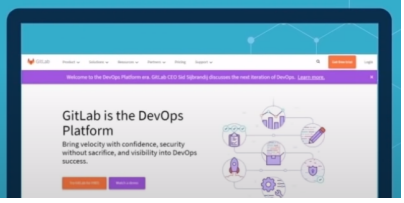
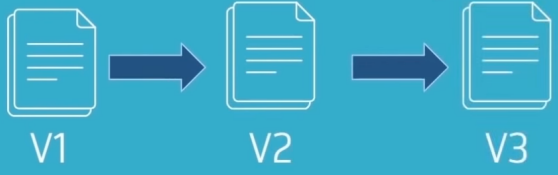
GitLab Overview

- GitLab is an Open-Source Software Development Platform
- Source Code Management System
- CI/CD
- Complete DevOps Platform



Git vs. GitLab

- Git is a Version Control System



git

Git vs. GitLab

- GitLab is a Source Code Management System

Why Use GitLab?

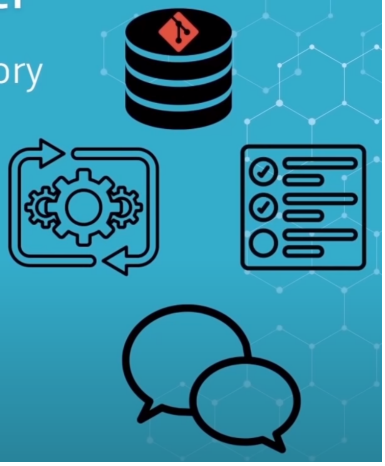
- GitLab enables collaboration among a team of developers with the GitLab Flow
- Built-in CI/CD functionality
- It can integrate with other tools
- And a plethora of other features

Key GitLab Terms - Group

- Allow you to manage settings across multiple projects
- Logically categorize multiple users or projects
- Provide a cross-project view

Key GitLab Terms - Project

- A container for a Git repository
- Built in CI/CD
- Issue Tracking
- Collaboration Tools



Key GitLab Terms - Member

- Users or Groups that have access to a project
- Members are assigned to roles
- Member roles include permissions to perform actions on projects or groups

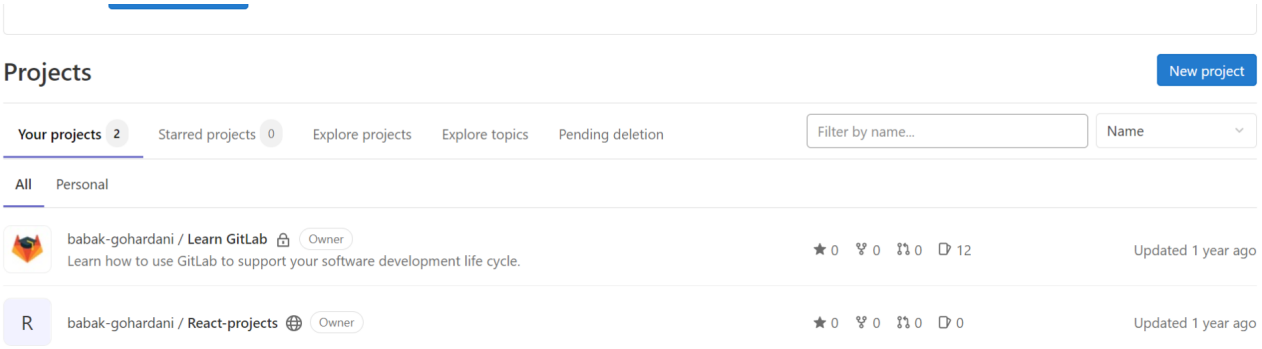
Key GitLab Terms - Merge Requests

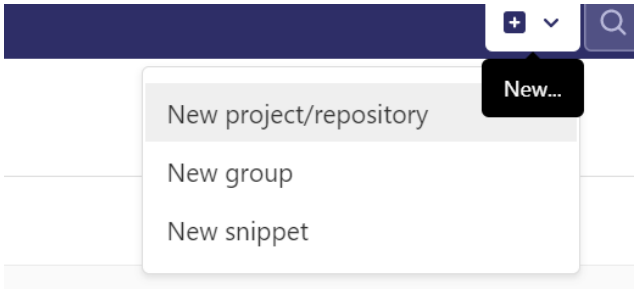
- A request to merge one branch into another
- Merge Requests provide a space to have a conversation with the team about the changes on a branch

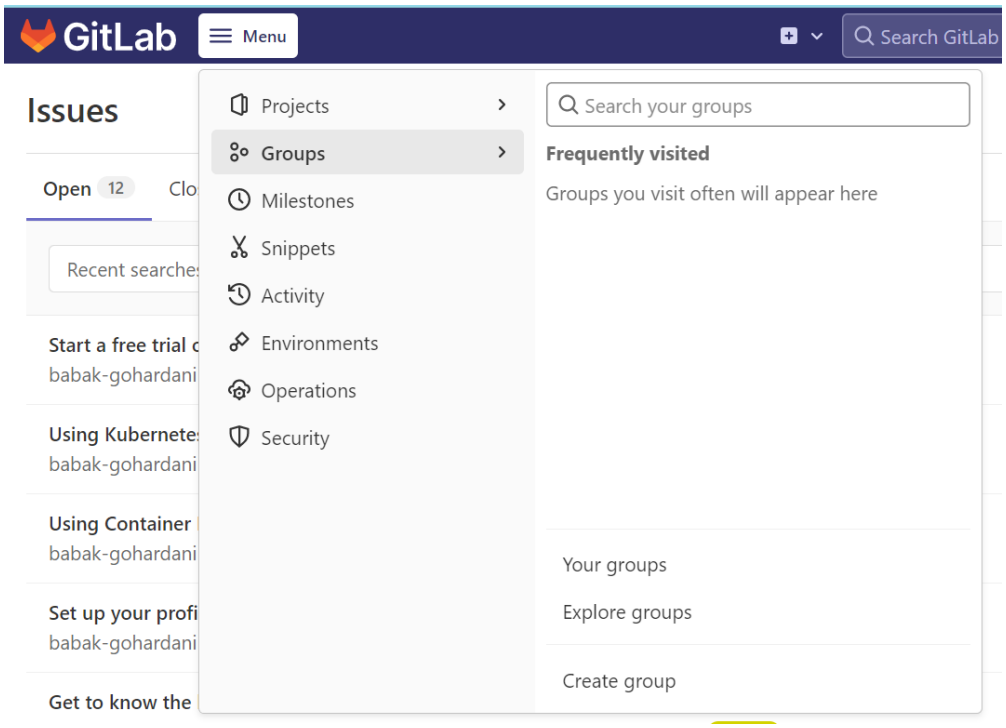
- It is the central place through which changes are reviewed and verified

Key GitLab Terms - Issue

- An issue is a way to track work related to a GitLab project
- Issues can be used to report bugs, track tasks, request new features, ask questions and more
- Your development workflow should begin with an issue


Projects
New project
Your projects 2
Starred projects 0
Explore projects
Explore topics
Pending deletion
Filter by name...
Name
All
Personal
babak-gohardani / Learn GitLab
Owner
Learn how to use GitLab to support your software development life cycle.
Updated 1 year ago
R
babak-gohardani / React-projects
Owner
Updated 1 year ago


New...
New project/repository
New group
New snippet


GitLab
Menu
Search GitLab
Issues
Open 12
Projects
Groups
Milestones
Snippets
Activity
Environments
Operations
Security
Search your groups
Frequently visited
Groups you visit often will appear here
Your groups
Explore groups
Create group

User Settings › SSH Keys

Search settings

SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab.

Add an SSH key

Add an SSH key for secure access to GitLab. Learn more.

Key

Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'.

Title

Example: MacBook key

Key titles are publicly visible.

Expiration date

mm/dd/yyyy

Key becomes invalid on this date.

Add key

Your SSH keys (0)

There are no SSH keys with access to your account.

