



# GITLAB-CI



Code d'Armor  
15 janvier 2019  
David Blaisonneau



**QUELQUES MOTS**



# GITLAB





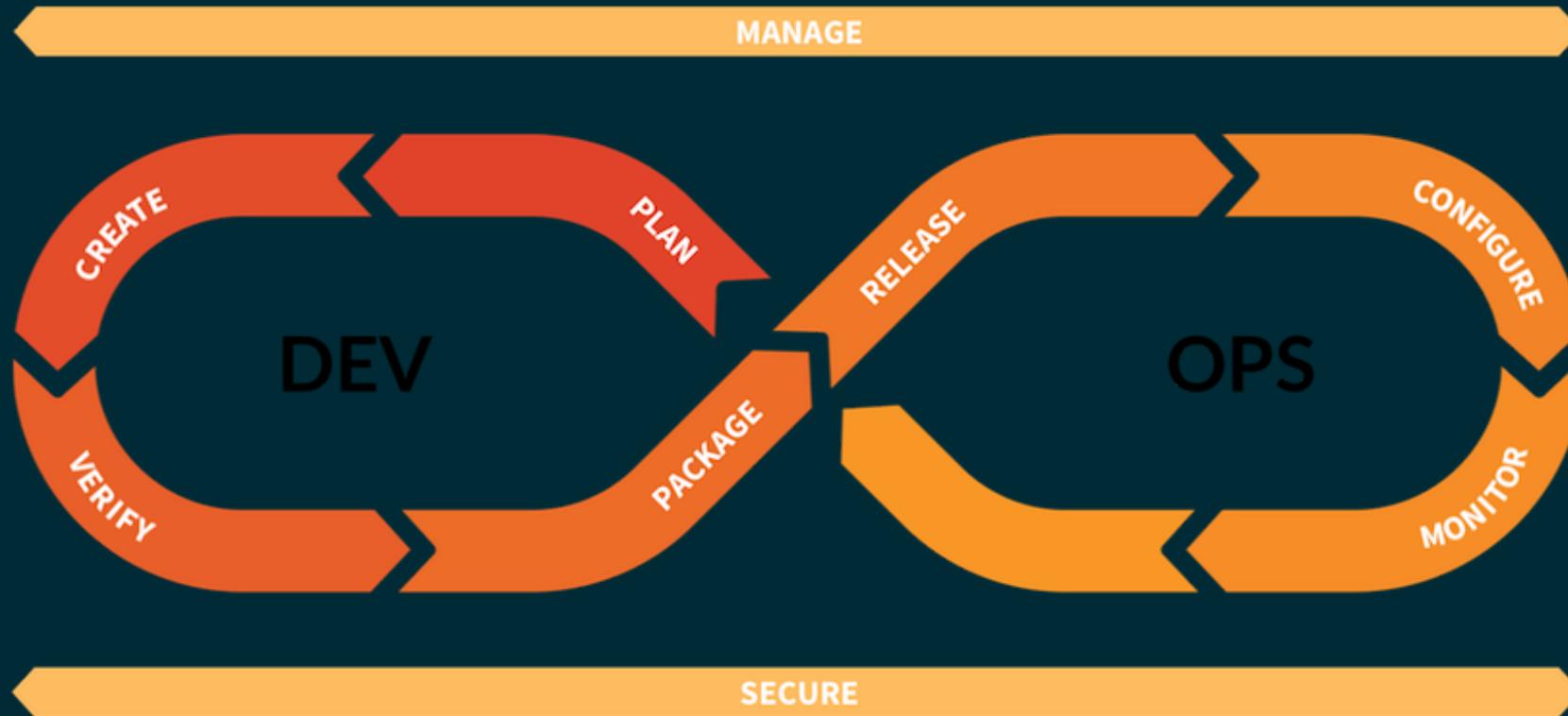
Gitlab est un logiciel de forge permettant de gérer toutes les étapes d'un cycle de développement

**DevOPS**. C'est:

- à l'origine une simple forge au dessus de GIT
- une application de gestion du Cycle **DevOPS**
- plusieurs versions:
  - [gitlab.com](https://gitlab.com)
  - en mode privé, pour son propre environnement



# LE CYCLE DEVOPS





# LES PRINCIPALES FONCTIONS DE GITLAB

On retrouve bien entendu:

- la gestion de projets et des groupes de projets
- la gestion des différents utilisateurs et des outils d'échanges: snippets, un wiki
- la gestion et revue de code
- un registry Docker
- ... et surtout de nombreuses fonctions autour du CI/CD !



