Formation MiNET

CI-CD

YAML

Yet Another Markup Language

“.gitlab-ci.yml” est un fichier YAML
tips: pas d’espace, que des TAB

YAML

YAML EST UN SUPERSET JSON

```yaml
hello-job:
    script:
        - echo "Hello World !"
```

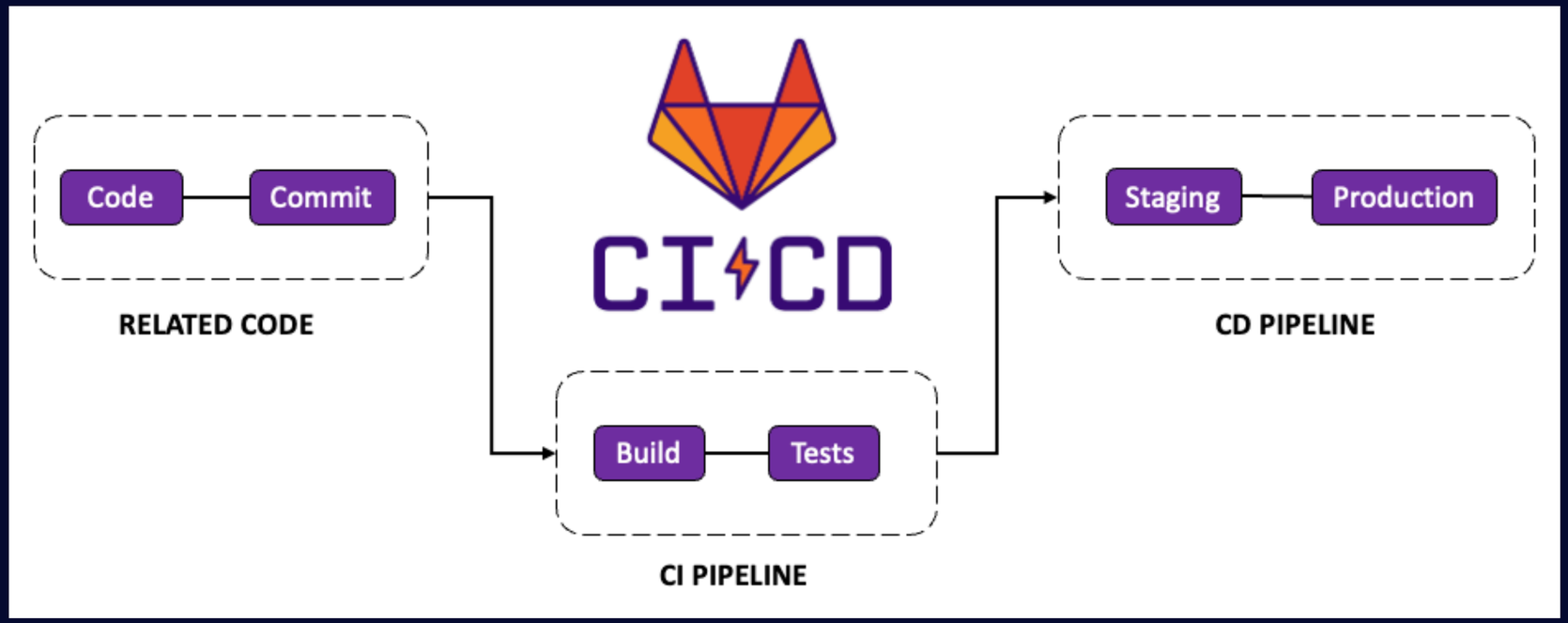
```json
{
    "hello-job": {
        "script": [
            "echo \"Hello World     !\""
            ]
    }
}
```

JSON to YAML Online Converter

Qu’est-ce que le CI-CD ?

