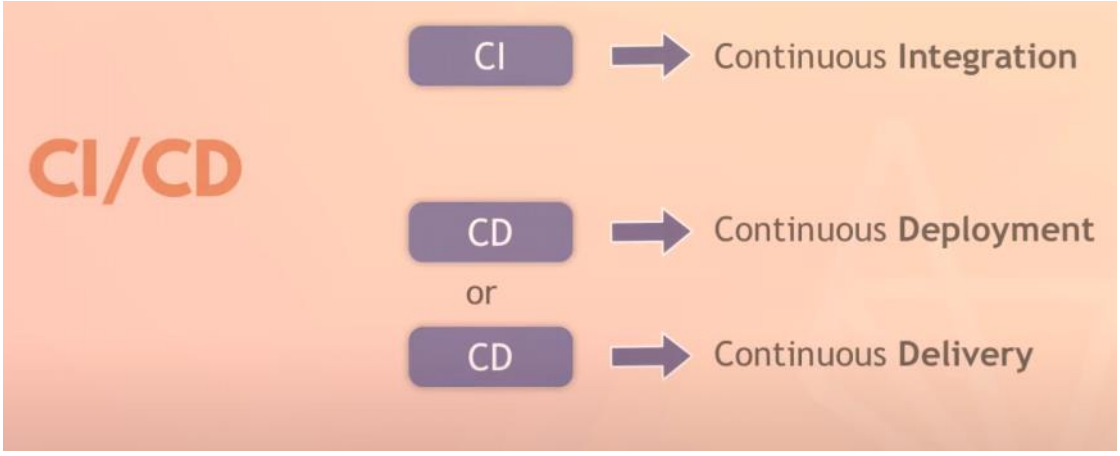
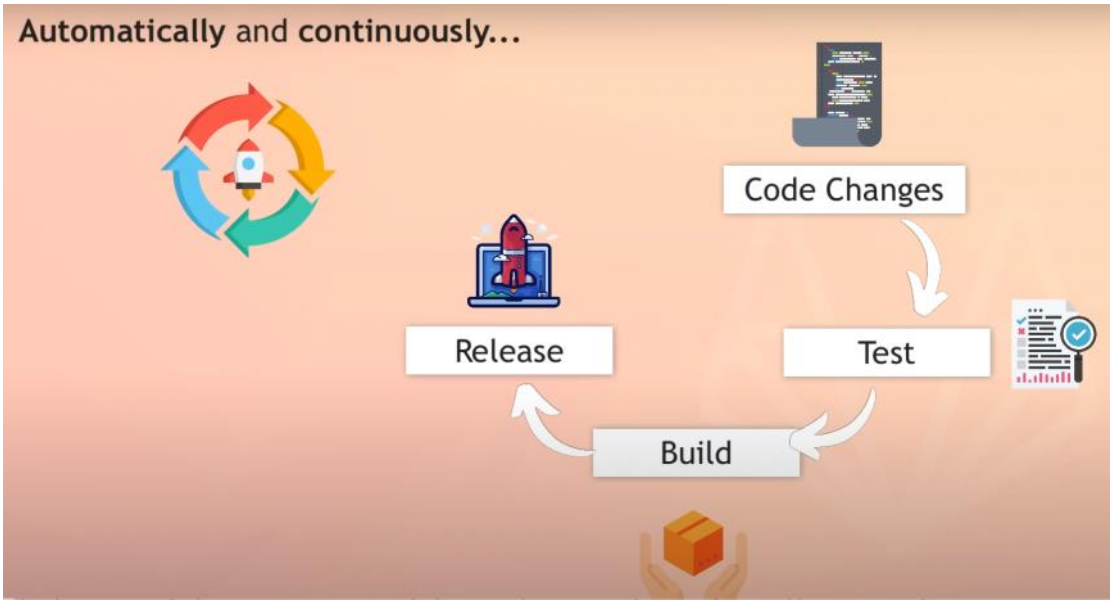
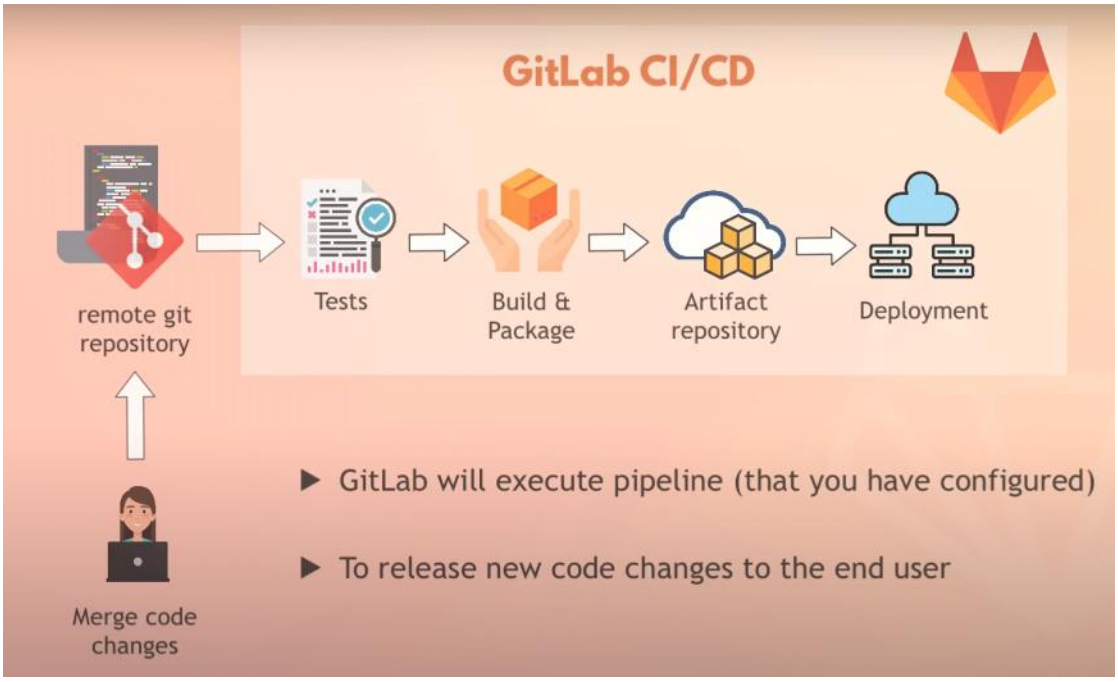
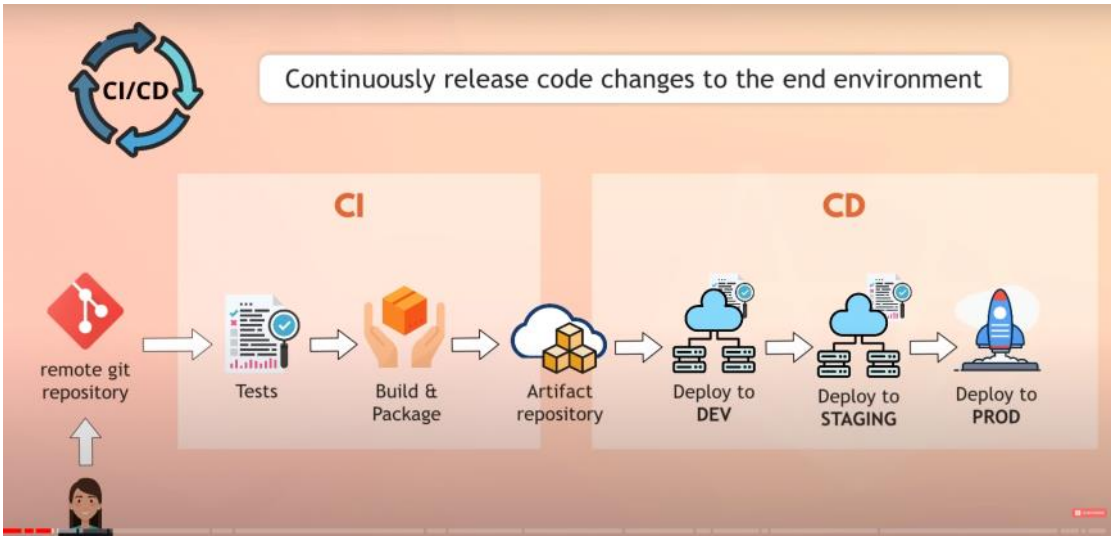
CI/CD GITLAB

