

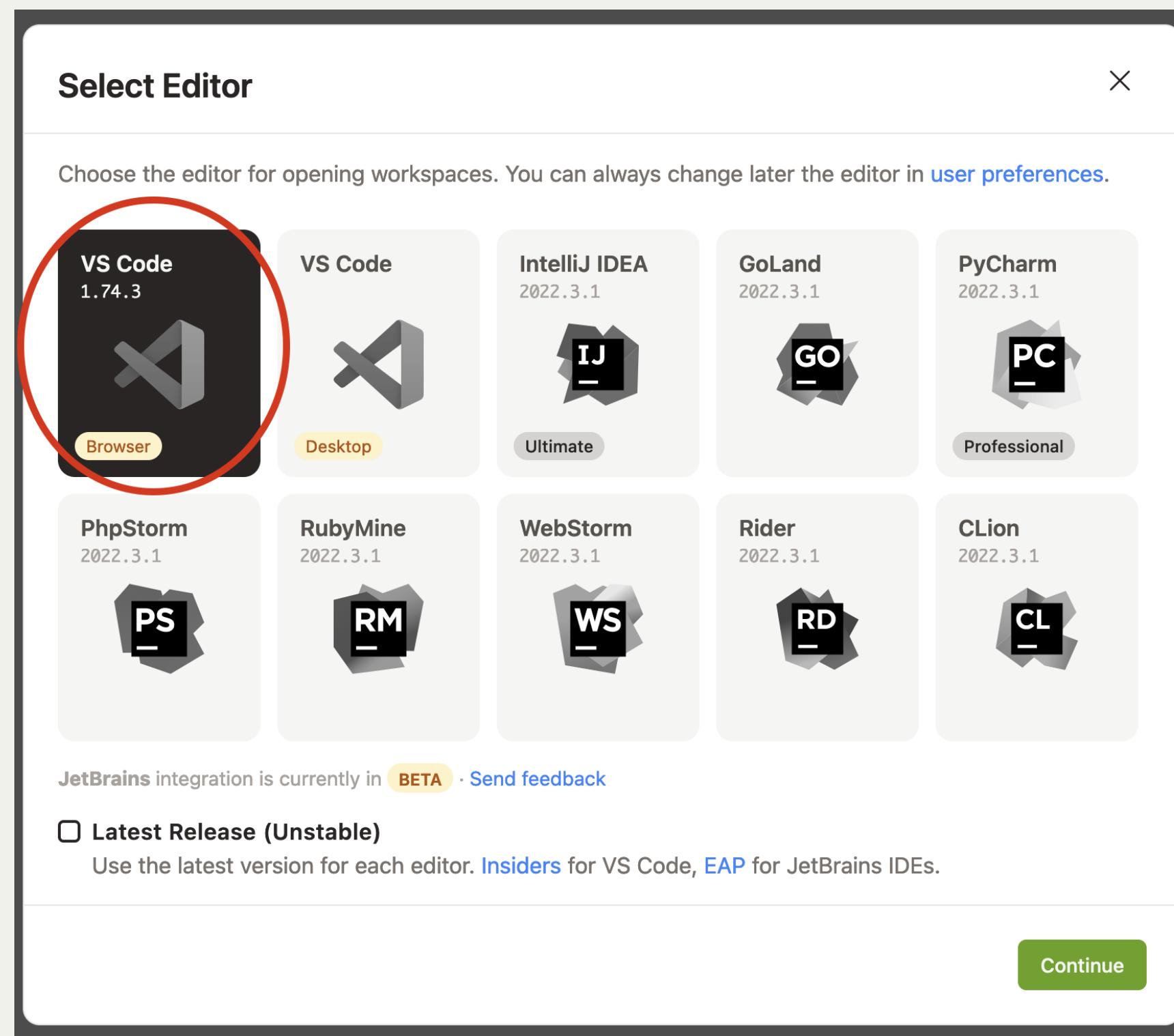
Send Code via SMS

▲ Passez à la slide suivante avant d'aller plus loin

Choix de l'Éditeur de Code



Choisissez l'éditeur "VSCode Browser" (la première tuile) :



▲ Passez à la slide suivante avant d'aller plus loin

Workspaces GitPod



- Vous arrivez sur la page listant les "workspaces" GitPod :
- Un workspace est une instance d'un environnement de travail virtuel (C'est un ordinateur distant)
- ⚠ Faites attention à réutiliser le même workspace tout au long de ce cours⚠

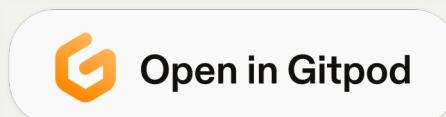
A screenshot of the GitPod web interface. At the top, there's a header with a logo, a 'Workspaces' button, a 'New Team' button, and a 'Feedback' button. Below the header, the title 'Workspaces' is displayed with the subtitle 'Manage recent and stopped workspaces.' A search bar labeled 'Filter Workspaces' and a limit selector 'Limit: 50' are present. A prominent green button labeled 'New Workspace' is located on the right. The main area shows two workspace entries: one named 'cicdlectures-gitpod-jcu32jcv4q2' which is active (indicated by a green dot) and another named 'cicdlectures-gitpod-vv8g7mywidp'. Each entry includes a GitHub URL, the branch name 'main', a status message 'No Changes', and a timestamp ('3 minutes ago' for the first and '4 minutes ago' for the second). Each entry also has a three-dot menu icon on the right.

Permissions GitPod <→ GitHub

- Pour les besoins de ce cours, vous devez autoriser GitPod à pouvoir effectuer certaines modifications dans vos dépôts GitHub
- Rendez-vous sur [la page des intégrations avec GitPod](#)
- Éditez les permissions de la ligne "GitHub" (les 3 petits points à droite) et sélectionnez uniquement :
 - user:email
 - public_repo
 - workflow

Démarrer l'environnement GitPod

Cliquez sur le bouton ci-dessous pour démarrer un environnement GitPod personnalisé:



Après quelques secondes (minutes?), vous avez accès à l'environnement:

- Gauche: navigateur de fichiers ("Workspace")
- Haut: éditeur de texte ("Get Started")
- Bas: Terminal interactif
- À droite en bas: plein de popups à ignorer (ou pas?)

Source disponible dans : <https://github.com/dduportal/esgi-gitpod>

Checkpoint

- Vous devriez pouvoir taper la commande `whoami` dans le terminal de GitPod:
 - Retour attendu: `gitpod`
- Vous devriez pouvoir fermer le fichier "Get Started" ...
 - ... et ouvrir le fichier `.gitpod.yml`

On peut commencer !

Ligne de commande

Guide de survie



Problématique

- Communication Humain <→ Machine
- Base commune de TOUS les outils

CLI

-  CLI == "Command Line Interface"
-  "Interface de Ligne de Commande"

REPL

Pour les théoriciens et curieux :

-  REPL == "Read–eval–print loop"

https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print_loop

Anatomie d'une commande

```
ls --color=always -l /bin
```

Copy

- Séparateur : l'espace
- Premier élément (`ls`) : c'est la commande
- Les éléments commençant par un tiret – sont des "options" et/ou drapeaux ("flags")
 - "Option" == "Optionnel"
- Les autres éléments sont des arguments (`/bin`)
 - Nécessaire (par opposition)

Manuel des commandes

- Afficher le manuel de <commande> :

```
man <commande> # Commande 'man' avec comme argument le nom de ladite commande
```

Copy

- Navigation avec les flèches haut et bas
 - Tapez / puis une chaîne de texte pour chercher
 - Touche n pour sauter d'occurrence en occurrence
- Touche q pour quitter



Essayez avec ls, chercher le mot color

- La majorité des commandes fournit également une option (--help), un flag (-h) ou un argument (help)
- Google c'est pratique aussi hein !

Raccourcis

Dans un terminal Unix/Linux/WSL :

- CTRL + C : Annuler le process ou prompt en cours
- CTRL + L : Nettoyer le terminal
- CTRL + A : Positionner le curseur au début de la ligne
- CTRL + E : Positionner le curseur à la fin de la ligne
- CTRL + R : Rechercher dans l'historique de commandes

🎓 Essayez-les !

Commandes de base 1/2

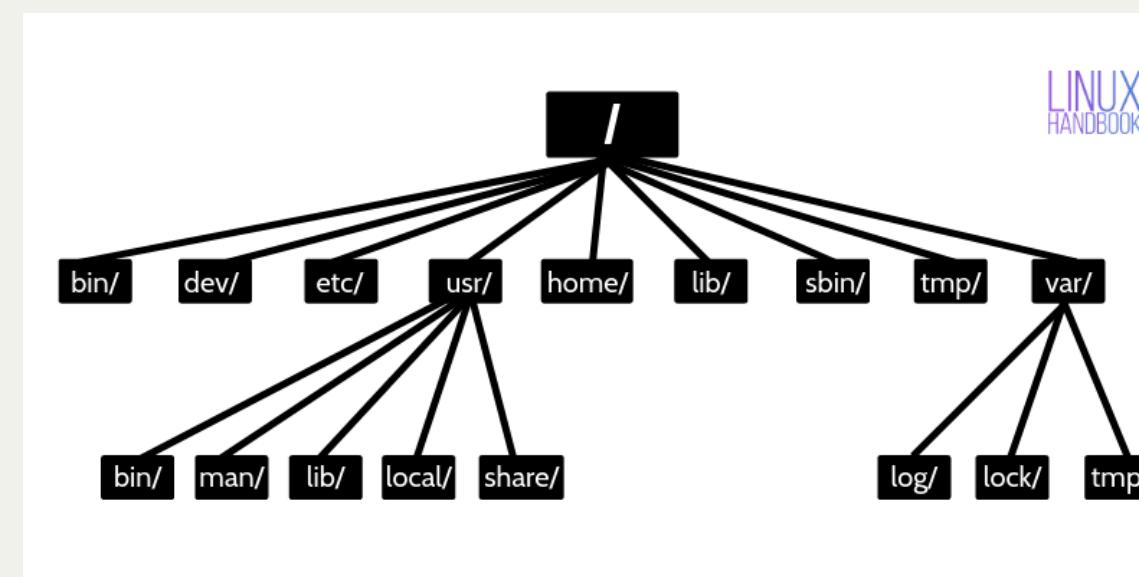
- `pwd` : Afficher le répertoire courant
 -  Option `-P` ?
- `ls` : Lister le contenu du répertoire courant
 -  Options `-a` et `-l` ?
- `cd` : Changer de répertoire
 -  Sans argument : que se passe t'il ?
- `cat` : Afficher le contenu d'un fichier
 -  Essayez avec plusieurs arguments
- `mkdir` : créer un répertoire
 -  Option `-p` ?

Commandes de base 2/2

- echo : Afficher un (des) message(s)
- rm : Supprimer un fichier ou dossier
- touch : Créer un fichier
- grep : Chercher un motif de texte

Arborescence de fichiers 1/2

- Le système de fichier a une structure d'arbre
 - La racine du disque dur c'est / : `ls -l /`
 - Le séparateur c'est également / : `ls -l /usr/bin`
- Deux types de chemins :
 - Absolu (depuis la racine): Commence par / (Ex. /usr/bin)
 - Sinon c'est relatif (e.g. depuis le dossier courant) (Ex . /bin ou local/bin/)



Source

Arborescence de fichiers 2/2

- Le dossier "courant" c'est . : `ls -l ./bin # Dans le dossier /usr`
- Le dossier "parent" c'est .. : `ls -l ../ # Dans le dossier /usr`
- ~ (tilde) c'est un raccourci vers le dossier de l'utilisateur courant : `ls -l ~`
- Sensible à la casse (majuscules/minuscules) et aux espaces : `ls`

```
ls -l /bin  
ls -l /Bin  
mkdir ~/\"Accent tué\"\nls -d ~/Accent\ tué
```

Copy

Un language (?)

- Variables interpolées avec le caractère "dollar" \$:

```
echo $MA_VARIABLE
echo "$MA_VARIABLE"
echo ${MA_VARIABLE}

# Recommendation
echo "${MA_VARIABLE}"

MA_VARIABLE="Salut tout le monde"

echo "${MA_VARIABLE}"
```

Copy

- Sous commandes avec \$ (<command>) :

```
echo ">> Contenu de /tmp :\n$(ls /tmp)"
```

Copy

- Des if, des for et plein d'autres trucs (<https://tldp.org/LDP/abs/html/>)

Codes de sortie

- Chaque exécution de commande renvoie un code de retour (🇬🇧 "exit code")
 - Nombre entier entre 0 et 255 (en **POSIX**)
- Code accessible dans la variable **éphémère** \$? :

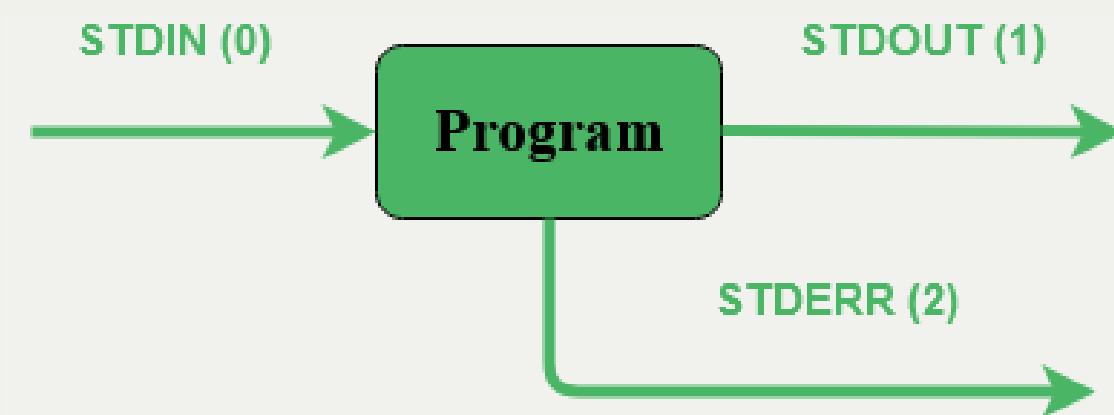
```
ls /tmp
echo $?

ls /do_not_exist
echo $?

# Une seconde fois. Que se passe-t'il ?
echo $?
```

Copy

Entrée, sortie standard et d'erreur



```
ls -l /tmp
echo "Hello" > /tmp/hello.txt
ls -l /tmp
ls -l /tmp >/dev/null
ls -l /tmp 1>/dev/null

ls -l /do_not_exist
ls -l /do_not_exist 1>/dev/null
ls -l /do_not_exist 2>/dev/null

ls -l /tmp /do_not_exist
ls -l /tmp /do_not_exist 1>/dev/null 2>&1
```

Copy

Pipelines

- Le caractère "pipe" | permet de chaîner des commandes
 - Le "stdout" de la première commande est branchée sur le "stdin" de la seconde
- Exemple : Afficher les fichiers/dossiers contenant le lettre d dans le dossier /bin :

```
ls -l /bin  
ls -l /bin | grep "d" --color=auto
```

Copy

Exécution 1/2

- Les commandes sont des fichiers binaires exécutables sur le système :

```
command -v cat # équivalent de "which cat"  
ls -l "$(command -v cat)"
```

Copy

- La variable d'environnement \$PATH liste les dossiers dans lesquels chercher les binaires
 -  Utiliser cette variable quand une commande fraîchement installée n'est pas trouvée

Exécution 2/2

- Exécution de scripts :
 - Soit appel direct avec l'interpréteur : sh ~/monscript.txt
 - Soit droit d'exécution avec un "shebang" (e.g. #!/bin/bash)

```
$ chmod +x ./monscript.sh  
$ head -n1 ./monscript.sh  
#!/bin/bash  
  
$ ./monscript.sh  
# Exécution
```