Search settings

SSH Key

Title: **GitLab KeyPair**

Created on: **May 10, 2022 10:52am**

Expires: **Never**

Last used on: **Never**

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIPsF0a4IV2FZWeiDl8ltO+jpmTAgKoY6MeGbHKCvmzI
```

Fingerprints

MD5: `50:e5:e3:2a:cc:02:4e:b5:60:a5:d5:7f:79:1e:14:84`

SHA256: `Xg0xHx9ydmwgvNyEtoXqcV6Py0rgDb7Qubpucq6oX1c`

Delete

New group › **Create group**

Group name

My awesome group

Must start with letter, digit, emoji, or underscore. Can also contain periods, dashes, spaces, and parentheses.

Group URL

https://gitlab.com/ my-awesome-group

Visibility level

Who will be able to see this group? View the documentation

- Private
  The group and its projects can only be viewed by members.
- Public
  The group and any public projects can be viewed without any authentication.

Now, personalize your GitLab experience

We'll use this to help surface the right features and information to you.

Role

Software Developer

Who will be using this group?

My company or team Just me

What will you use this group for?

Invite Members (optional)

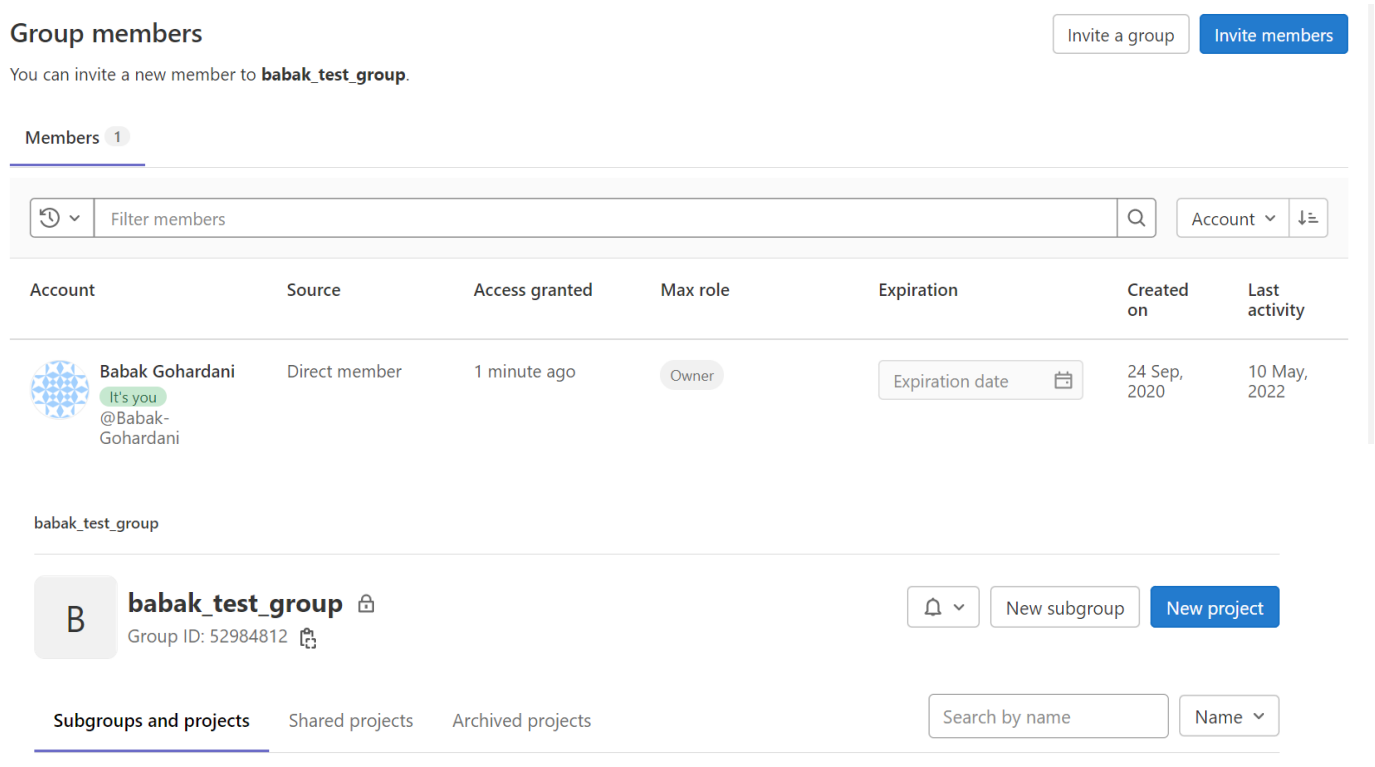
Invited users will be added with developer level permissions. View the documentation to see how to change this later.

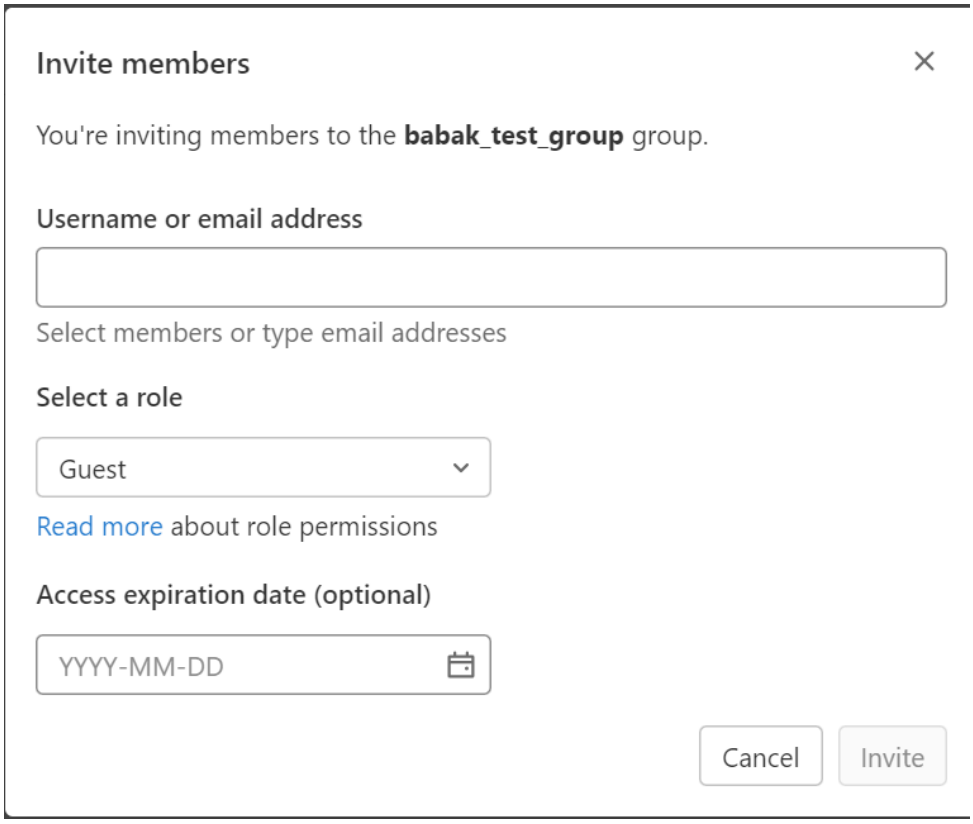
Email 1

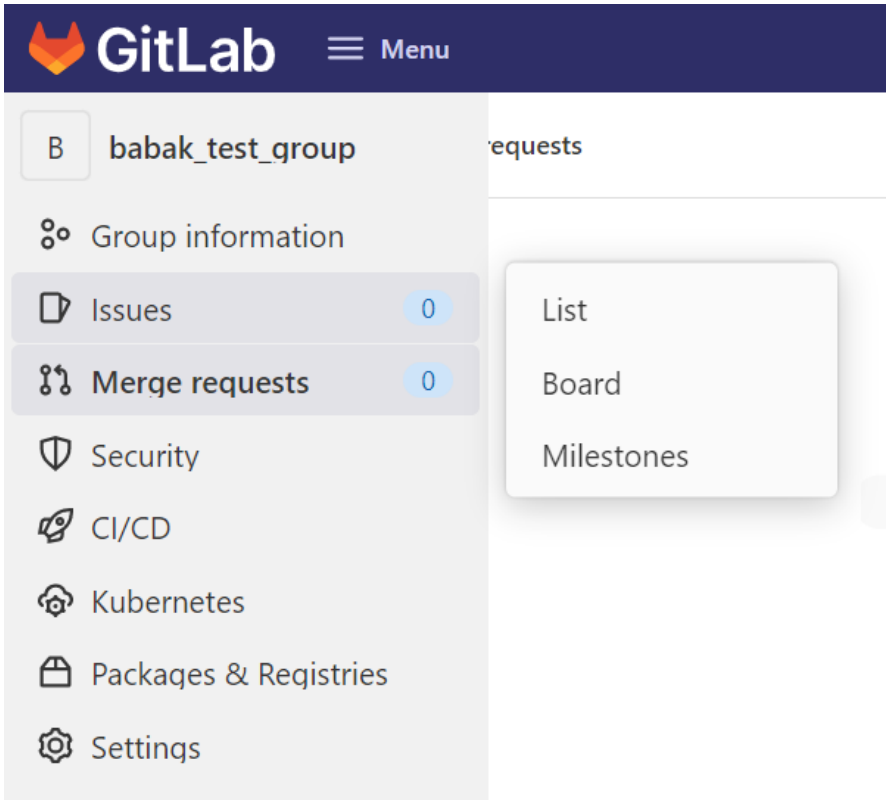
member1@company.com

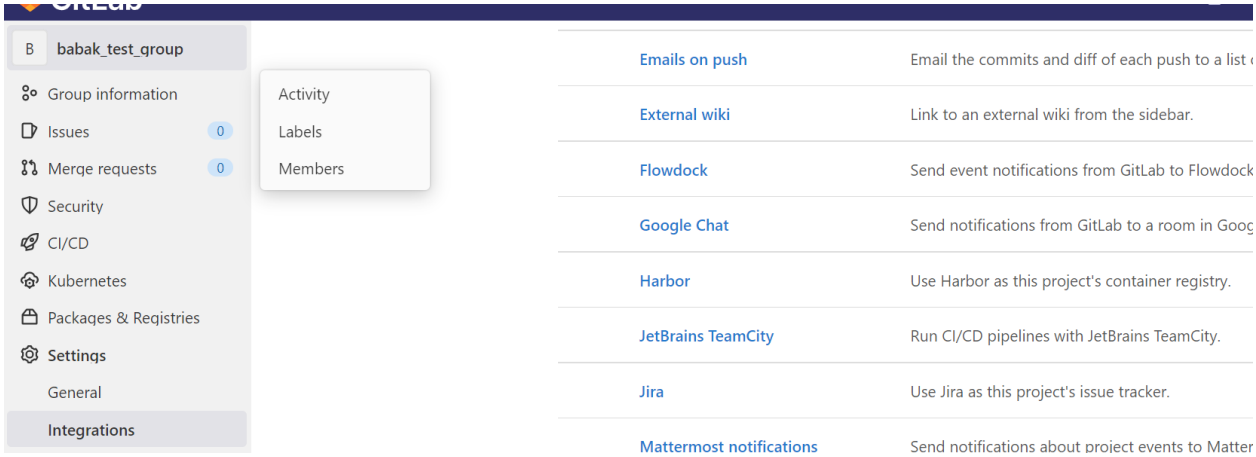
+ Invite another member

Create group Cancel


Group members
Invite a group
Invite members
You can invite a new member to babak_test_group.
Members 1
Filter members
Account
Account
Source
Access granted
Max role
Expiration
Created on
Last activity
Babak Gohardani
It's you
@Babak-Gohardani
Direct member
1 minute ago
Owner
Expiration date
24 Sep, 2020
10 May, 2022
babak_test_group
B
babak_test_group
Group ID: 52984812
New subgroup
New project
Subgroups and projects
Shared projects
Archived projects
Search by name
Name


Invite members
