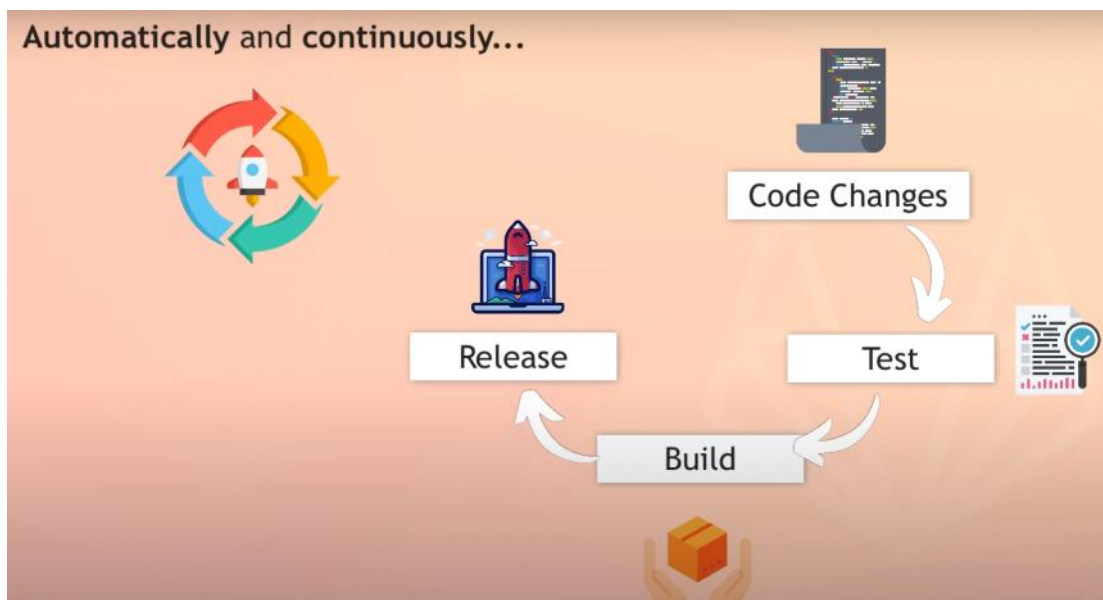
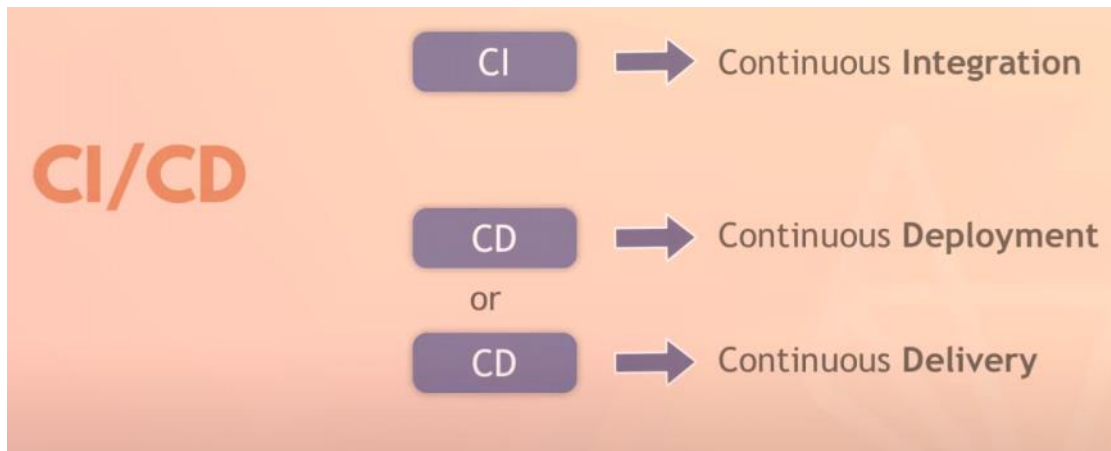
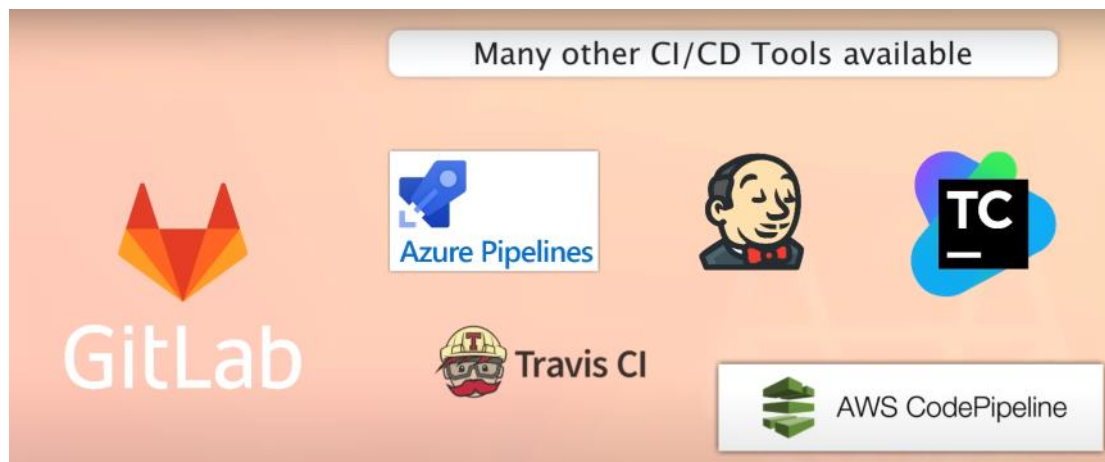