Copy

Git

Guide de survie



Problématique

- Support de communication
 - Humain → Machine
 - Humain <→ Humain
- Besoins de traçabilité, de définition explicite et de gestion de conflits
 - Collaboration requise pour chaque changement (revue, responsabilités)

Tracer le changement dans le code

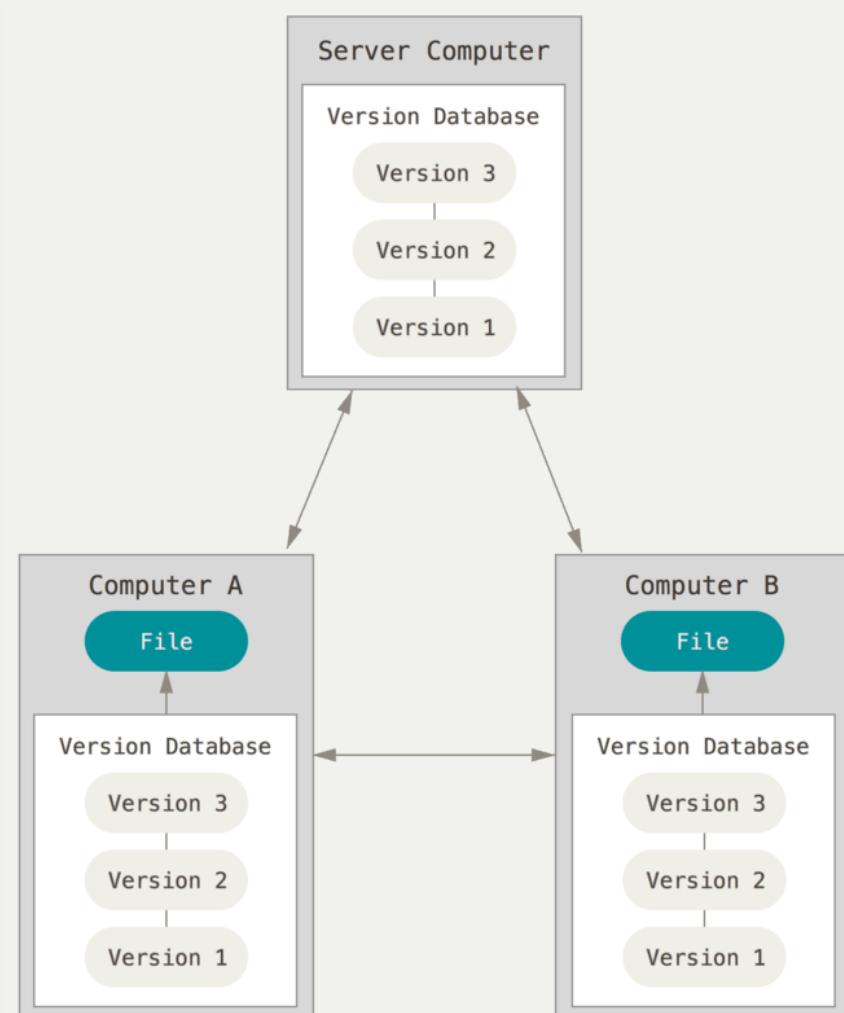
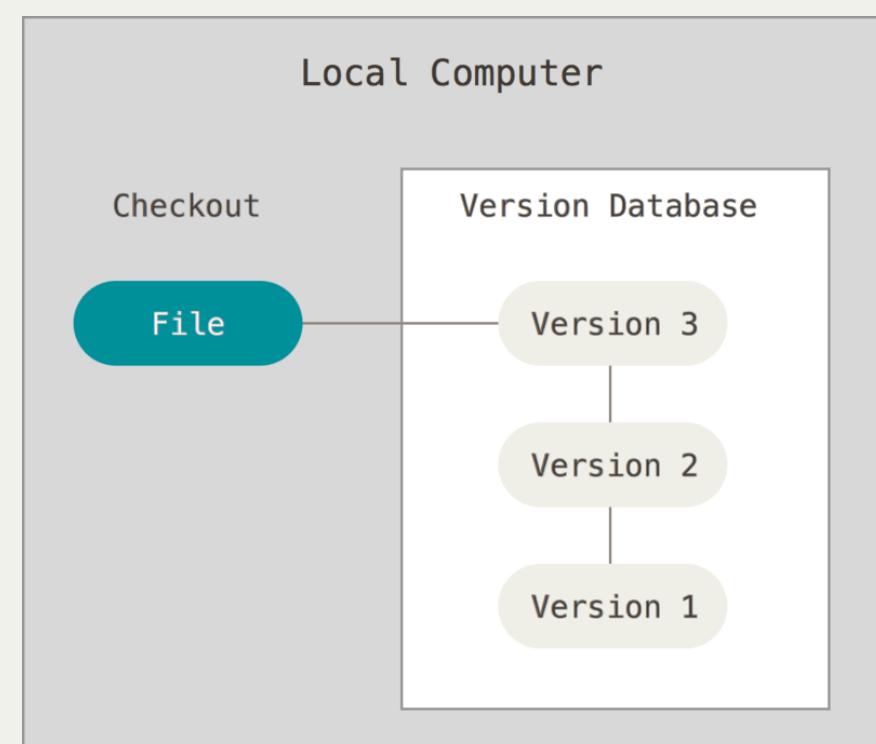
avec un **VCS** :  Version Control System

également connu sous le nom de SCM ( Source Code Management)

Pourquoi un VCS ?

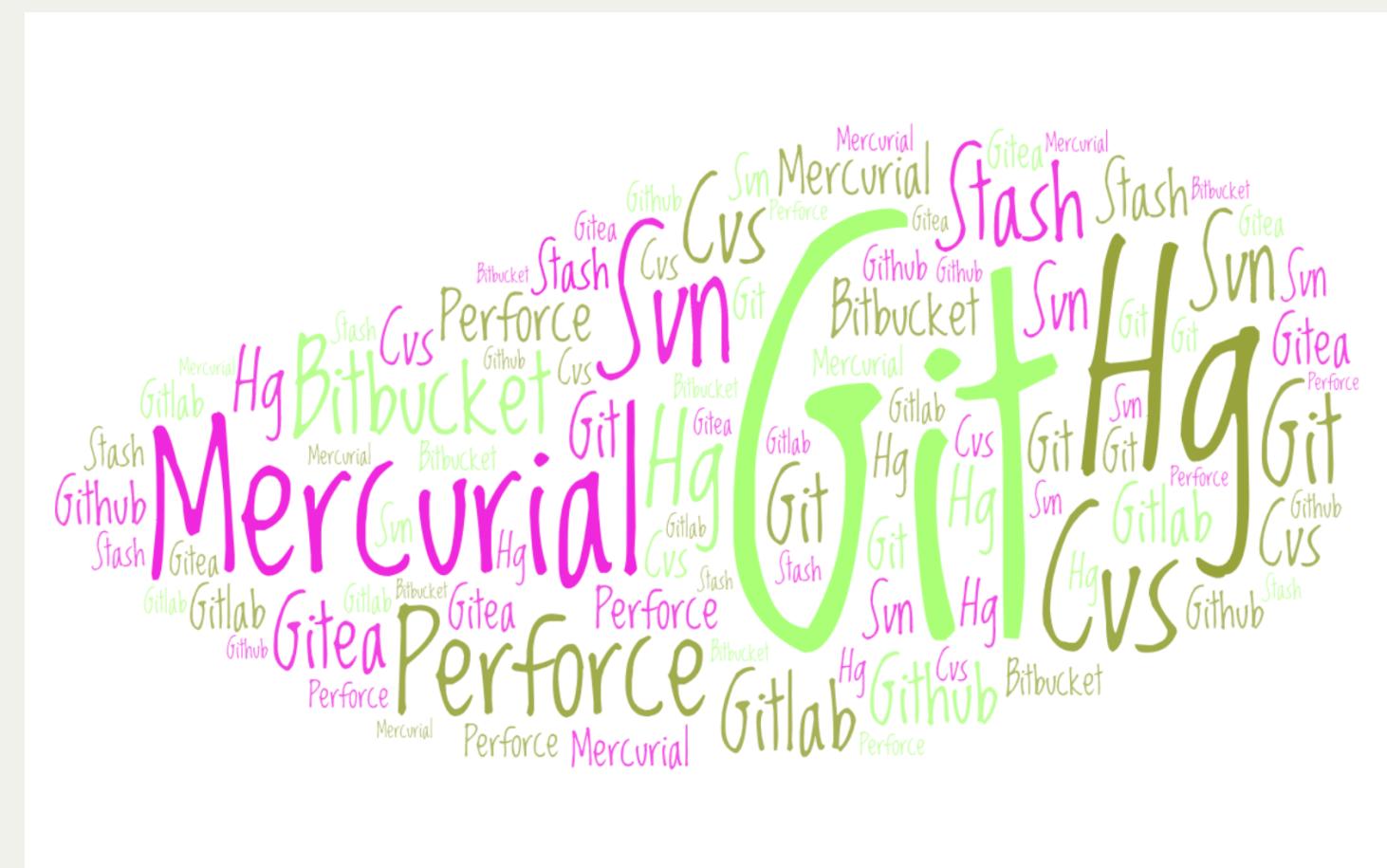
- Pour conserver une trace de **tous** les changements dans un historique
 - "Source unique de vérité" ( *Single Source of Truth*)
- Pour **collaborer** efficacement sur un même référentiel

Concepts des VCS



Source : <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>

Quel VCS utiliser ?



Nous allons utiliser **Git**

Git

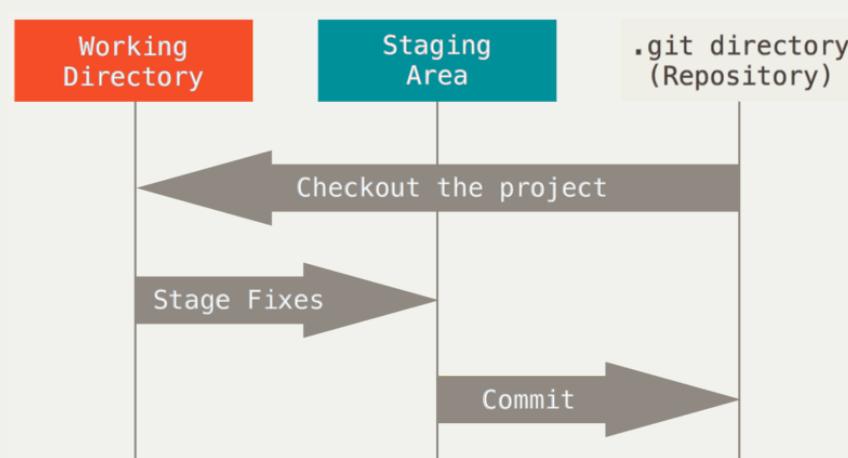
Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

<https://git-scm.com/>



Les 3 états avec Git

- L'historique ("Version Database") : dossier `.git`
- Dossier de votre projet ("Working Directory") - Commande
- La zone d'index ("Staging Area")



Source : https://git-scm.com/book/fr/v2/D%C3%A9marrage-rapide-Rudiments-de-Git#les_trois%C3%A9tats



Exercice avec Git - 1.1

- Dans le terminal de votre environnement GitPod:

- Créez un dossier vide nommé `projet-vcs-1` dans le répertoire `/workspace`, puis positionnez-vous dans ce dossier

```
mkdir -p /workspace/projet-vcs-1/  
cd /workspace/projet-vcs-1/
```

Copy

- Est-ce qu'il y a un dossier `.git/` ?
 - Essayez la commande `git status` ?
- Initialisez le dépôt git avec `git init`
 - Est-ce qu'il y a un dossier `.git/` ?
 - Essayez la commande `git status` ?

✓ Solution de l'exercice avec Git - 1.1

```
mkdir -p /workspace/projet-vcs-1/  
cd /workspace/projet-vcs-1/  
ls -la # Pas de dossier .git  
git status # Erreur "fatal: not a git repository"  
git init ./  
ls -la # On a un dossier .git  
git status # Succès avec un message "On branch master No commits yet"
```

Copy



Exercice avec Git - 1.2

- Créez un fichier README.md dedans avec un titre et vos nom et prénoms
 - Essayez la commande git status ?
- Ajoutez le fichier à la zone d'indexation à l'aide de la commande git add (...)
 - Essayez la commande git status ?
- Créez un commit qui ajoute le fichier README.md avec un message, à l'aide de la commande git commit -m <message>
 - Essayez la commande git status ?

✓ Solution de l'exercice avec Git - 1.2

```
echo "# Read Me\n\nObi Wan" > ./README.md
git status # Message "Untracked file"

git add ./README.md
git status # Message "Changes to be committted"
git commit -m "Ajout du README au projet"
git status # Message "nothing to commit, working tree clean"
```

Copy

Terminologie de Git - Diff et changeset

diff: un ensemble de lignes "changées" sur un fichier donné

```
10 cluster/addons/node-problem-detector/npd.yaml
...
@@ -26,28 +26,28 @@ subjects:
26   apiVersion: apps/v1
27   kind: DaemonSet
28 +   metadata:
29 -   name: npd-v0.8.0
30   namespace: kube-system
31   labels:
32     k8s-app: node-problem-detector
33 -   version: v0.8.0
34   kubernetes.io/cluster-service: "true"
35   addonmanager.kubernetes.io/mode: Recconcile
36   spec:
37     selector:
38       matchLabels:
39         k8s-app: node-problem-detector
40 -   version: v0.8.0
41   template:
42     metadata:
43       labels:
44         k8s-app: node-problem-detector
45 -   version: v0.8.0
46   kubernetes.io/cluster-service: "true"

26   apiVersion: apps/v1
27   kind: DaemonSet
28   metadata:
29 +   name: npd-v0.8.5
30   namespace: kube-system
31   labels:
32     k8s-app: node-problem-detector
33 +   version: v0.8.5
34   kubernetes.io/cluster-service: "true"
35   addonmanager.kubernetes.io/mode: Recconcile
36   spec:
37     selector:
38       matchLabels:
39         k8s-app: node-problem-detector
40 +   version: v0.8.5
41   template:
42     metadata:
43       labels:
44         k8s-app: node-problem-detector
45 +   version: v0.8.5
46   kubernetes.io/cluster-service: "true"
```

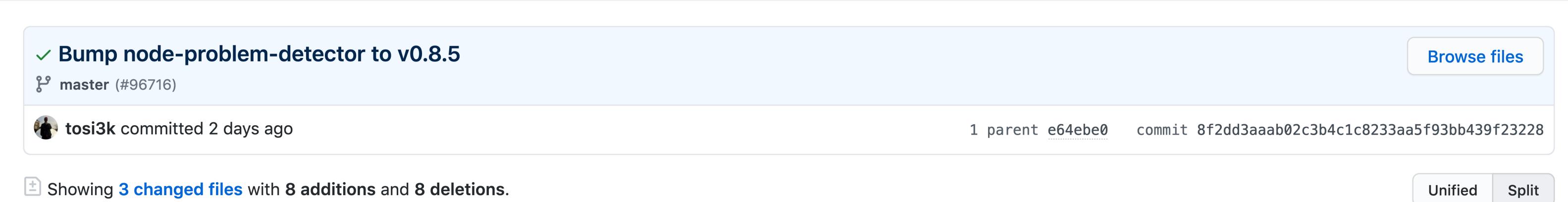
changeset: un ensemble de "diff" (donc peut couvrir plusieurs fichiers)

Showing 12 changed files with 314 additions and 200 deletions.

