Gitlab and Rancher deployment - CI-CD and pipelines

Table des matières

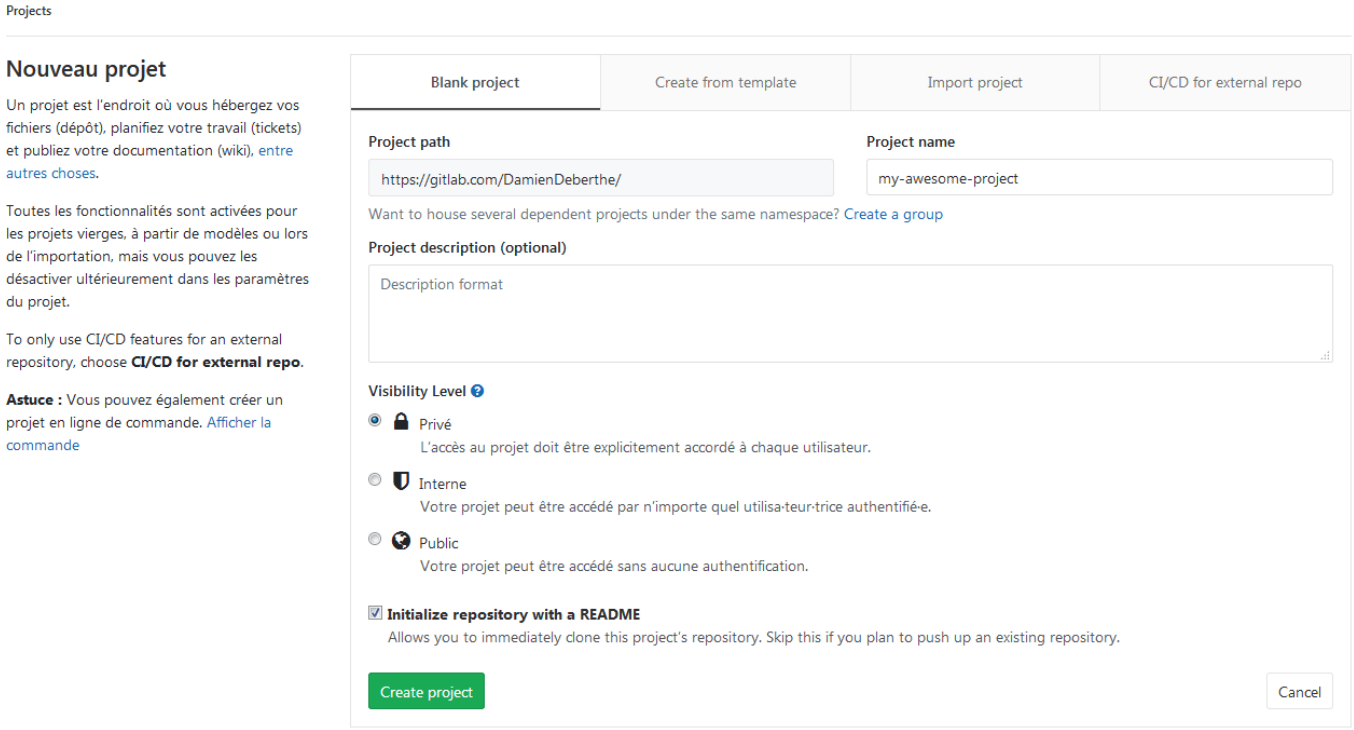
I. Introduction

The goal is to automatist the integration of the code.