# LA GESTION DE PROJETS

Pour la gestion des projets:

- différents niveaux d'utilisateurs (guest, reporter, developer, maintenir, owner)
- gestion des *issues* via un board de type KanBan
- milestones (agile sprint / release)
- time tracking, cycle analytics
- service desk



# LA GESTION DU CODE

Tout est fait pour simplifier le gestion du code

- WebIDE
- versionning du code (via Git)
- gestion simple des branches, des tags
- *merge request* et revue de code
- plein de beaux graphiques
- le blocage de fichiers



# LES *MERGE REQUEST*

Ils permettent entre autres:

- de présenter l'ajout de code
- de comparer le code résultant de plusieurs commits
- commenter les choix des développeurs
- visualiser les changements en mode "Live preview"

et bien entendu on y retrouve les résultats du CI/CD !



# QUELQUES EXEMPLES

- Orange Kubespray #7
- Gitlab #24345
- Gitlab #24181



# GITLAB-CI





# CONTINUOUS INTEGRATION

*L'intégration continue est un ensemble de pratiques utilisées en génie logiciel consistant à vérifier à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée. [...] Le principal but de cette pratique est de détecter les problèmes d'intégration au plus tôt lors du développement. De plus, elle permet d'automatiser l'exécution des suites de tests et de voir l'évolution du développement du logiciel.*

--wikipedia--



# CONTINUOUS DELIVERY

*La livraison continue est une approche d'ingénierie logicielle dans laquelle les équipes produisent des logiciels dans des cycles courts, ce qui permet de le mettre à disposition à n'importe quel moment. Le but est de construire, tester et diffuser un logiciel plus rapidement. L'approche aide à réduire le coût, le temps et les risques associés à la livraison de changement en adoptant une approche plus incrémentielle des modifications en production. Un processus simple et répétable de déploiement est un élément clé.*

