Anthony LOUREIRO

Daniel DA CUNHA GOMES

CI/CD Project with GitLab CI

1. Three Ubuntu Server virtual machines

2. Setting up the CI/CD Pipeline

3. Configuring the Docker server

4. Deploy the application

1. Three Ubuntu Server virtual machines

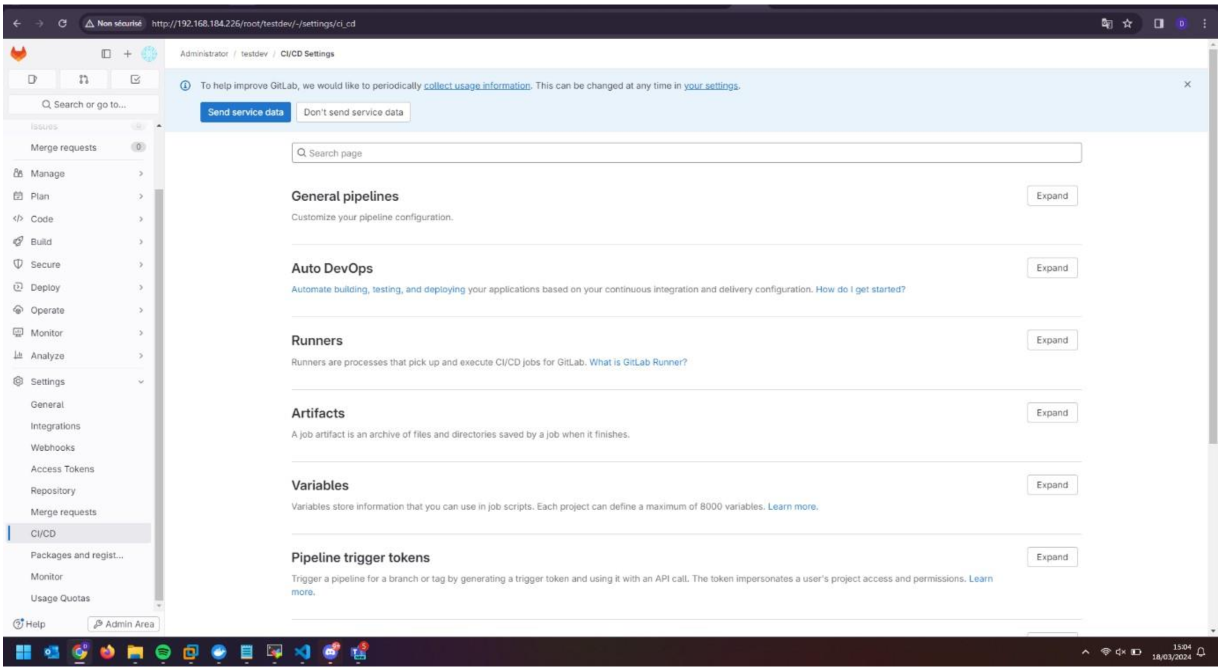
Prerequisites

Before getting started,we need to setup 3 Ubuntu VM :

1 - gitlab-instance: Host for the GitLab instance.

On the VM we need to Install GitLab on the gitlab-instance virtual machine

(We can see the GitLab instance is working)



To help improve GitLab, we would like to periodically collect usage information. This can be changed at any time in your settings.

General pipelines

Customize your pipeline configuration.

Auto DevOps

Automate building, testing, and deploying your applications based on your continuous integration and delivery configuration. How do I get started?

Runners

Runners are processes that pick up and execute CI/CD jobs for GitLab. What is GitLab Runner?

Artifacts

A job artifact is an archive of files and directories saved by a job when it finishes.

Variables

Variables store information that you can use in job scripts. Each project can define a maximum of 8000 variables. Learn more.

Pipeline trigger tokens

Trigger a pipeline for a branch or tag by generating a trigger token and using it with an API call. The token impersonates a user's project access and permissions. Learn more.