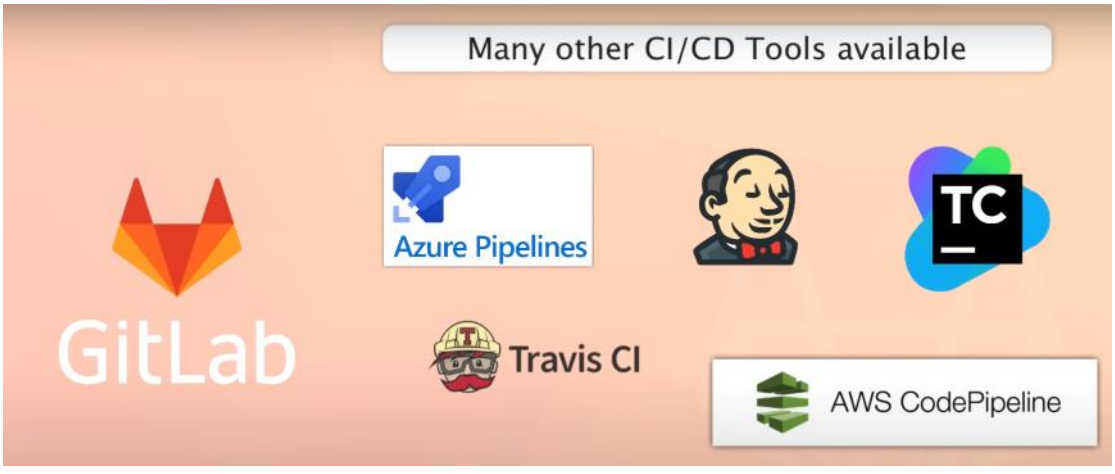
Friday, October 13, 2023 10:12 AM

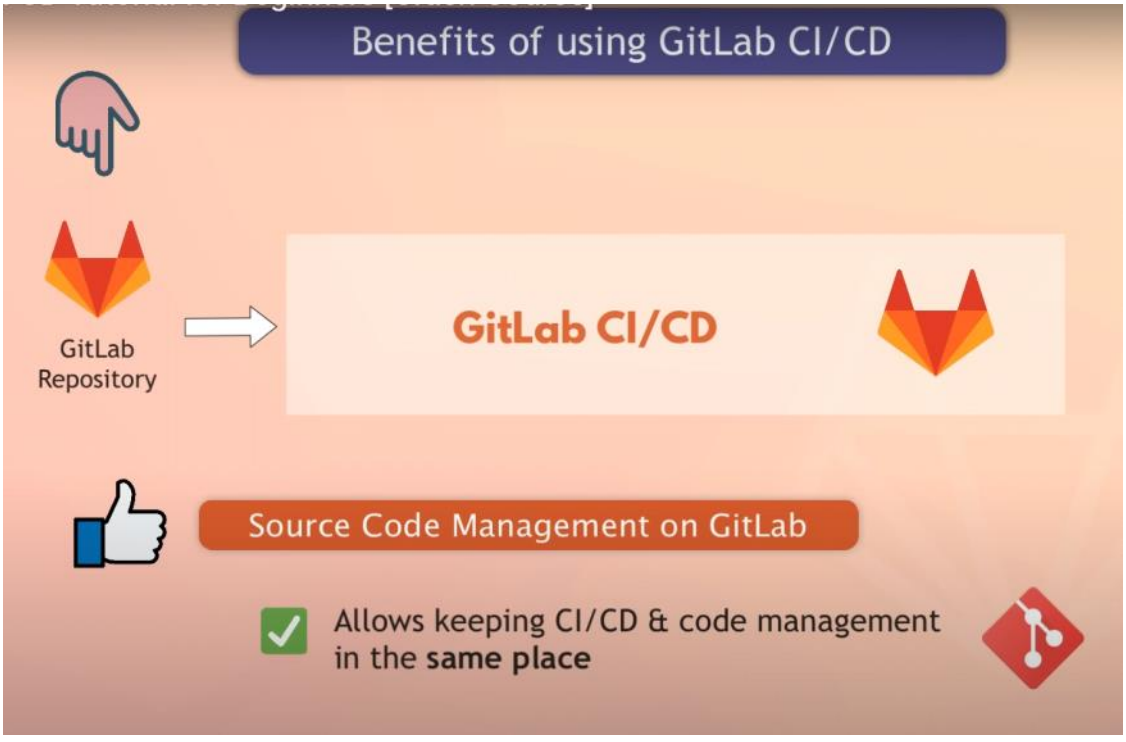


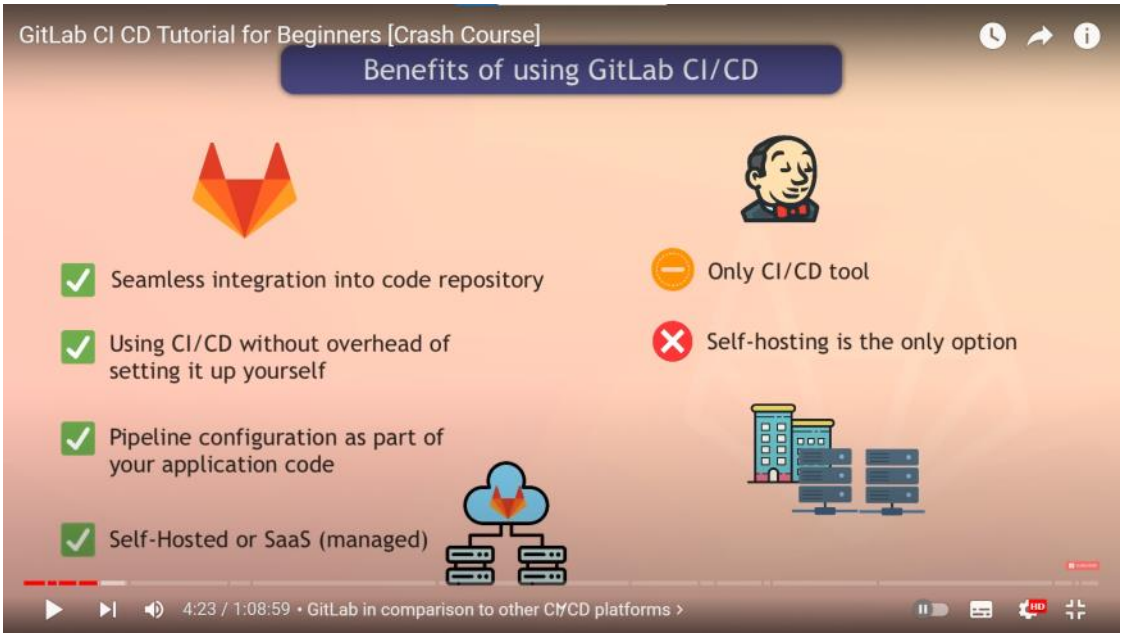


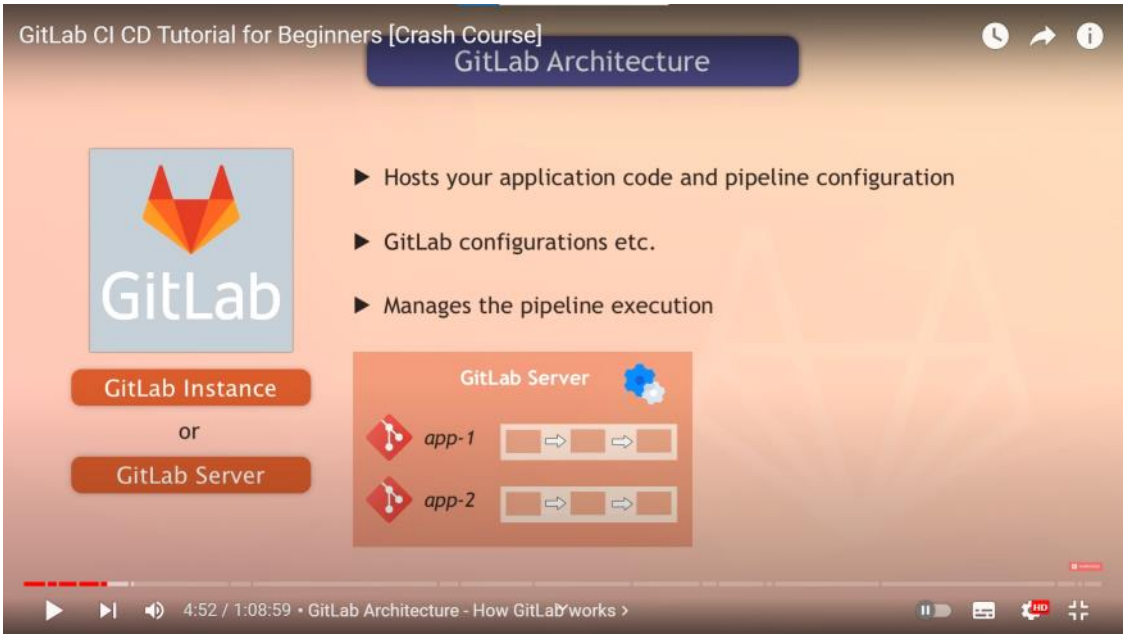

GitLab CI/CD
remote git repository
Tests
Build & Package
Artifact repository
Deployment
Merge code changes
GitLab will execute pipeline (that you have configured)
To release new code changes to the end user

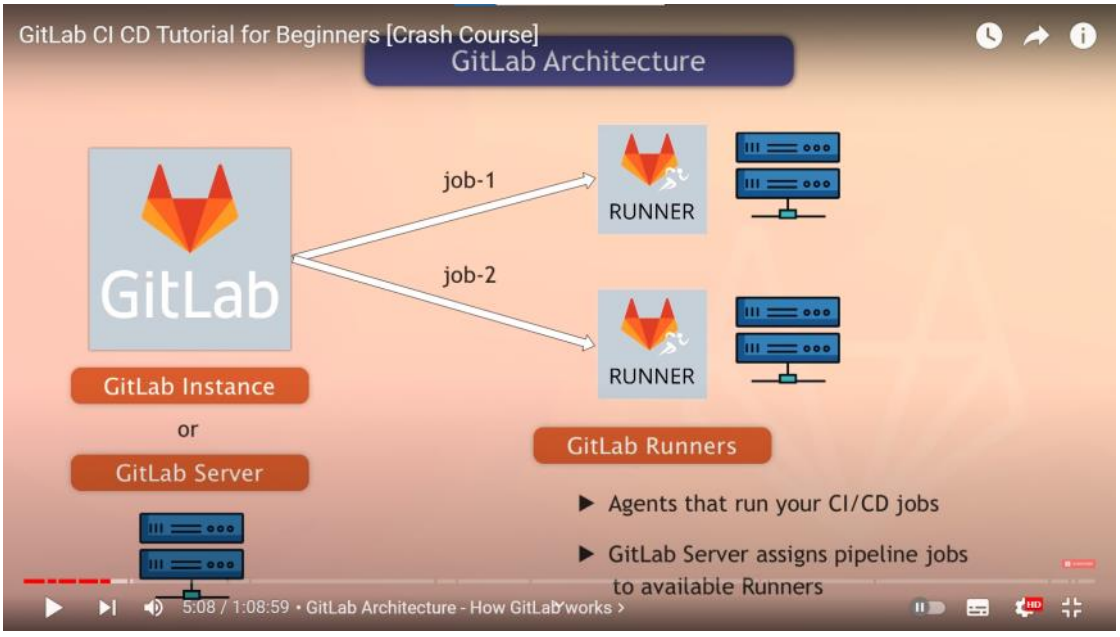

CI/CD
Continuously release code changes to the end environment
CI
CD
remote git repository
Tests
Build & Package
Artifact repository
Deploy to DEV
Deploy to STAGING
Deploy to PROD

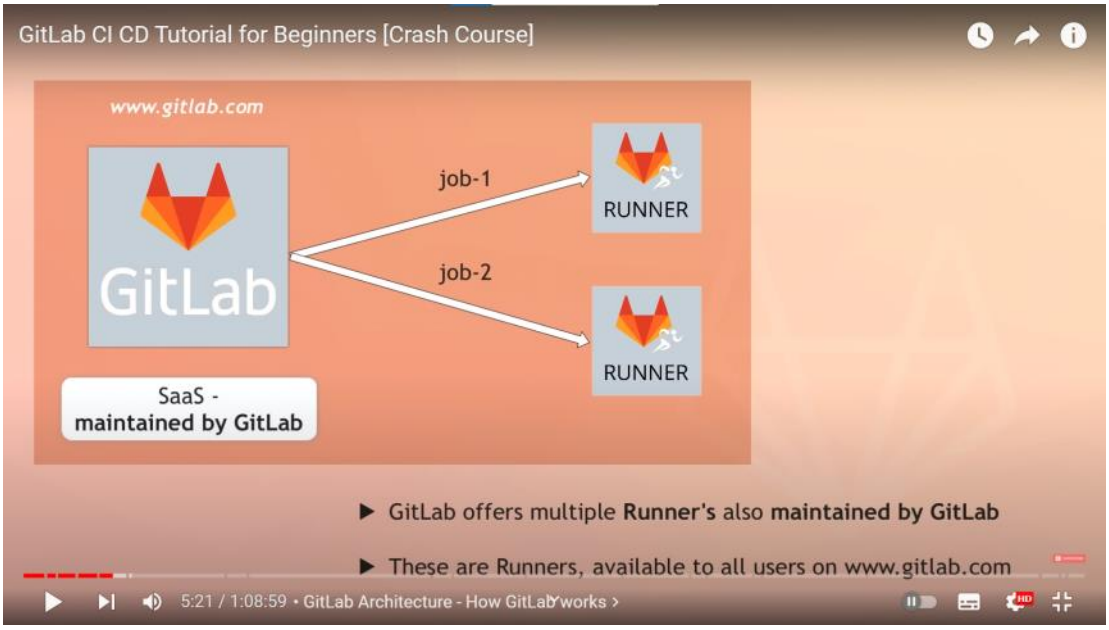

Many other CI/CD Tools available
GitLab
Azure Pipelines
Travis CI
AWS CodePipeline

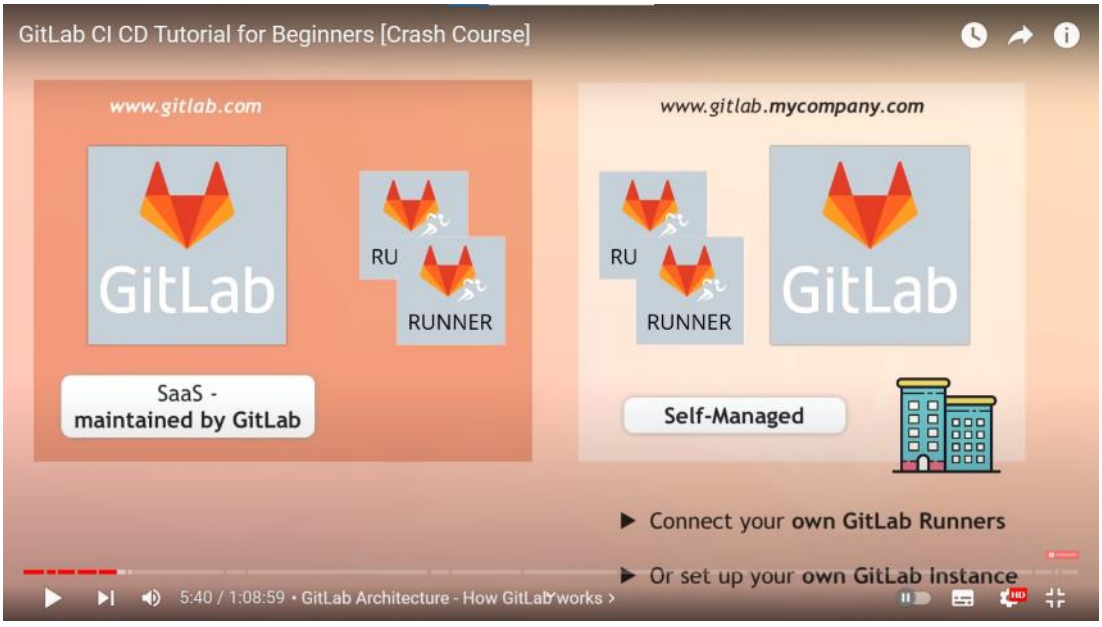

Benefits of using GitLab CI/CD
GitLab Repository
GitLab CI/CD
Source Code Management on GitLab
Allows keeping CI/CD & code management in the same place


GitLab CI CD Tutorial for Beginners [Crash Course]
Benefits of using GitLab CI/CD
Seamless integration into code repository
Using CI/CD without overhead of setting it up yourself
Pipeline configuration as part of your application code
Self-Hosted or SaaS (managed)
Only CI/CD tool
Self-hosting is the only option
4:23 / 1:08:59 • GitLab in comparison to other CI/CD platforms ›

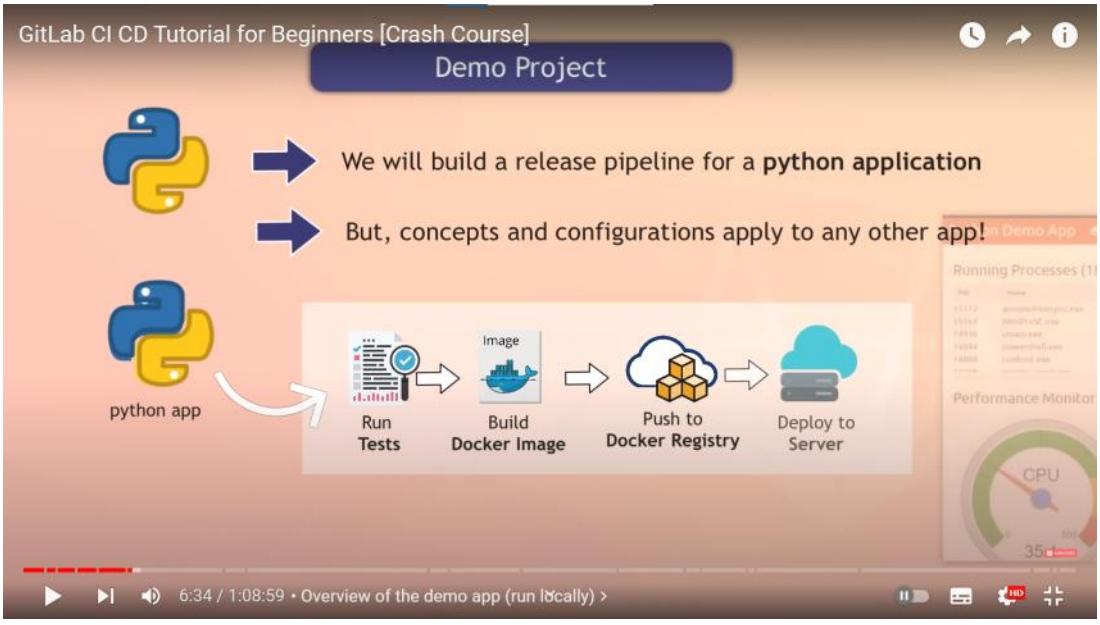

GitLab CI CD Tutorial for Beginners [Crash Course]
GitLab Architecture
GitLab
GitLab Instance
or
GitLab Server
Hosts your application code and pipeline configuration
GitLab configurations etc.
Manages the pipeline execution
GitLab Server
app-1
app-2
4:52 / 1:08:59 • GitLab Architecture - How GitLab works