--wikipedia--

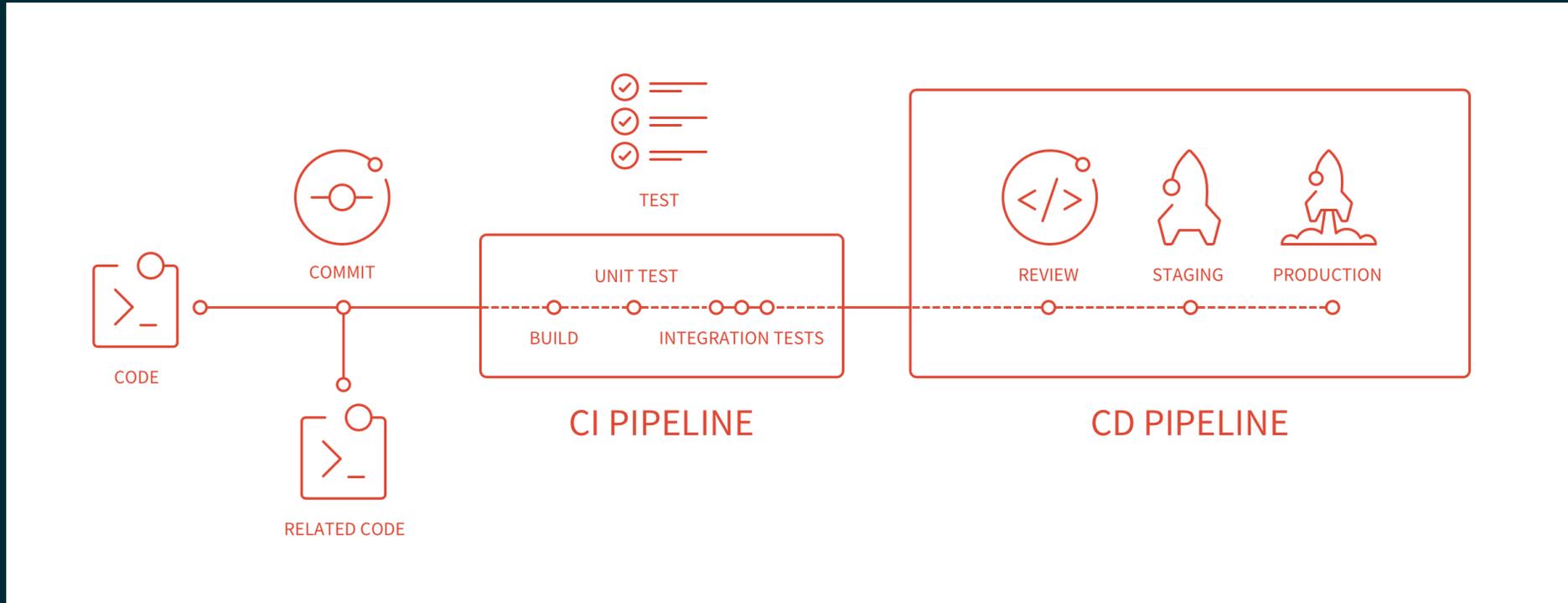


# L'ÉCOSYSTÈME



CI  CD

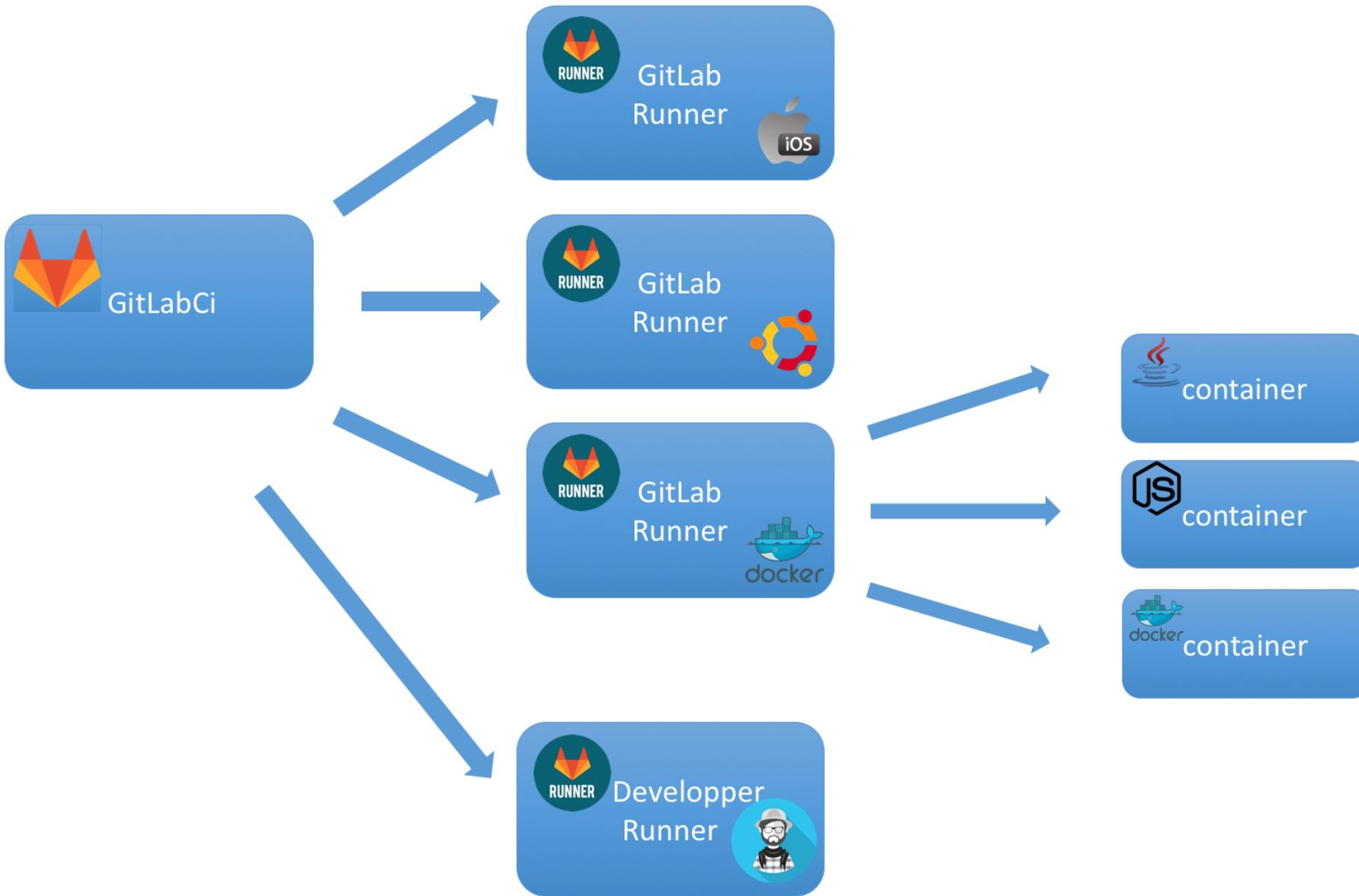
Travis / Jenkins / TeamCity





# LES RUNNERS

Les *runners* sont les environnements d'exécution des tâches du CI/CD. Ils peuvent être partagés, ou dédiés à un projet; sous différents environnements: shell, docker, virtualbox, ssh, kubernetes.