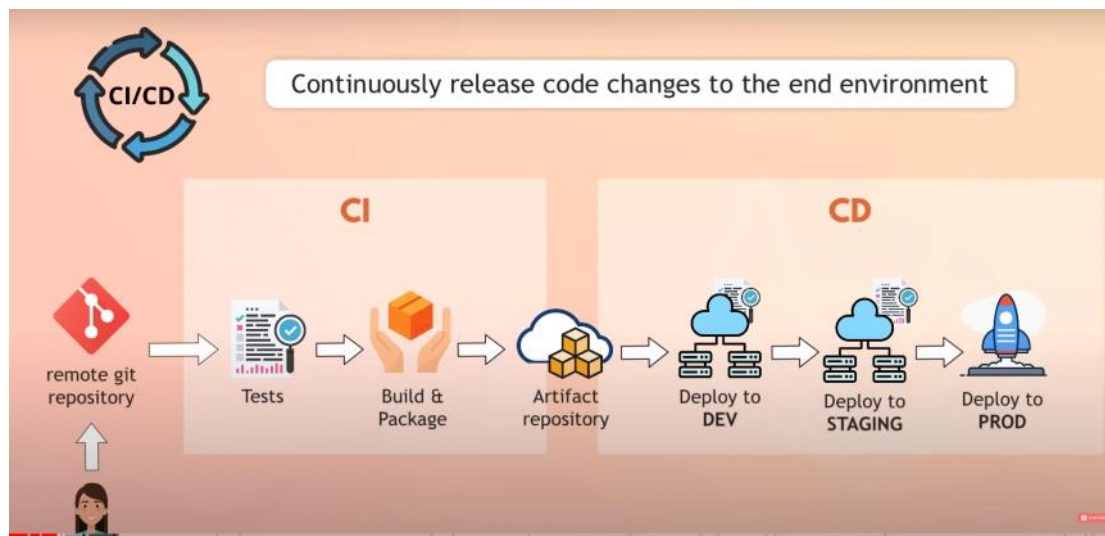
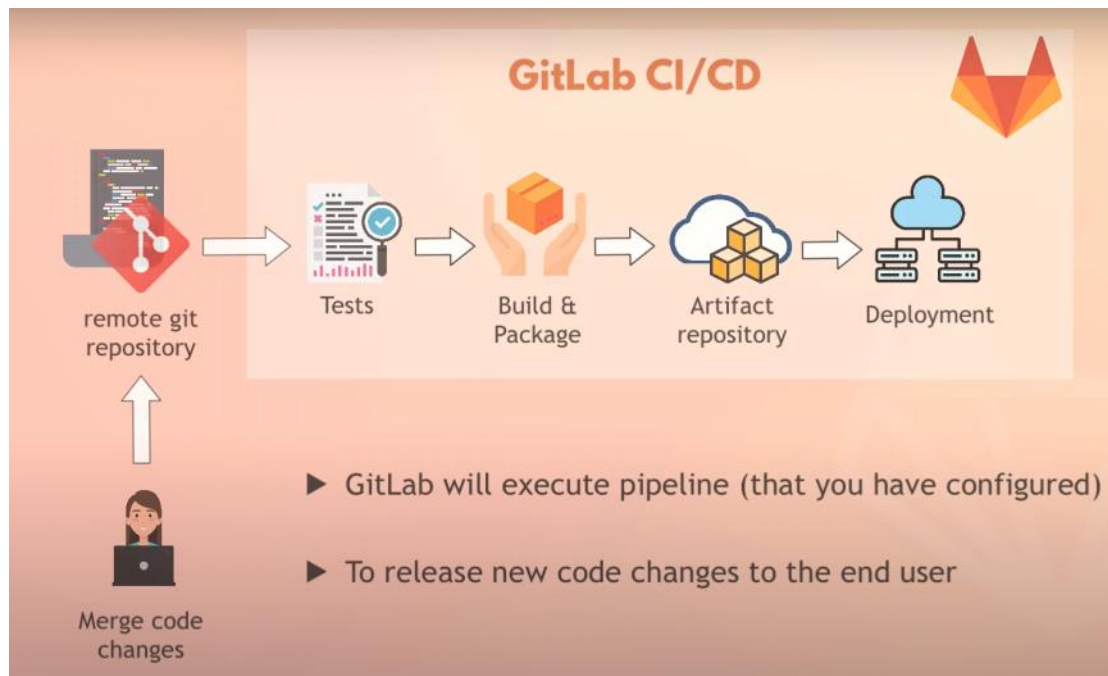