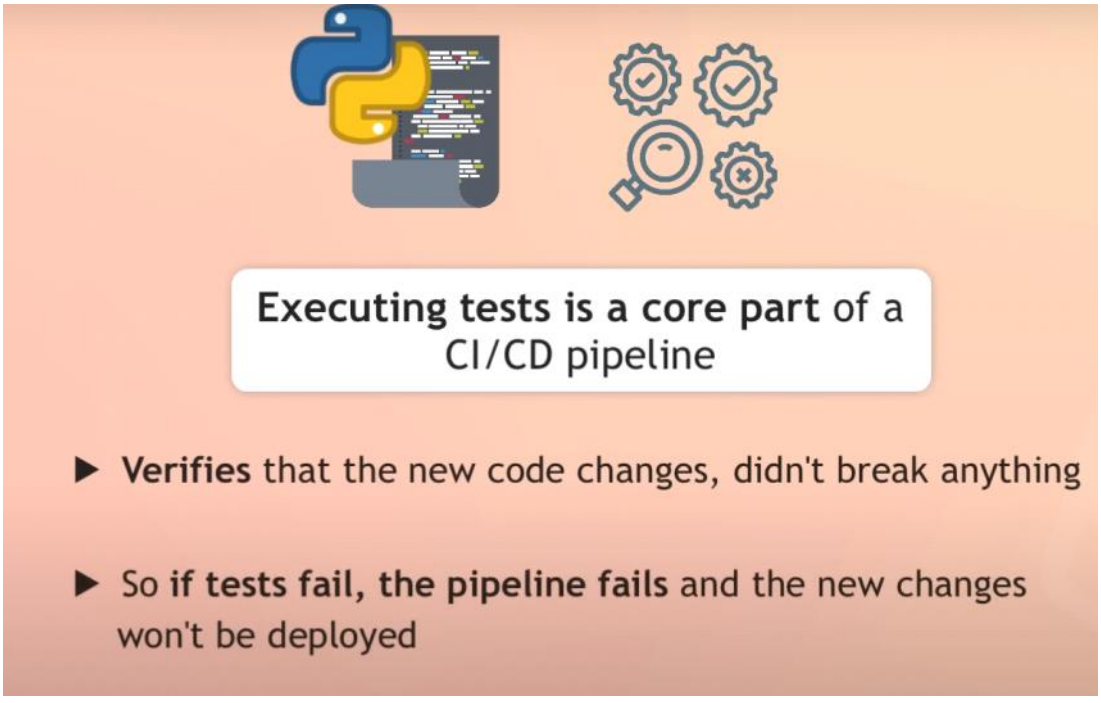

GitLab CI CD Tutorial for Beginners [Crash Course]
GitLab Architecture
GitLab
GitLab Instance
or
GitLab Server
job-1
job-2
RUNNER
RUNNER
GitLab Runners
Agents that run your CI/CD jobs
GitLab Server assigns pipeline jobs to available Runners
5:08 / 1:08:59 • GitLab Architecture - How GitLab works

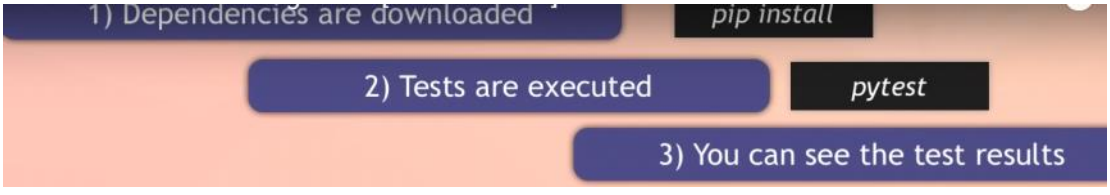




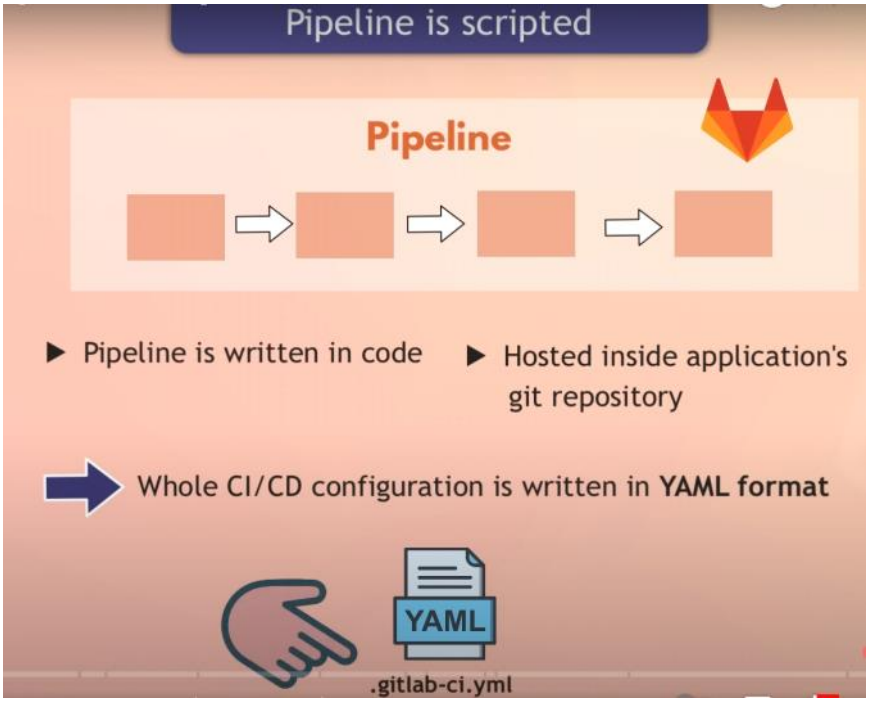
DEMO PROJECT:

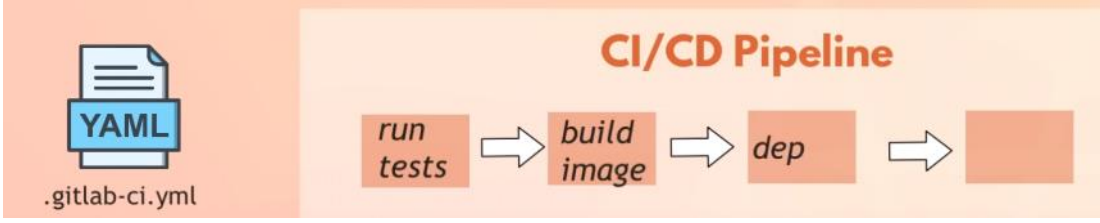

GitLab CI CD Tutorial for Beginners [Crash Course]
Demo Project
We will build a release pipeline for a python application
But, concepts and configurations apply to any other app!
python app
Run Tests
Image
Build Docker Image
Push to Docker Registry
Deploy to Server
6:34 / 1:08:59 • Overview of the demo app (run locally) >

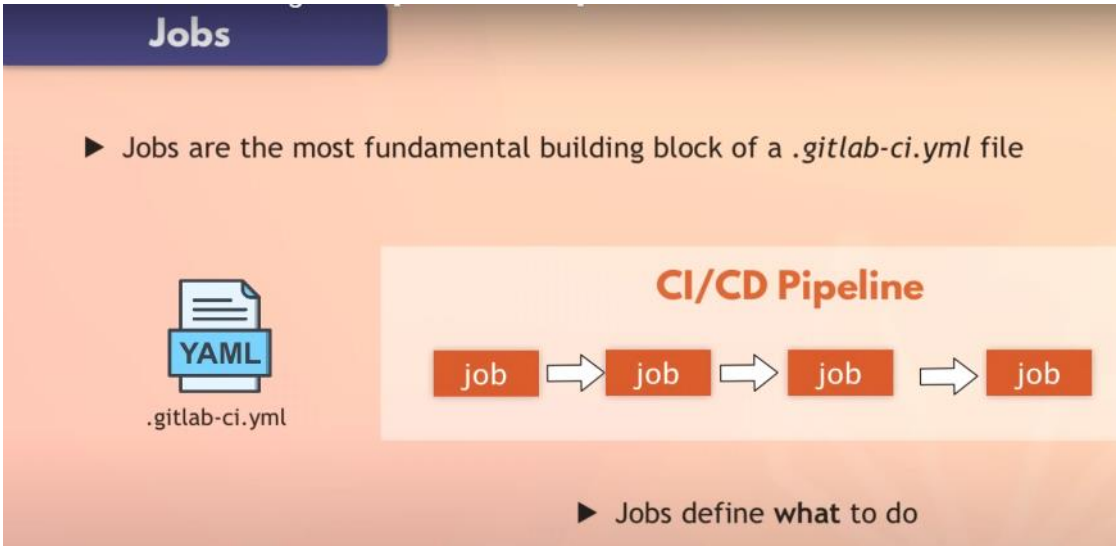

Executing tests is a core part of a CI/CD pipeline
Verifies that the new code changes, didn't break anything
So if tests fail, the pipeline fails and the new changes won't be deployed

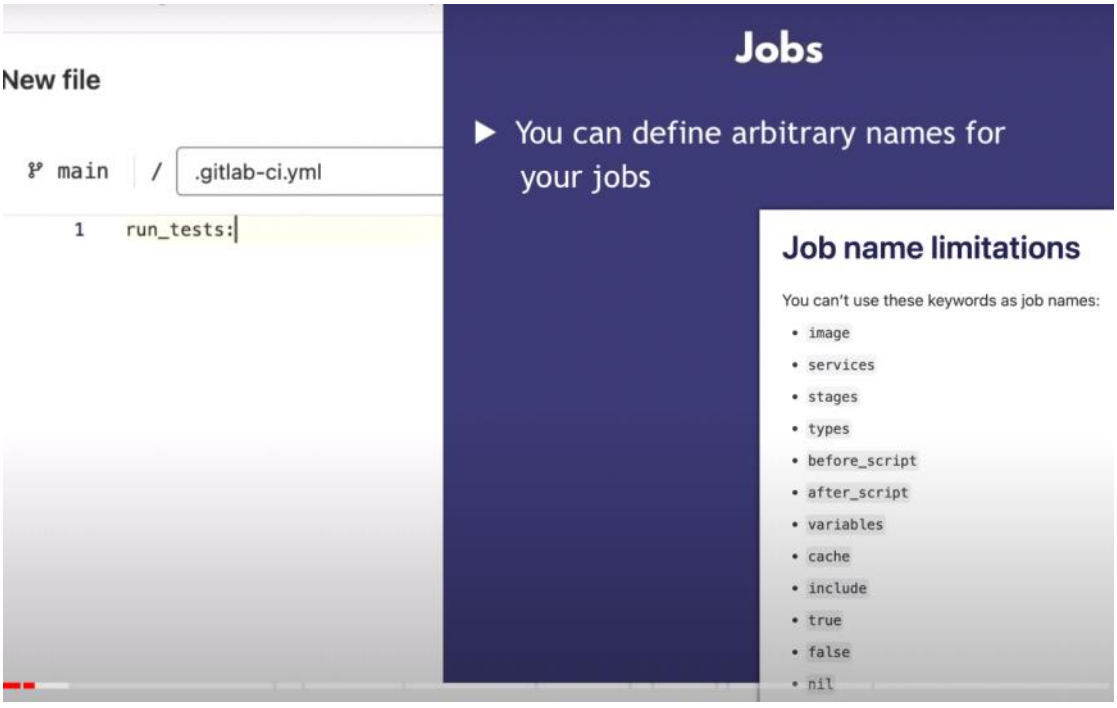

1) Dependencies are downloaded
pip install
2) Tests are executed
pytest
3) You can see the test results