- > 3 Jenkinsfile
- > 10 make.ps1
- > 456 tests/plugins-cli.Tests.ps1
- > 2 tests/plugins-cli.bats
- > 2 tests/plugins-cli/Dockerfile-windows
- > 16 tests/plugins-cli/custom-war/Dockerfile-windows

Terminologie de Git - Commit

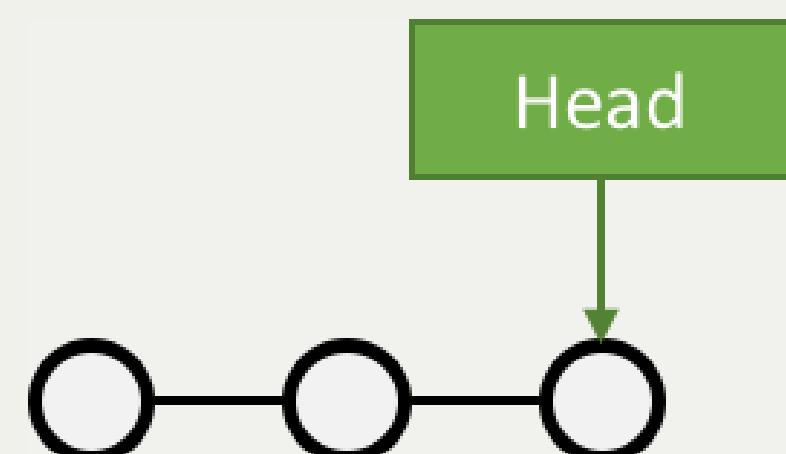
commit: un changeset qui possède un (commit) parent, associé à un message



A screenshot of a GitHub commit page for a commit titled "Bump node-problem-detector to v0.8.5". The commit was made by "tos13k" on the "master" branch. The commit message is "Bump node-problem-detector to v0.8.5". The commit has one parent, "e64ebe0", and a commit hash of "8f2dd3aaab02c3b4c1c8233aa5f93bb439f23228". Below the commit details, it says "Showing 3 changed files with 8 additions and 8 deletions." and includes "Unified" and "Split" diff options.

"HEAD": C'est le dernier commit dans l'historique

○ : a commit





Exercice avec Git - 2

- Afficher la liste des commits
- Afficher le changeset associé à un commit
- Modifier du contenu dans README .md et afficher le diff
- Annulez ce changement sur README .md

✓ Solution de l'exercice avec Git - 2

```
git log
git show # Show the "HEAD" commit
echo "# Read Me\n\nObi Wan Kenobi" > ./README.md

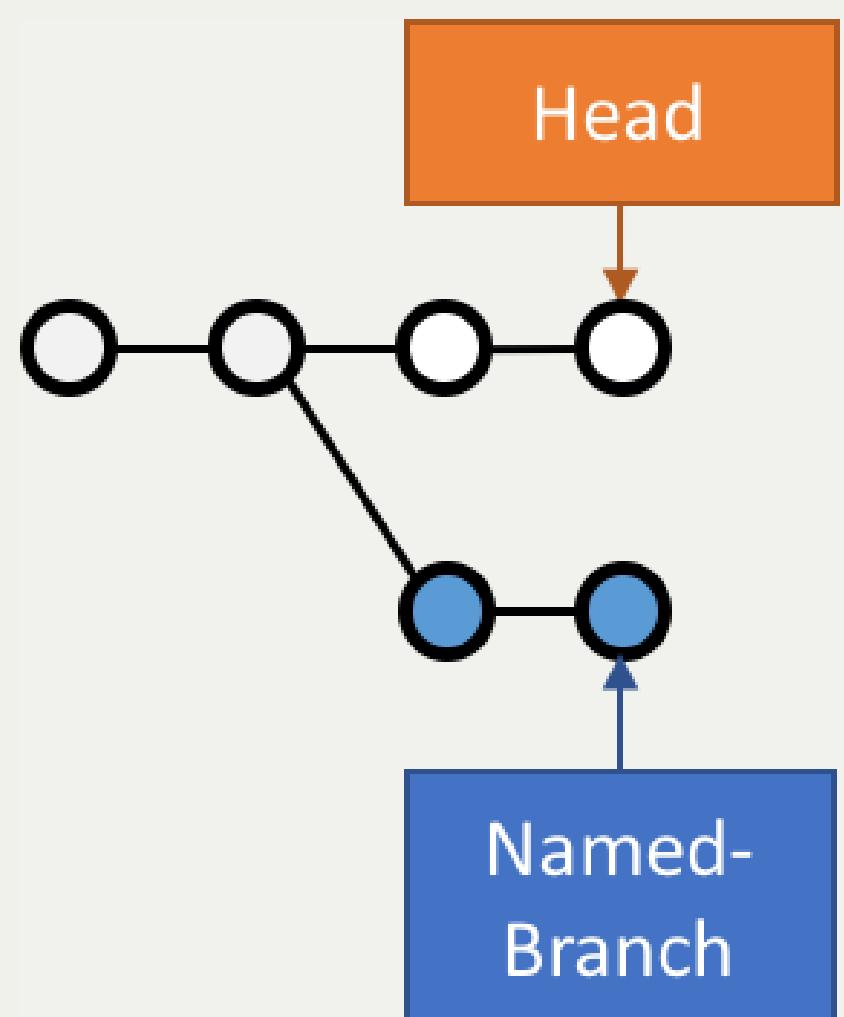
git diff
git status

git checkout -- README.md
git status
```

Copy

Terminologie de Git - Branche

- Abstraction d'une version "isolée" du code
- Concrètement, une **branche** est un alias pointant vers un "commit"





Exercice avec Git - 3

- Créer une branche nommée `feature/html`
- Ajouter un nouveau commit contenant un nouveau fichier `index.html` sur cette branche
- Afficher le graphe correspondant à cette branche avec `git log --graph`

✓ Solution de l'exercice avec Git - 3

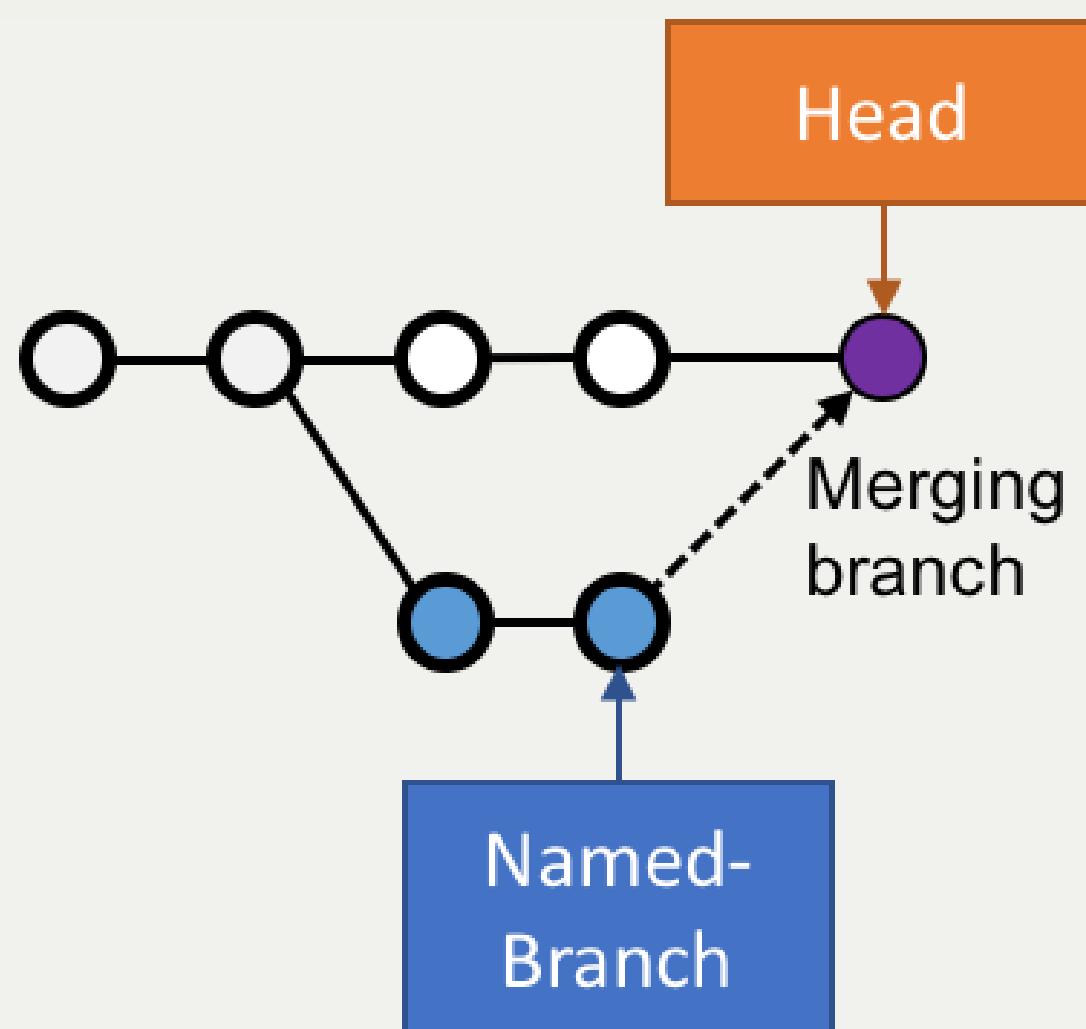
```
git branch feature/html && git switch feature/html
# Ou git switch --create feature/html
echo '<h1>Hello</h1>' > ./index.html
git add ./index.html && git commit --message="Ajout d'une page HTML par défaut" # -m / --message

git log
git log --graph
git lg # cat ~/.gitconfig => regardez la section section [alias], cette commande est déjà définie!
```

Copy

Terminologie de Git - Merge

- On intègre une branche dans une autre en effectuant un **merge**
 - Un nouveau commit est créé, fruit de la combinaison de 2 autres commits





Exercice avec Git - 4

- Merger la branche `feature/html` dans la branche principale
 - Δ Pensez à utiliser l'option `--no-ff`
- Afficher le graphe correspondant à cette branche avec `git log --graph`

✓ Solution de l'exercice avec Git - 4

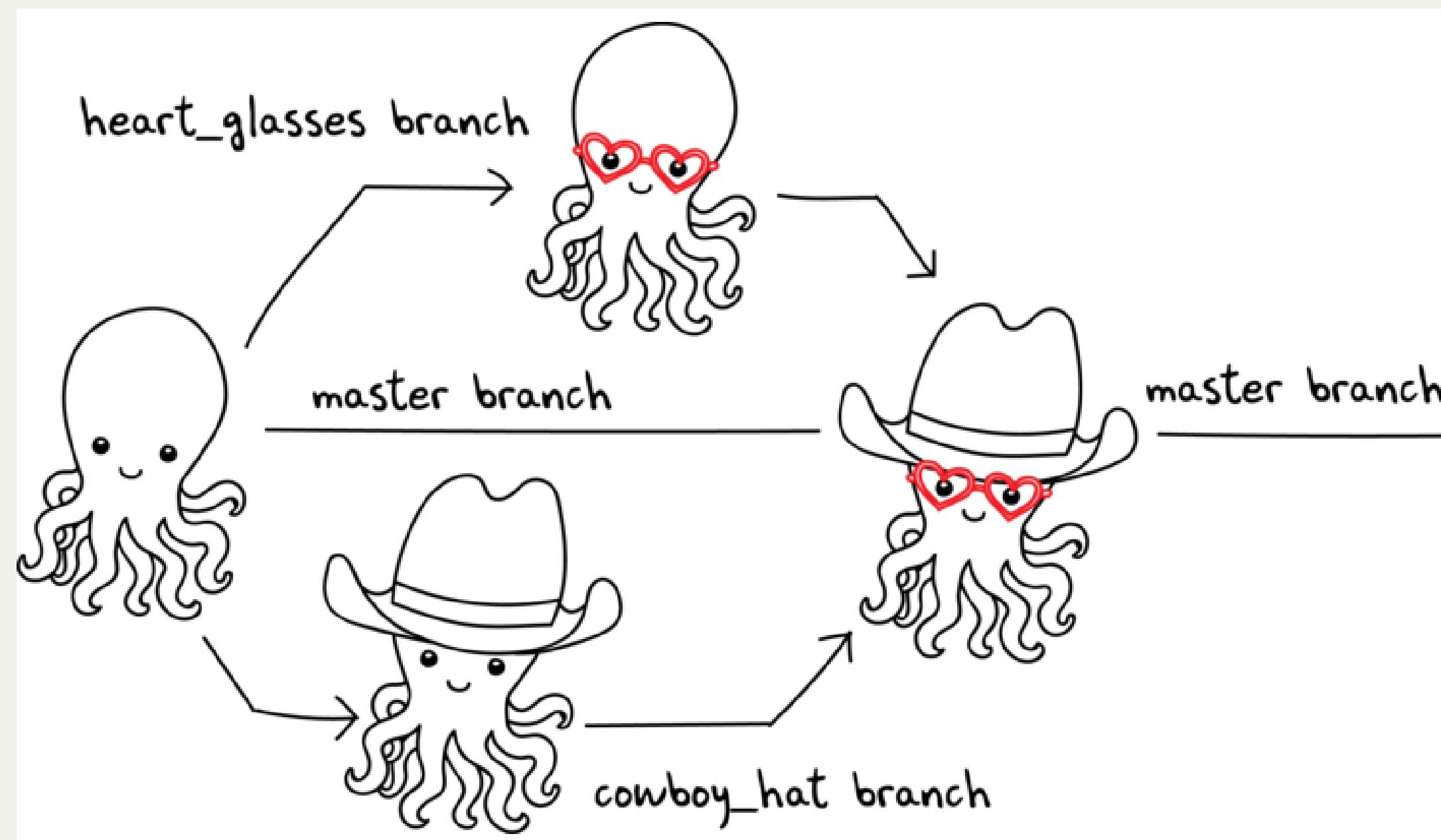
```
git switch main
git merge --no-ff feature/html # Enregistrer puis fermer le fichier 'MERGE_MSG' qui a été ouvert
git log --graph

# git lg
```

Copy

Feature Branch Flow

- **Une seule branche par fonctionnalité**



Exemple d'usages de VCS

- "Infrastructure as Code" :
 - Besoins de traçabilité, de définition explicite et de gestion de conflits
 - Collaboration requise pour chaque changement (revue, responsabilités)
- Code Civil:
 - <https://github.com/steeve/france.code-civil>
 - <https://github.com/steeve/france.code-civil/pull/40>
 - <https://github.com/steeve/france.code-civil/commit/b805ecf05a86162d149d3d182e04074ecf72c066>

Checkpoint

- git est un des (plus populaires) de système de contrôle de versions
 - Cet outil vous permet:
 - D'avoir un historique auditable de votre code source
 - De collaborer efficacement sur le code source (conflit git == "PARLEZ-VOUS")
- ⇒ C'est une ligne de commande (trop?) complète qui nécessite de pratiquer