📚 Source:

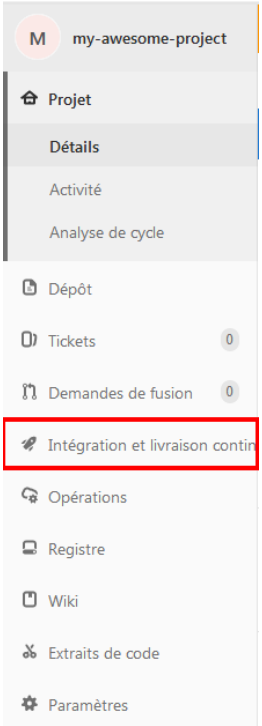
II. Gitlab repo and config

- Create new project:

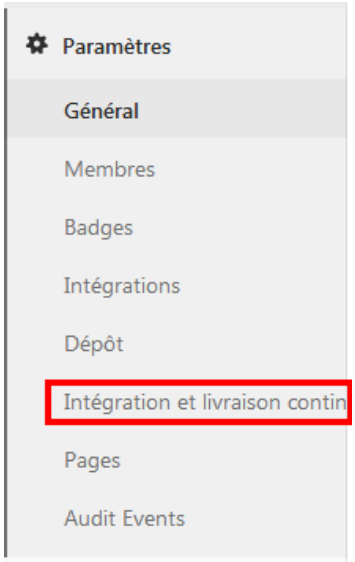


- The CI/CD information are these 2 menus:

For integration



For configuration



You can find this link for explications: https://gitlab.com/help/ci/quick_start/README

III. Gitlab-runner

The Gitlab-runner is an element that run the jobs that you define in "`.gitlab-ci.yml`".

At the beginning, Gitlab give us free runners from Digital Ocean and it's automatic and transparent. But it's very slow (between 1 to 3 min for each task). Better solution is to install our Runner.

📚 Source : https://gitlab.com/help/ci/runners/README.md
https://docs.gitlab.com/runner/

Let's do this!

1. Runner on Docker

a. Prerequisites

Install sudo, curl, docker: (*Why docker?* Because runner will launch Docker image for test and verify pipelines)

```
apt-get update && apt-get install -y sudo curl

curl -L
https://raw.githubusercontent.com/DamienDeberthe/Documentations/master/Do
cker/Scripts/docker-install-auto.sh | bash
```

b. Install the service "gitlab-runner" on Docker

Create the docker for the runner:

```
docker run -d --name gitlab-runner --restart always -v
/var/run/docker.sock:/var/run/docker.sock -v /data/gitlab-
runner:/etc/gitlab-runner gitlab/gitlab-runner:latest
```

c. Configure the service "gitlab-runner" on Docker

Execute these commands to connect the docker runner with the project on Gitlab:

```
docker exec -it gitlab-runner gitlab-runner register
```

Response like:

```
Please enter the gitlab-ci coordinator URL (e.g. https://gitlab.com/):
https://gitlab.com
Please enter the gitlab-ci token for this runner:
xxx
Please enter the gitlab-ci description for this runner:
[b3fec93a1a72]: runner-for-this-awesome-project
Please enter the gitlab-ci tags for this runner (comma separated):

Registering runner... succeeded                     runner=xxx
Please enter the executor: docker, virtualbox, docker-ssh+machine,
docker+machine, kubernetes, docker-ssh, parallels, shell, ssh:
docker
Please enter the default Docker image (e.g. ruby:2.1):
ruby:2.1
Runner registered successfully. Feel free to start it, but if it's
running already the config should be automatically reloaded!
```

You can find the configuration in the file "`/data/gitlab-runner/config.toml`"

→ To see the runners: Gitlab project -> Paramètre -> Intégration et livraison continue -> Exécuteurs: Runners activated for this project

It's better to disable the shared runners for increase speed. 

Source: https://www.sheevaboite.fr/articles/installer-gitlab-ci-moins-5-minutes-docker/

2. Runner on Debian (not tested and obsolete)

a. Prerequisites

Install sudo, curl:

```
apt-get update && apt-get install -y sudo curl
```

b. Install the service "gitlab-runner" on Debian

```
curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh | sudo bash
apt-get install gitlab-runner
```

📚 Source : https://docs.gitlab.com/runner/install/linux-repository.html

c. Configure the service "gitlab-runner" on Debian

On the server runner, execute these commands to connect the runner with the project on Gitlab:

```
gitlab-runner register
      https://gitlab.com
      token : find on the Gitlab project -> Paramètre -> Intégration et
      livraison continue -> Pipelines généraux : Jeton de l'exécuteur
      <name_of_this_runner>
      <tag>
      Bash? Docker?
```