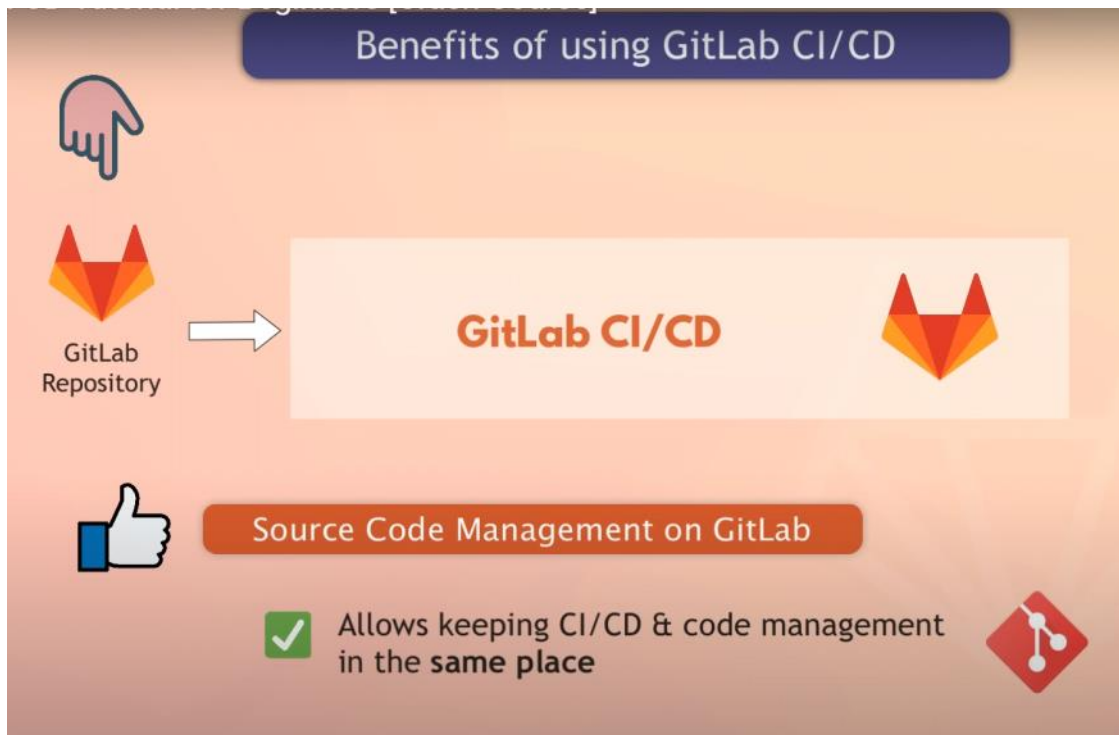
CI/CD GITLAB

Friday, October 13, 2023

10:12 AM







GitLab CI CD Tutorial for Beginners [Crash Course]


Benefits of using GitLab CI/CD

This section compares the benefits and drawbacks of using GitLab CI/CD. On the left, four green checkmarks highlight the benefits: seamless integration into the code repository, no overhead of setting up CI/CD, pipeline configuration as part of the application code, and the option to be self-hosted or SaaS (managed). On the right, two drawbacks are listed: it is only a CI/CD tool (marked with a yellow minus sign) and self-hosting is the only option (marked with a red X). The bottom of the slide features a video player interface with a progress bar at 4:23 / 1:08:59 and a title 'GitLab in comparison to other CI/CD platforms >'. The video player also includes standard controls like play, pause, and volume.

- ✓ 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 >

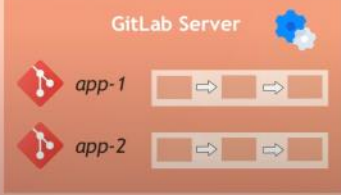
GitLab CI CD Tutorial for Beginners [Crash Course]
GitLab Architecture



GitLab Instance
or
GitLab Server


- ▶ Hosts your application code and pipeline configuration
- ▶ GitLab configurations etc.
- ▶ Manages the pipeline execution