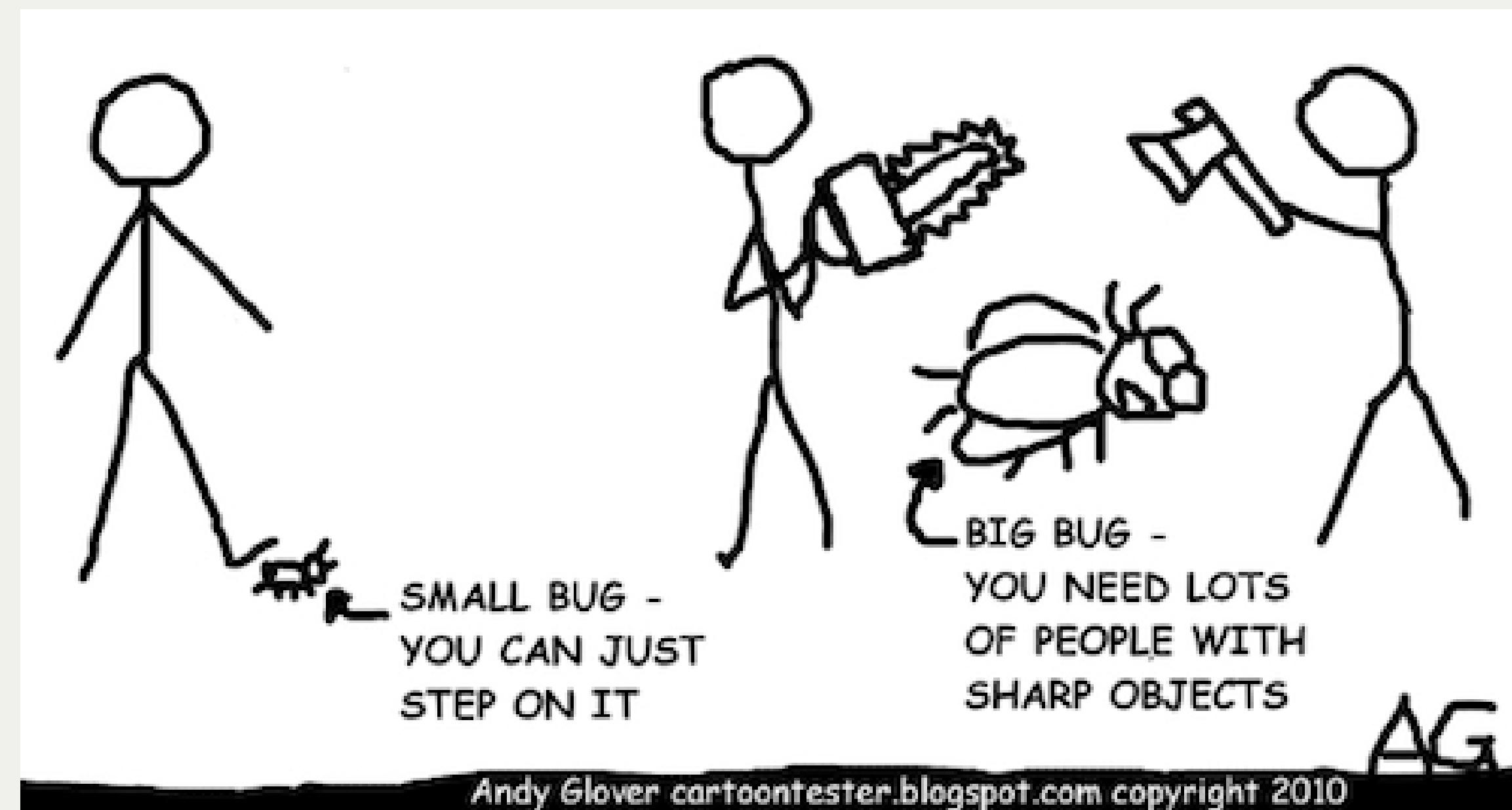
Intégration Continue (CI)

Continuous Integration doesn't get rid of bugs, but it does make them dramatically easier to find and remove.

— Martin Fowler

Pourquoi la CI ?

But : Déetecter les fautes au plus tôt pour en limiter le coût



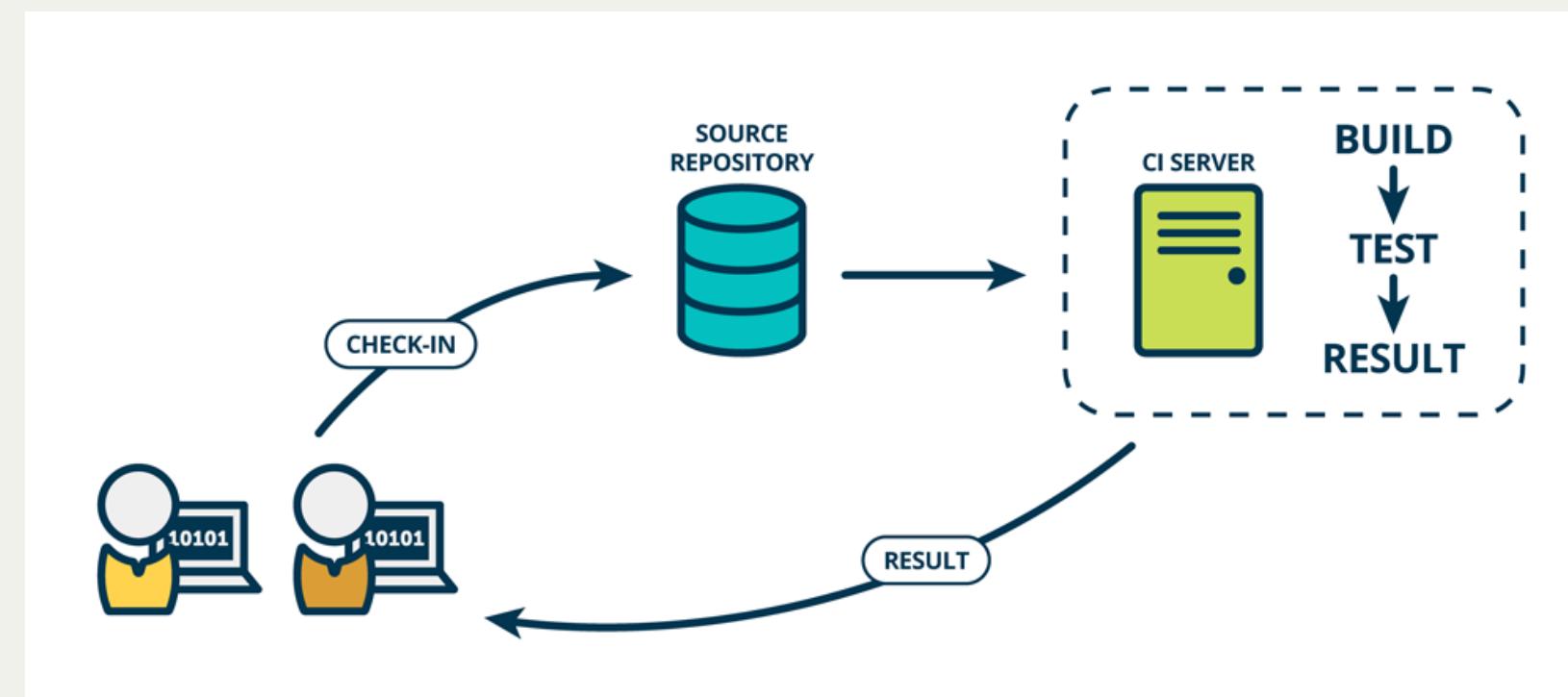
Source : <http://cartoontester.blogspot.be/2010/01/big-bugs.html>

Qu'est ce que l'Intégration Continue ?

Objectif : que l'intégration de code soit un *non-événement*

- Construire et intégrer le code **en continu**
- Le code est intégré **souvent** (au moins quotidiennement)
- Chaque intégration est validée par une exécution **automatisée**

Et concrètement ?



- Un•e dévelopeu•se•r ajoute du code/branche/PR :
 - une requête HTTP est envoyée au système de "CI"
- Le système de CI compile et teste le code
- On ferme la boucle : Le résultat est renvoyé au dévelopeu•se•r•s

Quelques moteurs de CI connus

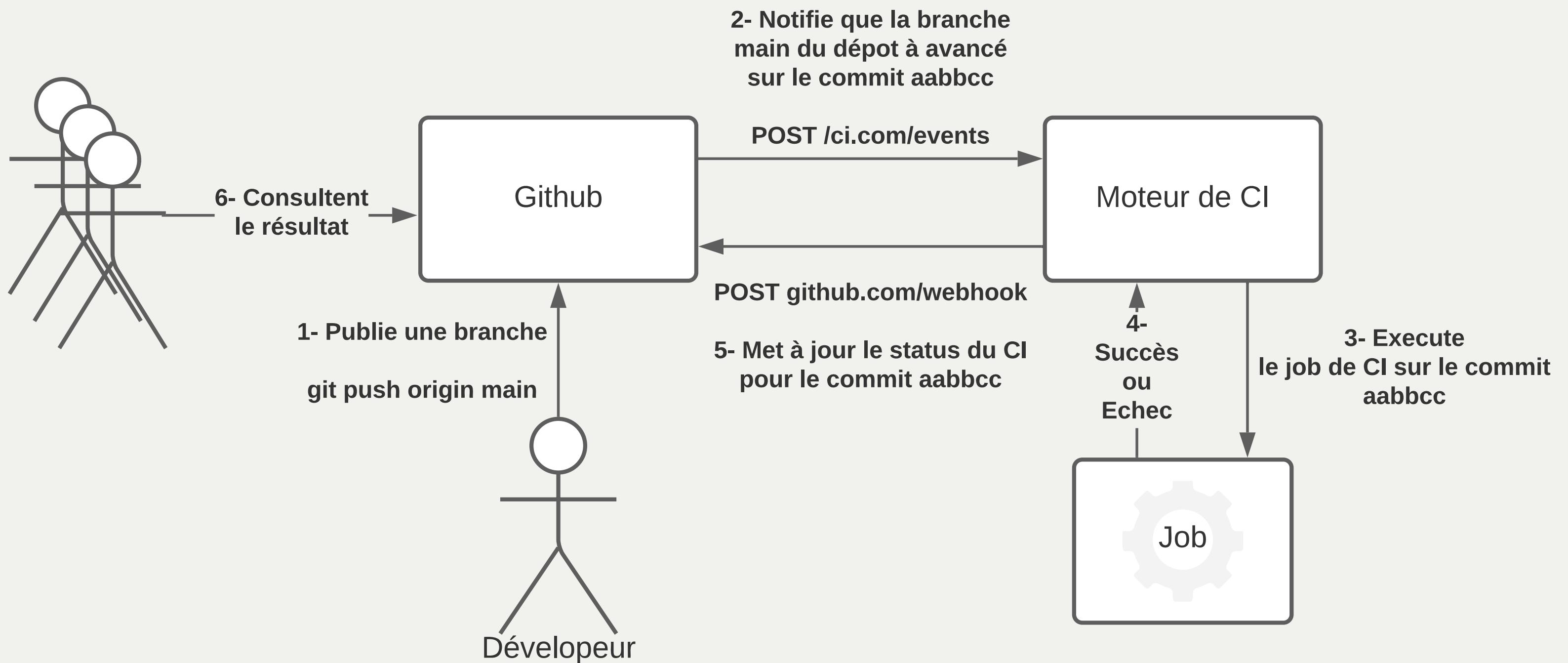
- A héberger soit-même : Jenkins, GitLab, Drone CI, CDS...
- Hébergés en ligne : Travis CI, Semaphore CI, Circle CI, Codefresh, GitHub Actions

GitHub Actions

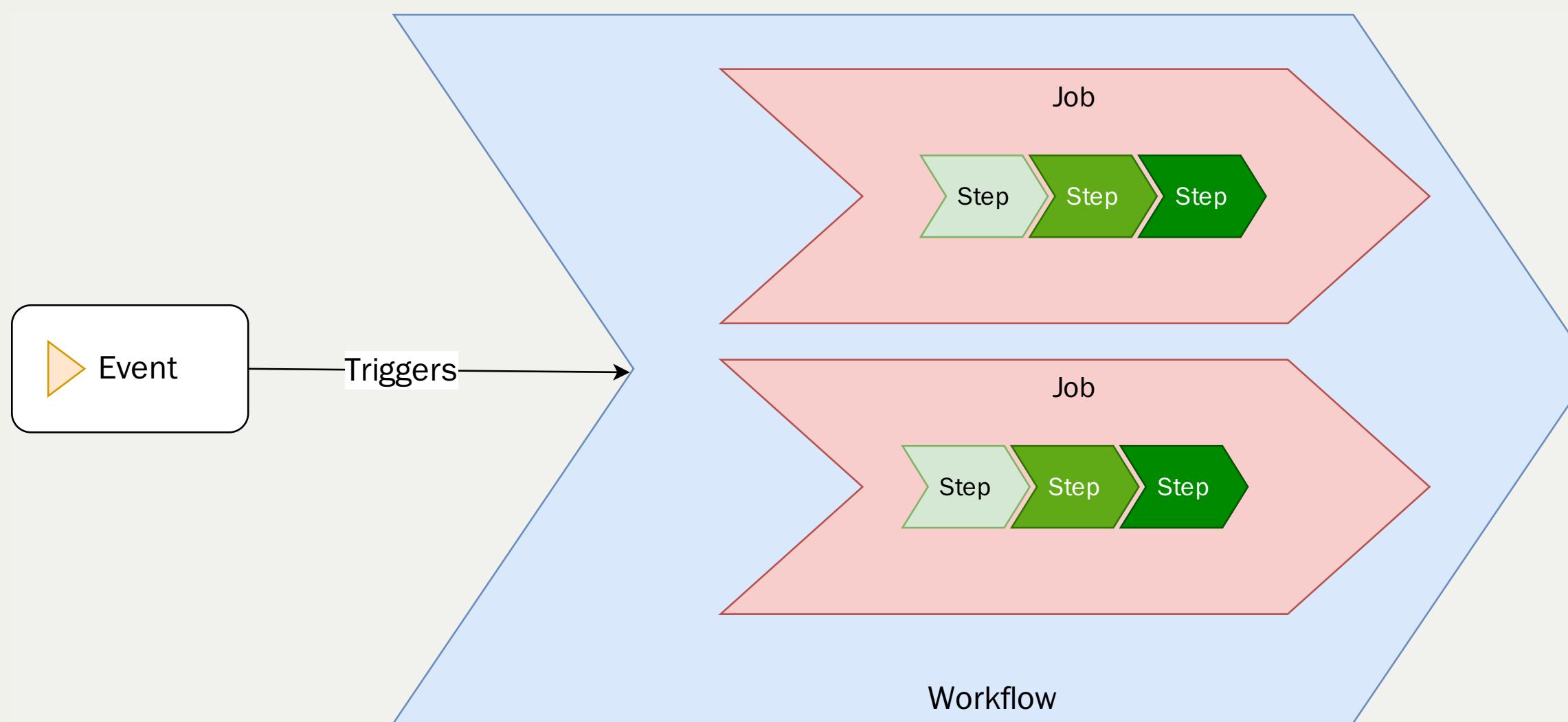
GitHub Actions est un moteur de CI/CD intégré à GitHub

- ✓ : Très facile à mettre en place, gratuit et intégré complètement
- ✗ : Utilisable uniquement avec GitHub, et DANS la plateforme GitHub