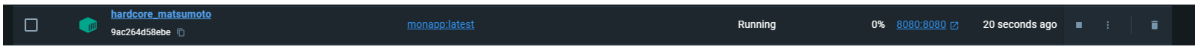
2 - gitlab-runner: Host for the GitLab runner.

On the VM we need to Install GitLab Runner on the gitlab-runner virtual machine to run our pipeline jobs

3 - dev-srv: Deployment host for the application.

On the VM we need to Install Docker on the dev-srv virtual machine to later on deploy our application.

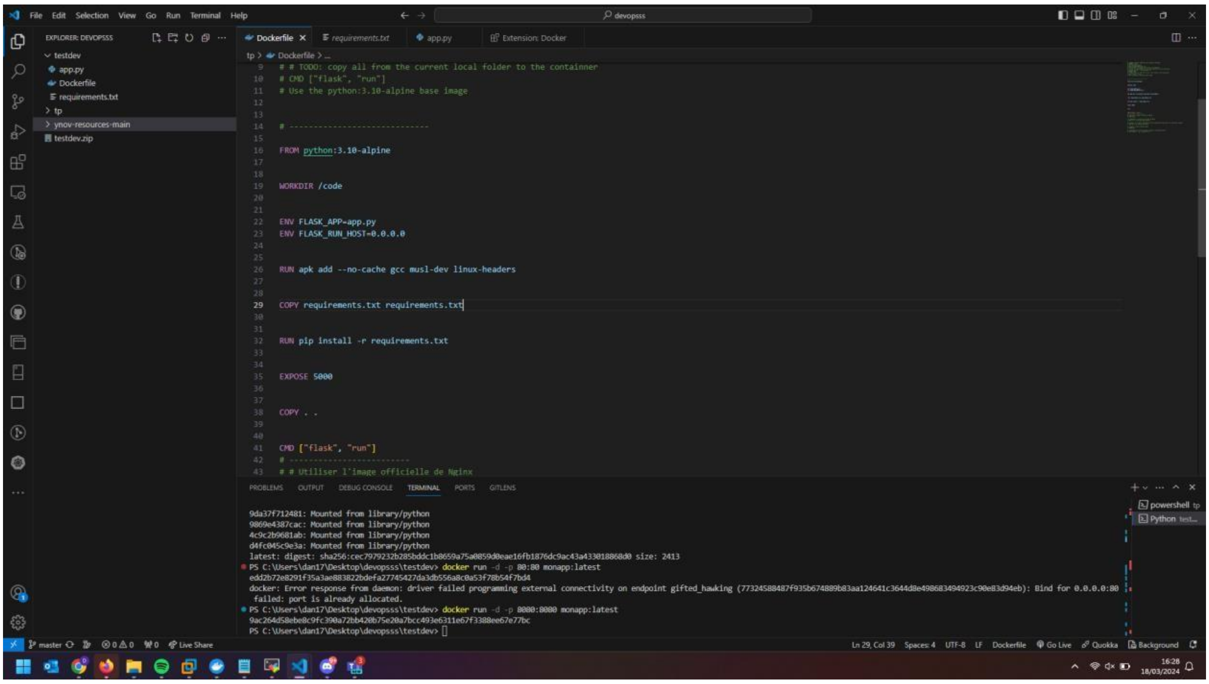
AND – a Docker Hub account to publish the containerized application image.



□ hardcore_matsumoto 9ac264d58ebe monapp:latest Running 0% 8080:8080 20 seconds ago