GitLab Server

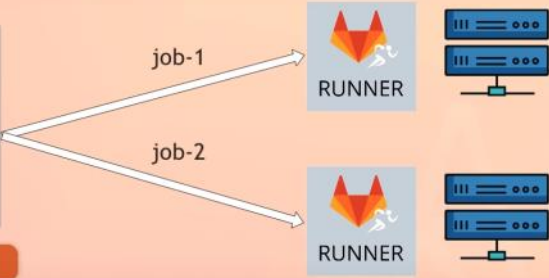


4:52 / 1:08:59 • GitLab Architecture - How GitLab works >

GitLab CI CD Tutorial for Beginners [Crash Course]
GitLab Architecture



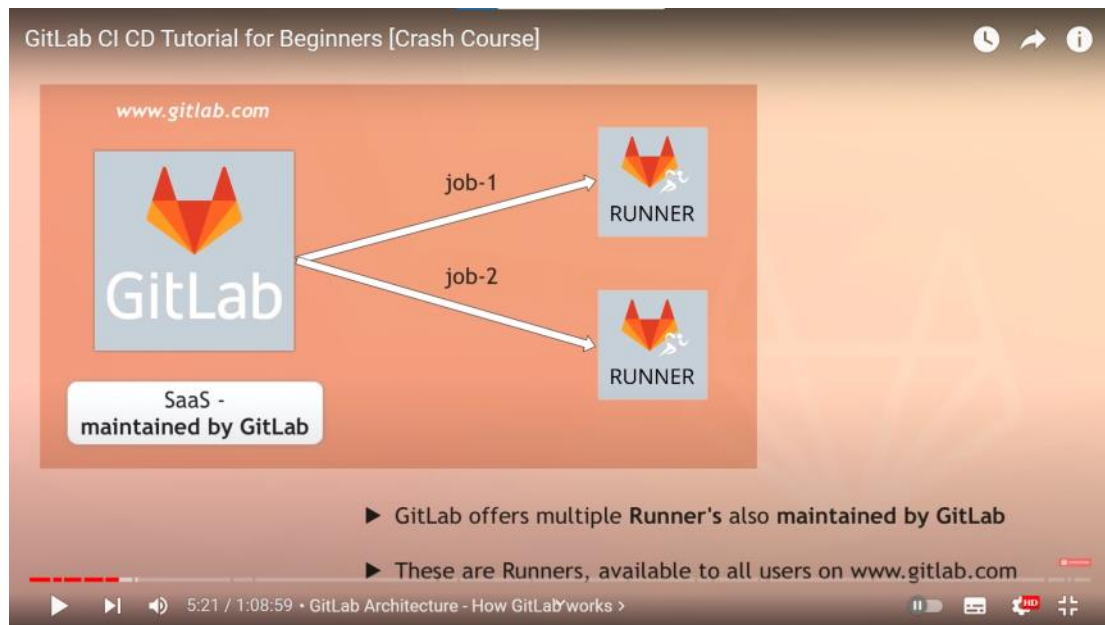
GitLab Instance
or
GitLab Server



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

Build Docker Image

Push to Docker Registry



Deploy to Server

Running Processes (11)

Performance Monitor

CPU 35%

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
```


Pipeline is scripted


Pipeline




- ▶ Pipeline is written in code
- ▶ Hosted inside application's git repository

 Whole CI/CD configuration is written in **YAML format**





.gitlab-ci.yml

CI/CD Pipeline




Jobs

- ▶ Jobs are the most fundamental building block of a `.gitlab-ci.yml` file


.gitlab-ci.yml

CI/CD Pipeline



- ▶ Jobs define **what** to do

New file

main / .gitlab-ci.yml

1 run_tests:

Jobs

► You can define arbitrary names for your jobs

Job name limitations

You can't use these keywords as job names:

- image
- services
- stages
- types
- before_script
- after_script
- variables
- cache
- include
- true
- false
- nil

Jobs

► You can define arbitrary names for your jobs

► Must contain at least the *script* clause

► *script* specify the commands to execute

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

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

Job Configuration to Run a Test:

main / .gitlab-ci.yml .gitlab-ci.yml Apply a template

1 run_tests:
2 script:
3 - make test

new file

main / .gitlab-ci.yml

```
1 run_tests:
2   script:
3     - make test
```

For the job to executed successful

- **make** command available
- **pip** available
- **python** available

These 3 programs need to be available on the machine, where the job is executed

GitLab Jobs are executed on Runners:

Where or on which environment is the job executed?



Runners can be executed on any environment:

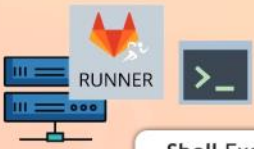


GitLab CI CD Tutorial for Beginners [Crash Course]

EXECUTOR

Different Types of Executors

- ▶ Executor determines the environment each job runs in



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



- ▶ Commands are executed inside a container
- ✓ Only Docker itself needs to be installed
- ✓ Each job runs in a separate & isolated container




- ▶ GitLab's managed Runners use **Docker containers**
- ▶ So, our jobs are executed in Docker containers




Which Docker Image is used?


- ▶ Containers run based on a certain Docker **Image**
- ▶ Depending on the Image you use, you have different tools available inside the container



mysql

→



 You will have **mysql** available inside the container

- ▶ By default: GitLab's managed Runners use a **Ruby** image to start the container



For our job to execute Python tests, we need Python not Ruby



run python tests



python

```

1  run_tests:
2    image: python:3.9-slim-buster
3    script:
4      - make test
5

```

```

1  run_tests:
2    image: python:3.9-slim-buster
3    script:
4      - make test
5

```

With this image you get python & pip

before_script

- ▶ Commands that should run before *script* command

after_script

- Define commands that run after each job, including failed jobs

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