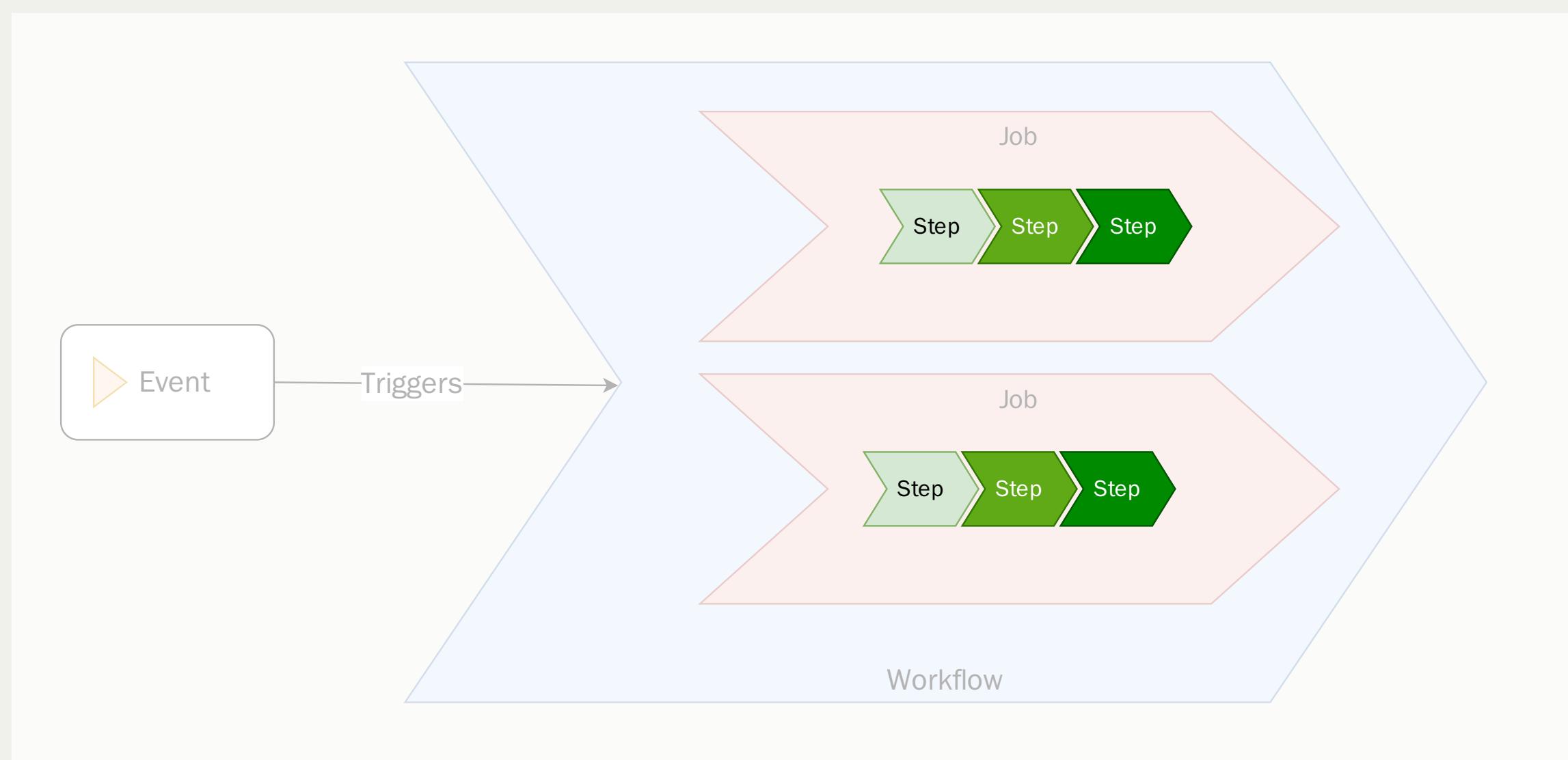
Anatomie de déclenchement de GitHub Actions



Concepts de GitHub Actions



Concepts de GitHub Actions - Step 1/2



Concepts de GitHub Actions - Step 2/2

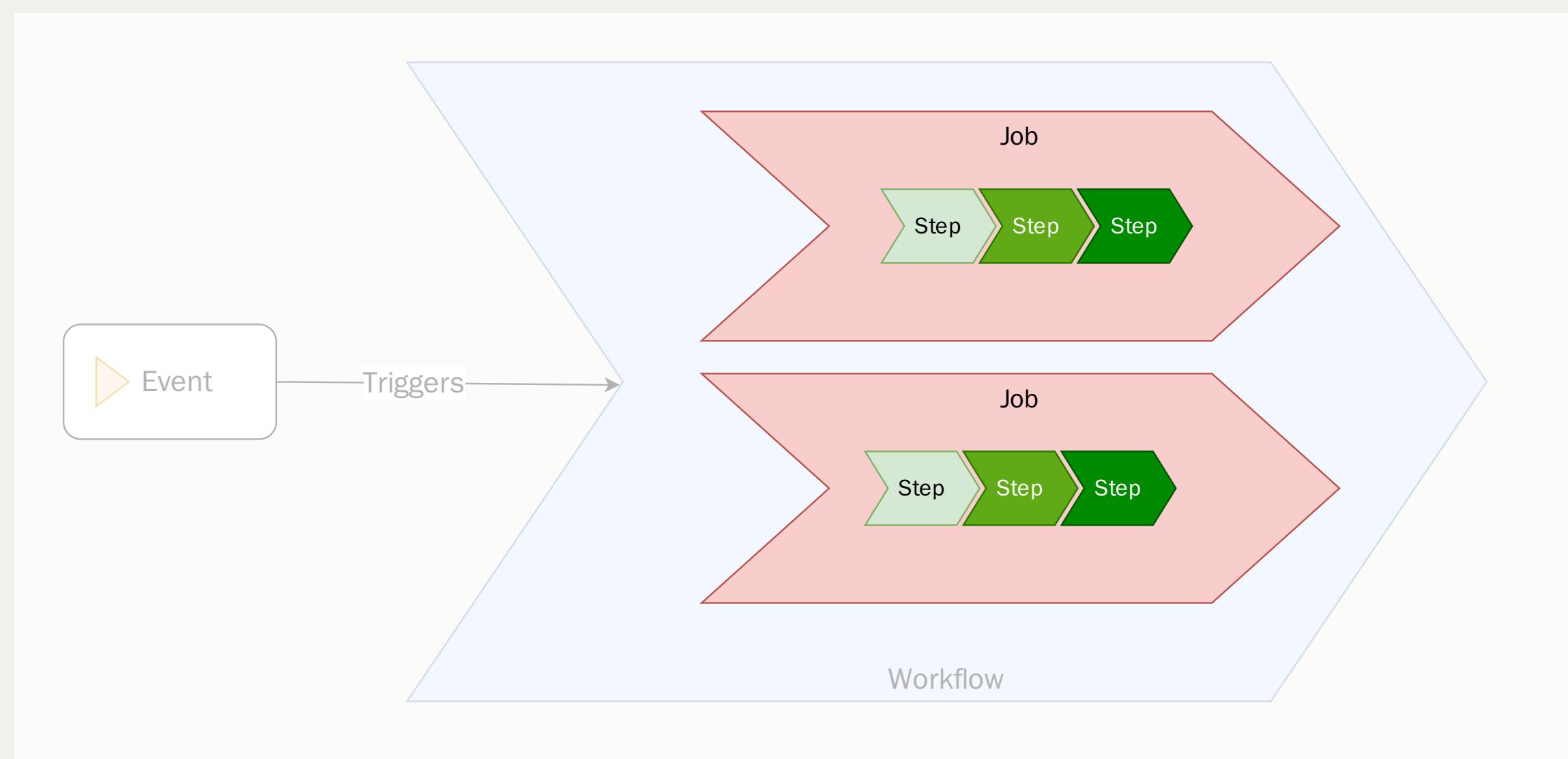
Une **Step** (étape) est une tâche individuelle à faire effectuer par le CI :

- Par défaut c'est une commande à exécuter - mot clef `run`
- Ou une "action" (quel est le nom du produit déjà ?) - mot clef `uses`
 - Réutilisables et partageables

```
steps: # Liste de steps
  # Exemple de step 1 (commande)
  - name: Say Hello
    run: echo "Hello ESGI"
  # Exemple de step 2 (une action)
  - name: 'Login to DockerHub'
    uses: docker/login-action@v2 # https://github.com/marketplace/actions/docker-login
    with:
      username: ${{ secrets.DOCKERHUB_USERNAME }}
      password: ${{ secrets.DOCKERHUB_TOKEN }}
```

Copy

Concepts de GitHub Actions - Job 1/2



Concepts de GitHub Actions - Job 2/2

Un **Job** est un groupe logique de tâches :

- Enchaînement *séquentiel* de tâches
- Regroupement logique : "qui a un sens"
 - Exemple : "compiler puis tester le résultat de la compilation"

```
jobs: # Map de jobs
  build: # 1er job, identifié comme 'build'
    name: 'Build Slides'
    runs-on: ubuntu-22.04 # cf. prochaine slide "Concepts de GitHub Actions - Runner"
    steps: # Collection de steps du job
      - name: 'Build the JAR'
        run: mvn package
      - name: 'Run Tests on the JAR file'
        run: mvn verify
  deploy: # 2nd job, identifié comme 'deploy'
    # ...
```

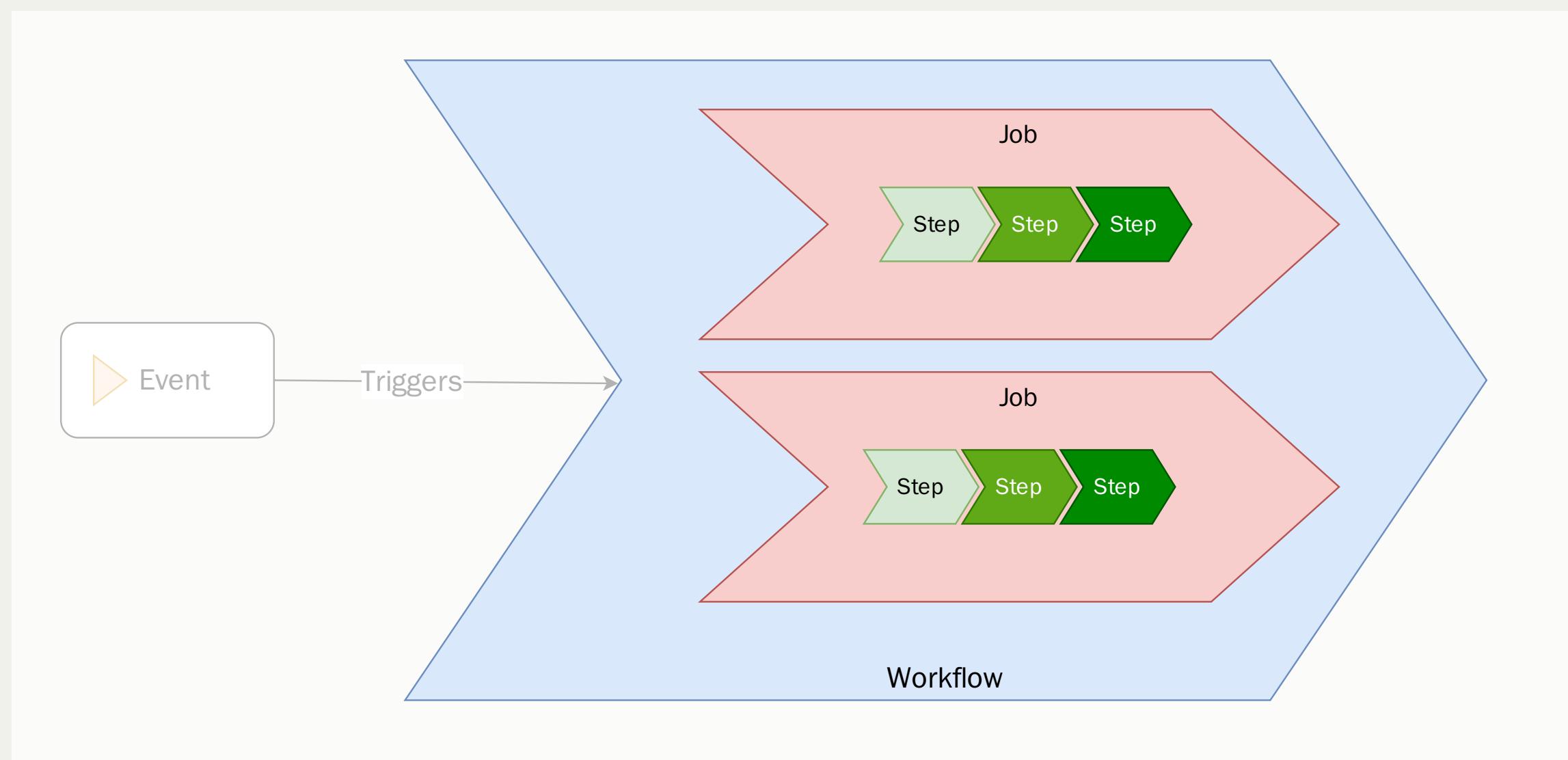
Copy

Concepts de GitHub Actions - Runner

Un **Runner** est un serveur distant sur lequel s'exécute un job.

- Mot clef `runs-on` dans la définition d'un job
- Défaut : machine virtuelle Ubuntu dans le cloud utilisé par GitHub
- D'autres types sont disponibles (macOS, Windows, etc.)
- Possibilité de fournir son propre serveur

Concepts de GitHub Actions - Workflow 1/2



Concepts de GitHub Actions - Workflow 2/2

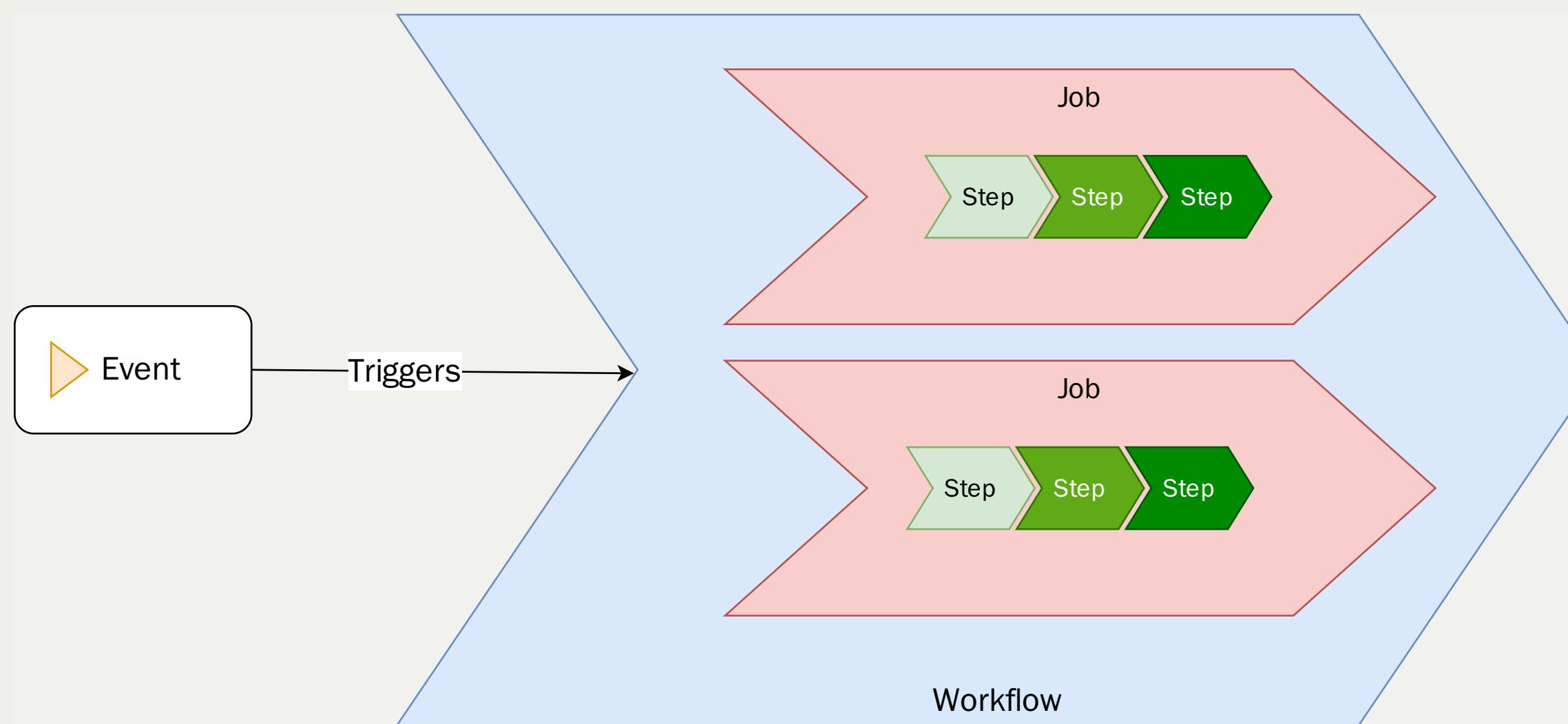
Un **Workflow** est une procédure automatisée composée de plusieurs jobs, décrite par un fichier **YAML**.