The configuration is in the file: `~/.gitlab-runner/config.toml`

Start the service:

```
service gitlab-runner start
```

For active runner and can hosted pipelines:

```
gitlab-runner run
```

→ To see the runners: Gitlab project -> Paramètre -> Intégration et livraison continue -> Exécuteurs: Runners activated for this project

📚 Sources: https://docs.gitlab.com/runner/register/index.html

IV. File gitlab-ci.yml

This file is in the repository and it contain the instructions that will be execute for all the environment (test, build, deploy…).

1. First file gitlab-ci.yml

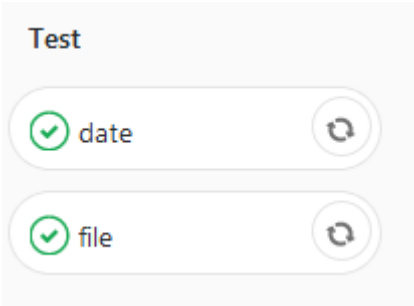
```
before_script:
  - mkdir /tmp/dir


stages:
  - test

date:
  stage: test
  script: date

file:
  stage: test
  script: echo "test1" > /tmp/dir/file
```

You will have 2 tasks in your pipeline:



- Source: https://gitlab.com/help/ci/yaml/README
- Source for pipelines: https://docs.gitlab.com/ee/ci/pipelines.html

2. Advanced file gitlab-ci.yml

```
before_script:
  - echo "before_script instructions here"


stages:
  - test
  - build
  - deploy

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

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

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

deploy_prod:
  stage: deploy
  script:
    - echo "Deploy to production server"
  environment:
    name: production
    url: https://example.com
  when: manual
  only:
  - master
```

- **script:** Chaque ligne représente des commandes à exécuter par le runner.
- **environment:** Défini les environnements spécifiques de déploiement des jobs (exemple: qa, review, staging, production, test, etc). Le nom des environnements est laissé libre au développeur
- **when:** Définir quand est ce que le job doit être lancé *(manual, on_failure/on_success, always)*.
- **only:** Ne Déclencher le job que lors de push ou commit sur certaines branches ou tags.
- **except**: Ne pas déclencher le job lors de push ou commit sur certaines branches ou tags.

📚 Sources: https://docs.gitlab.com/ee/ci/environments.html
https://www.supinfo.com/articles/single/6822-integration-continue-gitlab-gitlab-ci-cd

V. Deploy in production

With pipelines we will understand the order to deploy our application in production:

1. Build the environment
2. Test the code
3. Create the Docker image and push into Gitlab registry
4. Deploy in production (not finish)

In this case, we will create a simple webserver Apache (httpd) with a web page in HTML. The goal is when we change the HTML page, the Docker container is update on the Gitlab registry.

1. Build the environment

a. File HTML

Create our registry on Gitlab, and create a “index.html” with content like:

```
<!doctype html>
<html lang="fr">

<head>
  <meta charset="utf-8">
  <title>Congratulation</title>
</head>

<body>

        <center>

                <h1>
                        Congratulation !
                </h1>

                <br/>

                <h2>
                        If you see this page you have have succeed the
CI/CD deployment.
                </h2>

                <br/><br/><br/>

                <h4>
                        Proudly deploy with CI/CD pipelines on Gitlab
                </h4>

        </center>

</body>
</html>
```

b. File .gitlab-ci.yml

Create our first stage: build.

```
stages:
  - build

build:
  stage: build
  image: httpd:latest
  script:
    - apachectl -t
```

The pipeline will return OK.