You're inviting members to the babak_test_group group.
Username or email address
Select members or type email addresses
Select a role
Guest
Read more about role permissions
Access expiration date (optional)
YYYY-MM-DD
Cancel
Invite


GitLab
Menu
B
babak_test_group
Group information
Issues
0
Merge requests
0
Security
CI/CD
Kubernetes
Packages & Registries
Settings
List
Board
Milestones


B
babak_test_group
Group information
Issues
0
Merge requests
0
Security
CI/CD
Kubernetes
Packages & Registries
Settings
General
Integrations
Activity
Labels
Members
Emails on push
Email the commits and diff of each push to a list
External wiki
Link to an external wiki from the sidebar.
Flowdock
Send event notifications from GitLab to Flowdock
Google Chat
Send notifications from GitLab to a room in Goog
Harbor
Use Harbor as this project's container registry.
JetBrains TeamCity
Run CI/CD pipelines with JetBrains TeamCity.
Jira
Use Jira as this project's issue tracker.
Mattermost notifications
Send notifications about project events to Matter

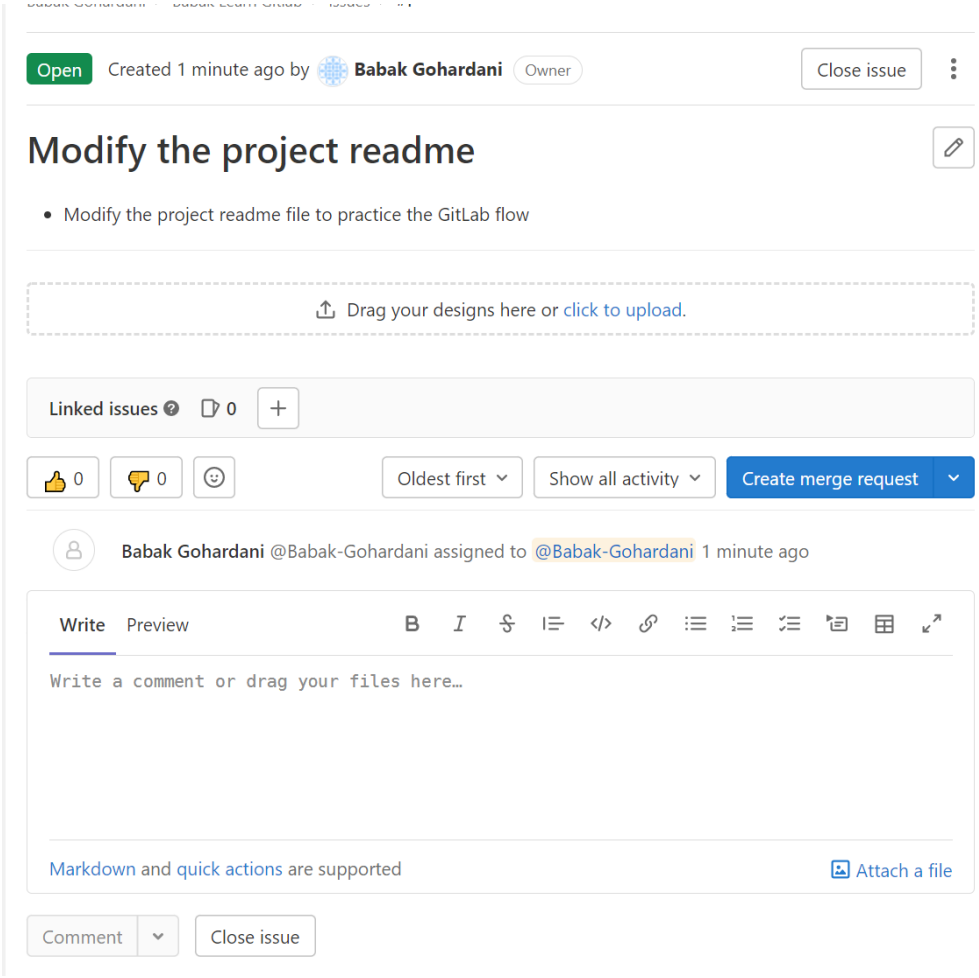
Import project

Migrate your data from an external source like GitHub, Bitbucket, or another instance of GitLab.

Project deployment target (optional)

Select the deployment target

- Select the deployment target
- Kubernetes (GKE, EKS, OpenShift, and so on)
- Managed container runtime (Fargate, Cloud Run, DigitalOcean App)
- Self-managed container runtime (Podman, Docker Swarm, Docker Compose)
- Heroku
- Virtual machine (for example, EC2)
- Mobile app store
- Registry (package or container)
- Infrastructure provider (Terraform, Cloudformation, and so on)
- Serverless backend (Lambda, Cloud functions)
- GitLab Pages
- Other hosting service
- No deployment planned


Open
Created 1 minute ago by
Babak Gohardani
Owner
Close issue
Modify the project readme
Modify the project readme file to practice the GitLab flow
Drag your designs here or click to upload.