- On parle de "Workflow/Pipeline as Code"
- Chemin : `.github/workflows/<nom du workflow>.yml`
- On peut avoir *plusieurs* fichiers donc *plusieurs* workflows

```
.github/workflows
├── ci-cd.yaml
└── bump-dependency.yaml
└── nightly-tests.yaml
```

Copy

Concepts de GitHub Actions - Évènement 1/2



Concepts de GitHub Actions - Évènement 2/2

Un **évenement** du projet GitHub (push, merge, nouvelle issue, etc.) déclenche l'exécution du workflow

- Plein de type d'évènements : push, issue, alarme régulière, favori, fork, etc.
 - Exemple : "Nouveau commit poussé", "chaque dimanche à 07:00", "une issue a été ouverte" ...
- Un workflow spécifie le(s) évènement(s) qui déclenche(nt) son exécution
 - Exemple : "exécuter le workflow lorsque un nouveau commit est poussé ou chaque jour à 05:00 par défaut"

Concepts de GitHub Actions : Exemple Complet

Workflow File :

```
name: Node.js CI
on: # Évènements déclencheurs
  - push:
    branch: main # Lorsqu'un nouveau commit est poussé sur la branche "main"
  - schedule:
    cron: */15 * * * * # Toutes les 15 minutes
jobs:
  test-linux:
    runs-on: ubuntu-22.04
    steps:
      - uses: actions/checkout@v3
      - run: npm install
      - run: npm test
  test-mac:
    runs-on: macos-12
    steps:
      - uses: actions/checkout@v3
      - run: npm install
      - run: npm test
```

Copy

Essayons GitHub Actions

- **But :** nous allons créer notre premier workflow dans GitHub Actions
- N'hésitez pas à utiliser la documentation de GitHub Actions:
 - Accueil
 - Quickstart
 - Référence



Exercice: Créez un dépôt (git dans) GitHub

- En étant authentifié dans GitHub,
- Créez un nouveau dépôt nommé esgi-devops-2023
 - Pas de "template" (modèle)
 - Visibilité publique
 - Initialisation avec un fichier README.md



Exercice: Récupérez le dépôt dans Gitpod

- Obtenez l'URL (HTTPS) du dépôt GitHub fraîchement créé
 - ⚡ Depuis la page du dépôt, cliquez sur le bouton vert intitulé "**Code**"
- Dans Gitpod,
 - Depuis un terminal, positionnez-vous dans le dossier `/workspace`,
 - Clonez le dépôt avec la commande `git clone https://github.com/xxx/esgi-devops-2023`
 - ⚡ Si le dépôt n'apparaît pas dans l'"Explorer" à gauche : `code -a /workspace/esgi-devops-2023`



Exercice: Exemple simple avec GitHub Actions

- Dans le dépôt esgi-devops-2023, sur la branch main,
 - Créez le fichier `.github/workflows/bonjour.yml` avec le contenu suivant :

```
name: Bonjour
on:
  - push
jobs:
  dire_bonjour:
    runs-on: ubuntu-22.04
    steps:
      - run: echo "Bonjour 🎉"
```

Copy

- Commitez puis poussez le fichier sur GitHub:

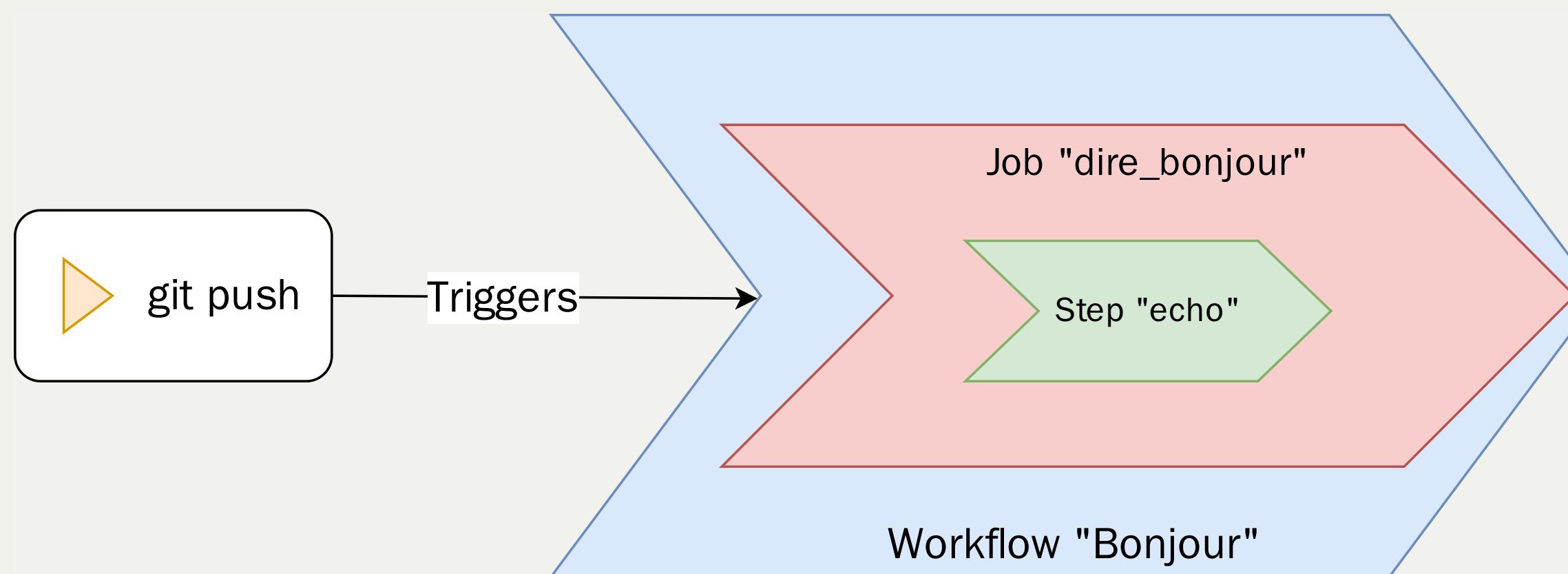
```
git add .github/workflows/bonjour.yml
git commit -m 'Create GitHub workflow Bonjour'
git push origin main
```

Copy



Exercice: Exemple simple avec GitHub Actions : Récapète

- Revenez sur la page GitHub de votre projet et naviguez dans l'onglet "Actions" :
 - Voyez-vous un workflow ? Et un Job ? Et le message affiché par la commande echo ?



Exemple GitHub Actions : Checkout

- Supposons que l'on souhaite utiliser le code du dépôt...
 - Essayez: modifiez le fichier `bonjour.yml` pour afficher le contenu de `README.md`:

```
name: Bonjour
on:
  - push
jobs:
  dire_bonjour:
    runs-on: ubuntu-22.04
    steps:
      - run: ls -l # Liste les fichiers du répertoire courant
      - run: cat README.md # Affiche le contenu du fichier `README.md` à la base du dépôt
```

Copy

- Est-ce que l'étape se passe bien ? (SPOILER: non ✗)



Exercice GitHub Actions : Checkout

- **But** : On souhaite récupérer ("checkout") le code du dépôt dans le job
- 🚧 C'est à vous d'essayer de *réparer* 🔧 le job :
 - L'étape doit être conservée et doit fonctionner
 - Utilisez l'action "checkout" (Documentation) du marketplace GitHub Action
 - Vous pouvez vous inspirer du Quickstart de GitHub Actions

✓ Solution GitHub Actions : Checkout

```
name: Bonjour
on:
  - push
jobs:
  dire_bonjour:
    runs-on: ubuntu-22.04
    steps:
      - uses: actions/checkout@v3 # Récupère le contenu du dépôt correspondant au commit du workflow en cours
      - run: ls -l # Liste les fichier du répertoire courant
      - run: cat README.md # Affiche le contenu du fichier `README.md` à la base du dépôt
```

Copy

Exemple : Environnement d'exécution

- Notre workflow doit s'assurer que "la vache" 🐄 doit nous lire 🗂 le contenu du fichier README.md
 - WAT 😳 ?
- Essayez la commande `cat README.md | cowsay` dans GitPod
 - Modifiez l'étape du workflow pour faire la même chose dans GitHub Actions
 - SPOILER: ✘ (la commande `cowsay` n'est pas disponible dans le runner GitHub Actions)



Exercice : Personnalisation dans le workflow

- **But :** exécuter la commande `cat README.md | cowsay` dans le workflow comme dans GitPod
- 🚀 C'est à vous de mettre à jour le workflow pour personnaliser l'environnement :
 - 🔎 Cherchez comment installer `cowsay` dans le runner GitHub (`runs-on...`)

✓ Solution : Personnalisation dans le workflow

```
name: Bonjour
on:
  - push
jobs:
  dire_bonjour:
    runs-on: ubuntu-22.04
    steps:
      - uses: actions/checkout@v3 # Récupère le contenu du dépôt correspondant au commit du workflow en cours
      - run: |
          sudo apt-get update
          sudo apt-get install --yes cowsay
      - run: cat README.md | cowsay
```

Copy

Checkpoint

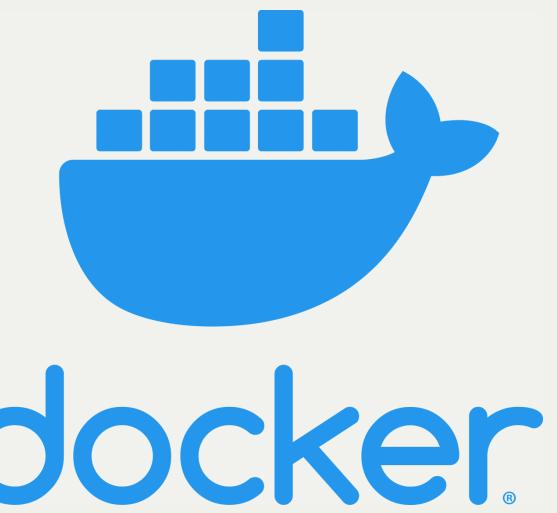
- L'intégration Continue est un ensemble de pratiques pour s'assurer que le code est **continuellement** "vérifié"
- GitHub Actions est un des nombreux systèmes permettant de faire de l'intégration continue

⇒ 😔 Problème : comment gérer les différences entre l'environnement d'IC et les machines de travail
?

Docker : Bases

"La Base"

Pourquoi ?



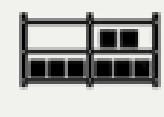
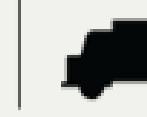
🤔 Quel est le problème ?

 Static Website	?	?	?	?	?	?	?
 Web Frontend	?	?	?	?	?	?	?
 Background Workers	?	?	?	?	?	?	?
 User DB	?	?	?	?	?	?	?
 Analytics DB	?	?	?	?	?	?	?
 Queue	?	?	?	?	?	?	?
	Development VM	QA Server	Single Prod Server	Production Cluster	Public cloud	Developer's Laptop	Customer Servers
			---				
<i>Source: https://blog.docker.com/2013/08/paas-present-and-future/</i>							

Problème de temps exponentiel

Déjà vu ?

L'IT n'est pas la seule industrie à résoudre des problèmes...

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

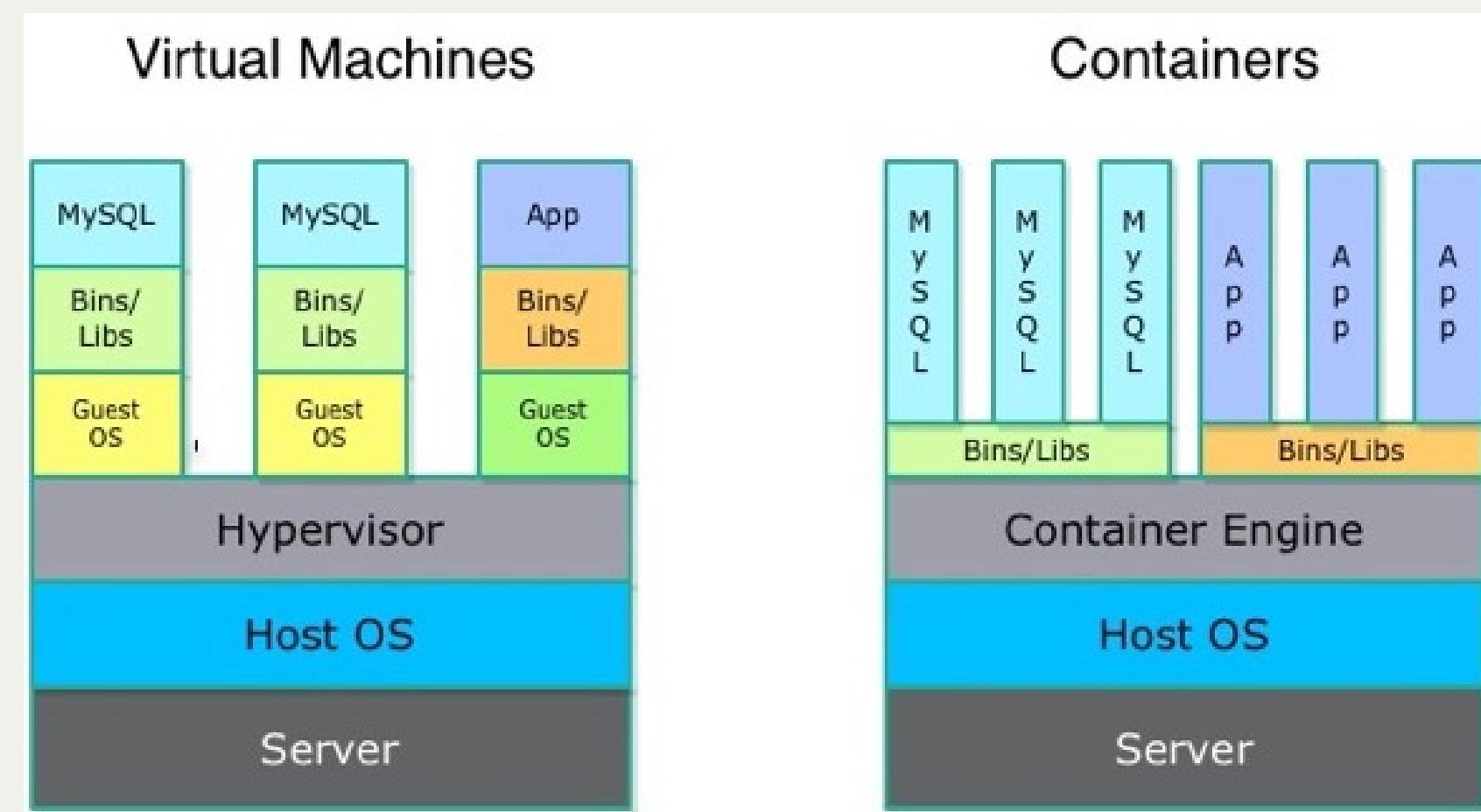
Solution: Le conteneur intermodal

"Separation of Concerns"



Comment ça marche ?