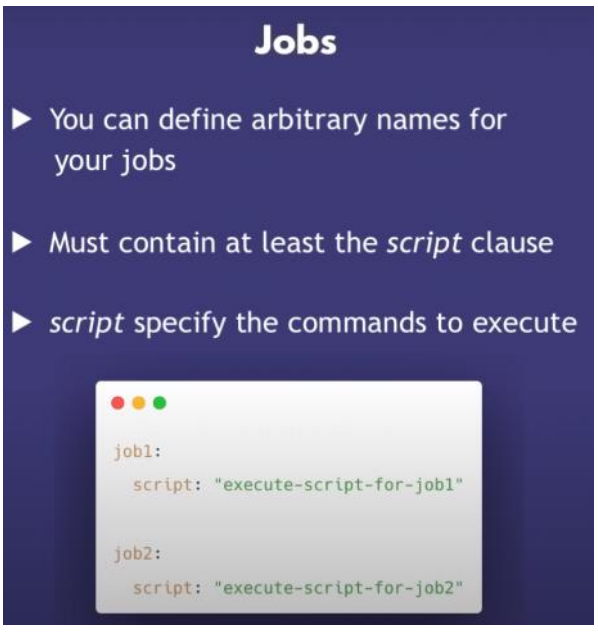
```
[\W (main)]$ export PORT=5004
[\W (main)]$ make run
```



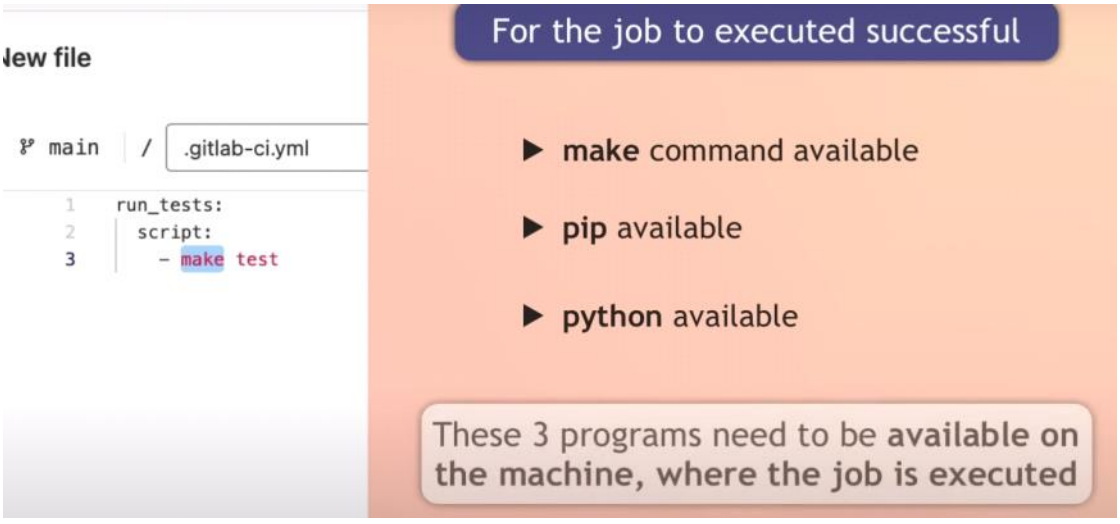




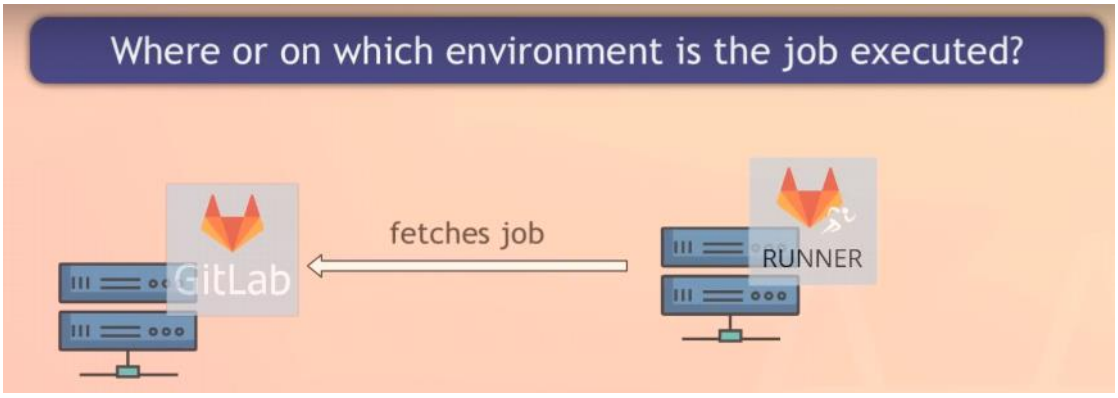




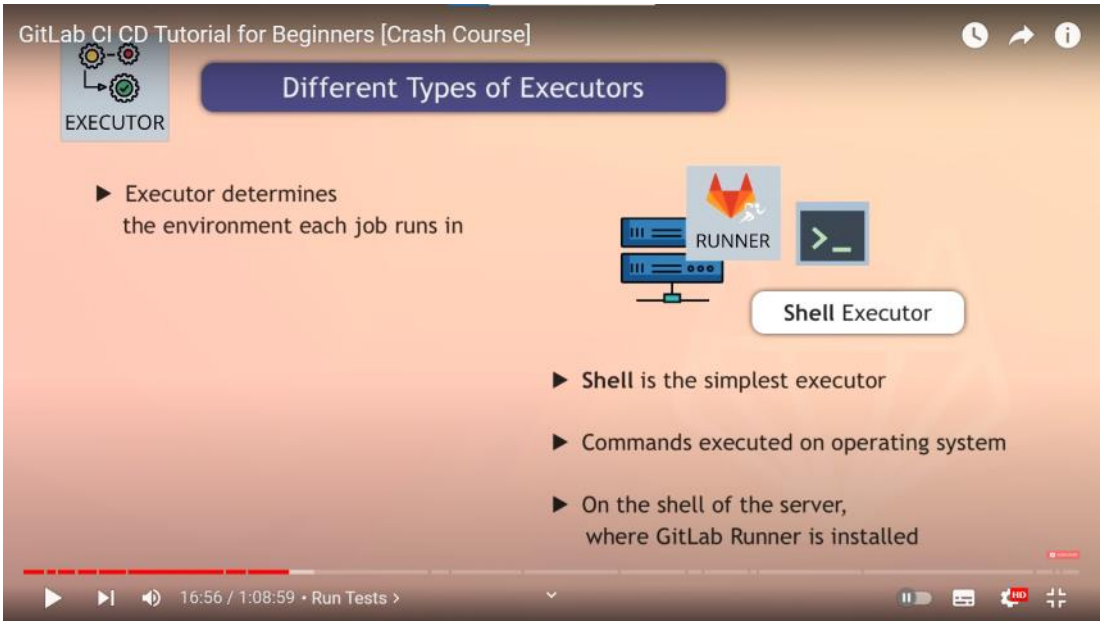
Job Configuration to Run a Test:

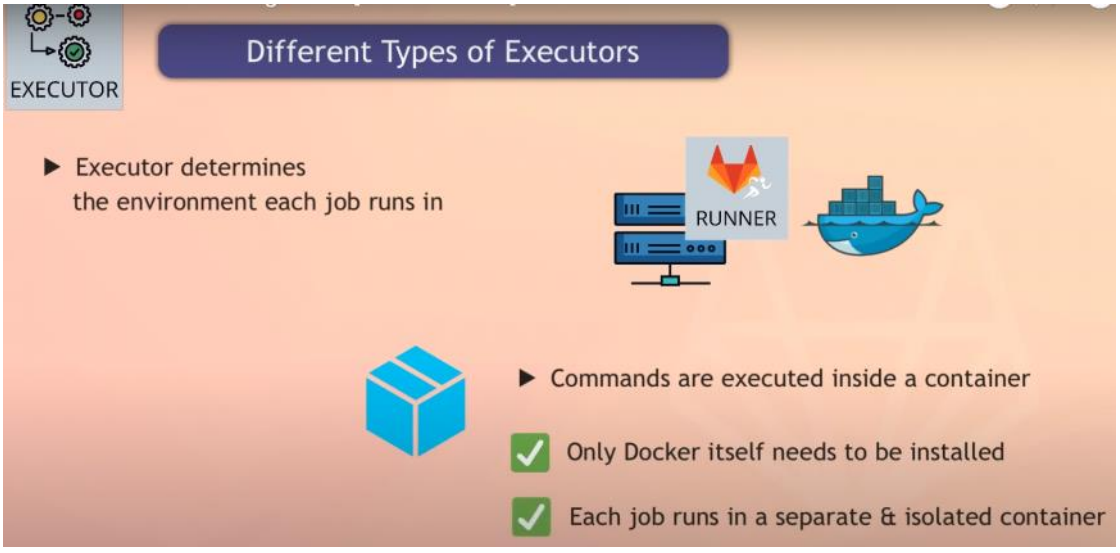


GitLab Jobs are executed on Runners:

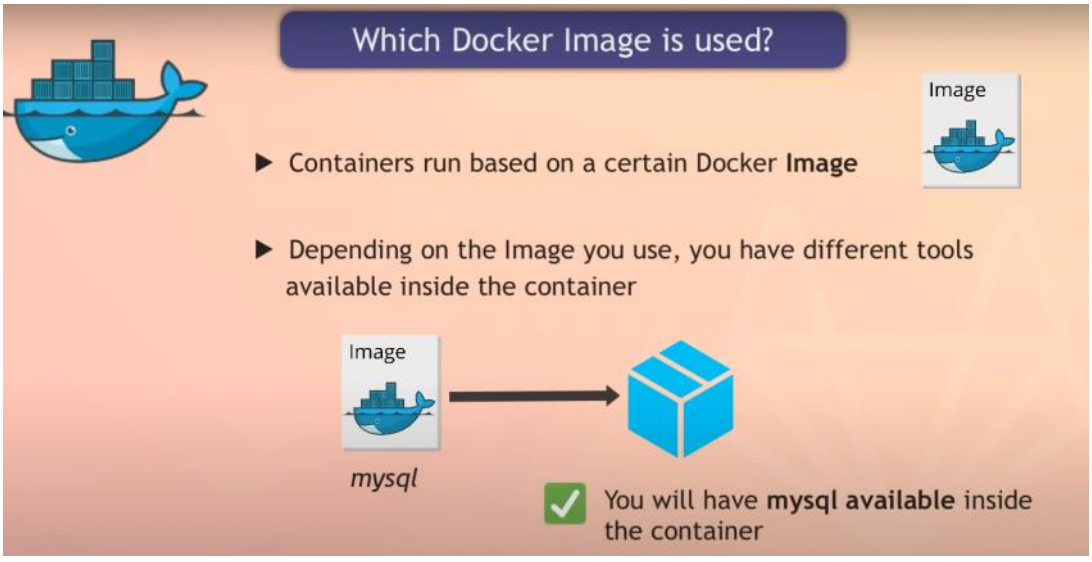


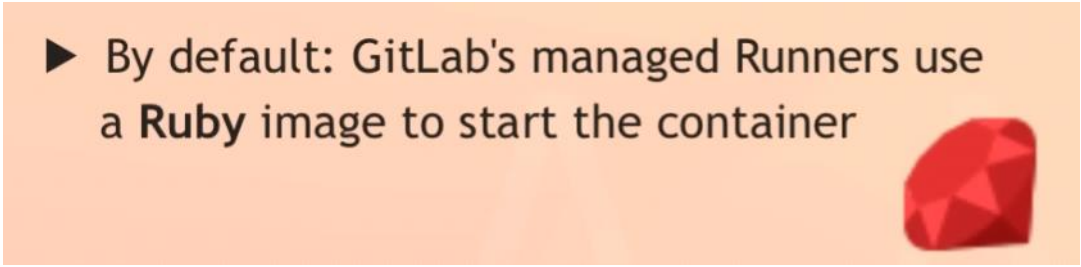
Runners can be executed on any environment:


GitLab CI CD Tutorial for Beginners [Crash Course]
EXECUTOR
Different Types of Executors
Executor determines the environment each job runs in
RUNNER
Shell Executor
Shell is the simplest executor
Commands executed on operating system
On the shell of the server, where GitLab Runner is installed
16:56 / 1:08:59 • Run Tests

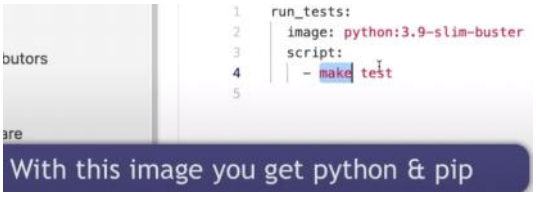

EXECUTOR
Different Types of Executors
Executor determines the environment each job runs in
RUNNER
Commands are executed inside a container
Only Docker itself needs to be installed
Each job runs in a separate & isolated container


RUNNER
GitLab
GitLab's managed Runners use Docker containers
So, our jobs are executed in Docker containers






```
run_tests:
  image: python:3.9-slim-buster
  script:
    - make test
```



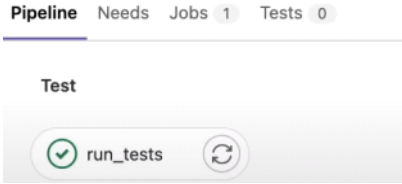
before_script

- Commands that should run before *script* command

Always update package information (apt-get update), before actually installing a tool. So you don't install outdated software.