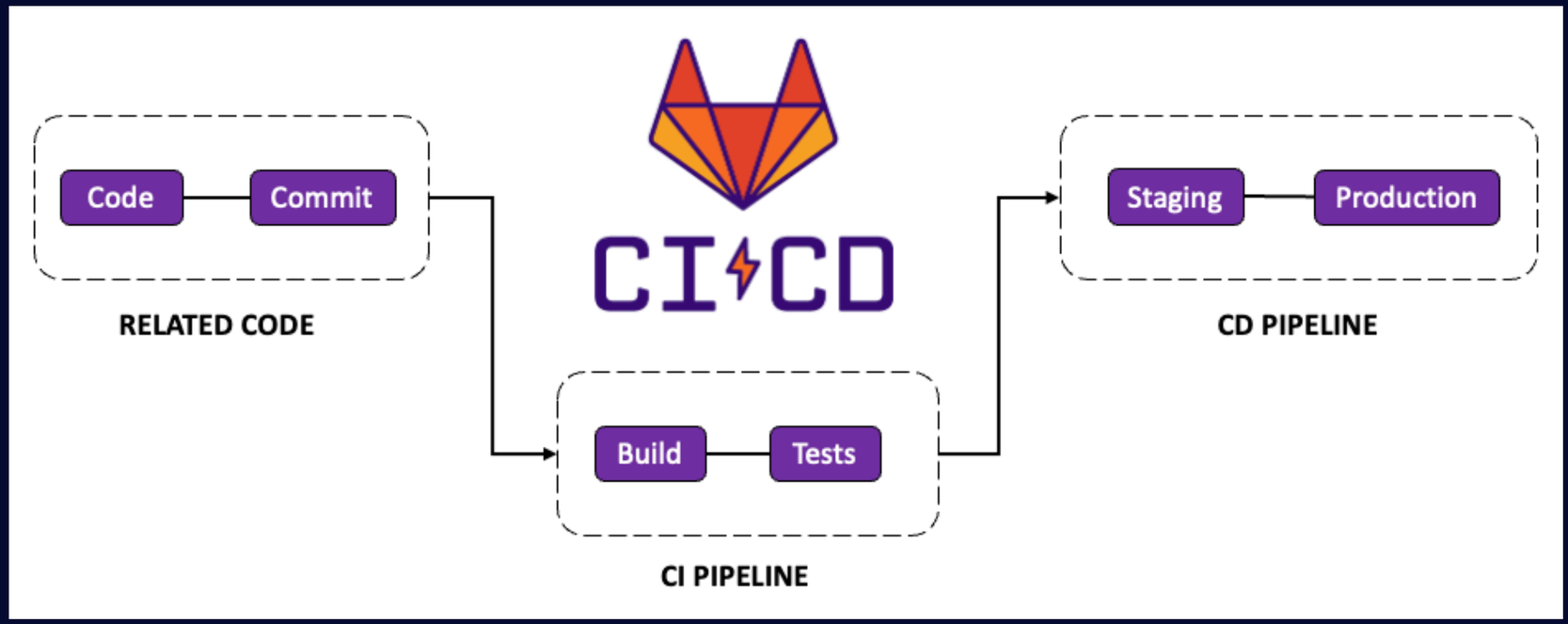
Qu’est-ce que le CI-CD ?

CI : Intégration continue



Qu’est-ce que le CI-CD ?