Linked issues
0
0
0
Oldest first
Show all activity
Create merge request
Babak Gohardani @Babak-Gohardani assigned to @Babak-Gohardani 1 minute ago
Write
Preview
Write a comment or drag your files here…
Markdown and quick actions are supported
Attach a file
Comment
Close issue

GitLab Flow

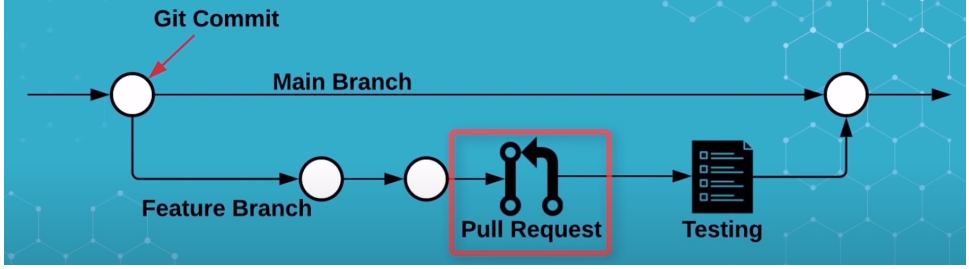

Branching Strategy
A software development workflow within the context of Git (or other Version Control System)
Describes how a development team will create, collaborate on, and merge branches of source code in a codebase

- Enables concurrent development in the codebase
- How to choose a branching strategy?
  - Depends on multiple factors
    - Team requirements
    - Source Code Management system
    - Application environments

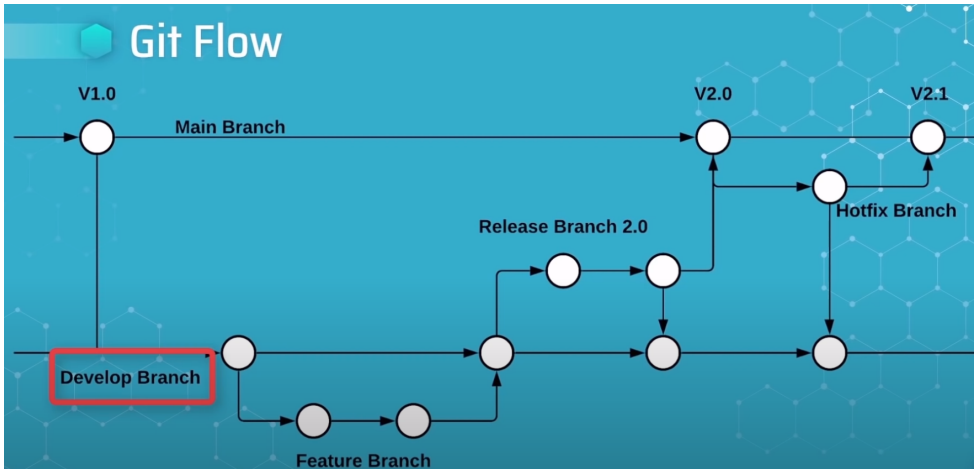
Common Git Branching Strategies

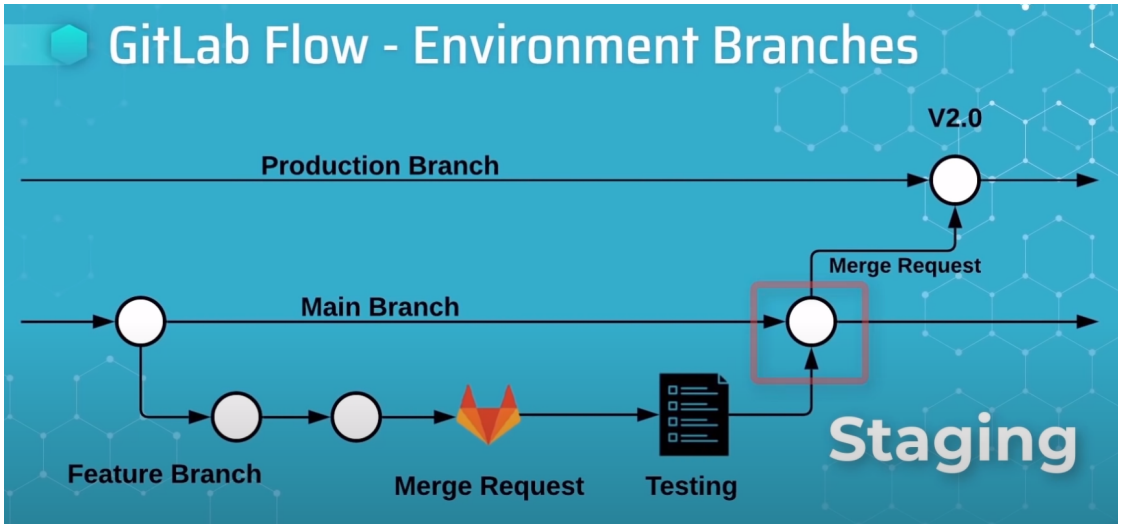
- GitHub Flow
- Git Flow
- GitLab Flow

GitHub Flow



Git Flow





* gitlab flow stars with creation of an issue.

Protected branches

Keep stable branches secure and force developers to use merge requests. What are protected branches?

By default, protected branches restrict who can modify the branch. Learn more.

Protect a branch

Branch: Select branch…

Wildcards such as `*-stable` or `production/*` are supported.

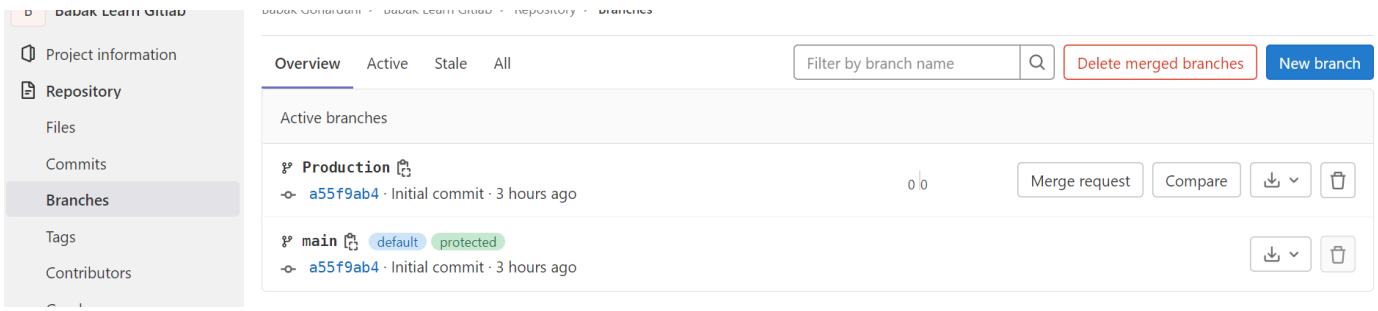
Allowed to merge: Select

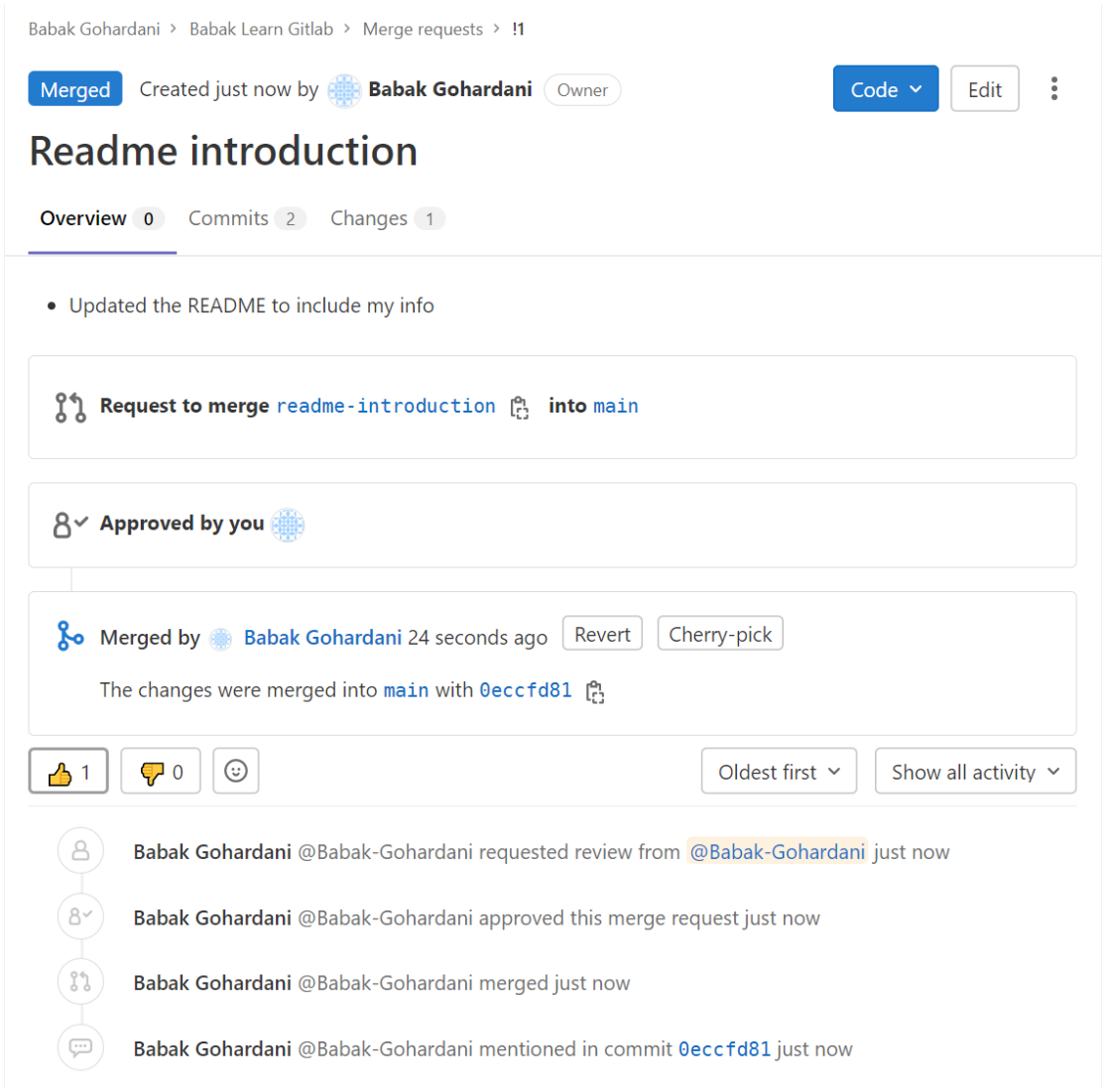
Allowed to push: Select

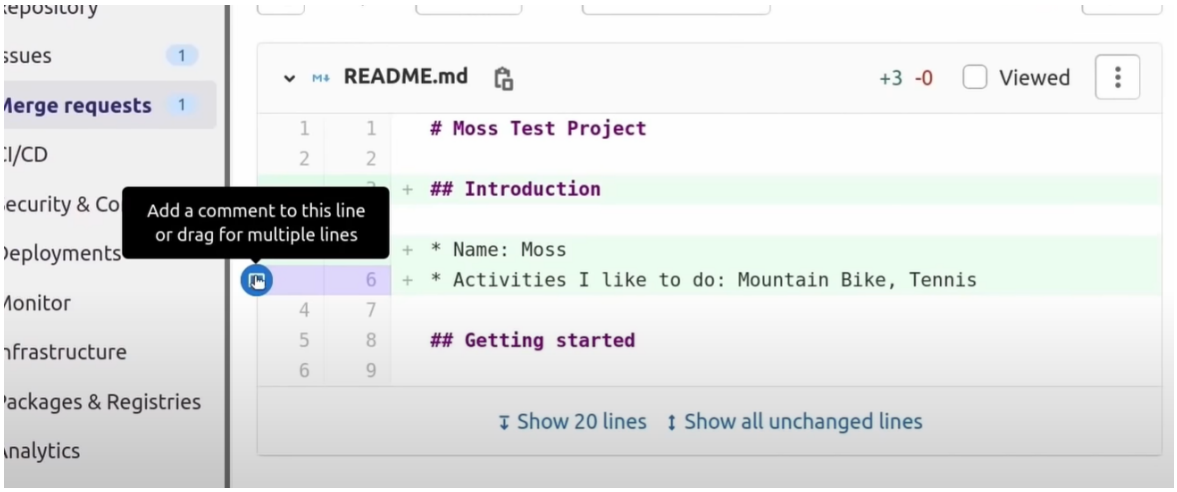
Allowed to force push:

Allow all users with push access to force push.

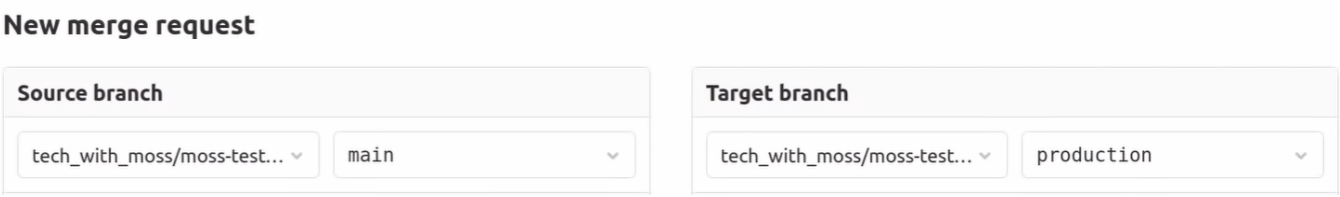
Protect




Babak Gohardani › Babak Learn Gitlab › Merge requests › !1
Merged
Created just now by Babak Gohardani
Owner
Code
Edit
Readme introduction
Overview 0
Commits 2
Changes 1
Updated the README to include my info
Request to merge readme-introduction into main
Approved by you
Merged by Babak Gohardani 24 seconds ago
Revert
Cherry-pick