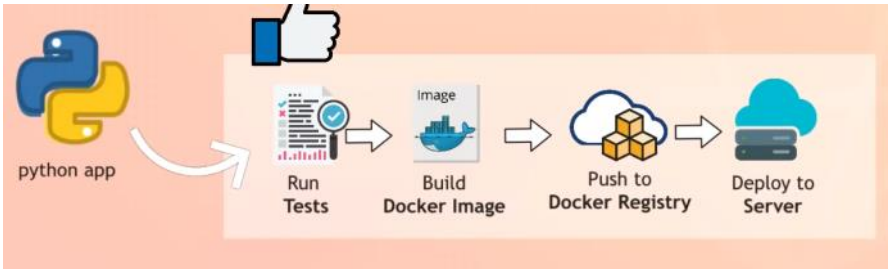
```
run_tests:
  image: python:3.9-slim-buster
  before_script:
    - apt-get update && apt-get install make
  script:
    - make test
```

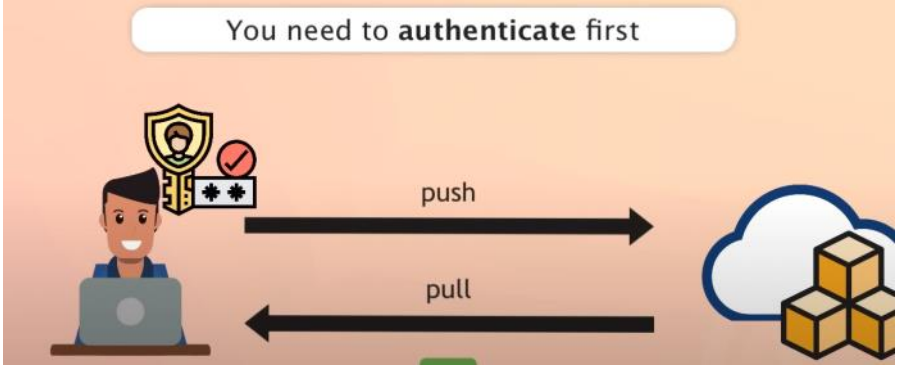
To Check Pipeline status:

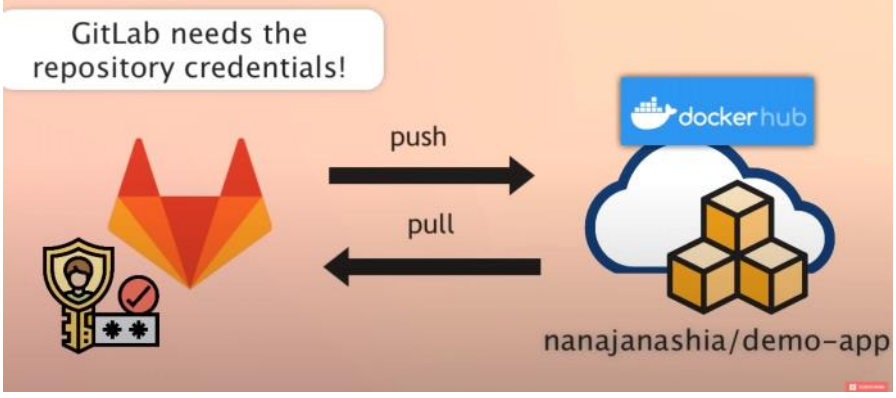


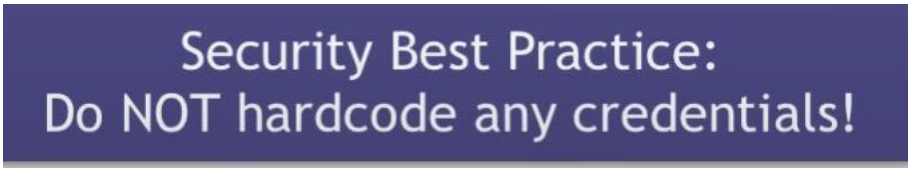
All jobs in that pipeline:

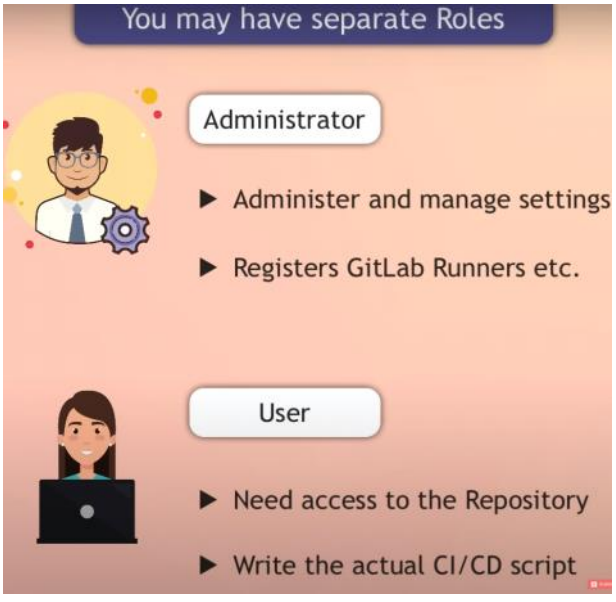
We have done first step of our pipeline(to run test):


python app
Run Tests
Image
Build Docker Image
Push to Docker Registry
Deploy to Server

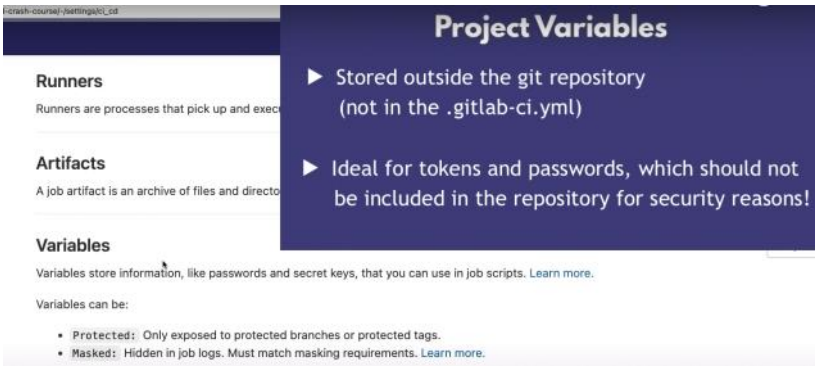

You need to **authenticate** first
push
pull

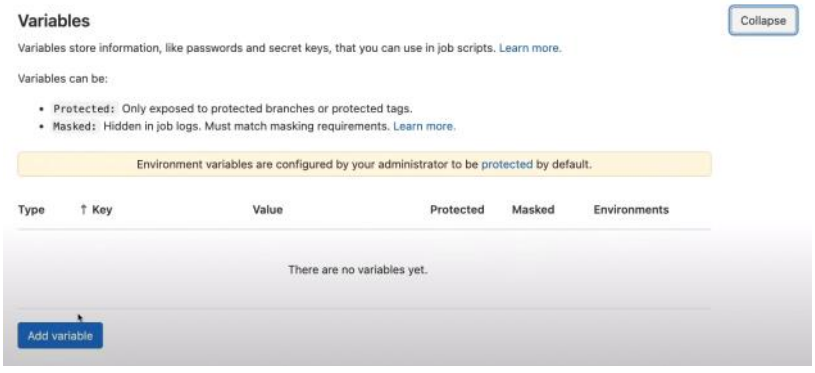

GitLab needs the repository credentials!
push
pull
docker hub
nanajanashia/demo-app

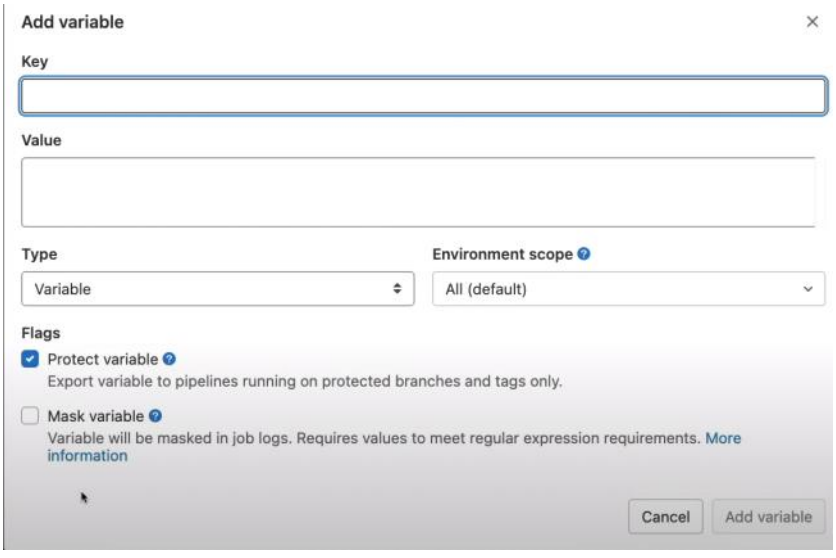

Security Best Practice:
Do NOT hardcode any credentials!

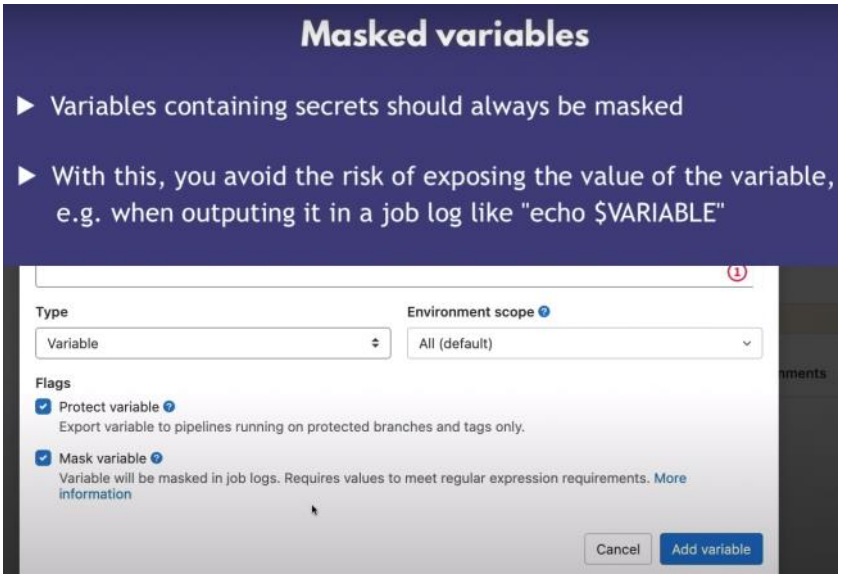

You may have separate Roles
Administrator
Administer and manage settings
Registers GitLab Runners etc.
User
Need access to the Repository
Write the actual CI/CD script

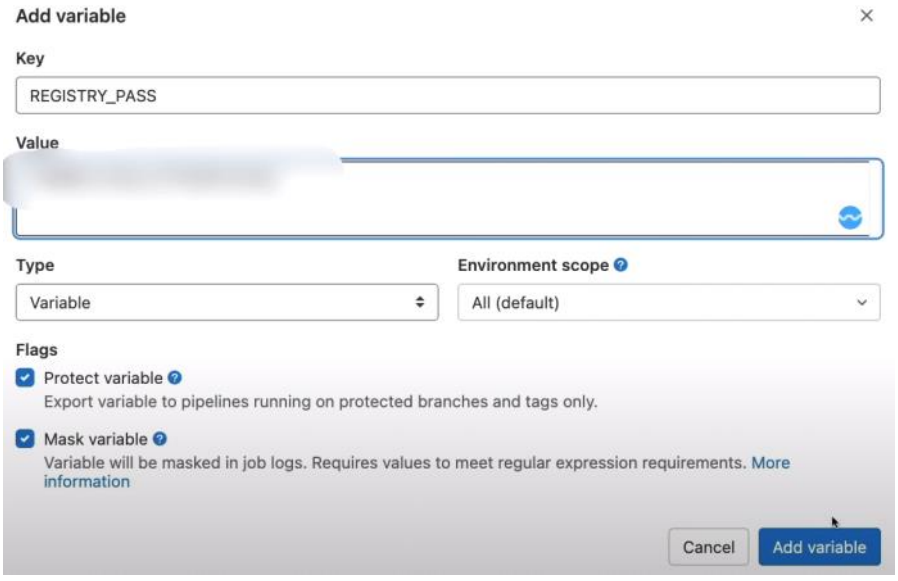
Adding user-credentials for docker in setting/CI-CD to fetch docker private repository