# PIPELINES/STAGES/JOBS

Un pipeline est un ensemble de tâches (jobs)  
ordonnées en plusieurs étapes (stages)



David Blaisonneau > rubyonrail-autodevops > Pipelines > #43314863

passed Pipeline #43314863 triggered 2 hours ago by David Blaisonneau

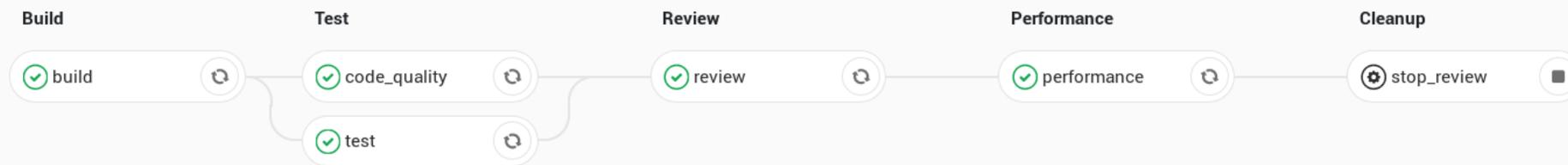
## Update welcome\_controller\_test.rb

6 jobs from [hello](#) in 14 minutes and 47 seconds

latest Auto DevOps

[2677f483](#)

**Pipeline** Jobs 6





# LES *MANIFEST*

Toute la description du CI/CD est centralisée dans un seul fichier YAML, le fichier `.gitlab-ci.yml`

```
job1:  
  script: "execute-script-for-job1"  
job2:  
  script: "execute-script-for-job2"
```



# JOB CONSOLE

Chaque job permet d'accéder en direct aux logs de l'exécution de la tâche.



# JOB ARTIFACTS

Pour sauvegarder les résultats d'une tâche, il est possible de sauvegarder une archive appelée artifact. Elle peut être disponible au téléchargement ou



# PIPELINES VARIABLES

De nombreuses variables prédéfinies sont disponibles pour chaque tâches, permettant de donner toutes les infos nécessaires à propos:

- du projet
- du commit
- du merge request
- de l'utilisateur
- du runner
- du pipeline ...



# ASSIGNER UNE VARIABLE D'ENVIRONNEMENT

Des variables d'environnement peuvent aussi être assignées à chaque pipeline, de manière publique ou



# DÉCLANCHEMENT D'UN PIPELINE

- un push sur une branches GIT: push/merge-request/master/branches/tags
- un évènement API / trigger
- un autre pipeline
- un déclenchement programmé (cron)
- un formulaire web



# SCHEDULES



# ENVIRONNEMENTS

Un environnement permet de spécifier différents milieux de déploiement différents: testing / production / site1 / site2.

Environment	Deployment	Job	Commit	Updated	
<a href="#">production</a>	#1 by	production #145543278	<a href="#">816a358c</a> Rails template	7 hours ago	
<a href="#">review/hello</a>	#4 by	review #145626094	<a href="#">2677f483</a> Update welcome_controller_t...	4 hours ago	



# KUBERNETES CLUSTERS

GitLab peut être connecté très simplement à un cluster *Kubernetes* existant, ou même le créer tout seul sur *Google Cloud Platform*. On peut ainsi y déployer ses applications ou mettre à disposition un ensemble de runners.



# REVIEW APPS





# AUTODEVOPS

Il est possible de laisser GitLab générer tout seul toutes les étapes du CI/CD, de manière ~~magique~~ automatique.

## Auto DevOps

Auto DevOps will automatically build, test, and deploy your application based on a predefined Continuous Integration and Delivery configuration. [Learn more about Auto DevOps](#)

**Enable Auto DevOps**

The Auto DevOps pipeline configuration will be used when there is no `.gitlab-ci.yml` in the project.

**Instance default (disabled)**

Follow the instance default to either have Auto DevOps enabled or disabled when there is no project specific `.gitlab-ci.yml`.