"Virtualisation Légère"



Conteneur != VM

"Separation of concerns": 1 "tâche" par conteneur

VM

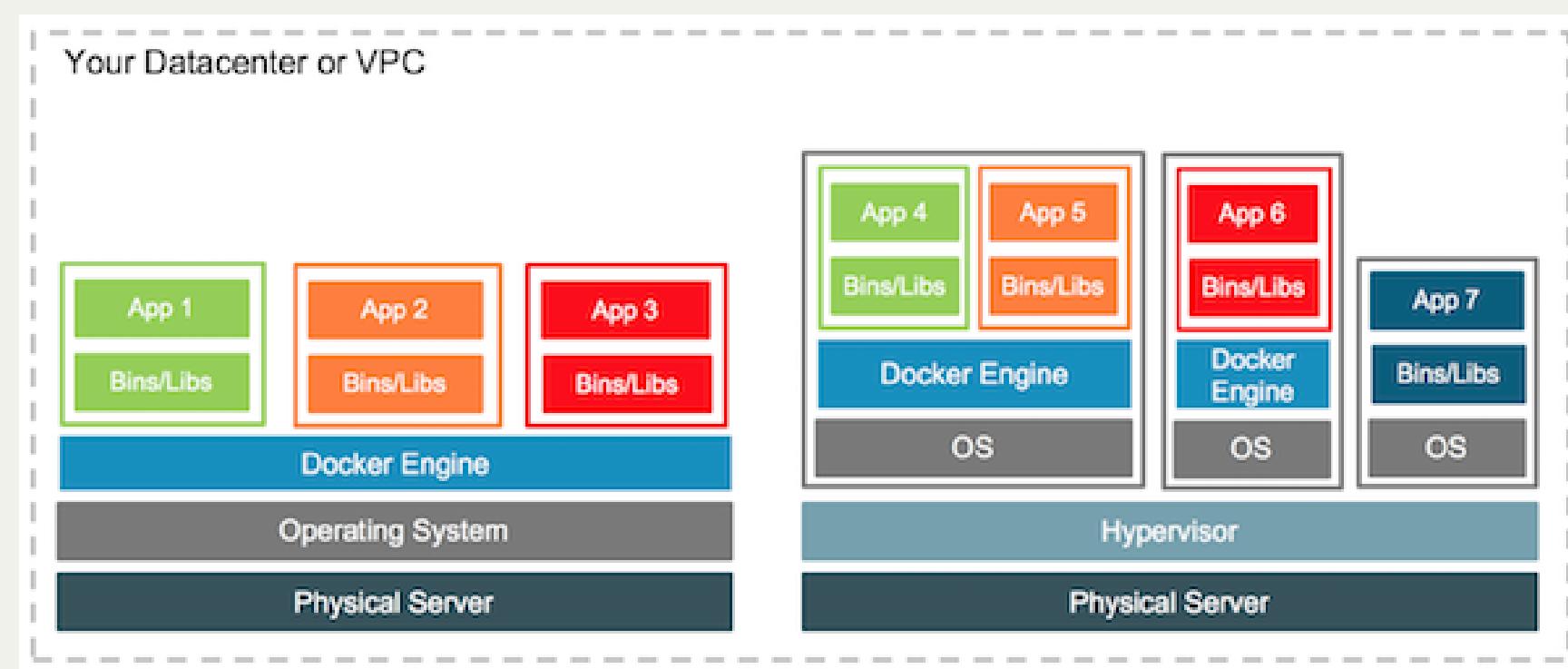


Containers



VMs && Conteneurs

Non exclusifs mutuellement



Comment ça marche ?



Exercice : Votre premier conteneur

C'est à vous (ouf) !

- Retournez dans Gitpod
- Dans un terminal, tapez la commande suivante :

```
docker container run hello-world  
# Équivalent de l'ancienne commande 'docker run'
```

Copy

□ Anatomie

- Un service "Docker Engine" tourne en tâche de fond et publie une API REST
- La commande `docker run ...` a envoyé une requête POST au service
- Le service a téléchargé une **Image** Docker depuis le registre **DockerHub**,
- Puis a exécuté un **conteneur** basé sur cette image



Exercice : Où est mon conteneur ?

C'est à vous !

```
docker container ls --help
# ...
docker container ls
# ...
docker container ls --all
```

Copy

⇒ 🤔 comment comprenez vous les résultats des 2 dernières commandes ?

✓ Solution : Où est mon conteneur ?

Le conteneur est toujours présent dans le "Docker Engine" même en étant arrêté

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES	Copy
109a9cdd3ec8	hello-world	"/hello"	33 seconds ago	Exited (0) 17 seconds ago		festive_faraday	

- Un conteneur == une commande "conteneurisée"
 - cf. colonne "**COMMAND**"
- Quand la commande s'arrête : le conteneur s'arrête
 - cf. code de sortie dans la colonne "**STATUS**"



Exercice : Cycle de vie d'un conteneur simple

- Lancez un nouveau conteneur nommé `bonjour`
 - `docker container run --help` ou Documentation en ligne
- Affichez les "logs" du conteneur (==traces d'exécution écrites sur le stdout + stderr de la commande conteneurisée)
 - `docker container logs --help` ou Documentation en ligne
- Lancez le conteneur avec la commande `docker container start`
 - Regardez le résultat dans les logs
- Supprimez le container avec la commande `docker container rm`

✓ Solution : Cycle de vie d'un conteneur simple

```
docker container run --name=bonjour hello-world
# Affiche le texte habituel

docker container logs bonjour
# Affiche le même texte : pratique si on a fermé le terminal

docker container start bonjour
# N'affiche pas le texte mais l'identifiant unique du conteneur 'bonjour'

docker container logs bonjour
# Le texte est affiché 2 fois !

docker container ls --all
# Le conteneur est présent
docker container rm bonjour
docker container ls --all
# Le conteneur n'est plus là : il a été supprimé ainsi que ses logs

docker container logs bonjour
# Error: No such container: bonjour
```

Copy



Que contient "hello-world" ?

- C'est une "image" de conteneur, c'est à dire un modèle (template) représentant une application auto-suffisante.
 - On peut voir ça comme un "paquetage" autonome
- C'est un système de fichier complet:
 - Il y a au moins une racine /
 - Ne contient que ce qui est censé être nécessaire (dépendances, librairies, binaires, etc.)

Docker Hub

- <https://hub.docker.com/> : C'est le registre d'images "par défaut"
 - Exemple : Image officielle de conteneur "Ubuntu"
- 🎓 Cherchez l'image hello-world pour en voir la page de documentation
 - 💡 pas besoin de créer de compte pour ça
- Il existe d'autre "registres" en fonction des besoins (GitHub GHCR, Google GCR, etc.)



Exercice : conteneur interactif

- Quel distribution Linux est utilisée dans le terminal Gitpod ?
 - Regardez le fichier `/etc/os-release`
- Exécutez un conteneur interactif basé sur `alpine:3.17` (une distribution Linux ultra-légère) et regardez le contenu du fichier au même emplacement
 - `docker container run --help`
 - Demandez un `tty` à Docker
 - Activez le mode interactif
- Exécutez la même commande dans un conteneur basé sur la même image mais en **NON** interactif
 - Comment surcharger la commande par défaut ?

✓ Solution : conteneur interactif

```
$ cat /etc/os-release
# ... Ubuntu ....  
  
$ docker container run --tty --interactive alpine:3.17
/ # cat /etc/os-release
# ... Alpine ...
# Notez que le "prompt" du terminal est différent DANS le conteneur
/ # exit
$ docker container ls --all  
  
$ docker container run alpine:3.17 cat /etc/os-release
# ... Alpine ...
```

Copy



Exercice : conteneur en tâche de fond

- Exécutez un conteneur, basé sur l'image nginx en tâche de fond ("Background"), nommé webserver-1
 - ☀ On parle de processus "détaché" (ou bien "démonisé")
 - ⚠ Pensez bien à docker container ls
- Regardez le contenu du fichier /etc/os-release dans ce conteneur
 - ☀ docker container exec
- Essayez d'arrêter, démarrer puis redémarrer le conteneur
 - ⚠ Pensez bien à docker container ls à chaque fois
 - ☀ stop, start, restart

✓ Solution : conteneur en tâche de fond

```
docker container run --detach --name=webserver-1 nginx
# <ID du conteneur>

docker container ls
docker container ls --all

docker container exec webserver-1 cat /etc/os-release
# ... Debian ...

docker container stop webserver-1
docker container ls
docker container ls --all

docker container start webserver-1
docker container ls
docker container ls --all

docker container start webserver-1
docker container ls
```

Copy

Checkpoint

- Docker essaye de résoudre le problème de l'empaquetage le plus "portable" possible
 - On n'en a pas encore vu les effets, ça arrive !
 - Vous avez vu qu'un conteneur permet d'exécuter une commande dans un environnement "préparé"
 - Catalogue d'images Docker par défaut : Le **Docker Hub**
 - Vous avez vu qu'on peut exécuter des conteneurs selon 3 modes :
 - "One shot"
 - Interactif
 - En tâche de fond
- ⇒  Mais comment ces images sont-elles fabriquées ? Quelle confiance leur accorder ?

Docker Images

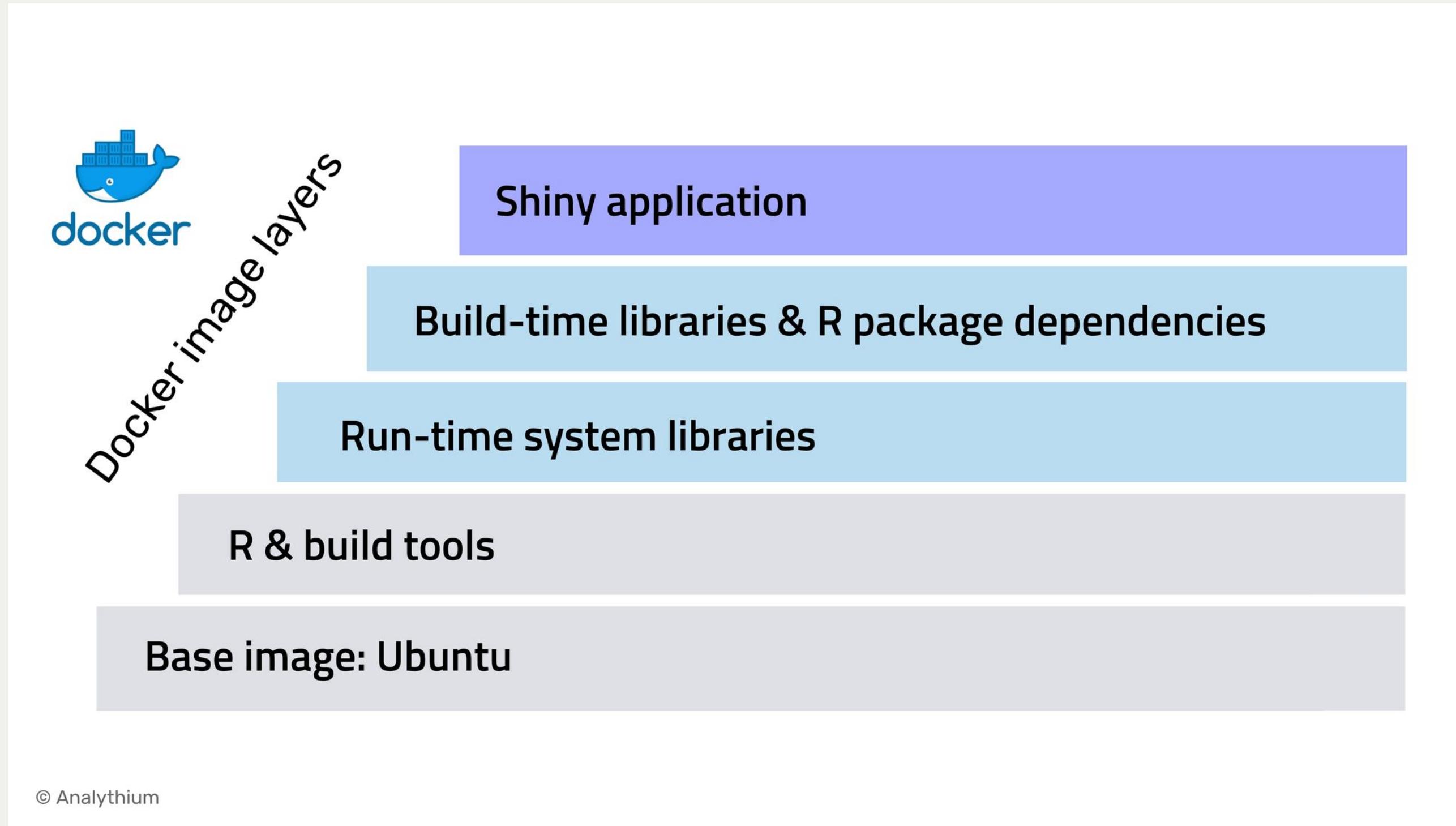


Pourquoi des images ?

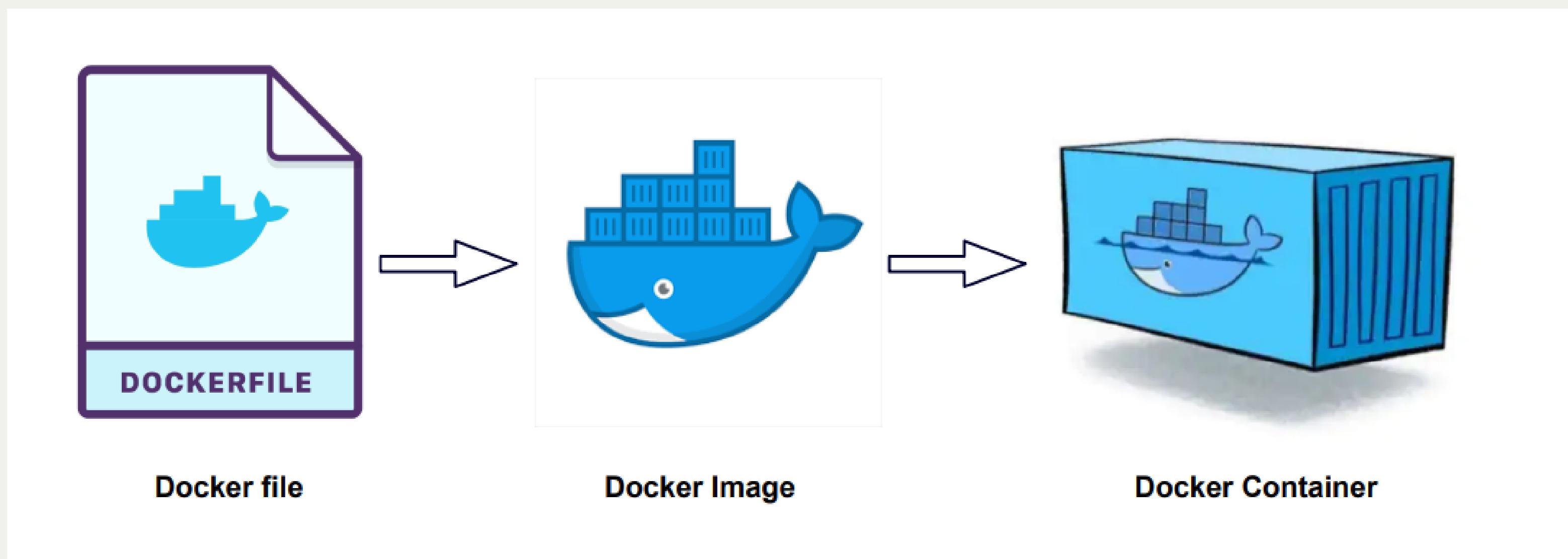
- Un **conteneur** est toujours exécuté depuis une **image**.
- Une **image de conteneur** (ou "Image Docker") est un modèle ("template") d'application auto-suffisant.