The changes were merged into main with 0eccfd81
1
0
Oldest first
Show all activity
Babak Gohardani @Babak-Gohardani requested review from @Babak-Gohardani just now
Babak Gohardani @Babak-Gohardani approved this merge request just now
Babak Gohardani @Babak-Gohardani merged just now
Babak Gohardani @Babak-Gohardani mentioned in commit 0eccfd81 just now


ssues 1
Merge requests 1
I/CD
ecurity & Co
Deployments
Monitor
nfrastructure
Packages & Registries
Analytics
README.md
+3 -0
Viewed
Moss Test Project
Introduction
* Name: Moss
* Activities I like to do: Mountain Bike, Tennis
Getting started
Add a comment to this line or drag for multiple lines
Show 20 lines
Show all unchanged lines

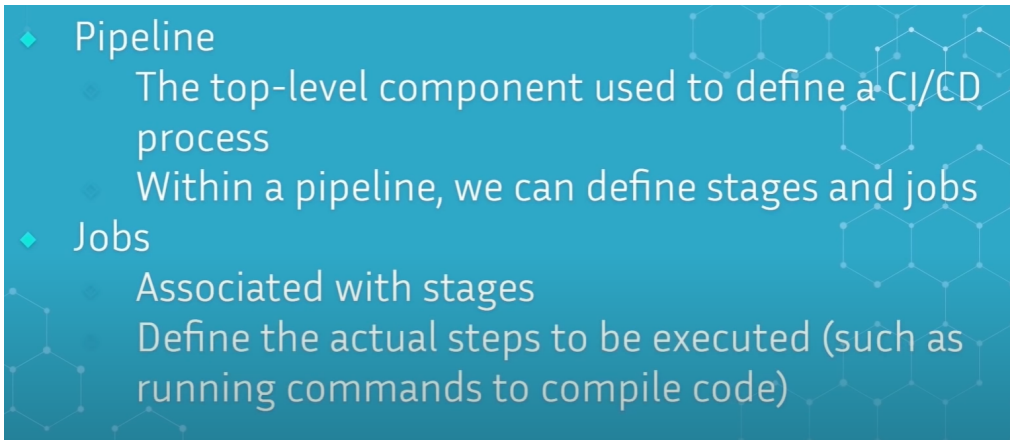
- After we make changes after a merge request has been issued, it will automatically associate the changes with the open merge request and its issues.

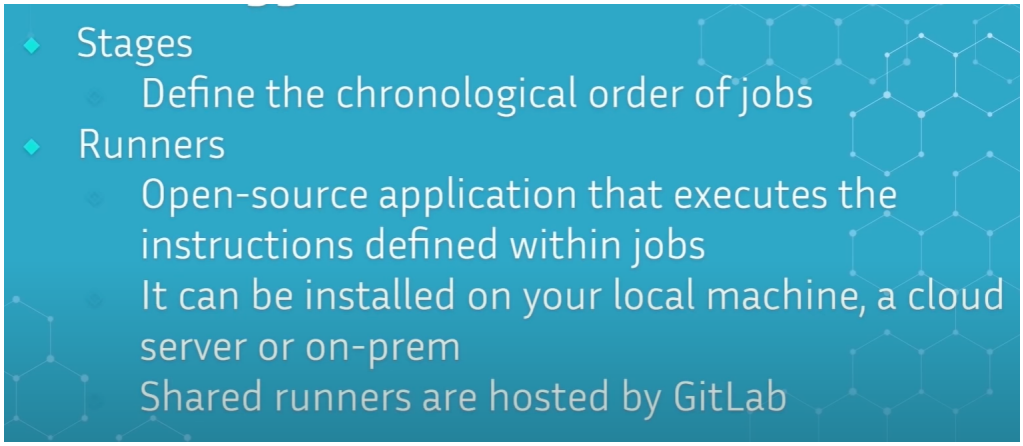


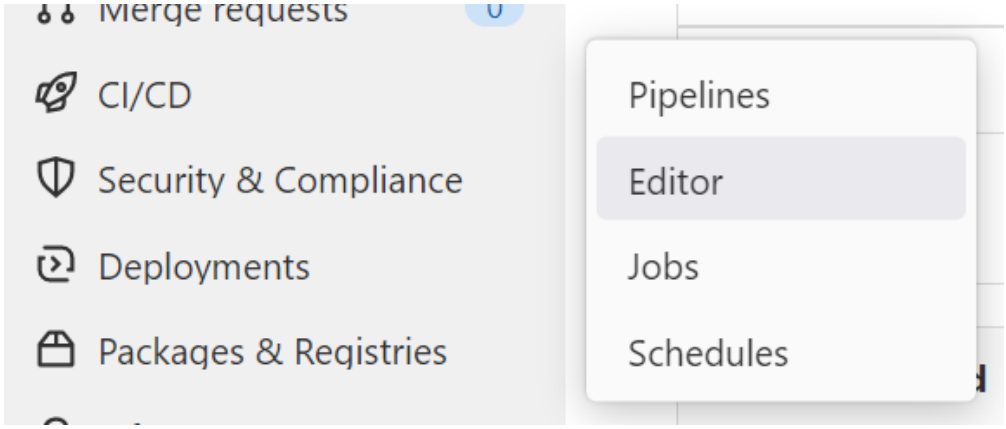
- Each merge request should invoke automated testing with CI pipeline
- Repository -> create Tag => Deployment -> releases : automatically generated
- Create a Tag for each merge of Main branch into Prod branch

GitLab Flow:

- **Create main repo**
- **Create production repo**
- **Create an issue**
- **Create feature branch based on the issue**
- **Merge feature branch into main => invoke automated tests => delete feature branch after merge**
- **Merge main branch into Production branch (can invoke test again if needed) => create release Tag for Prod => close the issue => deploy into main app, recreate container, …**







.gitlab-ci.yml is the exact file in GitLab for CI/CD process. It has 33 stages:

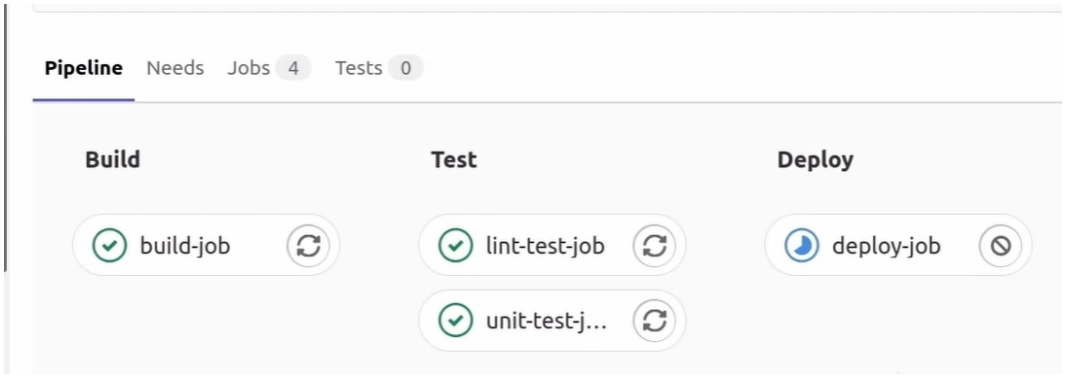
- Build
- Test
- Deploy

Each section can have its own custom shell commands and each is also visible separately on GitLab web UI.

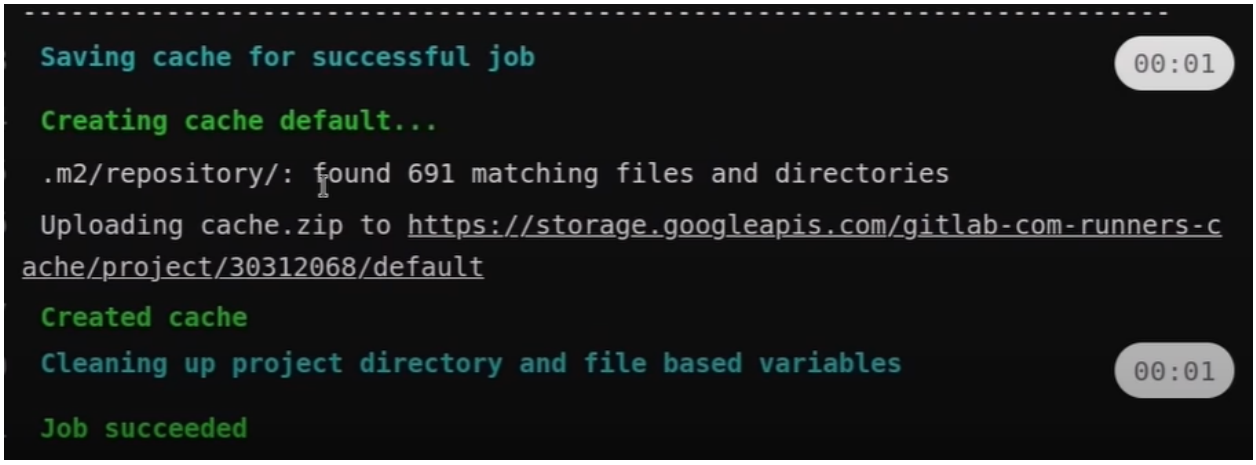
.gitlab-ci.yml 1.27 KiB

Edit in pipeline editor | Replace | Delete