CD : Déploiement continu
Cd : Livraison continue



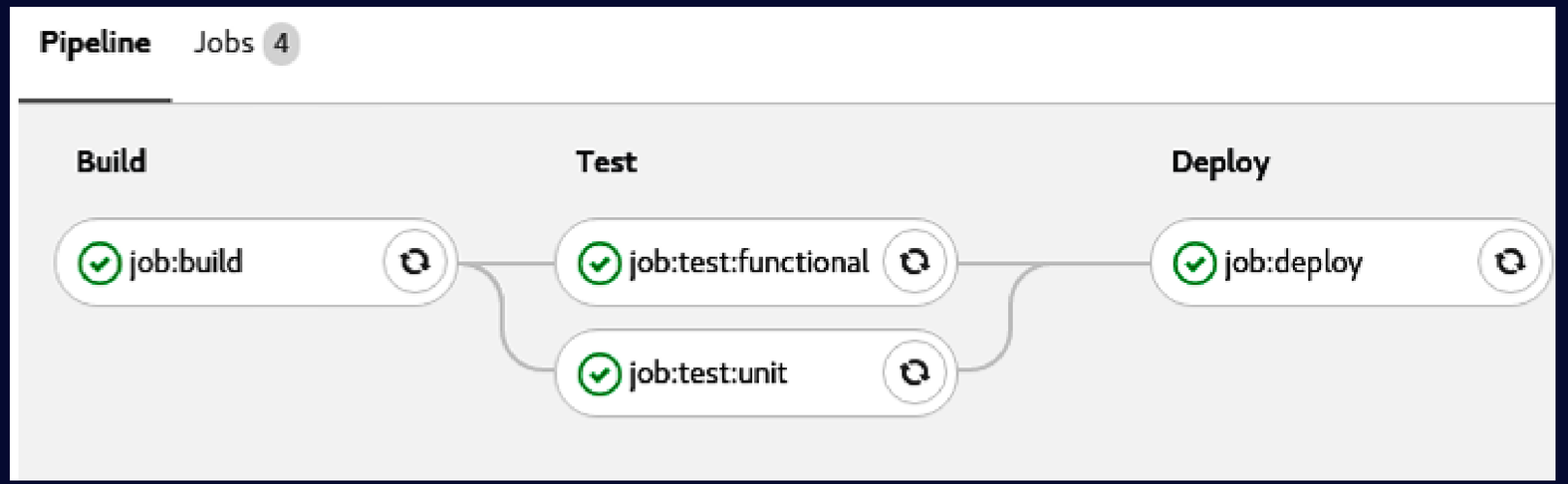
INTRODUCTION AUX PIPELINES

Définition

Un pipeline définit la chaîne de ce qui doit être fait dans le processus d'intégration ou de livraison continue.

GITLAB CI PIPELINE STRUCTURE

- Un **pipeline** c’est une séquence de **stages**
- Un **stage** est composé de **jobs**
- A **job** est un “script” executé par un **runner**



EXAMPLE

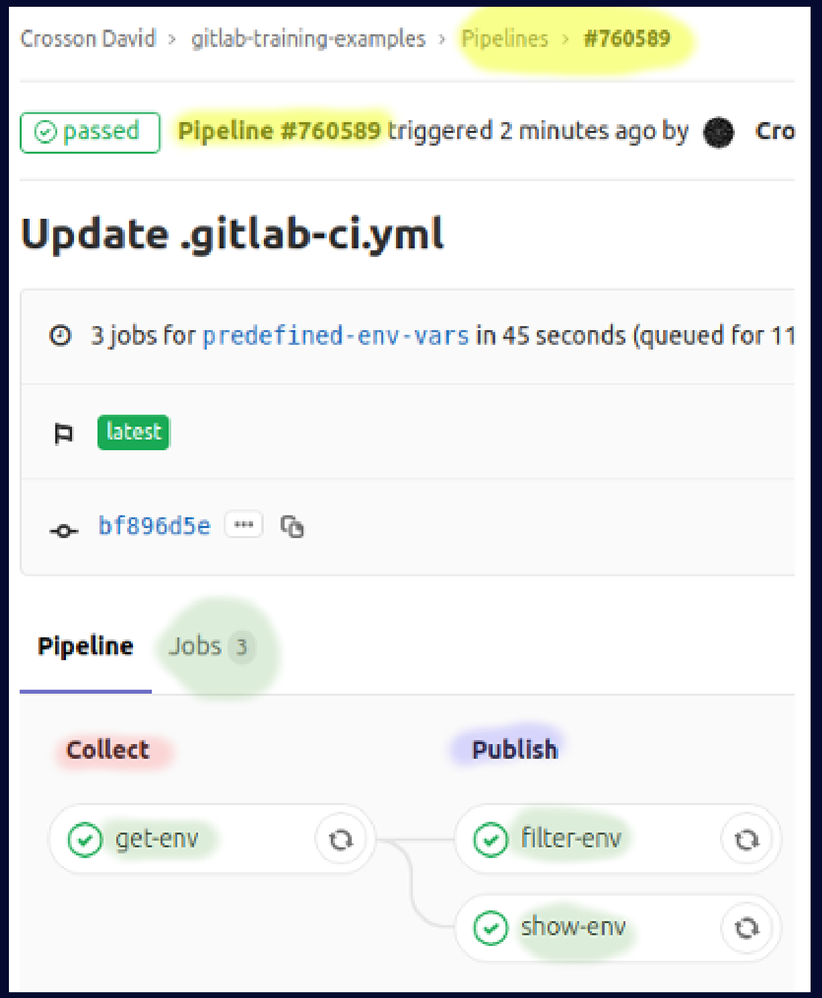
```yaml
stages:
  - collect
  - publish

get-env:
  stage: collect
  tags:
    - shared
  script:
    - env | sort > sorted-env.txt
  artifacts:
    paths:
      - sorted-env.txt

show-env:
  stage: publish
  tags:
    - shared
  script:
    - cat sorted-env.txt

filter-env:
  stage: publish
  tags:
    - shared
  script:
    - grep -i CI_RUNNER_DESCRIPTION sorted-env.txt
```

stages :
Décrit une liste ordonnée d'action à etre executé



PIPELINE TRIGGERING

- Une pipeline démarre automatiquement
  - new pushes
  - branches création
  - planification
- Pipeline peut être démarré de manière explicite
  - API
  - UI GITLAB