⇒ Permet de fournir un livrable portable (ou presque).

🤔 Application Auto-Suffisante ?



C'est quoi le principe ?



Docker file

Docker Image

Docker Container



Pourquoi fabriquer sa propre image ?

Problème :

```
cat /etc/os-release
# ...
git --version
# ...

# Même version de Linux que dans GitPod
docker container run --rm ubuntu:20.04 git --version
# docker: Error response from daemon: failed to create shim task: OCI runtime create failed: runc create failed: unable t

# En interactif ?
docker container run --rm --tty --interactive ubuntu:20.04 git --version
```

Copy



Fabriquer sa première image

- **But :** fabriquer une image Docker qui contient git
- Dans votre workspace Gitpod, créez un dossier nommé docker-git /
- Dans ce dossier, créer un fichier Dockerfile avec le contenu ci-dessous :

```
FROM ubuntu:20.04
RUN apt-get update && apt-get install --yes --no-install-recommends git
```

Copy

- Fabriquez votre image avec la commande docker image build --tag=docker-git <chemin/vers/docker-git /
- Testez l'image fraîchement fabriquée
 - ⚒ docker image ls

✓ Fabriquer sa première image

```
cat <<EOF >Dockerfile
FROM ubuntu:20.04
RUN apt-get update && apt-get install --yes --no-install-recommends git
EOF

docker image build --tag=docker-git ./

docker image ls | grep docker-git

# Doit fonctionner
docker container run --rm docker-git:latest git --version
```

Copy

Conventions de nommage des images

```
[REGISTRY/] [NAMESPACE/] NAME [:TAG | @DIGEST]
```

Copy

- Pas de Registre ? Défaut: registry.docker.com
- Pas de Namespace ? Défaut: library
- Pas de tag ? Valeur par défaut: latest
 - Δ Friends don't let friends use latest
- Digest: signature unique basée sur le contenu

Conventions de nommage : Exemples

- ubuntu:20.04 ⇒ registry.docker.com/library/ubuntu:20.04
- dduportal/docker-asciidoc ⇒
registry.docker.com/dduportal/docker-asciidoc:latest
- ghcr.io/dduportal/docker-asciidoc:1.3.2@sha256:xxxx



Utilisons les tags

- Rappel : ⚠ Friends don't let friends use latest
- Il est temps de "taguer" votre première image !

```
docker image tag docker-git:latest docker-git:1.0.0
```

Copy

- Testez le fonctionnement avec le nouveau tag
- Comparez les 2 images dans la sortie de docker image ls

✓ Utilisons les tags

```
docker image tag docker-git:latest docker-git:1.0.0

# 2 lignes
docker image ls | grep docker-git
# 1 ligne
docker image ls | grep docker-git | grep latest
# 1 ligne
docker image ls | grep docker-git | grep '1.0.0'

# Doit fonctionner
docker container run --rm docker-git:1.0.0 git --version
```

Copy



Mettre à jour votre image (1.1.0)

- Mettez à jour votre image en version 1.1.0 avec les changements suivants :
 - Ajoutez un `LABEL` dont la clef est `description` (et la valeur de votre choix)
 - Configurez `git` pour utiliser une branche `main` par défaut au lieu de `master` (commande
`git config --global init.defaultBranch main`)
- Indices :
 - Commande `docker image inspect <image name>`
 - Commande `git config --get init.defaultBranch` (dans le conteneur)
 - Ajoutez des lignes **à la fin** du Dockerfile
 - Documentation de référence des Dockerfile

✓ Mettre à jour votre image (1.1.0)

```
cat ./Dockerfile
FROM ubuntu:20.04
RUN apt-get update && apt-get install --yes --no-install-recommends git
LABEL description="Une image contenant git préconfiguré"
RUN git config --global init.defaultBranch main

docker image build -t docker-git:1.1.0 ./docker-git/
# Sending build context to Docker daemon 2.048kB
# Step 1/4 : FROM ubuntu:20.04
# ---> e40cf56b4be3
# Step 2/4 : RUN apt-get update && apt-get install --yes --no-install-recommends git
# ---> Using cache
# ---> 926b8d87f128
# Step 3/4 : LABEL description="Une image contenant git préconfiguré"
# ---> Running in 0695fc62ecc8
# Removing intermediate container 0695fc62ecc8
# ---> 68c7d4fb8c88
# Step 4/4 : RUN git config --global init.defaultBranch main
# ---> Running in 7fb54ecf4070
# Removing intermediate container 7fb54ecf4070
# ---> 2858ff394edb
Successfully built 2858ff394edb
Successfully tagged docker-git:1.1.0

docker container run --rm docker-git:1.0.0 git config --get init.defaultBranch
docker container run --rm docker-git:1.1.0 git config --get init.defaultBranch
# main
```

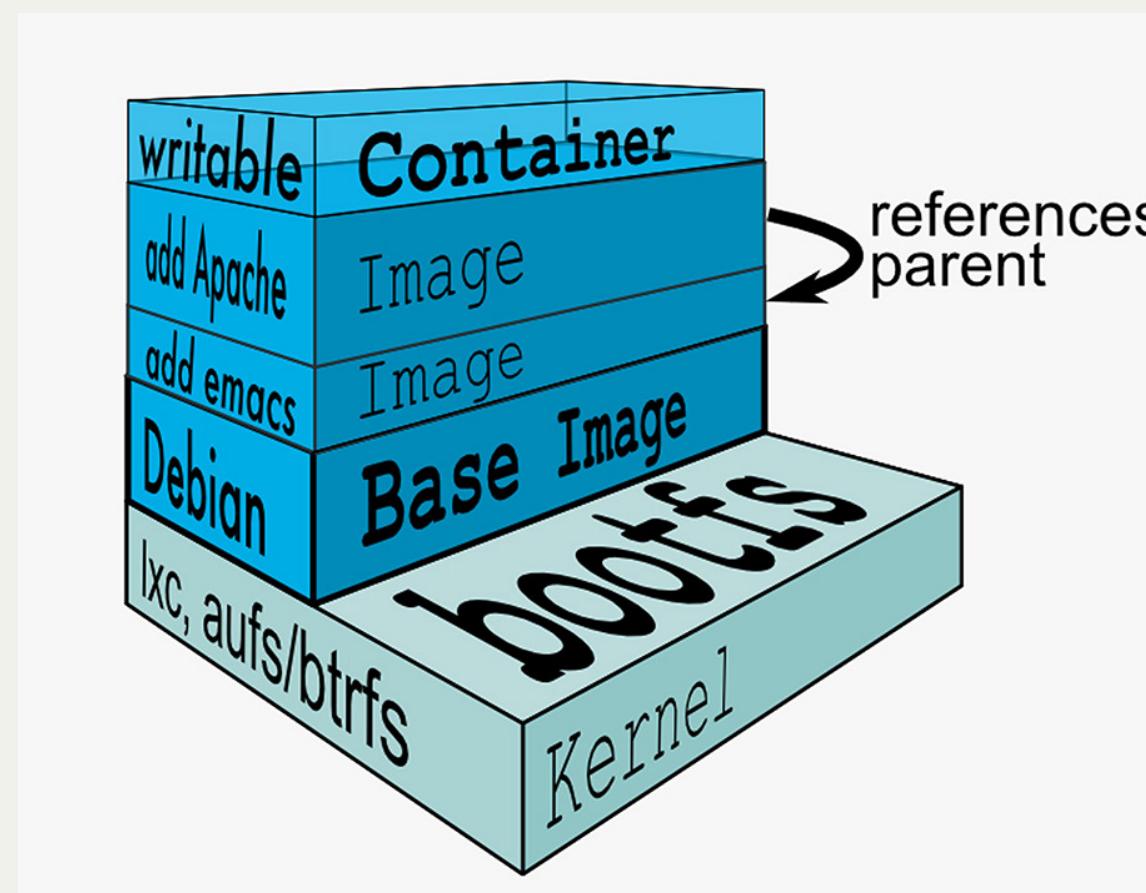
Copy

Cache d'images & Layers

```
Step 2/4 : RUN apt-get update && apt-get install --yes --no-install-recommends git  
---> Using cache
```

Copy

💡 En fait, Docker n'a PAS exécuté cette commande la seconde fois ⇒ ça va beaucoup plus vite !



🎓 Essayez de voir les layers avec (dans Gitpod) `dive <image>:<tag>`



Cache d'images & Layers

- **But :** manipuler le cache d'images
- Commencez par vérifier que le cache est utilisé : relancez la dernière commande `docker image build` (plusieurs fois s'il le faut)
- Invalidez le cache en ajoutant le paquet APT `make` à installer en même temps que `git`
 - Δ Tag 1.2.0
- Vérifiez que le cache est bien présent de nouveau

✓ Cache d'images & Layers

```
# Build one time
docker image build -t docker-git:1.1.0 ./docker-git/
# Second time is fully cached
docker image build -t docker-git:1.1.0 ./docker-git/

cat Dockerfile
# FROM ubuntu:20.04
# RUN apt-get update && apt-get install --yes --no-install-recommends git make
# LABEL description="Une image contenant git préconfiguré"
# RUN git config --global init.defaultBranch main

# Build one time
docker image build -t docker-git:1.2.0 ./docker-git/
# Second time is fully cached
docker image build -t docker-git:1.2.0 ./docker-git/

## Vérification
# Renvoie une erreur
docker run --rm docker-git:1.1.0 make --version
# Doit fonctionner
docker run --rm docker-git:1.2.0 make --version
```

Copy

Checkpoint

- Une image Docker fournit un environnement de système de fichier auto-suffisant (application, dépendances, binaries, etc.) comme modèle de base d'un conteneur
- Les images Docker ont une convention de nommage permettant d'identifier les images très précisément
- On peut spécifier une recette de fabrication d'image à l'aide d'un `Dockerfile` et de la commande `docker image build`

⇒  et si on utilisait Docker pour nous aider dans l'intégration continue ?

CI avec Docker



Environnement d'exécution du CI

Problème : On souhaite avoir les mêmes outils dans notre CI ainsi que dans nos environnement de développement

- Environnement d'exécutions différents :
 - Système d'exploitation ? (macOS, Windows, Ubuntu Linux, Arch Linux, etc.)
 - Architecture du processeur ? (Intel, AMD, ARM, PowerPC, Risc-V)

⇒ Est-ce que Docker 🐳 peut aider ?

Personnaliser l'environnement GitHub Actions

Plusieurs solutions existent, chacune avec ses avantages / inconvénients :

- Personnaliser l'environnement dans votre workflow: (Δ sensible aux mises à jour, \checkmark facile à mettre en place)
 -  C'est ce qu'on a fait dans nos workflows précédemment
- Spécifier un environnement préfabriqué pour le workflow (Δ complexe, \checkmark portable)
 - C'est ce qu'on va essayer avec Docker 



Exercice : Environnement préfabriqué simple

- **But :** exécuter la commande `cat README.md | cowsay` dans le workflow avec un environnement le plus proche possible du développement
 - En utilisant le même environnement que GitPod (**même version de Ubuntu** et de `cowsay`)
- 🚀 C'est à vous de mettre à jour le workflow pour exécuter les étapes dans la même image Docker que GitPod :
 - 🌐 Image utilisée dans GitPod
 - 🌐 Utilisation d'un container comme runner GitHub Actions
 - 🌐 Contraintes d'exécution de container dans GitHub Actions (`--user=root`)

✓ Solution : Environnement préfabriqué simple

```
name: Bonjour
on:
  - push
jobs:
  dire_bonjour:
    runs-on: ubuntu-20.04 # Same as GitPod
    container:
      image: ghcr.io/dduportal/esgi-gitpod
      options: --user=root
    steps:
      - uses: actions/checkout@v3 # Récupère le contenu du dépôt correspondant au commit du workflow en cours
      - run: cat README.md | cowsay
```

Copy

- Quel est l'impact en terme de temps d'exécution du changement précédent ?
- **Problème :** Le temps entre une modification et le retour est crucial





Problème : Accélérer le workflow

- **Problème :**

- Optimiser prématulement est contre-productif (commencez par faire un système qui marche comme prévu)
- Mais il faut bien s'y coller à un moment donné



Exercice : Environnement préfabriqué local

- **But :** Utiliser un Dockerfile pour fabriquer une image à utiliser
- 😞 GitHub Actions ne permet pas de spécifier un Dockerfile
- C'est à vous. Mettez à jour votre workflow pour :
 - Fabriquer une image Docker contenant cowsay et basée sur le même Ubuntu que Gitpod
 - Copier le fichier README dans l'image
 - Exécuter la commande cowsay avec l'aide d'un conteneur avec

```
bash -c 'README.md | cowsay'
```

Copy

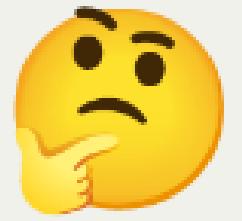
✓ Exercice : Environnement préfabriqué local

```
# Dockerfile
FROM ubuntu:20.04
RUN apt-get update && apt-get install --yes cowsay
COPY ./README.md /README.md
```

Copy

```
# bonjour.yml
name: Bonjour
on:
- push
jobs:
  dire_bonjour:
    runs-on: ubuntu-22.04
    steps:
      - uses: actions/checkout@v3 # Récupère le contenu du dépôt correspondant au commit du workflow en cours
      - name: "Build Image"
        run: docker image build --tag=cowsay:latest ./
      - name: "Say Hello"
        run: docker container run --rm cowsay:latest bash -c "cat /README.md | cowsay"
```

Copy



Réfléchissons ensemble

- Impact sur le temps ?
- Quelles limites voyez-vous ?

Work in progress

🚧 Revenez plus tard

Bibliographie

Ligne de commande

- <https://tldp.org>
- <https://en.wikipedia.org/wiki/POSIX>
- https://en.wikipedia.org/wiki/Read%20%93eval%20%93print_loop
- <https://linuxhandbook.com/linux-directory-structure/>

Git / VCS

- <https://docs.github.com>
- <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>
- <http://martinfowler.com/bliki/VersionControlTools.html>
- <http://martinfowler.com/bliki/FeatureBranch.html>
- <https://about.gitlab.com/2014/09/29/gitlab-flow/>
- <https://www.atlassian.com/git/tutorials/comparing-workflows>
- <http://nvie.com/posts/a-successful-git-branching-model/>

CI/CD

- <http://martinfowler.com/articles/continuousIntegration.html>
- <http://martinfowler.com/bliki/ContinuousDelivery.html>
- <https://jaxenter.com/implementing-continuous-delivery-117916.html>
- <https://technologyconversations.com/2014/04/29/continuous-delivery-introduction-to-concepts-and-tools/>
- <http://blog.arungupta.me/continuous-integration-delivery-deployment-maturity-model>
- <http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment>

Docker

- <https://dduportal.github.io/cours/cnam-docker-2018>
- <https://kodekloud.com/blog/docker-for-beginners/>
- <https://www.slideshare.net/dotCloud/why-docker>
- <https://docs.docker.com/engine/reference/builder/>
- <https://www.r-bloggers.com/2021/05/best-practices-for-r-with-docker/>
- <https://github.com/wagoodman/dive>

Autre

- <https://cicd-lectures.github.io/slides/>
- <https://dduportal.github.io/cours/>
- <https://www.jenkins.io/node/>

Merci !

✉️ [damien.duportal <chez> gmail.com](mailto:damien.duportal@chez.com)

Slides: <https://dduportal.github.io/esgi-containers-ci-cd/2023>



Source on [GitHub](https://github.com/dduportal/esgi-containers-ci-cd) : <https://github.com/dduportal/esgi-containers-ci-cd>