```yaml
image: maven:latest

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

variables:
  MAVEN_CLI_OPTIONS: "--batch-mode"  # non-interactive mode
  MAVEN_OPTIONS: "-Dmaven.repo.local=$CI_PROJECT_DIR/.m2/repository"

cache:
  paths:
    - .m2/repository

# This job runs in the build stage, which runs first.
build-job:
  stage: build
  script:
    - echo "Compiling the code..."
    - mvn $MAVEN_CLI_OPTIONS compile

# This job runs in the test stage.
unit-test-job:
  stage: test    # It only starts when the job in the build stage completes successfully.
  script:
    - echo "Running unit tests"
    - mvn $MAVEN_CLI_OPTIONS test
  artifacts:
    when: always
    reports:
      junit:
        - target/surefire-reports/TEST-*.xml

# This job also runs in the test stage.
lint-test-job:
  stage: test    # It can run at the same time as unit-test-job (in parallel).
  script:
    - echo "Linting code"

# This job runs in the deploy stage.
deploy-job:
  stage: deploy  # It only runs when *both* jobs in the test stage complete successfully.
  environment: staging
  script:
    - echo "Deploying application..."
    - mvn $MAVEN_CLI_OPTIONS package
    - mvn $MAVEN_CLI_OPTIONS exec:java -Dexec.mainClass="com.mycompanny.app.App"
```



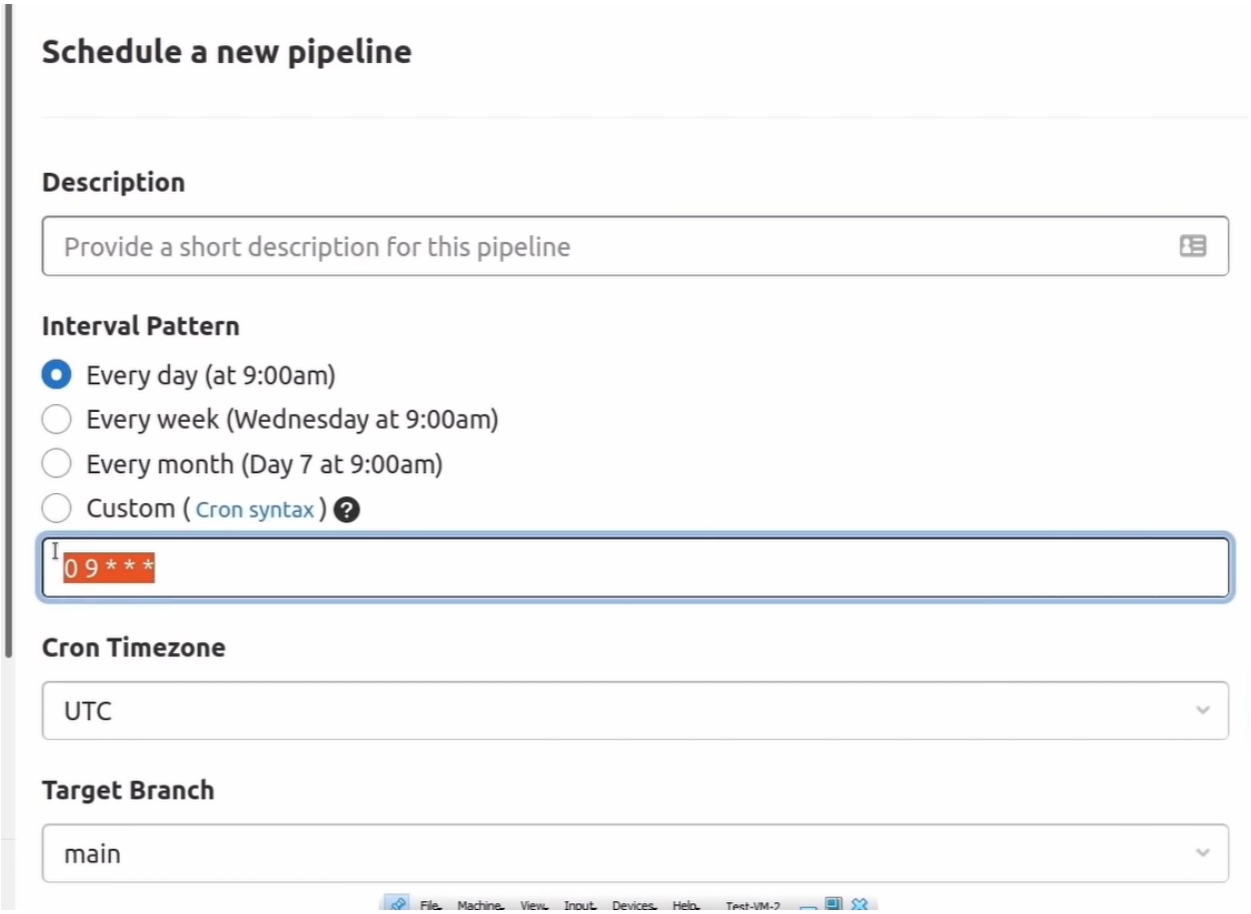

```
1  Running with gitlab-runner 14.3.0-rc1 (ed15bfbf)
2    on docker-auto-scale fa6cab46
3  Preparing the "docker+machine" executor                    00:24
4  Using Docker executor with image maven:latest ...
5  Pulling docker image maven:latest ...
6  Using docker image sha256:a3d57fce9bc8f09b2cd40faecc870f884deab19f7b3d05fc5
   8d1d8abc14f76ef for maven:latest with digest maven@sha256:d15844702506b4b0d
   8e28e037174081af6d107cca239176ae7fe4da267bb89a3 ...
8  Preparing environment                                      00:03
9  Running on runner-fa6cab46-project-30312068-concurrent-0 via runner-fa6cab4
   6-srm-1633802567-b0741217...
11 Getting source from Git repository                         00:02
12 $ eval "$CI_PRE_CLONE_SCRIPT"
13 Fetching changes with git depth set to 50...
14 hint: Using 'master' as the name for the initial branch. This default branc
   h name
15 hint: is subject to change. To configure the initial branch name to use in
```



1 tests | 0 failures | 0 errors | 100% success rate | 6.00ms

Jobs

Job	Duration	Failed	Errors	Skipped	Passed	Total
unit-test-job	6.00ms	0	0	0	1	1



- AutoDevOps : GitLab will try to set up a CI?CD pipeline itself based on your current project tech stack.

Gitlab Releases Overview

- Software packages generated from GitLab pipeline job artifacts
- Release notes
- Release evidence that includes everything associated with the release, such as issues, milestones or test reports

Package Registry

- Use GitLab as a public or private registry
- Supports a number of package managers
- Publish and share packages to the registry
- Packages can be published via CI/CD
- Packages are associated to a project and groups that project is added to

Container Registry

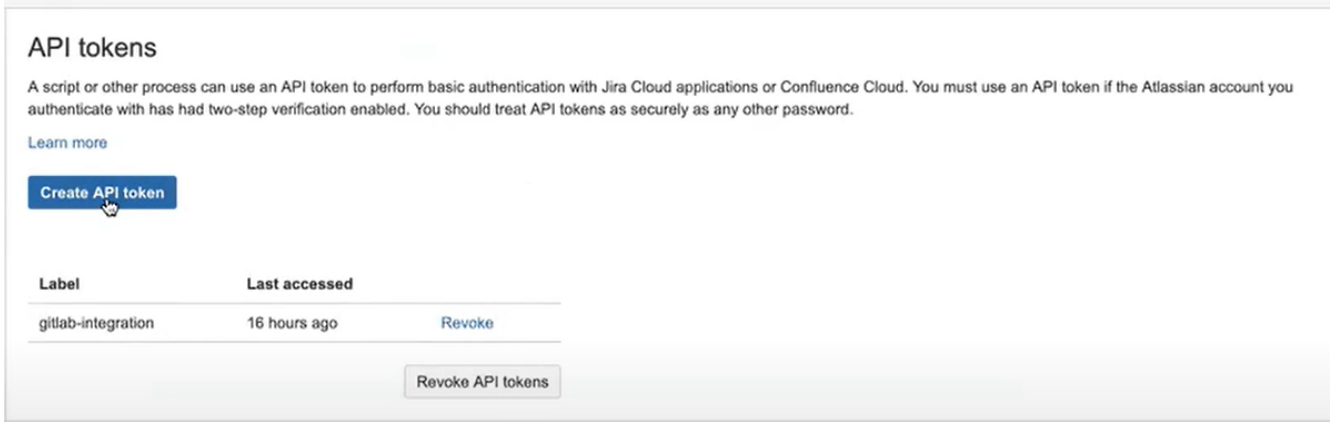
- Private container registry for publishing and consuming container images
- Every GitLab project has its own container registry
- The container images are associated with a project and any groups that project has been added to
- Utilize container images from a GitLab pipeline
- Build and publish container images to the registry from a GitLab pipeline

- Example of how to work with container registry and docker