```
1 run_tests:
2   image: python:3.9-slim-buster
3   before_script:
4     - apt-get update && apt-get install make
5   script:
6     - make test
7
```

Commit message

Target Branch

To Check Pipeline status:

CI/CD

- Pipelines
- Editor
- Jobs
- Schedules
- Test Cases

Status	Pipeline	Triggerer	Stages
passed 00:00:45 19 seconds ago	Add .gitlab-ci.yml #539951229 main -> f608a8e9 latest		

Pipeline Needs Jobs 1 Tests 0

Test

☒ run_tests

All jobs in that pipeline:

Nana Janashia > gitlab-ci-cd-crash-course > Jobs

All 1

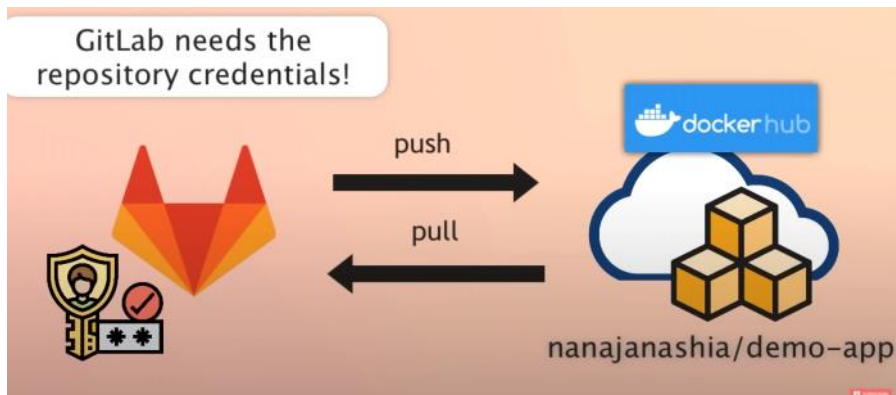
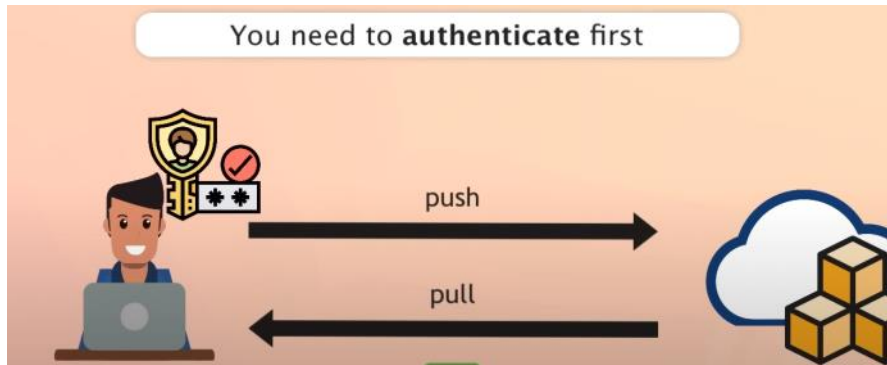
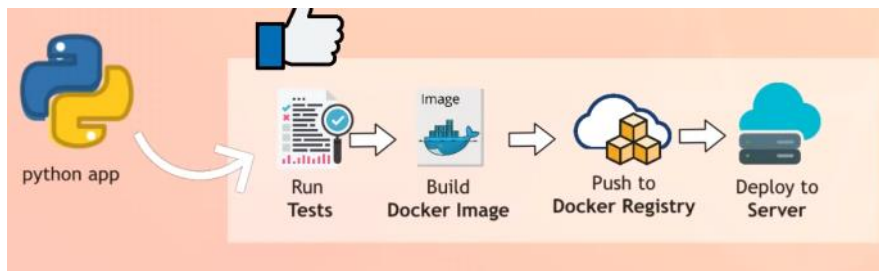
Pending 0

Running 0

Finished 1

Status	Name	Job	Pipeline	Stage	Duration	Coverage
passed	run_tests	#2460889217 main f608a8e9	#539951229 by	test	00:00:45 just now	

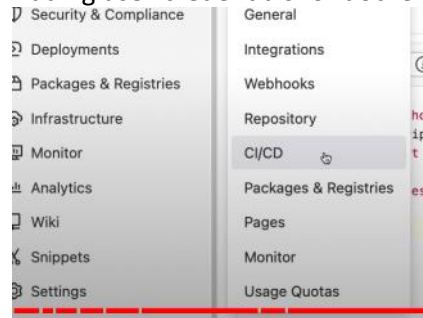
We have done first step of our pipeline(to run test):



Security Best Practice:
Do NOT hardcode any credentials!



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



Runners

Runners are processes that pick up and execute jobs.

Artifacts

A job artifact is an archive of files and directories produced by a job.

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.](#)

Project Variables

- ▶ Stored outside the git repository (not in the .gitlab-ci.yml)
- ▶ Ideal for tokens and passwords, which should not be included in the repository for security reasons!

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
There are no variables yet.					

[Add variable](#)

Add variable

Key

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](#)

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 outputting 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](#)

Buttons: Cancel, Add variable

Add variable

Key: REGISTRY_PASS

Value: [Masked]

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](#)

Buttons: Cancel, Add variable

Variables

Variables store information, like passwords and secret keys, that you can use in job scripts. [Learn more.](#)

Variables can be:

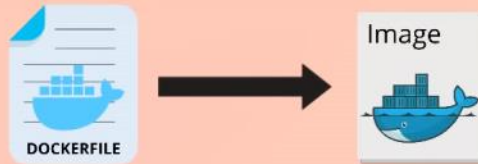
- Protected: Only exposed to protected branches or protected tags.
- Masked: Hidden in job logs. Must match masking requirements. [Learn more.](#)

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)

Buttons: Add variable, Reveal values

We want to build a Docker Image from a Dockerfile



Docker file is already present in our python project

A diagram showing a Docker image being pushed to a registry. On the left is the Docker logo (a stylized orange and yellow 'D'). A thick black arrow points from the logo to a cloud icon on the right. Inside the cloud are three yellow cubes, representing a Docker registry. Above the arrow is the word 'push'.

- ▶ `docker build -t`

`hub.docker.com/nanajanashia/demo-app`

- ▶ The repository location is included in the image name
- ▶ `hub.docker.com` is the default. If you use another registry, you need to specify