2. Test the code

a. File .gitlab-ci.yml

This part need more explication…

```
stages:
  - build
  - test
  - docker

build:
  stage: build
  image: httpd:latest
  script:
    - apachectl -t


test:
  stage: test
  image: httpd:latest
  script: apachectl -t
  coverage: /All files\s*\|\s*([\d\.]+)/
```

3. Create the Docker image and push into Gitlab registry

a. File docker-compose.yml

Docker-compose contain all the options you want to configure for your Docker image:

```
version: "2"
services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    image: registry.gitlab.com/damiendeberthe/my-awesome-project
    ports:
      - "80:80"
```

- Image: chose the registry where you will push the future Docker image (in this case, in the repository of the Gitlab project)

b. File Dockerfile

The Dockerfile for your Docker container:

```
FROM httpd:latest
    MAINTAINER DamienDeberthe

    WORKDIR /usr/local/apache2/htdocs
    COPY . .

    # For security
    RUN rm .gitlab-ci.yml Dockerfile docker-compose.yml README.md
```

- "COPY . ." : copy the file in the repo Gitlab into the work directory `/usr/local/apache2/htdocs`.

c. File .gitlab-ci.yml

```
stages:
  - build
  - test
  - docker

build:
  stage: build
  image: httpd:latest
  script:
    - apachectl -t


test:
  stage: test
  image: httpd:latest
  script: apachectl -t
  coverage: /All files\s*\|\s*([\d\.]+)/


docker:
  stage: docker
  image: docker:latest
  services:
    - docker:dind
  script:
    - apk add --update py-pip && pip install docker-compose
    - docker login -u gitlab-ci-token -p "xxx" "registry.gitlab.com"
    - docker-compose build
    - docker push "registry.gitlab.com/damiendeberthe/my-awesome-project"
```

- The `gitlab-ci-token` is created in the menu: User Settings → Token → Select API, give a name and an expiration date.
- Docker push: select the same registry as you chose in the `docker-compose.yml`

You will see a new Docker image in your Gitlab registry.

4. Deploy in production

a. Manually

Connect your Docker server to your Gitlab account:

```
docker login registry.gitlab.com
```

Simple install

```
docker create -p 80:80 --name=my-awesome-project-web
registry.gitlab.com/damiendeberthe/my-awesome-project:latest
docker start my-awesome-project-web
```

OU

```
docker run -d -p 80:80 --name=my-awesome-project-web
registry.gitlab.com/damiendeberthe/my-awesome-project:latest
```

Update

```
docker stop my-awesome-project-web ;
docker rm my-awesome-project-web ;
docker rmi registry.gitlab.com/damiendeberthe/my-awesome-project:latest ;
docker create -p 80:80 --name=my-awesome-project-web
registry.gitlab.com/damiendeberthe/my-awesome-project:latest \
&& docker start my-awesome-project-web
```

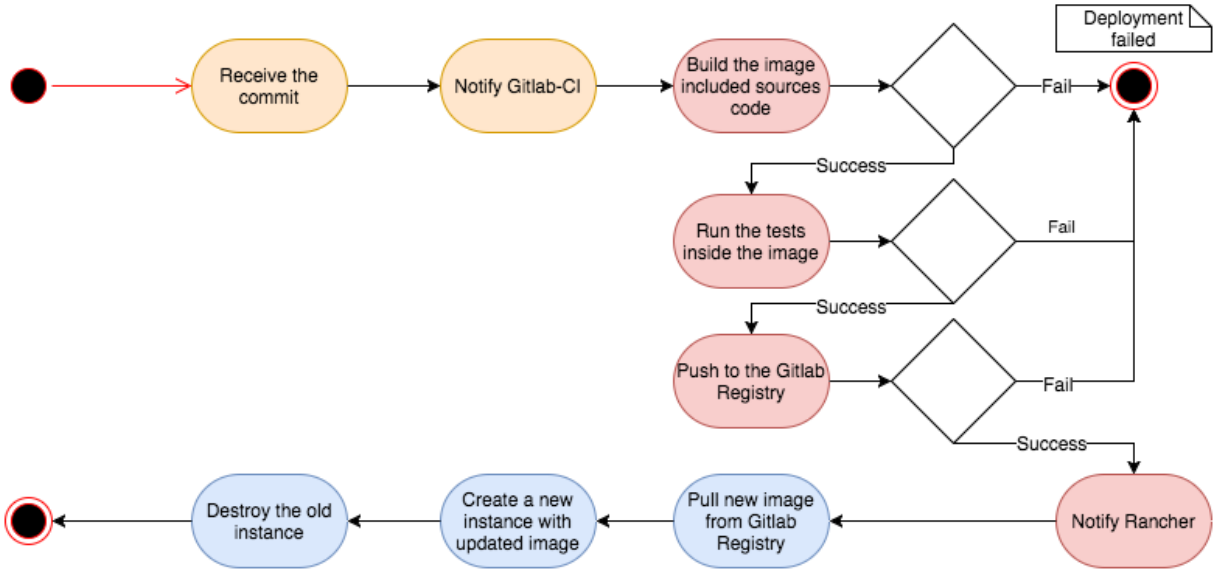
OU

```
docker stop my-awesome-project-web ;
docker rm my-awesome-project-web ;
docker rmi registry.gitlab.com/damiendeberthe/my-awesome-project:latest ;
docker run -d -p 80:80 --name=my-awesome-project-web
registry.gitlab.com/damiendeberthe/my-awesome-project:latest
```