```
build:
  image: docker:19.03.12
  stage: build
  services:
    - docker:19.03.12-dind
  script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
    - docker build -t $CI_REGISTRY/group/project/image:latest .
    - docker push $CI_REGISTRY/group/project/image:latest
```

GitLab integration with Jira

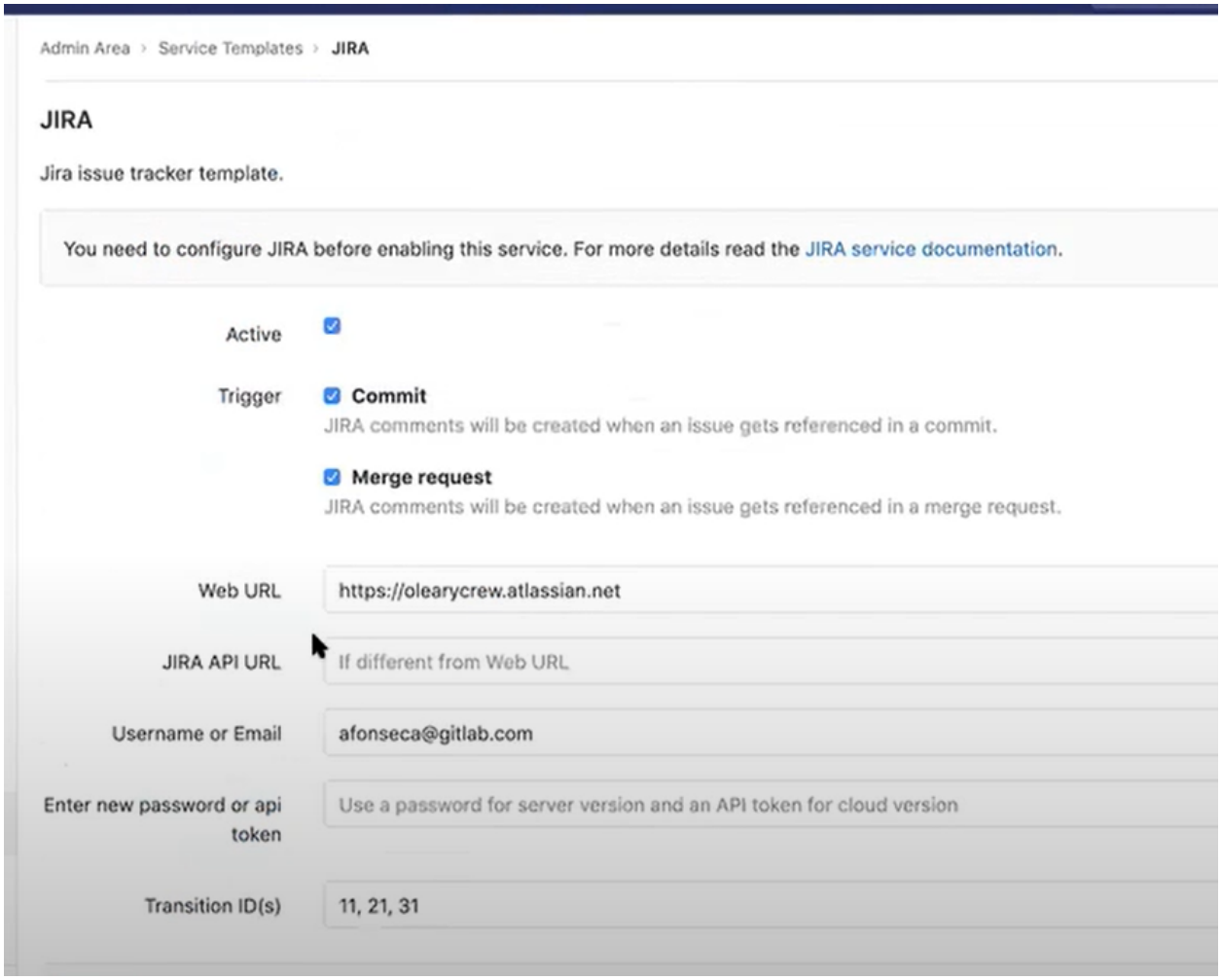
Go to Jira account -> setting -> security -> generate token



In GitLab can be done in two ways:

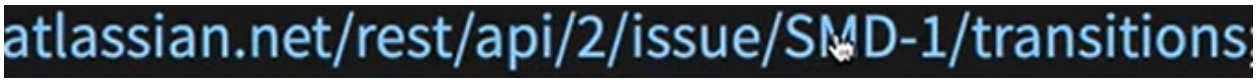
1. As GitLab admin and for instance-wide level
2. Enabled as normal user in project based level

Gitlab Admin page => Service Templates => Jira section



Username and Token need to belong to same account of Jira

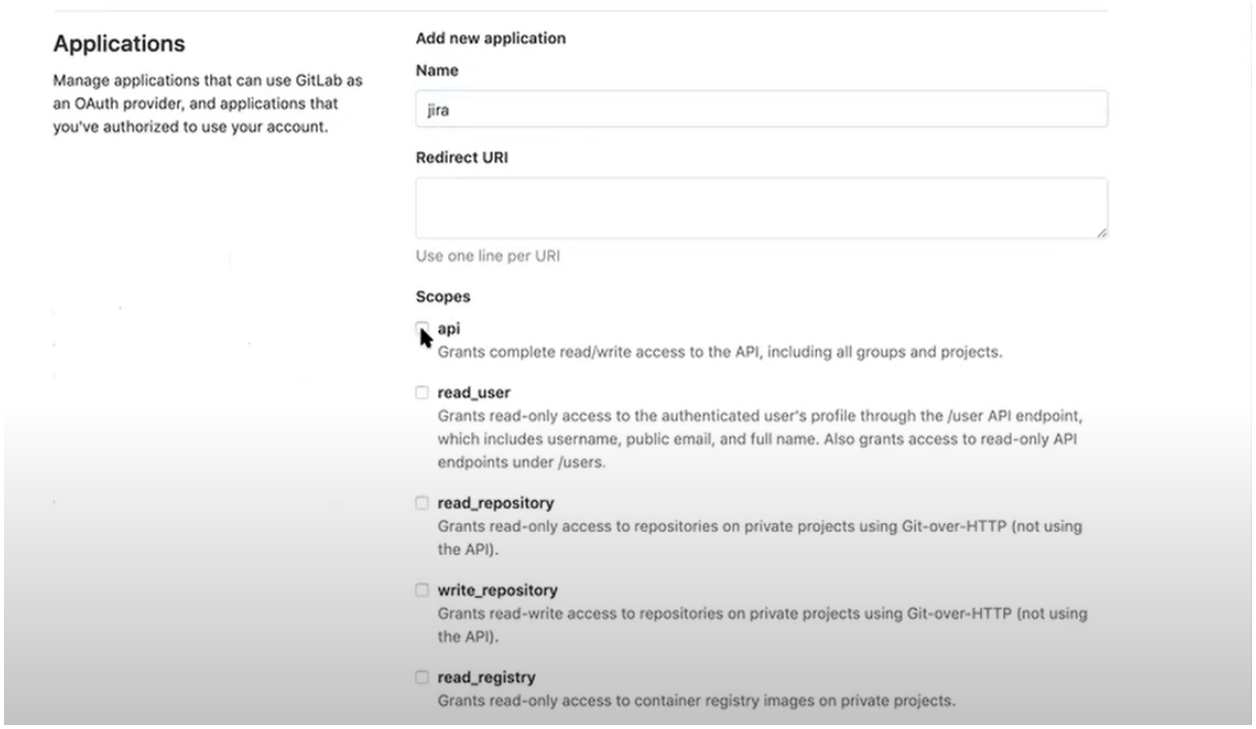
Transition Id can be found here:

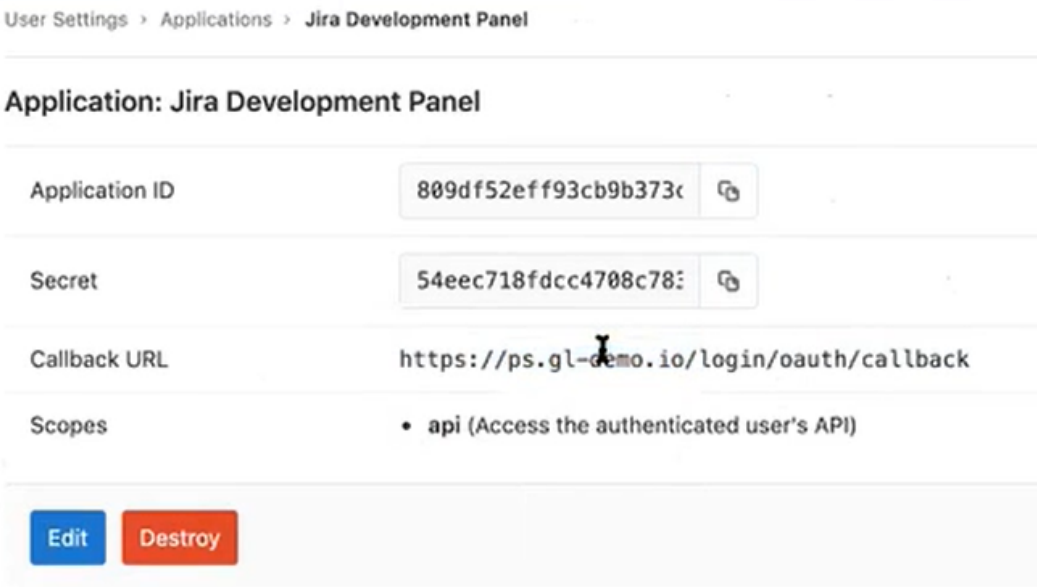