We can see a container on Docker Hub

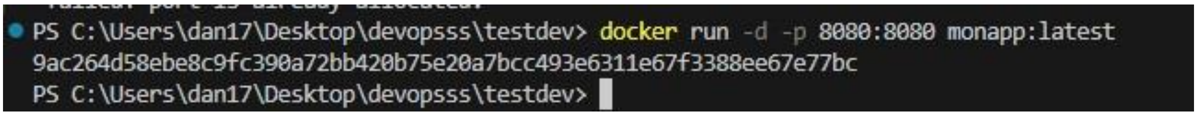
Begin by containerizing the application using Docker. This involves creating a Docker file that specifies the environment and dependencies needed to run the application.



```
# # TODO: copy all from the current local folder to the container
# CMD ["flask", "run"]
# Use the python:3.10-alpine base image


# ----------------------------

FROM python:3.10-alpine


WORKDIR /code


ENV FLASK_APP=app.py
ENV FLASK_RUN_HOST=0.0.0.0


RUN apk add --no-cache gcc musl-dev linux-headers


COPY requirements.txt requirements.txt


RUN pip install -r requirements.txt


EXPOSE 5000


COPY . .


CMD ["flask", "run"]
# -------------------------
# # Utiliser l'image officielle de Nginx
```

```
9da37f712481: Mounted from library/python
9869e4387cac: Mounted from library/python
4c9c2b9681ab: Mounted from library/python
d4fc045c9e3a: Mounted from library/python
latest: digest: sha256:cec7979232b285bddc1b8659a75a0859d0eae16fb1876dc9ac43a433018868d0 size: 2413
PS C:\Users\dan17\Desktop\devopsss\testdev> docker run -d -p 80:80 monapp:latest
edd2b72e8291f35a3ae883822bdefa27745427da3db556a8c0a53f78b54f7bd4
docker: Error response from daemon: driver failed programming external connectivity on endpoint gifted_hawking (77324588487f935b674889b83aa124641c3644d8e498683494923c90e83d94eb): Bind for 0.0.0.0:80
 failed: port is already allocated.
PS C:\Users\dan17\Desktop\devopsss\testdev> docker run -d -p 8080:8080 monapp:latest
9ac264d58ebe8c9fc390a72bb420b75e20a7bcc493e6311e67f3388ee67e77bc
PS C:\Users\dan17\Desktop\devopsss\testdev> 
```

We can see that's we configure the Docker File well, and the image below tell us, the Docker container is working

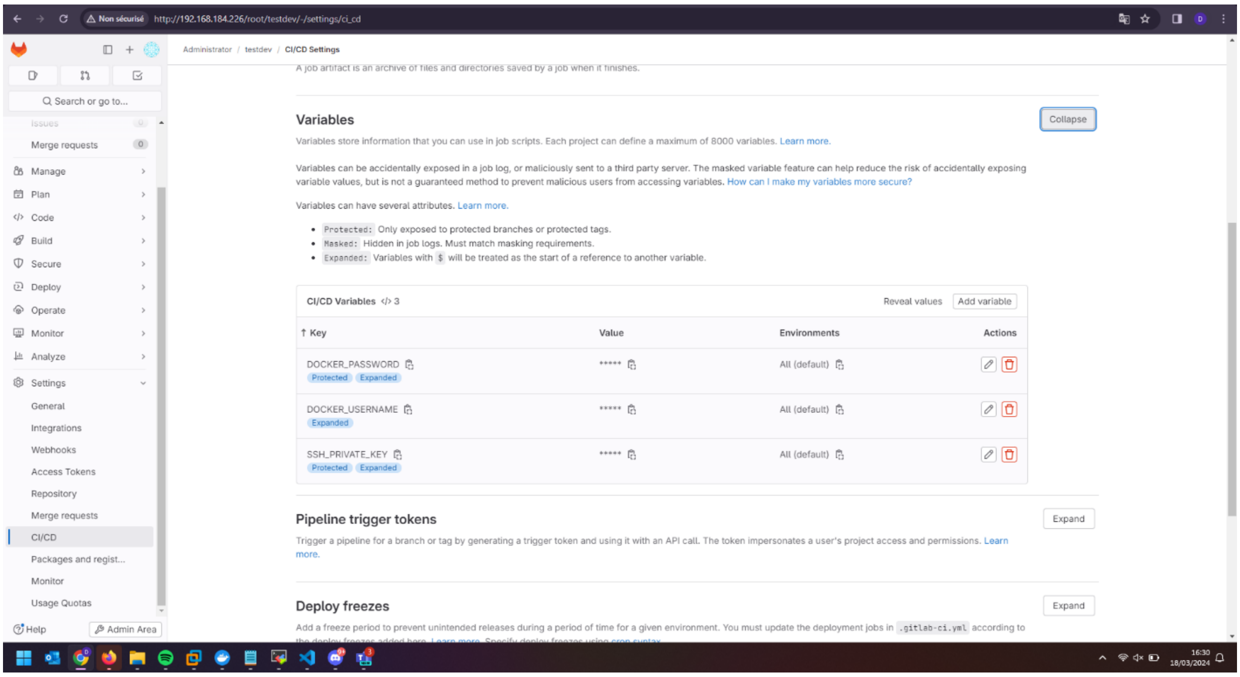


```
PS C:\Users\dan17\Desktop\devopsss\testdev> docker run -d -p 8080:8080 monapp:latest
9ac264d58ebe8c9fc390a72bb420b75e20a7bcc493e6311e67f3388ee67e77bc
PS C:\Users\dan17\Desktop\devopsss\testdev> 
```