```
1 run_tests:
2   image: python:3.9-slim-buster
3   before_script:
4     - apt-get update && apt-get install make
5   script:
6     - make test
7
8 build_image:
9   script:
10    - docker build -t nanajanashia/demo-app .
```

```
1 run_tests:
2   image: python:3.9-slim-buster
3   before_script:
4     - apt-get update && apt-get install make
5   script:
6     - make test
7
8
9 build_image:
10  before_script:
11    - docker login -u $REGISTRY_USER -p $REGISTRY_PASS
12  script:
13    - docker build -t nanajanashia/demo-app:python-app-1.0 .
14    - docker push nanajanashia/demo-app:python-app-1.0
```

Another way using variables :

```
9 build_image:
10  variables:
11    IMAGE_NAME: nanajanashia/demo-app
12    IMAGE_TAG: python-app-1.0
13  before_script:
14    - docker login -u $REGISTRY_USER -p $REGISTRY_PASS
15  script:
16    - docker build -t $IMAGE_NAME:$IMAGE_TAG .
17    - docker push $IMAGE_NAME:$IMAGE_TAG
```

Variable Definitions

- 1) Define variable in a job:
Only that job can use it
- 2) Define variable at top level of the file:
Globally available and all jobs can use it

Defining variables on Global level/pipeline level:

```
1 variables:
2   IMAGE_NAME: nanajanashia/demo-app
3   IMAGE_TAG: python-app-1.0
4
5
6 run_tests:
7   image: python:3.9-slim-buster
8   before_script:
9     - apt-get update && apt-get install make
10  script:
11    - make test
12
13
14 build_image:
15   before_script:
16     - docker login -u $REGISTRY_US
17   script:
18     - docker build -t $IMAGE_NAME:
19     - docker push $IMAGE_NAME:$IMA
```

1) Define
Only t

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

```
14 build_image:
15   image: docker:20.10.16
16   before_script:
17     - docker login -u $REGISTRY_USER -p $REGISTRY_PASS
18   script:
19     - docker build -t $IMAGE_NAME:$IMAGE_TAG .
20     - 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
```

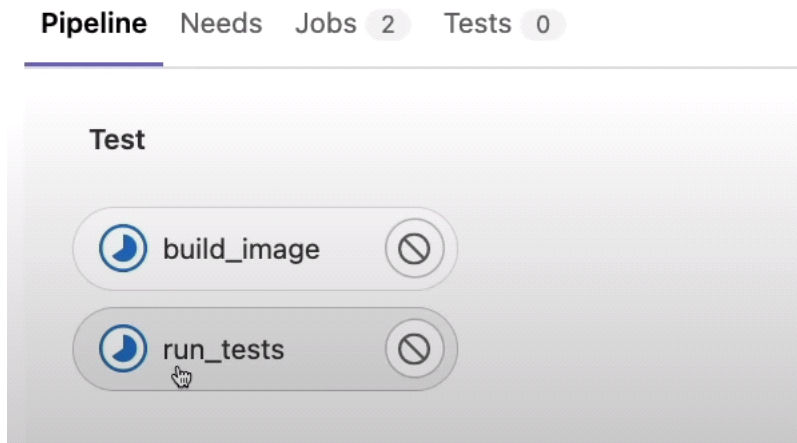
Adding docker certificates:

```

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:

Tags and Scans

 VULNERABILITY SCANNING - DISABLED
[Enable](#)

This repository contains 25+ tag(s).

TAG	OS	PULLED	PUSHED
 python-app-1.0		---	a few seconds ago

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

GitLab CI CD Tutorial for Beginners [Crash Course]

GitLab

Project information

Repository

Issues

Merge requests

CI/CD

Pipelines

Editor

Jobs

Schedules

Test Cases

Security & Compliance

Deployments

Packages & Registries

Infrastructure

Monitor

latest

3068526a

No related merge requests found

Pipeline

Needs

Jobs

Test

build_image

run_tests

stage

run_tests

build_image

Multiple jobs in the same stage are executed at the same time! No specific order guaranteed!