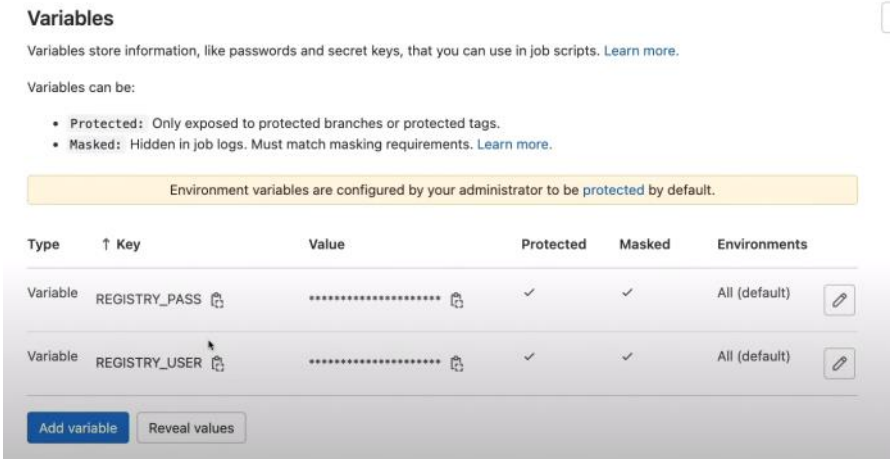


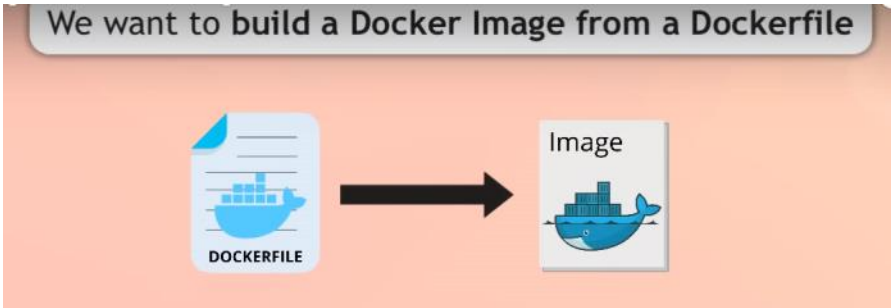




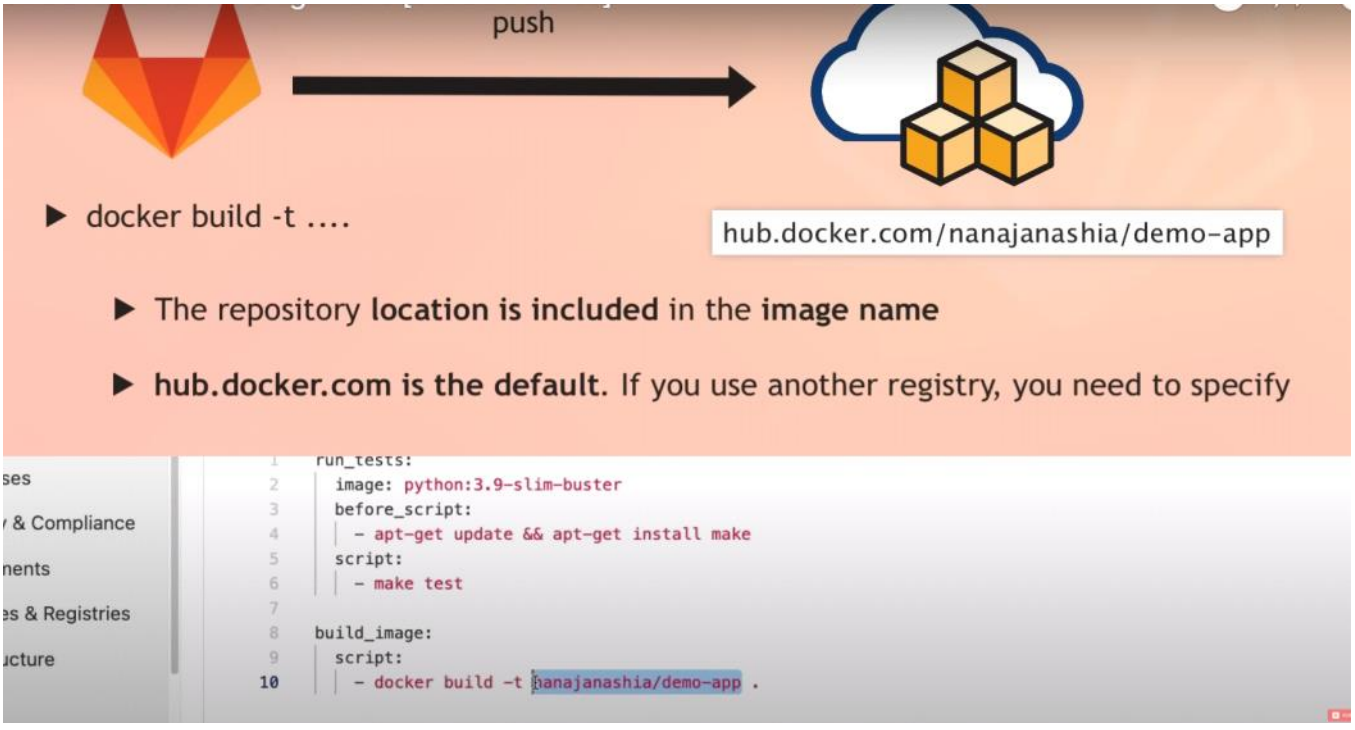

Masked variables
Variables containing secrets should always be masked
With this, you avoid the risk of exposing the value of the variable, e.g. when outputing it in a job log like "echo $VARIABLE"
Type
Variable
Environment scope
All (default)
Flags
Protect variable
Export variable to pipelines running on protected branches and tags only.
Mask variable
Variable will be masked in job logs. Requires values to meet regular expression requirements. More information
Cancel
Add variable

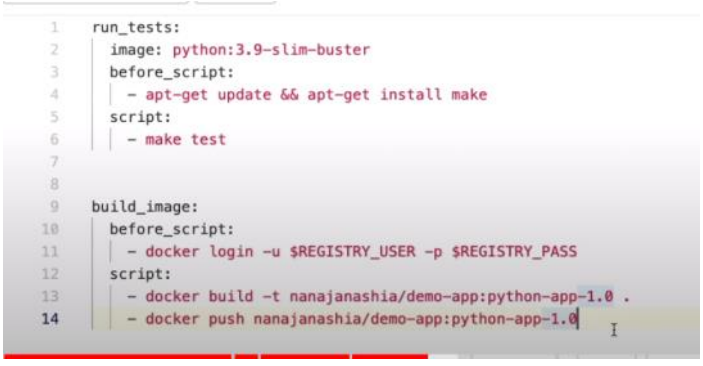

Add variable
Key
REGISTRY_PASS
Value
Type
Variable
Environment scope
All (default)
Flags
Protect variable
Export variable to pipelines running on protected branches and tags only.
Mask variable
Variable will be masked in job logs. Requires values to meet regular expression requirements. More information
Cancel
Add variable


Variables
Variables store information, like passwords and secret keys, that you can use in job scripts. Learn more.
Variables can be:
Protected: Only exposed to protected branches or protected tags.
Masked: Hidden in job logs. Must match masking requirements. Learn more.
Environment variables are configured by your administrator to be protected by default.
Type
Key
Value
Protected
Masked
Environments
Variable
REGISTRY_PASS
All (default)
Variable
REGISTRY_USER
All (default)
Add variable
Reveal values

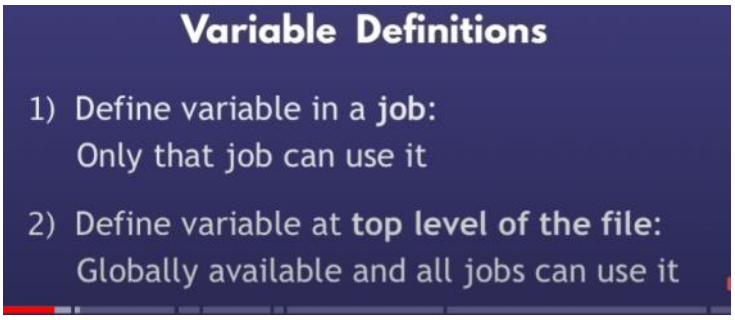


Docker file is already present in our python project





Another way using variables :



Defining variables on Global level/pipeline level:

```
variables:
  IMAGE_NAME: nanajanashia/demo-app
  IMAGE_TAG: python-app-1.0


run_tests:
  image: python:3.9-slim-buster
  before_script:
    - apt-get update && apt-get install make
  script:
    - make test


build_image:
  before_script:
    - docker login -u $REGISTRY_US
  script:
    - docker build -t $IMAGE_NAME:
    - docker push $IMAGE_NAME:$IMA
```

1) Define
Only t

Adding docker image so that docker client functionality could be used:

```
build_image:
  image: docker:20.10.16
  before_script:
    - docker login -u $REGISTRY_USER -p $REGISTRY_PASS
  script:
    - docker build -t $IMAGE_NAME:$IMAGE_TAG .
    - docker push $IMAGE_NAME:$IMAGE_TAG
```

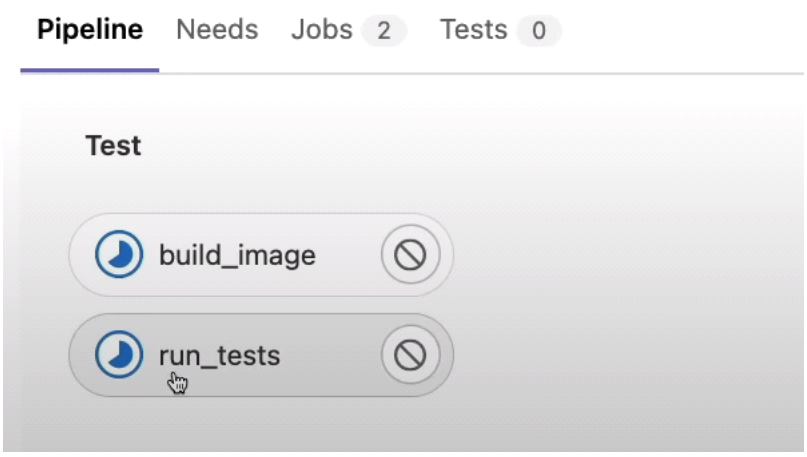
Adding docker demon service:

```
build_image:
  image: docker:20.10.16
  services:
    - docker:20.10.16-dind
  before_script:
    - docker login -u $REGISTRY_USER -p $REGISTRY_PASS
  script:
    - docker build -t $IMAGE_NAME:$IMAGE_TAG .
    - docker push $IMAGE_NAME:$IMAGE_TAG
```

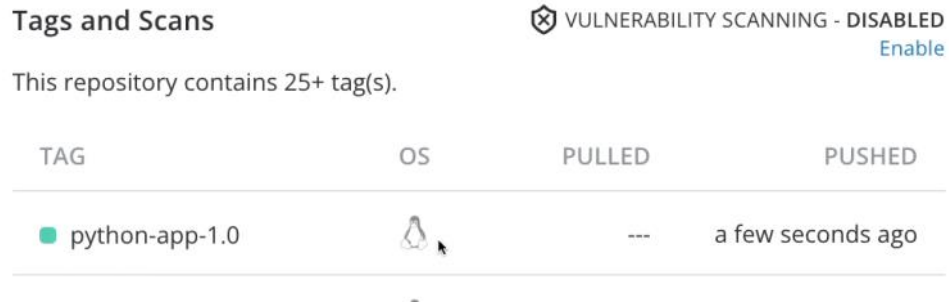
Adding docker certificates:

```yaml
build_image:
  image: docker:20.10.16
  services:
    - docker:20.10.16-dind
  variables:
    DOCKER_TLS_CERTDIR: "/certs"
  before_script:
    - docker login -u $REGISTRY_USER -p $REGISTRY_PASS
  script:
    - docker build -t $IMAGE_NAME:$IMAGE_TAG .
    - docker push $IMAGE_NAME:$IMAGE_TAG
```

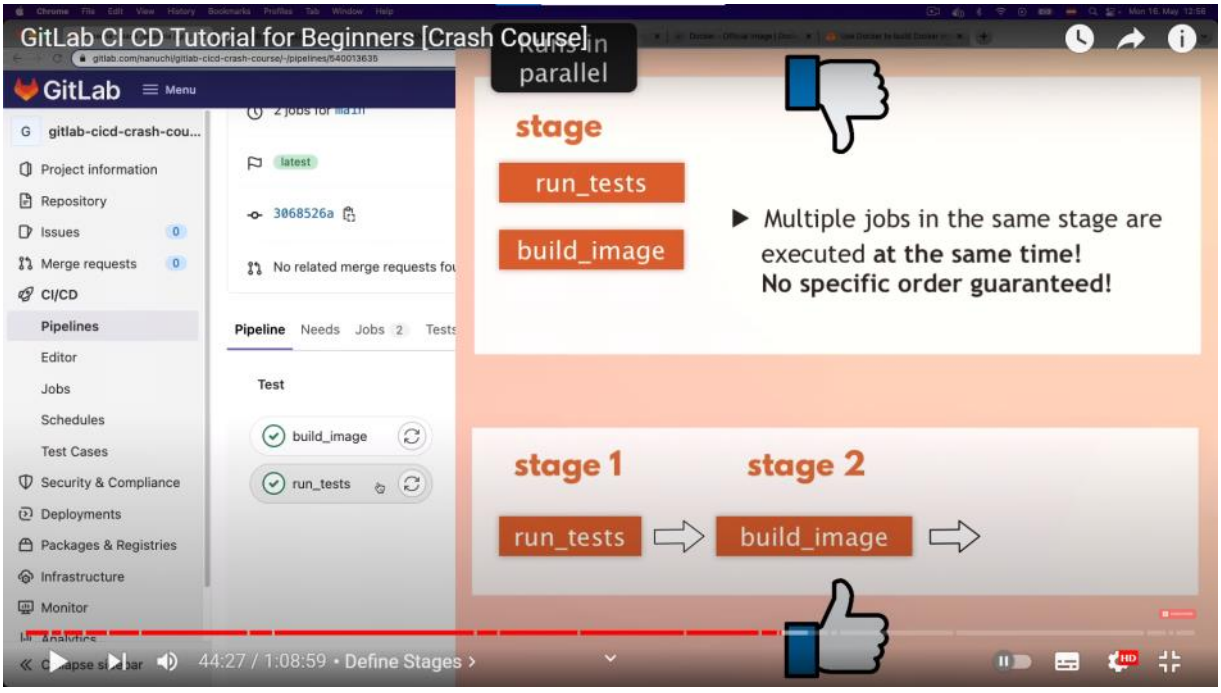
Commit changes and check jobs in pipeline:

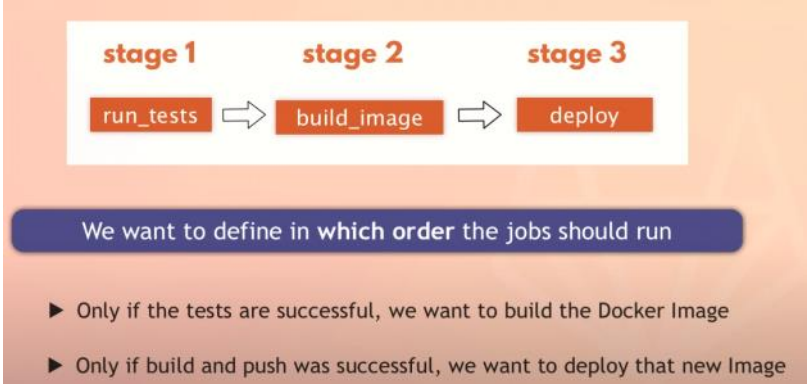


Checking on DockerHub if image is successfully pushed on our private repository:

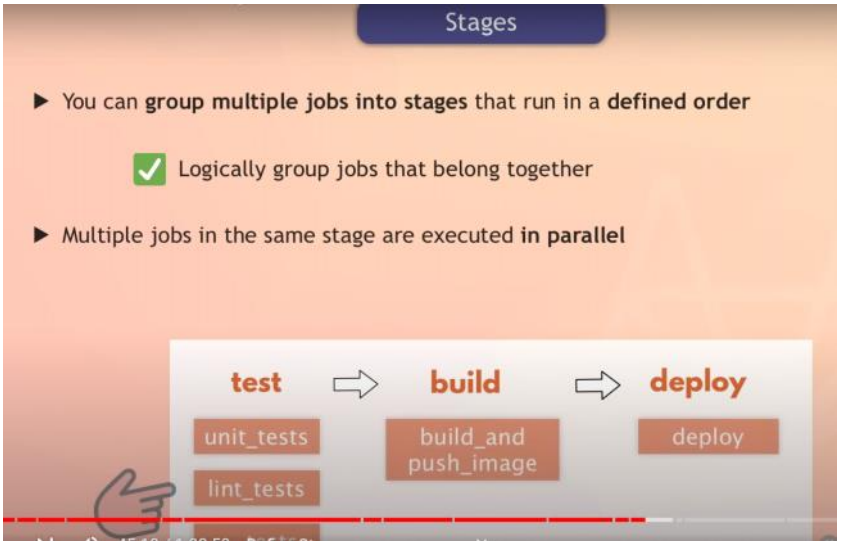


How to run these job in sequent not all at once:



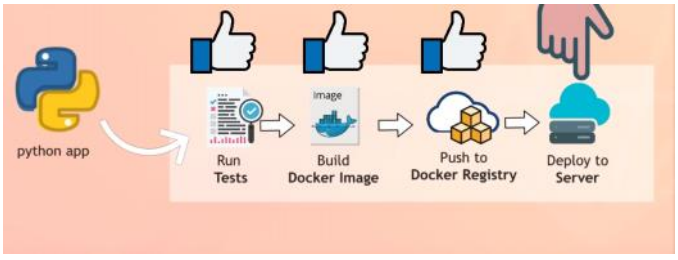


Using stages to accomplish it:

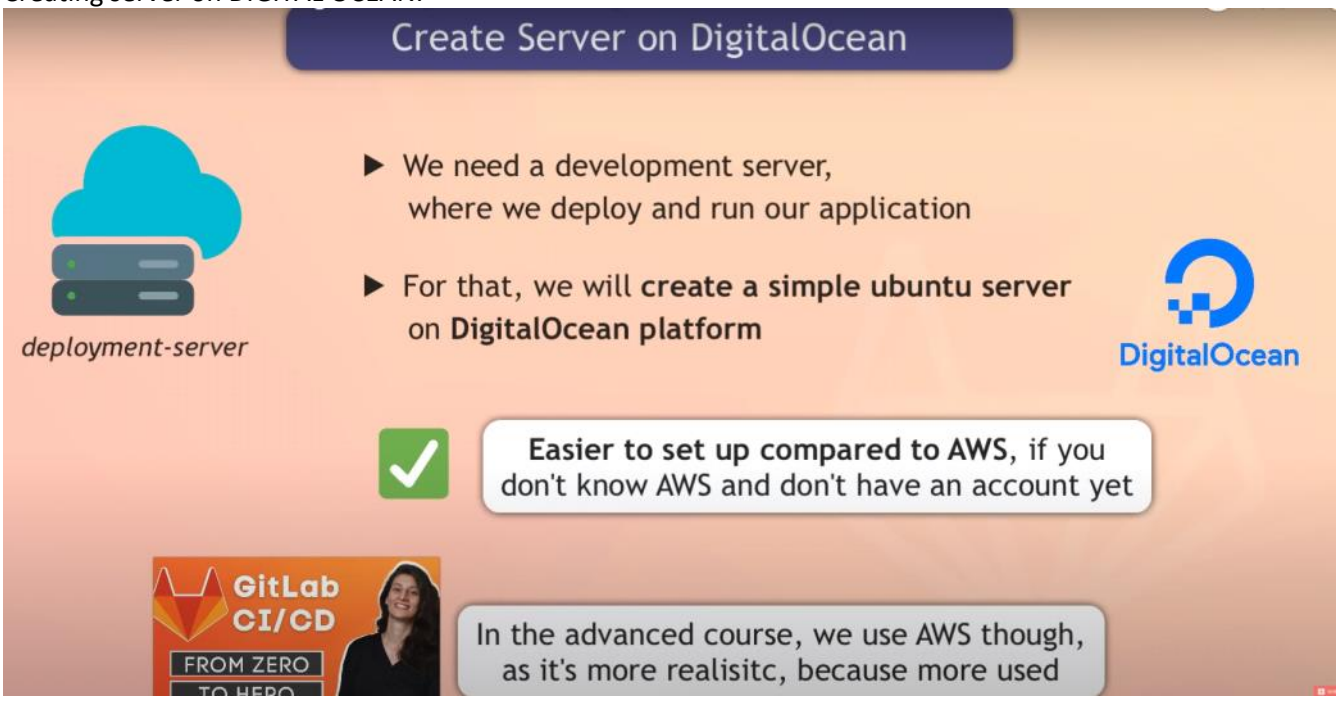


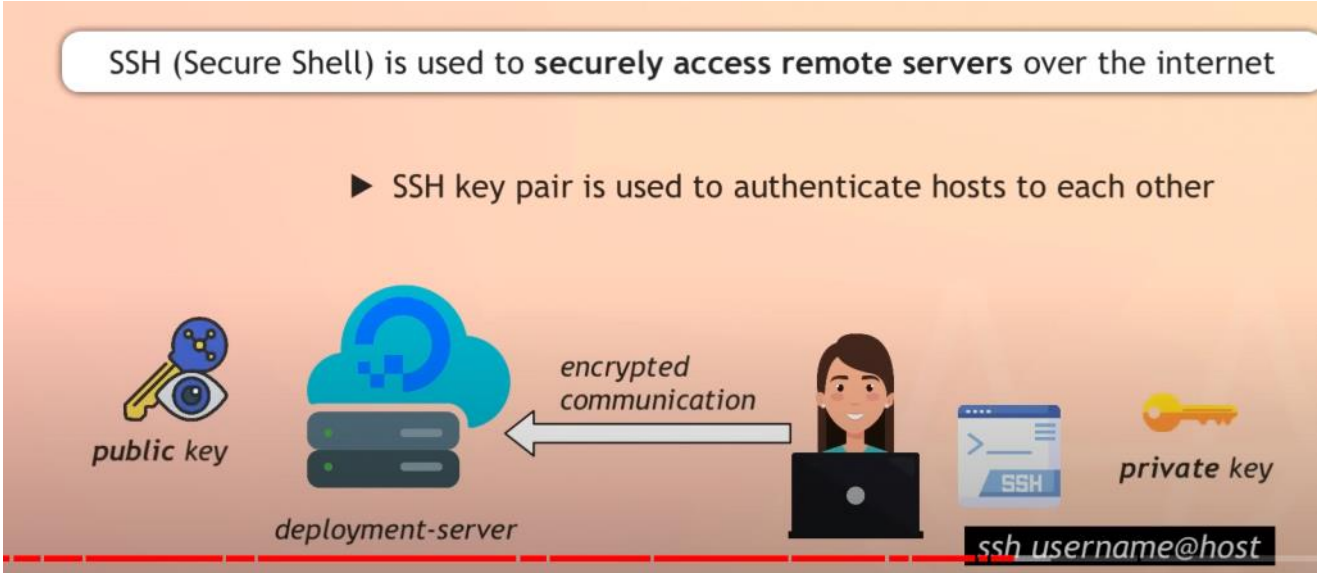
```
stages:
  - test
  - build

run_tests:
  stage: test
  image: python:3.9-slim-buster
  before_script:
    - apt-get update && apt-get install make
  script:
    - make test

build_image:
  stage: build
  image: docker:20.10.16
```

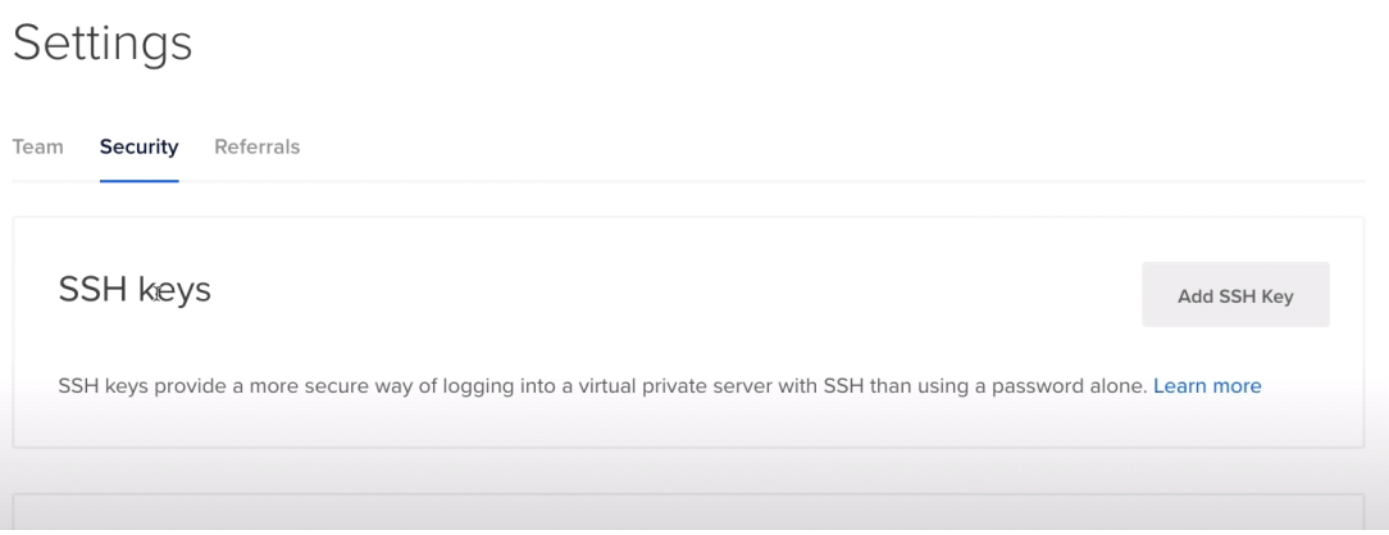
Last step remaining to deploy to servers:

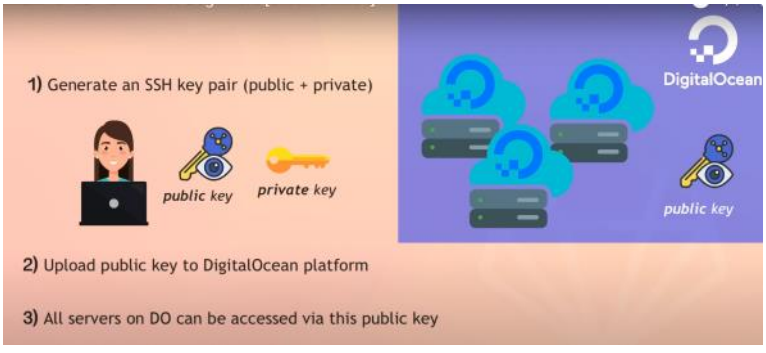


Creating server on DIGITAL OCEAN:









Generating ssh key pair on local :

```
[\W]$ ssh-keygen
```

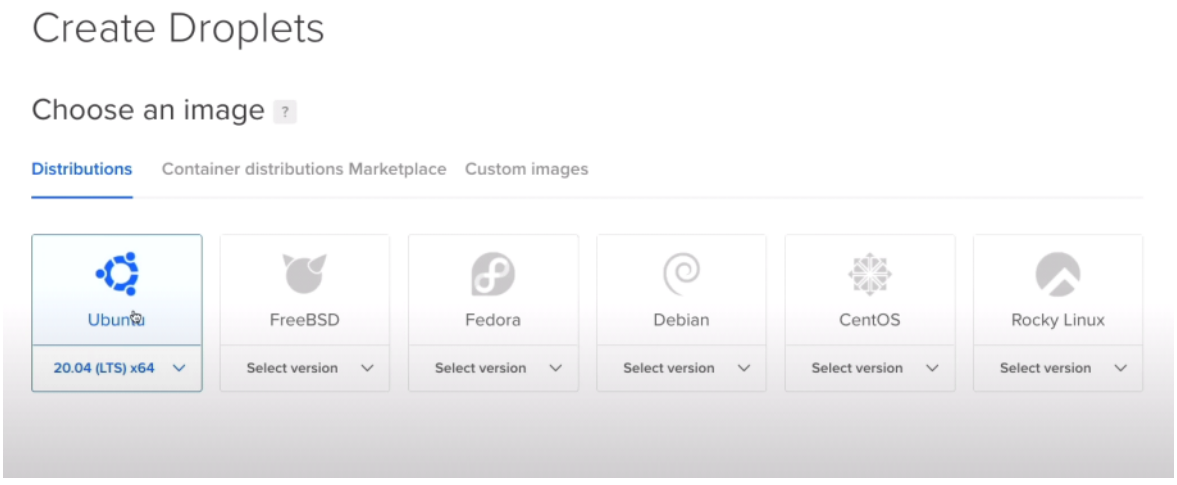
```
[\W]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/Users/nanajanashia/.ssh/id_rsa): /Users/nanajanashia/.ss
h/digital_ocean_key
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /Users/nanajanashia/.ssh/digital_ocean_key
Your public key has been saved in /Users/nanajanashia/.ssh/digital_ocean_key.pub
The key fingerprint is:
SHA256:hJX0e4shDKxIkyS+TMbvvkcXjvfnYvM/rmipZPsXZjM nanajanashia@Nanas-MacBook-Pro.local
The key's randomart image is:
+---[RSA 3072]----+
|. .    .o.       |
|oo . . o..       |
| =+   + . .      |
|+.oo . =   .     |
| o... o S o .    |
|  .  o + . E .   |
|   .. oo. = =    |
|  .  .o .B.o .   |
|   oo  o=oB++o.  |
+----[SHA256]-----+
```

```
[\W]$ ls /Users/nanajanashia/.ssh | grep digital
digital_ocean_key
digital_ocean_key.pub
[\W]$
```

Getting public key for it:

```
[\W]$ cat /Users/nanajanashia/.ssh/digital_ocean_key.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCcMaimAMP5xMjsbiRmjkxR9nKiAuomiSCI6000D7b/HMDo36Yj2l56yH
z+8/lat+FJycVFZ58Dgns3FqGIaiE776FJ2cLziCL67w3YMvPvbHT6w7Xc5eML2d+AP/AsrVIBMSqvGrqGb6jV5bfuTO7c
zNX1sGa39UbDXpVD6FRv/eAqh5pQP8C/QhTk8Y2dIT+WrbBGPW7LCG5gEA73DTCfPZCNovhv0nqUIFCJecwReLmhN8PotL
dxFi8aBEFKod6eOgnMQuv8myM2SrHtOjRuWZJzP7mjNPVFo9uW6jl6c6Cul6fXj8oelRptFQkwnqcA9jE6YQSWiudjAo38
aRwGGst6b5dCXlX/IVGsyM+bgrxDC+shr6i21bCtOFfK0cSGyOiZp1gG3HtZFW6+rxj2q0pwPsvmfz+Hzu5DYFdQIkHwo3
3HSUzy9G7LL/XID42UQjq/AUey+4B5uF9L3iBQC8fHobFmFo4BCI1+GA7T7egVuo+xmXibP5wZ3CrzwtM= nanajanashi
a@Nanas-MacBook-Pro.local
[\W]$
```

Creating deployment server via droplets option in digital ocean:



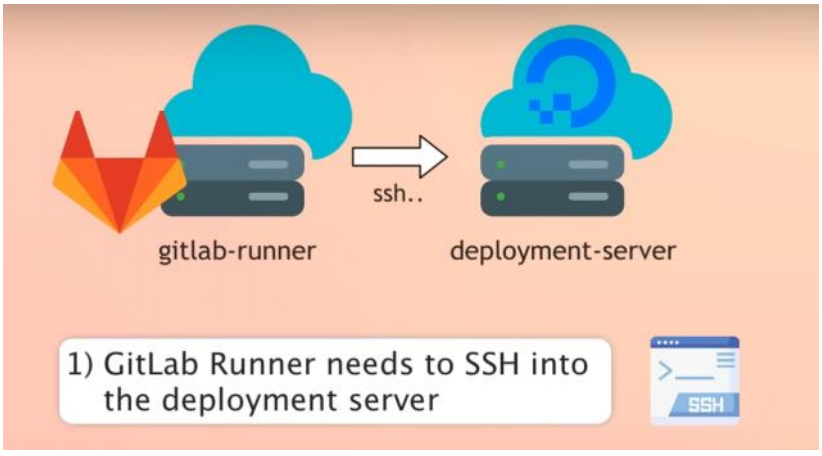
Connecting to our created server on digital ocean from local machine:

```
[\W]$ ssh -i ~/.ssh/digital_ocean_key root@161.35.223.117
```

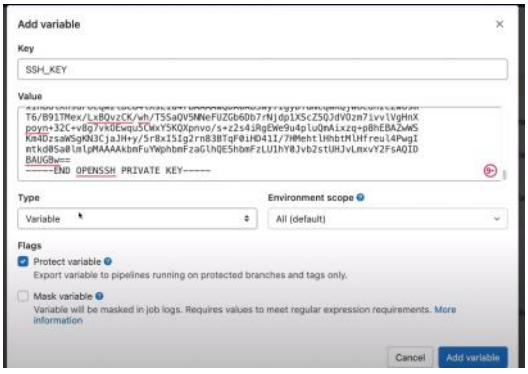
That’s public ip address at end,of our created server on digital ocean.

Command to install docker on our server:
->>apt update
->>apt install docker.io

Connnecting Gitlab to our digital ocean server:



Adding ssh key credentials of digital ocean server in gitlab secret variables in settings:




```
deploy:
  stage: deploy
  script:
    - ssh -o StrictHostKeyChecking=no -i $SSH_KEY root@161.35.223.117
```

Authenticating docker repository to pull image & running the container

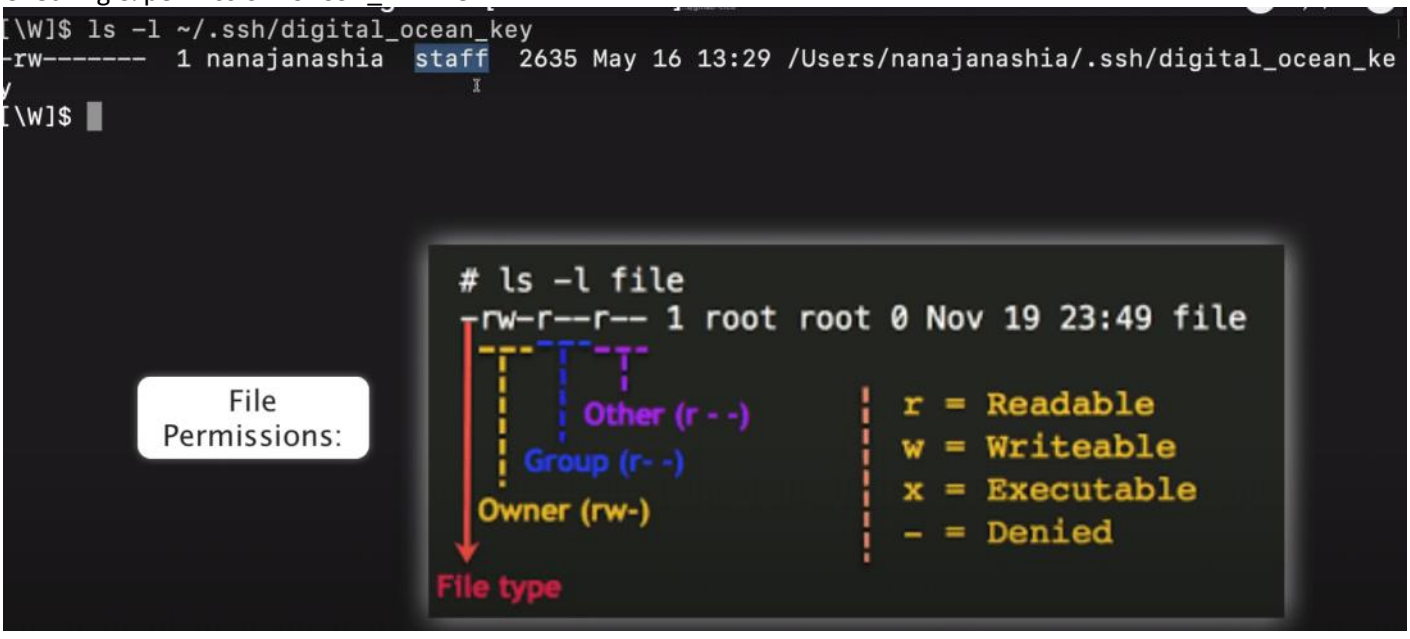
```
deploy:
  stage: deploy
  script:
    - ssh -o StrictHostKeyChecking=no -i $SSH_KEY root@161.35.223.117 "
        docker login -u $REGISTRY_USER -p $REGISTRY_PASS &&
        docker run -p 5000:5000 $IMAGE_NAME:$IMAGE_TAG"
```

Stopping all containers before running our container , so that they may not be 2 container running on same port:

```
docker ps -aq | xargs docker stop | xargs docker rm
```

```
deploy:
  stage: deploy
  script:
    - ssh -o StrictHostKeyChecking=no -i $SSH_KEY root@161.35.223.117 "
        docker login -u $REGISTRY_USER -p $REGISTRY_PASS &&
        docker ps -aq | xargs docker stop | xargs docker rm &&
        docker run -p 5000:5000 $IMAGE_NAME:$IMAGE_TAG"
```

Checking & permission for SSH_KEY file:

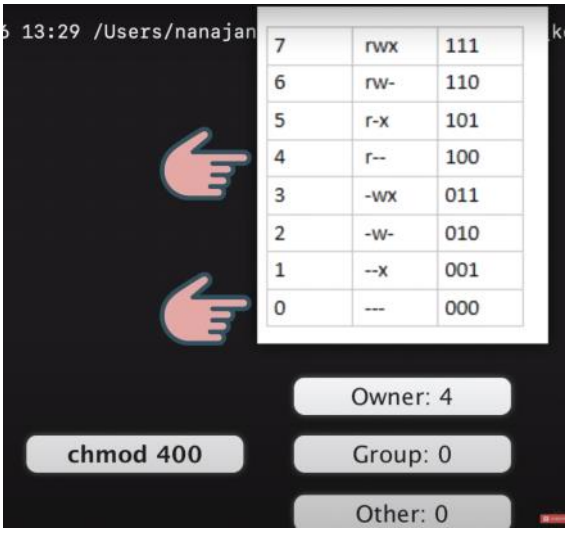


Changing permission for SSH_KEY file on Gitlab:

```
before_script:
  - chmod 400 $SSH_KEY
```

```
deploy:
  stage: deploy
  before_script:
    - chmod 400 $SSH_KEY
  script:
    - ssh -o StrictHostKeyChecking=no -i $SSH_KEY root@161.35.223.117 "
        docker login -u $REGISTRY_USER -p $REGISTRY_PASS &&
        docker ps -aq | xargs docker stop | xargs docker rm &&
        docker run -p 5000:5000 $IMAGE_NAME:$IMAGE_TAG"
```

Permission Code Chart:

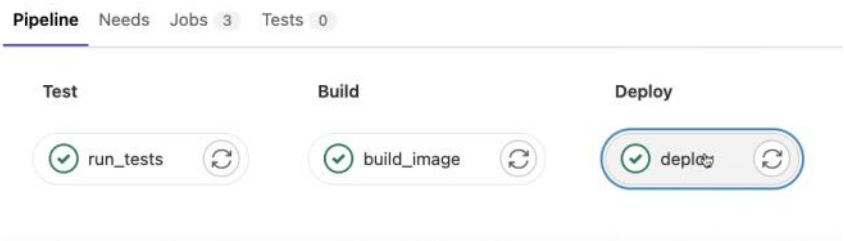


Changing docker to run in detached mode:

```
docker run -d -p 5000:5000 $IMAGE_NAME:$IMAGE_TAG"
```

```
deploy:
  stage: deploy
  before_script:
    - chmod 400 $SSH_KEY
  script:
    - ssh -o StrictHostKeyChecking=no -i $SSH_KEY root@161.35.223.117 "
        docker login -u $REGISTRY_USER -p $REGISTRY_PASS &&
        docker ps -aq | xargs docker stop | xargs docker rm &&
        docker run -d -p 5000:5000 $IMAGE_NAME:$IMAGE_TAG"
```

Checking Pipeline:


Pipeline
Needs
Jobs 3
Tests 0
Test
Build
Deploy
run_tests
build_image
deploy