2. Setting up the CI/CD Pipeline

Configure the CI/CD pipeline using GitLab CI. This can be achieved by creating a .gitlab-ci.yml file at the root of our project to create our pipeline

This file used some CI/CD variables that we need to add to our Gitlab CI/CD in the CI/CD settings like you can see below :



http://192.168.184.226/root/testdev/-/settings/ci_cd
Administrator / testdev / CI/CD Settings
Collapse
Expand
Expand

A job artifact is an archive of files and directories saved by a job when it finishes.

Variables

Variables store information that you can use in job scripts. Each project can define a maximum of 8000 variables. Learn more.

Variables can be accidentally exposed in a job log, or maliciously sent to a third party server. The masked variable feature can help reduce the risk of accidentally exposing variable values, but is not a guaranteed method to prevent malicious users from accessing variables. How can I make my variables more secure?

Variables can have several attributes. Learn more.

- `Protected:` Only exposed to protected branches or protected tags.
- `Masked:` Hidden in job logs. Must match masking requirements.
- `Expanded:` Variables with `$` will be treated as the start of a reference to another variable.

CI/CD Variables </> 3 Reveal values Add variable

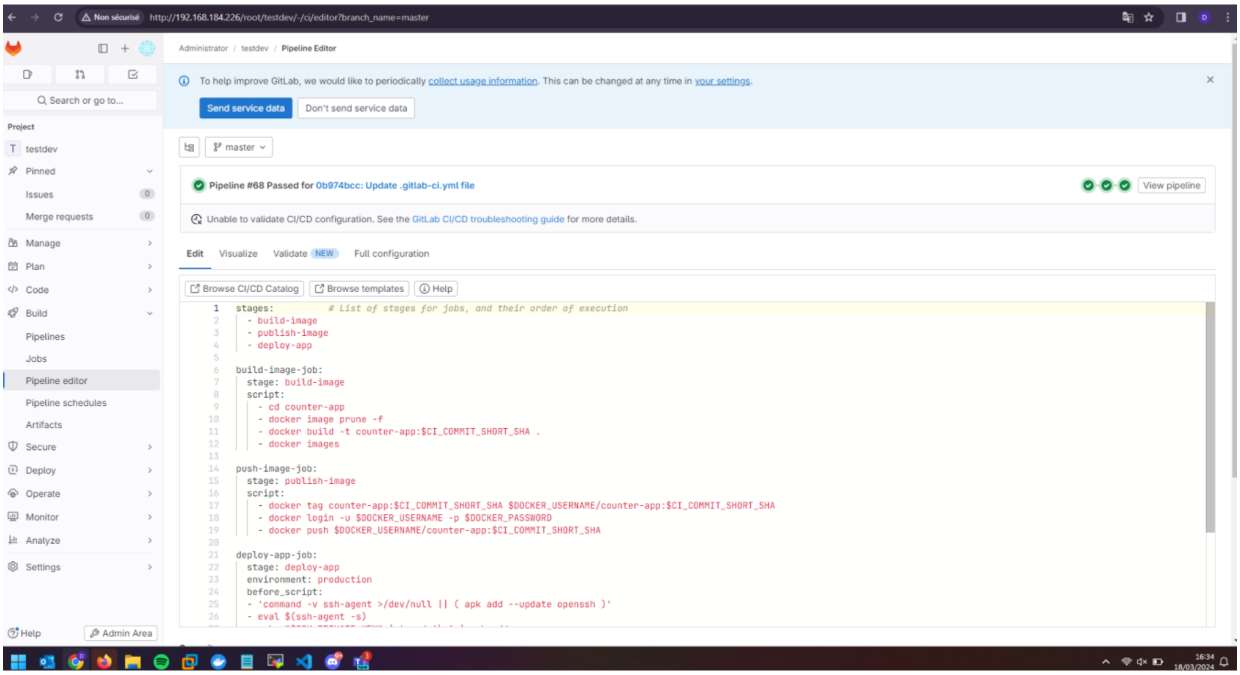
↑ Key	Value	Environments	Actions
DOCKER_PASSWORD Protected Expanded	*****	All (default)	
DOCKER_USERNAME Expanded	*****	All (default)	
SSH_PRIVATE_KEY Protected Expanded	*****	All (default)	