📚 Source: https://connect.adfab.fr/outils/gitlab-1-git-clone

b. Automatically with Rancher

First, it's important to know that we use the **Rancher v.1.6**.



Variables environment

First if all, on Gitlab, we need to add variables for Rancher: "Paramètres" → "Intégration et livraison continues"

- **RANCHER_URL**: <http://your-awesome-rancher.domain.com>
  - The URL to access at your Rancher. If you use gitlab.com, you have to give access with a domain name on WAN network.
- **RANCHER_ACCESS_KEY**: <Access_Key>
  - On Rancher: In your environment → API → Keys → Add
    Here is the "clef d'accès"
- **RANCHER_SECRET_KEY**: <Secret_key>
  Here is the "clé secret" (mot de passe or password)
- **RANCHER_ENV**: <Environment_name>
- **RANCHER_STACK**: <Stack_name>
- **RANCHER_SERVICE**: <Service_name>

The *<Environment_name>*, *<Stack_name>* and *<Service_name>* need to exist before to continue.

Connection to registry Gitlab

We need to connect Rancher at our Gitlab account for pull the Docker container.

On Rancher, in our environment: "Infrastructure" → "Registry":

- Adress: registry.gitlab.com
- User: <your_mail_on_Gitlab>
- Password: <your_pass_on_Gitlab>

File .gitlab-ci.yml with CDRX

```
stages:
  - build
  - test
  - docker
  - deploy

build:
  stage: build
  image: httpd:latest
  script:
    - apachectl -t
  only:
  - master

test:
  stage: test
  image: httpd:latest
  script: apachectl -t
  coverage: /All files\s*\|\s*([\d\.]+)/
  only:
  - master

docker:
  stage: docker
  image: docker:latest
  services:
    - docker:dind
  script:
    - apk add --update py-pip && pip install docker-compose
    - docker login -u gitlab-ci-token -p "xxx" "registry.gitlab.com"
    - docker-compose build
    - docker push "registry.gitlab.com/damiendeberthe/my-awesome-project"
  only:
  - master

deploy:
  stage: deploy
  image: cdrx/rancher-gitlab-deploy
  script:
    - upgrade --environment $RANCHER_ENV --stack $RANCHER_STACK --service
$RANCHER_SERVICE --start-before-stopping --no-wait-for-upgrade-to-finish
  only:
  - master
  when: manual
```

At the stage « deploy » we use the image `cdrx/rancher-gitlab-deploy` and we deploy automatically (with the option `--start-before-stopping`) our Docker container into Rancher.

Be careful! You need to manually execute the stage “deploy” in the pipeline.

Help: https://github.com/cdrx/rancher-gitlab-deploy

- Source: https://anthonykgross.fr/p/deploiement-continu-docker-gitlab-rancher
  Need to test: https://mikaoelitiana.name/fr/deployer-image-docker-rancher-gitlab/