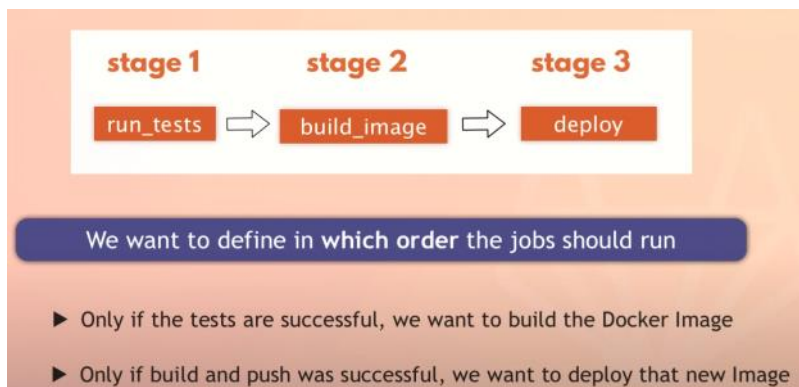
stage 1

run_tests

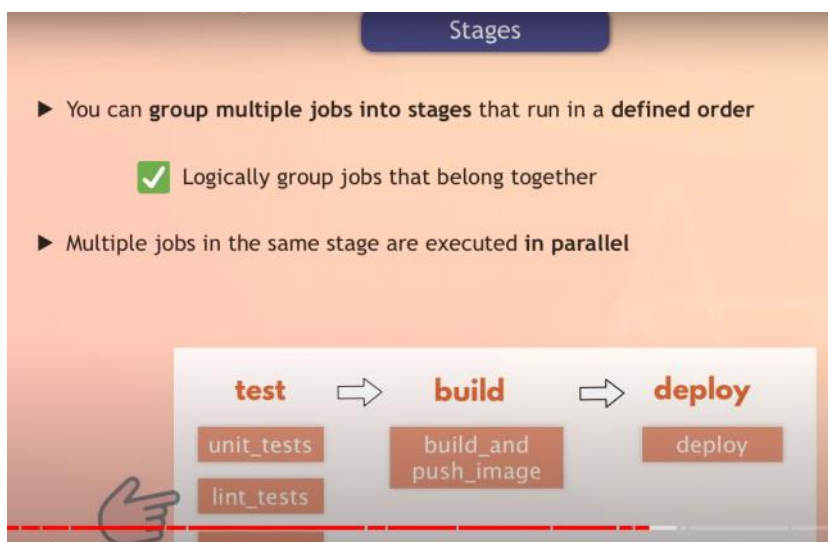
stage 2

build_image

44:27 / 1:08:59 • Define Stages



Using stages to accomplish it:

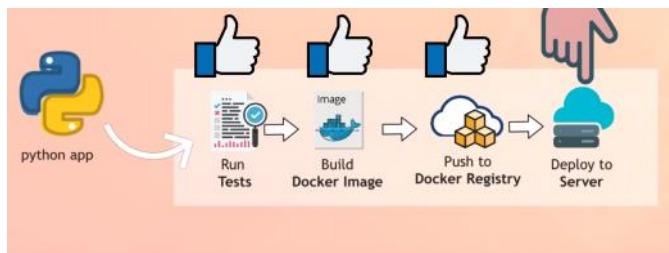



```

4
5 stages:
6   - test
7   - build
8
9 run_tests:
10  stage: test
11  image: python:3.9-slim-buster
12  before_script:
13    - apt-get update && apt-get install make
14  script:
15    - make test
16
17    abc build
18 build_image:
19   stage: build
20   image: docker:20.10.16

```

Last step remaining to deploy to servers:



Creating server on DIGITAL OCEAN:

Create Server on DigitalOcean



deployment-server


- ▶ We need a development server, where we deploy and run our application
- ▶ For that, we will create a simple ubuntu server on DigitalOcean platform



DigitalOcean



Easier to set up compared to AWS, if you don't know AWS and don't have an account yet

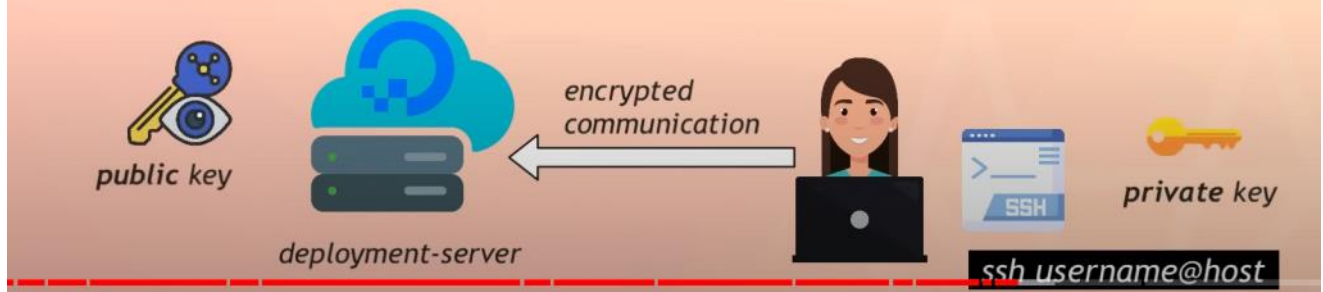


GitLab
CI/CD
FROM ZERO
TO HERO

In the advanced course, we use AWS though, as it's more realistic, because more used

SSH (Secure Shell) is used to **securely access remote servers** over the internet

► SSH key pair is used to authenticate hosts to each other



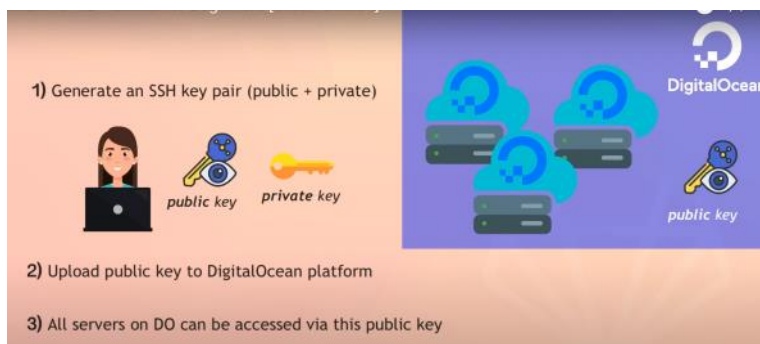
Settings

Team **Security** Referrals

SSH keys

Add SSH Key

SSH keys provide a more secure way of logging into a virtual private server with SSH than using a password alone. [Learn more](#)