### Domain

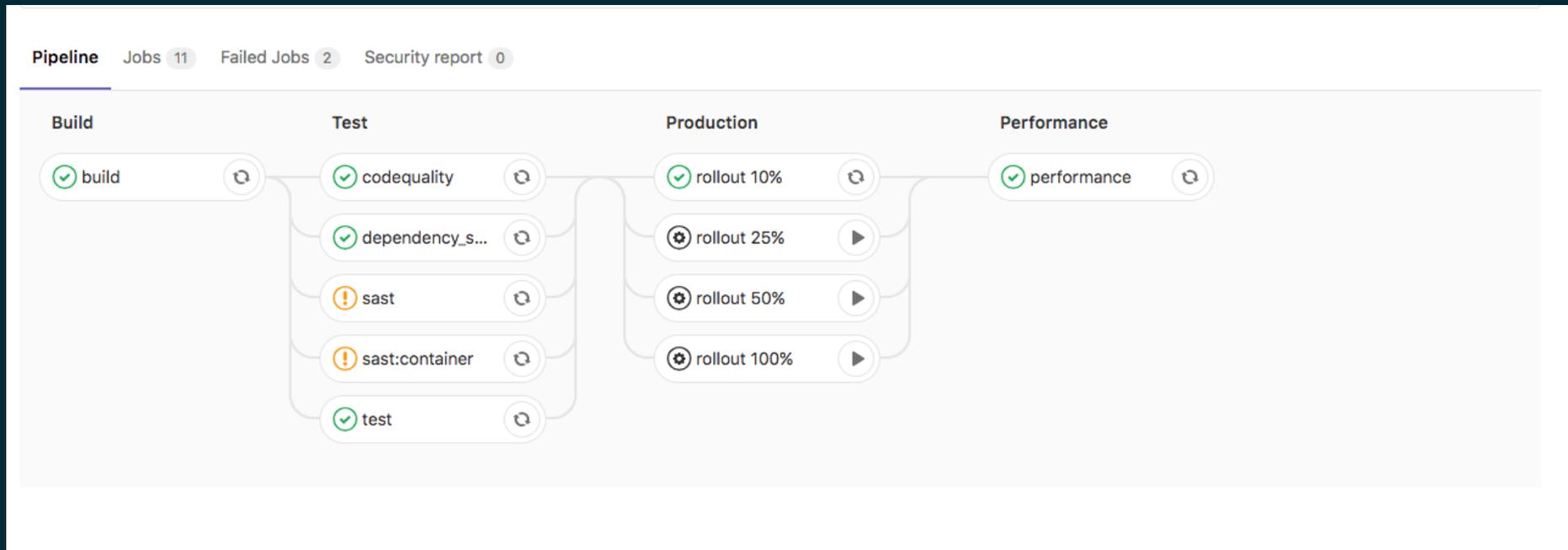
35.202.251.21.nip.io

You need to specify a domain if you want to use Auto Review Apps and Auto Deploy stages. `35.202.251.21.nip.io` can be used as an alternative to a custom domain. [?](#)

Do not set up a domain here if you are setting up multiple Kubernetes clusters with Auto DevOps. [?](#)



# AUTODEVOPS - ROLLOUT





# SERVERLESS

Dans sa dernière release, GitLab permet avec l'intégration Kubernetes l'utilisation de knative, une application de déploiement **serverless**.

Le serverless permet ainsi de créer de petits ensembles de code "scalable", disponibles via une API Rest sous forme de F(unctions)aaS.



GitLab Projects Groups Activity Milestones Snippets

Search or jump to...

Administrator > functions > Serverless

functions

- Project
- Repository
- Issues 0
- Merge Requests 0
- CI / CD
- Operations**
  - Metrics
  - Environments
  - Error Tracking
  - Serverless**

▼ \*

**functions-echo** functions-echo-buildtemplate  
Deploying functions from GitLab with Knative 27 minutes ago

`http://functions-echo.functions-1.functions.gitlabcloud.net`



# QUELQUES JOBS





# SIMPLE SCRIPTS

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



# UTILISATION DES *STAGES*

```
stages:  
  - build  
  - test  
  - deploy  
  
job 1:  
  stage: build  
  script: make build dependencies  
  
job 2:  
  stage: build  
  script: make build artifacts  
  
job 3:  
  stage: test  
  script: make test
```



# LINTING: ANSIBLE LINTING

```
ansible_linting:  
  image: sdesbure/ansible-lint:latest  
  script:  
    - ansible-lint -x ANSIBLE0010,ANSIBLE0013 os_*.yaml  
  stage: lint
```