Pipeline trigger tokens

Trigger a pipeline for a branch or tag by generating a trigger token and using it with an API call. The token impersonates a user's project access and permissions. Learn more.

Deploy freezes

Add a freeze period to prevent unintended releases during a period of time for a given environment. You must update the deployment jobs in `.gitlab-ci.yml` according to



To help improve GitLab, we would like to periodically collect usage information. This can be changed at any time in your settings.

Pipeline #68 Passed for 0b974bcc: Update .gitlab-ci.yml file

Unable to validate CI/CD configuration. See the GitLab CI/CD troubleshooting guide for more details.

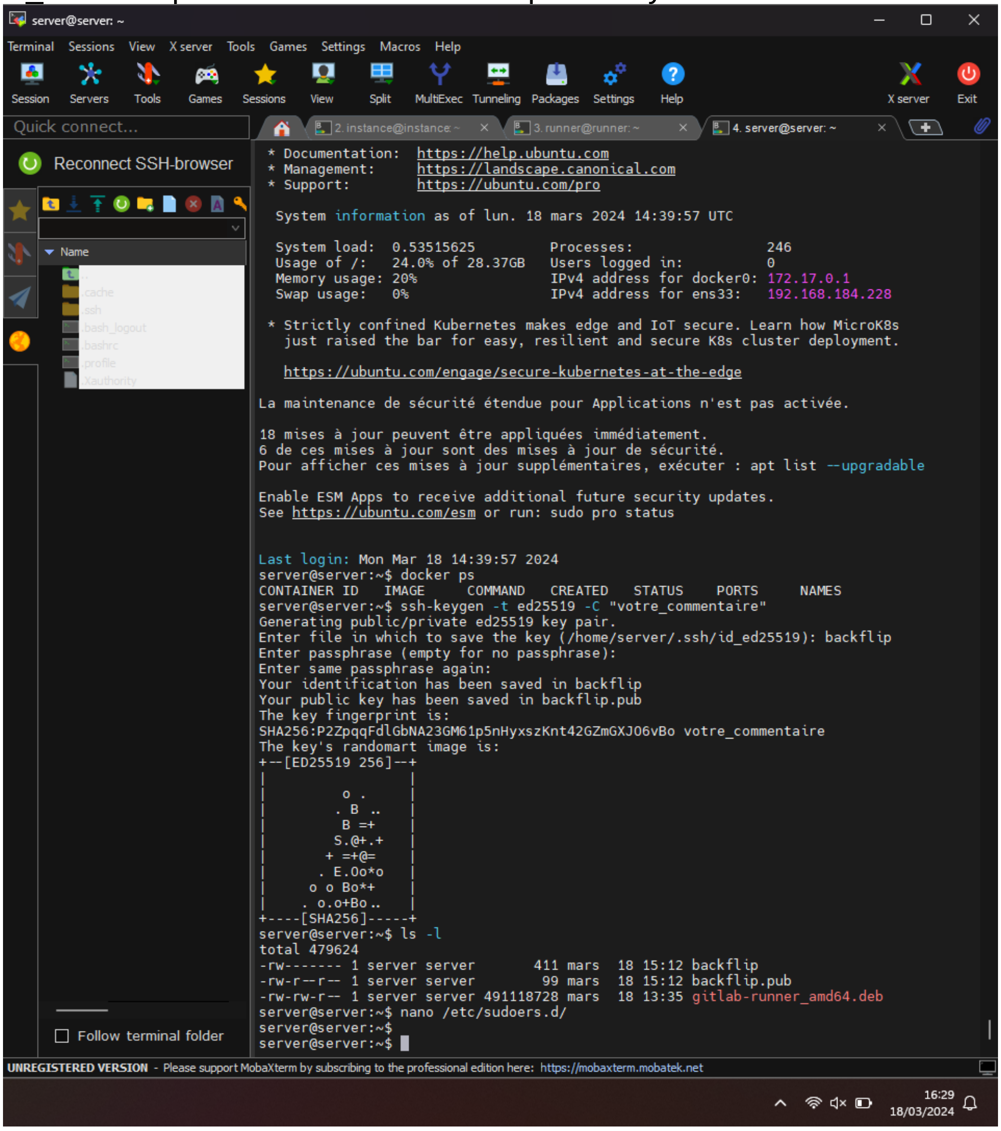
```yaml
stages:          # List of stages for jobs, and their order of execution
  - build-image
  - publish-image
  - deploy-app

build-image-job:
  stage: build-image
  script:
    - cd counter-app
    - docker image prune -f
    - docker build -t counter-app:$CI_COMMIT_SHORT_SHA .
    - docker images

push-image-job:
  stage: publish-image
  script:
    - docker tag counter-app:$CI_COMMIT_SHORT_SHA $DOCKER_USERNAME/counter-app:$CI_COMMIT_SHORT_SHA
    - docker login -u $DOCKER_USERNAME -p $DOCKER_PASSWORD
    - docker push $DOCKER_USERNAME/counter-app:$CI_COMMIT_SHORT_SHA

deploy-app-job:
  stage: deploy-app
  environment: production
  before_script:
  - 'command -v ssh-agent >/dev/null || ( apk add --update openssh )'
  - eval $(ssh-agent -s)
```

3. Configuring the Docker server

Now that our gitlab is configured correctly and docker is installed on our docker server. We need at first to give root rights to our machine user so our pipeline jobs will be able to execute docker commands. For this we will modify the file in the folder « etc » *sudoers* by adding at the end of the file this line for my user « warshall » :

```
warshall  ALL=(ALL) NOPASSWD: ALL
```

To allow our pipeline to connect to our remote machine by ssh, we need to generate SSH private & public keys.
For this we will run the command « ssh-keygen -t ed25519 -C "GitLab SSH key" », that will generate 2 files in the folder « .ssh » :
id_ed25519 -> which contains the private key
id_ed25519.pub -> which contains the public key