JOBs

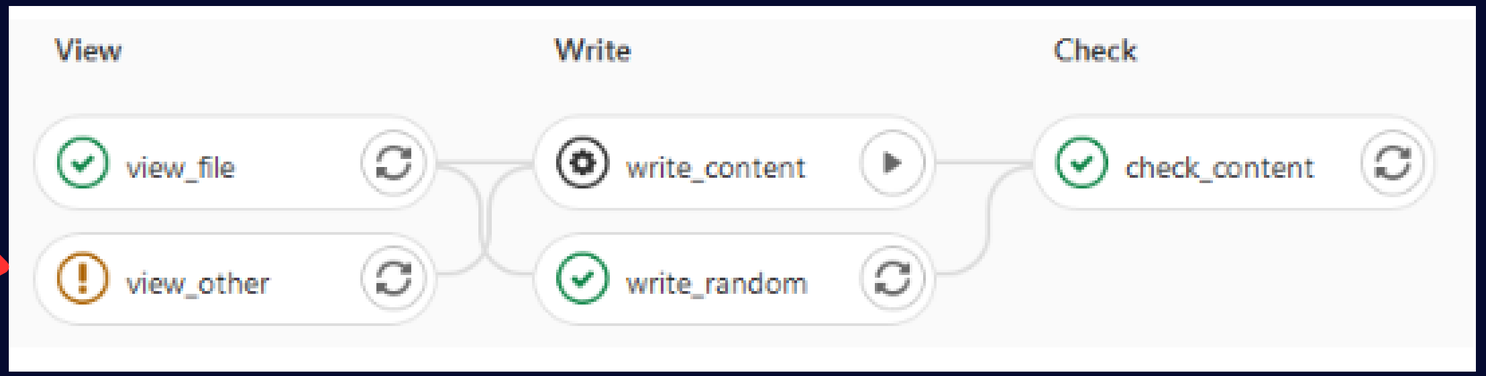
Par défaut, tous les “jobs” d'une étape sont automatiquement exécutés.

when: manual

Le nouveau job échoue, le pipeline s'arrête

passed

L’option “allow_failure:true” modifie ce comportement.



```yaml
example_job:
    script:
        - failingCommand || true
        - echo "Still running..."
```

```yaml
example_job:
    allow_failure: true
    script:
        - exit 1
        - echo "** NOT EXECUTED**"
```

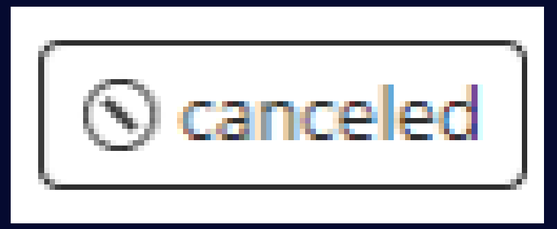
```yaml
example_job:
    script:
        - set +e # Continue execution on error
        - failingCommand
        - echo "Still running"
```

set -euo pipefail

when:

on_success : job is run if all previous jobs were ok
on_failure : job is run if any previous job has failed
never : job is excluded from the pipeline
manual : job is run manually
always : job is run whatever the status of previous jobs

RÉSUMÉ DE L'ÉTAT DES PIPELINES



GITLAB VARIABLES

```
show-env:
  script:
    - unset DEFAULT_CA_CERTS
    - env | sort
    - echo $CI_PROJECT_NAME
    - echo $CI_PROJECT_PATH
    - echo $CI_COMMIT_REF_NAME
    - echo $CI_COMMIT_SHA
    - echo $CI_PIPELINE_ID
    - echo $CI_PIPELINE_SOURCE
```

JOB ARTIFACTS

Fichier et des dossiers produit par un jobs

```
build_job1:
  artifacts:
    paths:
      - libraries/

build_job2:
  artifacts:
    paths:
      - binaries/
    name: "$CI_COMMIT_REF_NAME-$CI_JOB_NAME"
    expire_in: 2 weeks
```