# LINTING: SSH KEY CHECK

```
sshkey_check:
  stage: lint
  script:
    - "keylist=$(cat users.yml|yq -r '.users[].sshkey')"
    - "for key in ${keylist}; do \
      ssh-keygen -l -f files/keys/${key}.key.pub; \
    done"
```



# DEPLOY PYTHON TO PRODUCTION

```
test:
  script:
    # this configures Django application to use attached postgres d
    - export DATABASE_URL=postgres://postgres:@postgres:5432/python
    - apt-get update -qy
    - apt-get install -y python-dev python-pip
    - pip install -r requirements.txt
    - python manage.py test

staging:
  type: deploy
  script:
    - apt-get update -qy
    - apt-get install -y ruby-dev
    - gem install dpl
    dpl --provider=heroku --app=gitleb-ci --python test staging --a
```



# AUTOBUILD DOCKER DOCKER





# AUTO BUILDS

```
image: docker:git
services:
  - docker:dind
variables:
  DOCKER_DRIVER: overlay

before_script:
  - docker login -u gitlab-ci-token -p "$CI_BUILD_TOKEN" "$CI_REGISTRY"

development:
  stage: deploy
  script:
    - docker build -t "$CI_REGISTRY_IMAGE:latest" .
    - docker push "$CI_REGISTRY_IMAGE:latest"
  only:
    - master
```





# GITLAB CI FOR GITLAB PAGES

Le système de *Pages* permet de mettre en place un site web statique hébergé par GitLab. Ces pages *statiques* peuvent bien entendu être générées lors de l'exécution de la tâche.

Les possibilités sont multiples:

- Blog sur base markdown
- Mise en page de résultats de tests
- Résultats d'un build précédent.
- ...



## Group



### pages

Example websites hosted by GitLab Pages

Leave group

Watch

**Projects** Subgroups

Filter by name...

Last updated

New Project



**nanoc**

Example Nanoc site using GitLab Pages: <https://pages.gitlab.io/nanoc>



updated about an hour ago



**hugo**

Example Hugo site using GitLab Pages: <https://pages.gitlab.io/hugo>



updated a week ago



**pages.gitlab.io**

Main website for GitLab Pages: <https://about.gitlab.com/features/pages/>



updated a month ago



**middleman**

Example Middleman site using GitLab Pages: <https://pages.gitlab.io/middleman>



updated 2 weeks ago



**plain-html**

Example plain HTML site using GitLab Pages: <https://pages.gitlab.io/plain-html>



updated 2 weeks ago



**jekyll**

Example Jekyll site using GitLab Pages: <https://pages.gitlab.io/jekyll>



updated 4 weeks ago



**pelican**

Example Pelican site using GitLab Pages: <https://pages.gitlab.io/pelican>



updated 3 weeks ago



**jekyll-branched**





# PAGES EXAMPLES

```
pages:  
  stage: deploy  
  image: jekyll:latest  
  script:  
    - cd doc  
    - jekyll build -d ../public  
  artifacts:  
    paths:  
      - public
```



# UN PEU DE PRATIQUE

- Publier une image docker et la scanner
- AutoDevOPs
- Construire un package android
- Le projet Gitlab-ce



# PUBLIER UNE IMAGE DOCKER

Construire une image Docker, la publier et la tester

🔺 <https://gitlab.com/dblaisonneau/docker-node-red-smarthome/>

🔺 <https://www.objectif-libre.com/fr/blog/2018/07/26/scanning-docker-images-with-clair-and-gitlab/>



# SERVERLESS

 <https://gitlab.com/dblaisonneau/knative-ruby-app/>



# AUTODEVOPS

Créer un projet depuis un template, publier sur Kubernetes, effectuer un merge request, tester ce changement et le passer en prod.



[https://gitlab.com/help/topics/autodevops/quick\\_start\\_guide.md](https://gitlab.com/help/topics/autodevops/quick_start_guide.md)

- `app/views/welcome/index.html.erb`

```
<h1>Hi Code d'Armor !</h1>
```

- `test/controllers/welcome_controller_test.rb`



# ANDROID

Construire une image Android

 <https://about.gitlab.com/2018/10/24/setting-up-gitlab-ci-for-android-projects/>

 <https://gitlab.com/jlenny/androidblog-2018>



# GITLAB

Le projet Gitlab Community Edition

 <https://gitlab.com/gitlab-org/gitlab-ce/pipelines>



# QUELQUES MOTS POUR CONCLURE



# MERCI

**VOUS DEVEZ AVOIR DES QUESTIONS, NON ?**

 david-dev@blaisonneau.fr

 dblaisonneau