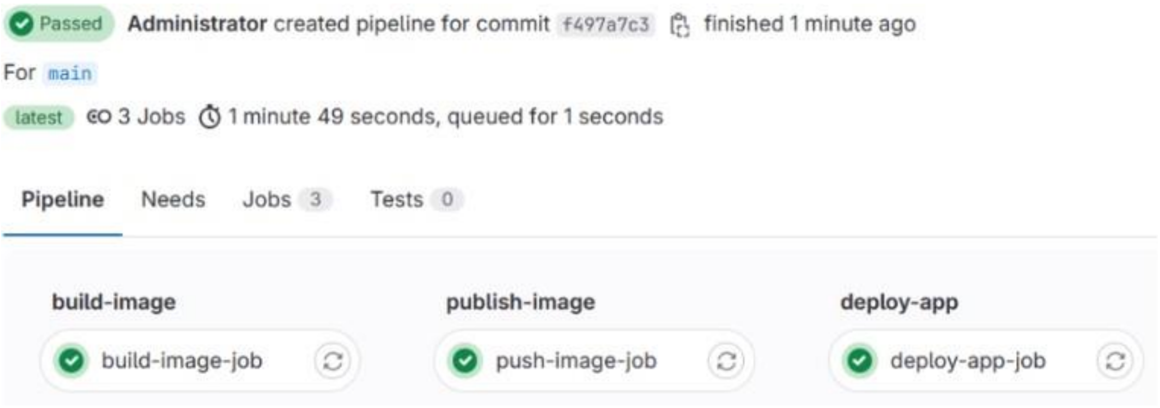
```
 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

  System information as of lun. 18 mars 2024 14:39:57 UTC

  System load:  0.53515625        Processes:                246
  Usage of /:   24.0% of 28.37GB  Users logged in:          0
  Memory usage: 20%               IPv4 address for docker0: 172.17.0.1
  Swap usage:   0%                IPv4 address for ens33:   192.168.184.228

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

La maintenance de sécurité étendue pour Applications n'est pas activée.

18 mises à jour peuvent être appliquées immédiatement.
6 de ces mises à jour sont des mises à jour de sécurité.
Pour afficher ces mises à jour supplémentaires, exécuter : apt list --upgradable

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status


Last login: Mon Mar 18 14:39:57 2024
server@server:~$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
server@server:~$ ssh-keygen -t ed25519 -C "votre_commentaire"
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/server/.ssh/id_ed25519): backflip
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in backflip
Your public key has been saved in backflip.pub
The key fingerprint is:
SHA256:P2ZpqqFdlGbNA23GM61p5nHyxszKnt42GZmGXJO6vBo votre_commentaire
The key's randomart image is:
+--[ED25519 256]--+
|                 |
|         o .     |
|        . B ..   |
|         B =+    |
|        S.@+.+   |
|       + =+@=    |
|      . E.Oo*o   |
|     o o Bo*+    |
|    . o.o+Bo..   |
+----[SHA256]-----+
server@server:~$ ls -l
total 479624
-rw------- 1 server server       411 mars  18 15:12 backflip
-rw-r--r-- 1 server server        99 mars  18 15:12 backflip.pub
-rw-rw-r-- 1 server server 491118728 mars  18 13:35 gitlab-runner_amd64.deb
server@server:~$ nano /etc/sudoers.d/
server@server:~$
server@server:~$
```

4. Deploy the application

build-image: Build the Docker image of the application.

publish-image: Publish the Docker image to Docker Hub.

deploy-app: Deploy the application to the dev-srv server. This stage involves pulling the Docker image from Docker Hub and running it on the deployment server using SSH.

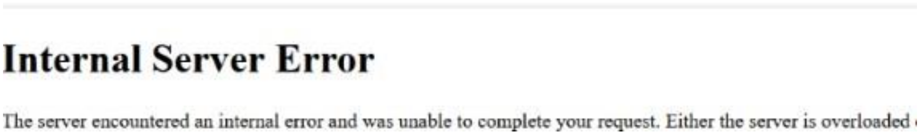


Passed **Administrator** created pipeline for commit `f497a7c3` finished 1 minute ago

For `main`

latest 3 Jobs 1 minute 49 seconds, queued for 1 seconds

Our pipeline CI/CD will build the image and push it on Docker hub and then he create a container



Internal Server Error

The server encountered an internal error and was unable to complete your request. Either the server is overloaded

Now we can see we have an error 500 because we don't have redis configured, but it proves that our application is correctly deployed