CONDITIONAL EXECUTION

rules:

```
job-name:
  rules:
    - if: $CI_COMMIT_BRANCH == "master"
      when: manual

    - if: $CI_COMMIT_TAG != null
```

```yaml
job:
  script: "echo Hello, Rules!"
  rules:
    - if: $CI_COMMIT_REF_NAME == "master"
      when: never

    - if: $MY_VARIABLE =~ /^feature/
      when: manual
      allow_failure: true

    - if: $CI_PIPELINE_SOURCE == "push"
      changes: [ Dockerfile ]
      variables:
        DOCKER_VERSION: 19.03

    - exists: [ spec/**.rb ]
      when: always
```

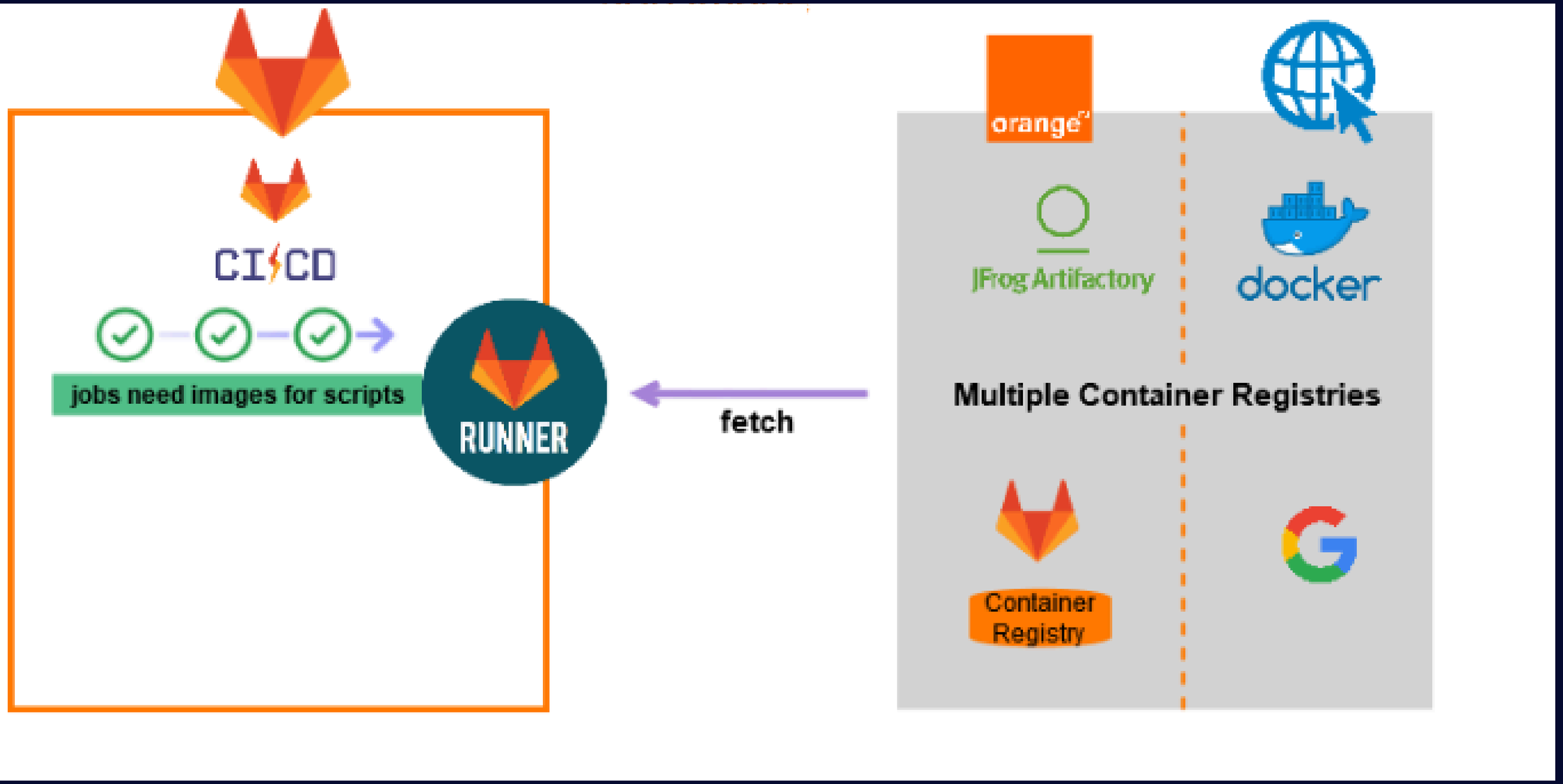
JOB IMAGE

Jobs sont executé dans des containers

default runners = **busybox**

ruby

image:


CI/CD
jobs need images for scripts
RUNNER
fetch
orange
JFrog Artifactory
docker
Multiple Container Registries
Container Registry

Le CI-CD en pratique

Le .gitlab-ci.yml

Services / **RevProxy**

```
20  stages:          # List of stages for jobs, and their order of execution
21    - test
22    - deploy
23
24
25
26  unit-test:   # This job runs in the test stage.
27    stage: test    # It only starts when the job in the build stage completes successfully.
28    image: debian
29    script:
30      - var=$(pwd)
31      - export http_proxy="http://192.168.102.61:82"
32      - export https_proxy="http://192.168.102.61:82"
33      - apt-get -y update
34      - apt -y install nginx
35      - rm -r /etc/nginx/*
36      - cd /etc/
37      - mkdir ssl
38      - cd ssl
39      - mkdir certs
40      - cd $var
41      - mv dhparam.pem /etc/ssl/certs/
42      - cp -r  ./* /etc/nginx/
43      - cd /etc/nginx/
44      - rm  /etc/nginx/sites-enabled/*
45      - ln -s /etc/nginx/sites-available/* sites-enabled/
46      - nginx -t
47      - exit $?
48      - cd  /etc/nginx/
49      - ls -a
50      - nginx -t
```

- **Le fichier où l’on décrit les jobs à exécuter dans chaque pipeline**
- **Il est écrit en YAML**
- **S’apprend sur le tas**

Le CI-CD en pratique

Gitlab runners

Aplication à installer sur les environments de test ou de prod qui exécute les jobs des pipelines

Le CI-CD à MiNET

Reverse proxy

Services / **RevProxy**

```yaml
stages:          # List of stages for jobs, and their order of execution
  - test
  - deploy



unit-test:   # This job runs in the test stage.
  stage: test    # It only starts when the job in the build stage completes successfully.
  image: debian
  script:
    - var=$(pwd)
    - export http_proxy="http://192.168.102.61:82"
    - export https_proxy="http://192.168.102.61:82"
    - apt-get -y update
    - apt -y install nginx
    - rm -r /etc/nginx/*
    - cd /etc/
    - mkdir ssl
    - cd ssl
    - mkdir certs
    - cd $var
    - mv dhparam.pem /etc/ssl/certs/
    - cp -r  ./* /etc/nginx/
    - cd /etc/nginx/
    - rm  /etc/nginx/sites-enabled/*
    - ln -s /etc/nginx/sites-available/* sites-enabled/
    - nginx -t
    - exit $?
    - cd  /etc/nginx/
    - ls -a
    - nginx -t
```

NGINX

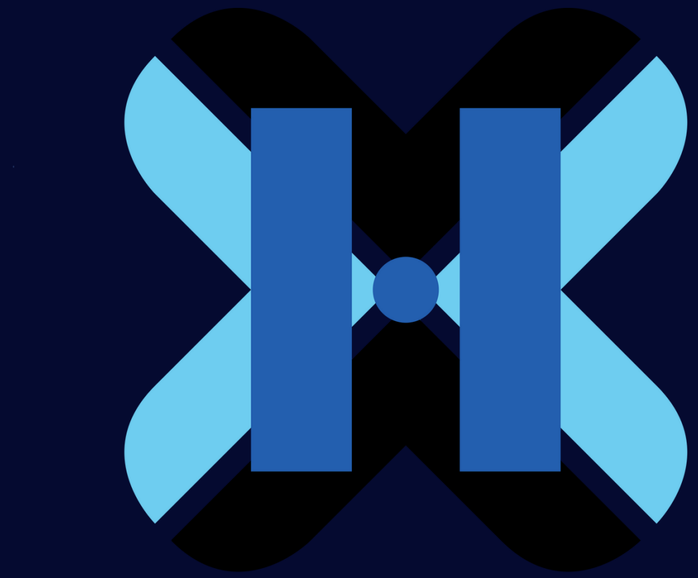
minet

Le CI-CD à MiNET

formation.minet.net

Le CI-CD à MiNET

Et plein d’autres services encore



HOSTING