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/.ssh/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:h3X0e4shDKxIkyS+TMbvkcXjvfnYvM/rmipZPsXzM 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=0B++o.|
+----[SHA256]-----+
```

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

Getting public key for it:

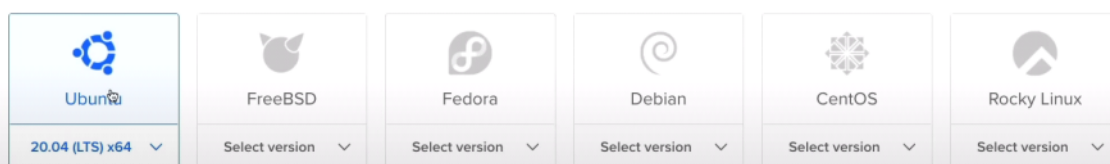
```
[W]$ cat /Users/nanajanashia/.ssh/digital_ocean_key.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGCcMaimAMP5xMjsbiRmjKxR9nKiAuomiSCI6000D7b/HMD036Yj2156yH
z+8/lat+FJycVFZ58Dgns3FqGIaiE776FJ2cLziCL67w3YmVpVbHT6w7Xc5eML2d+AP/AsrVIBMSqvGrqGb6jV5bfuT07c
zNX1sGa39UbDXpVD6FRv/eAqh5pQP8C/QhTk8Y2dIT+WrbBGPwL7CG5gEA73DTCfPZCNovhv0nqUIFCJecwReLmhN8PotL
dxFi8aBEFKod6eOgnMQuv8myM2SrHtOjRuWZJzP7mjNPVfo9uW6jl6c6Cul6fXj8oelRptFQkwnqcA9jE6YQSWiudjAo38
aRwGGst6b5dCX1X/IVGsyM+bgrxDC+shr6i21bCt0fK0cSGy0iZp1gG3HtZFW6+rxj2q0pwPsvmfz+Hzu5DYFdQIkHwo3
3HSUzy9G7LL/XID42UJq/AUey+4B5uF9L3iBQC8fHobFmFo4BCI1+GA7T7egVuo+xmXibP5wZ3CrzwtM= nanajanashi
a@Nanas-MacBook-Pro.local
[W]$
```

Creating deployment server via droplets option in digital ocean:

Create Droplets

Choose an image ?

Distributions Container distributions Marketplace Custom images



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

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

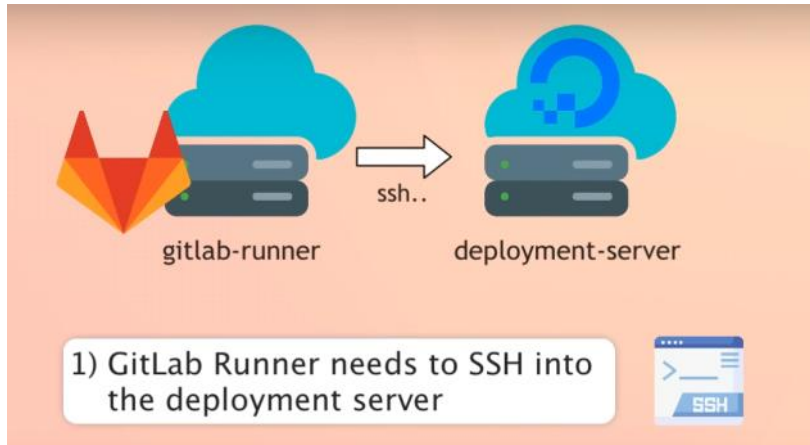
That's public ip address at end,of our created server on digital ocean.

Command to install docker on our server:

->>apt update

->>apt install docker.io

Connecting Gitlab to our digital ocean server:



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

The screenshot shows the 'Add variable' dialog in GitLab. The 'Key' field is labeled 'SSH_KEY'. The 'Value' field contains a long, base64-encoded string representing an SSH private key. The 'Type' is set to 'Variable' and the 'Environment scope' is set to 'All (default)'. There are checkboxes for 'Protect variable' (checked) and 'Mask variable' (unchecked). The 'Flags' section includes a note about exporting variables to pipelines. At the bottom, there are 'Cancel' and 'Add variable' buttons.

In the automated pipeline we want to disable this interactive step

```
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:

```
[W]$ ls -l ~/.ssh/digital_ocean_key
-rw----- 1 nanajanashia staff 2635 May 16 13:29 /Users/nanajanashia/.ssh/digital_ocean_key
```

File
Permissions:

```
# ls -l file
-rw-r--r-- 1 root root 0 Nov 19 23:49 file
```

File type

Owner (rw-)
Group (r- -)
Other (r- -)

r = Readable
w = Writeable
x = Executable
- = Denied

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:

6 13:29 /Users/nanajan

7	rwx	111
6	rw-	110
5	r-x	101
4	r--	100
3	-wx	011
2	-w-	010
1	--x	001
0	---	000

Owner: 4

Group: 0

Other: 0

chmod 400

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:

Test

run_tests

Build

build_image

Deploy

deploy