** As a result, every time that a comment gets posted in gitlab, we will have a notification in Jira.

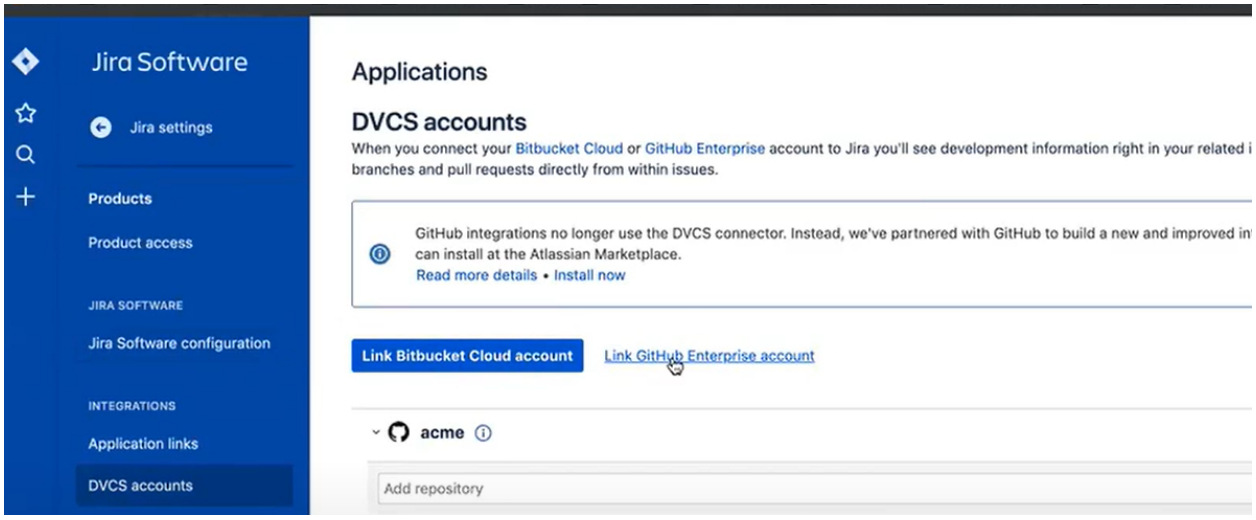
Additional integration (to show gitlab commits and merges in Jira side Panel):

User setting => applications => add new application

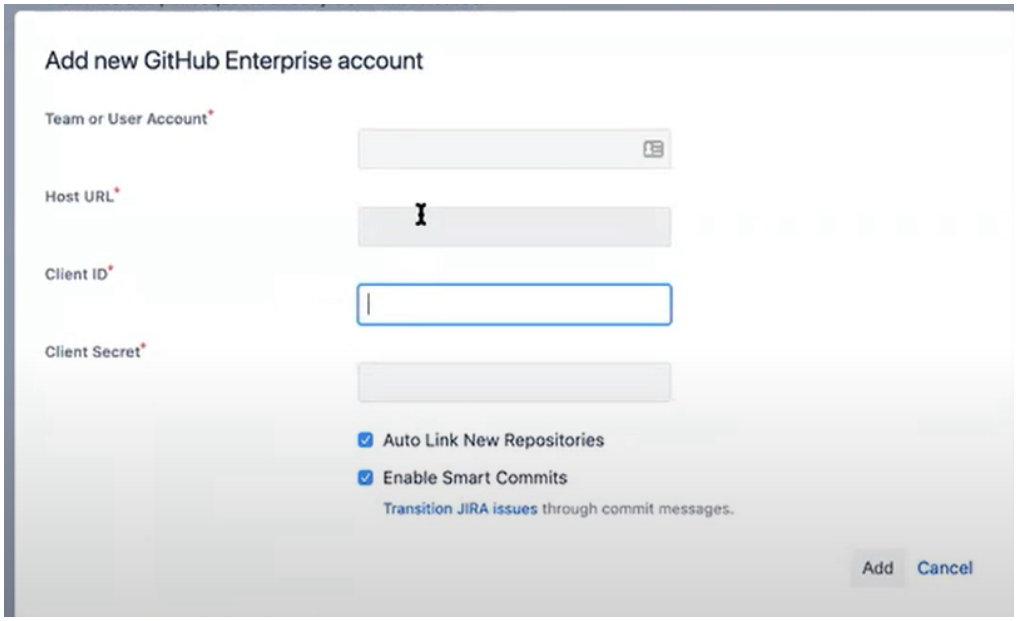




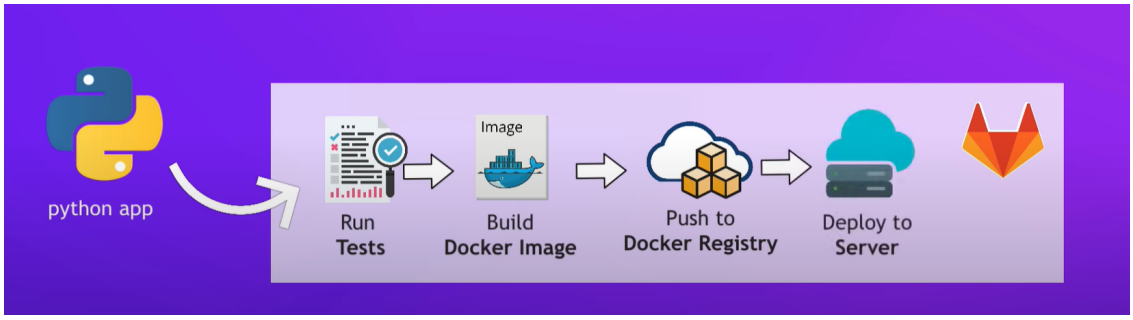
On the Jira side (can use github link)

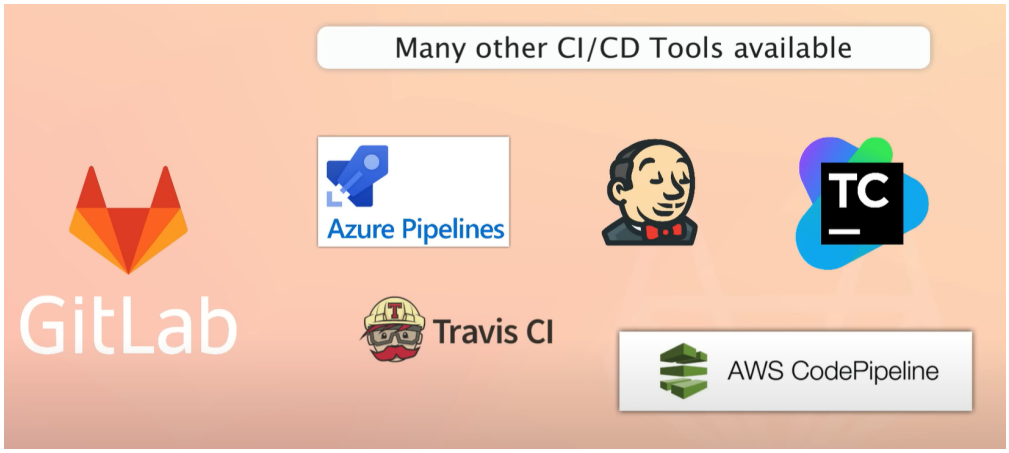


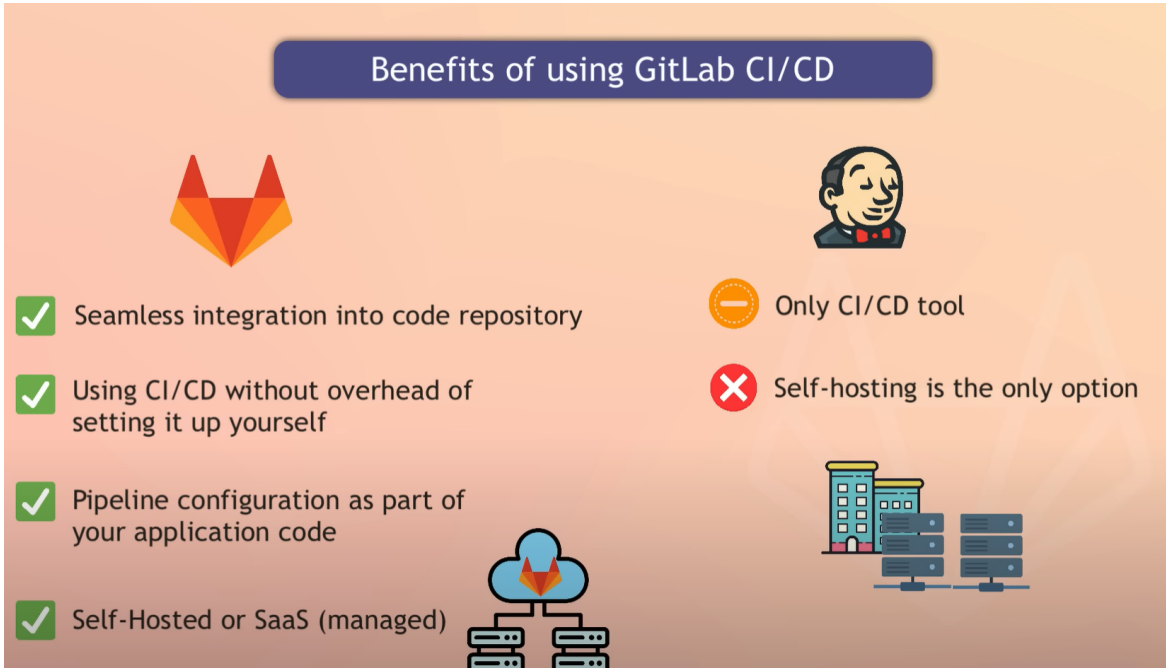
Team or user account = highest level group account name
Host url = link to gitlab instance
Client id = application id
Client secret = secret

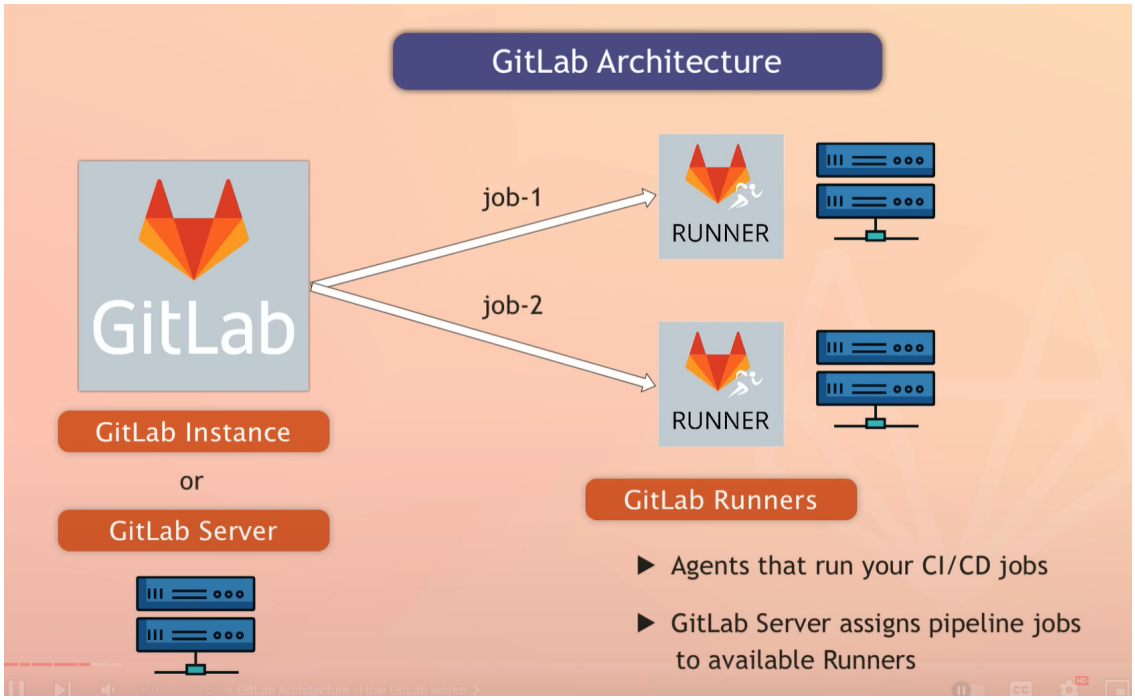


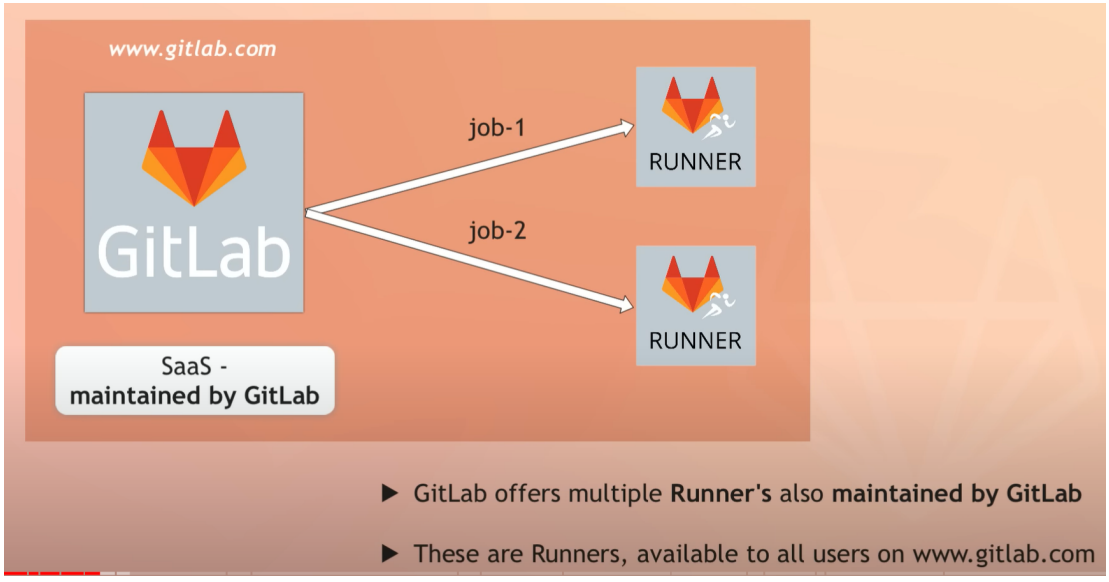
** GitLab CI/CD **


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

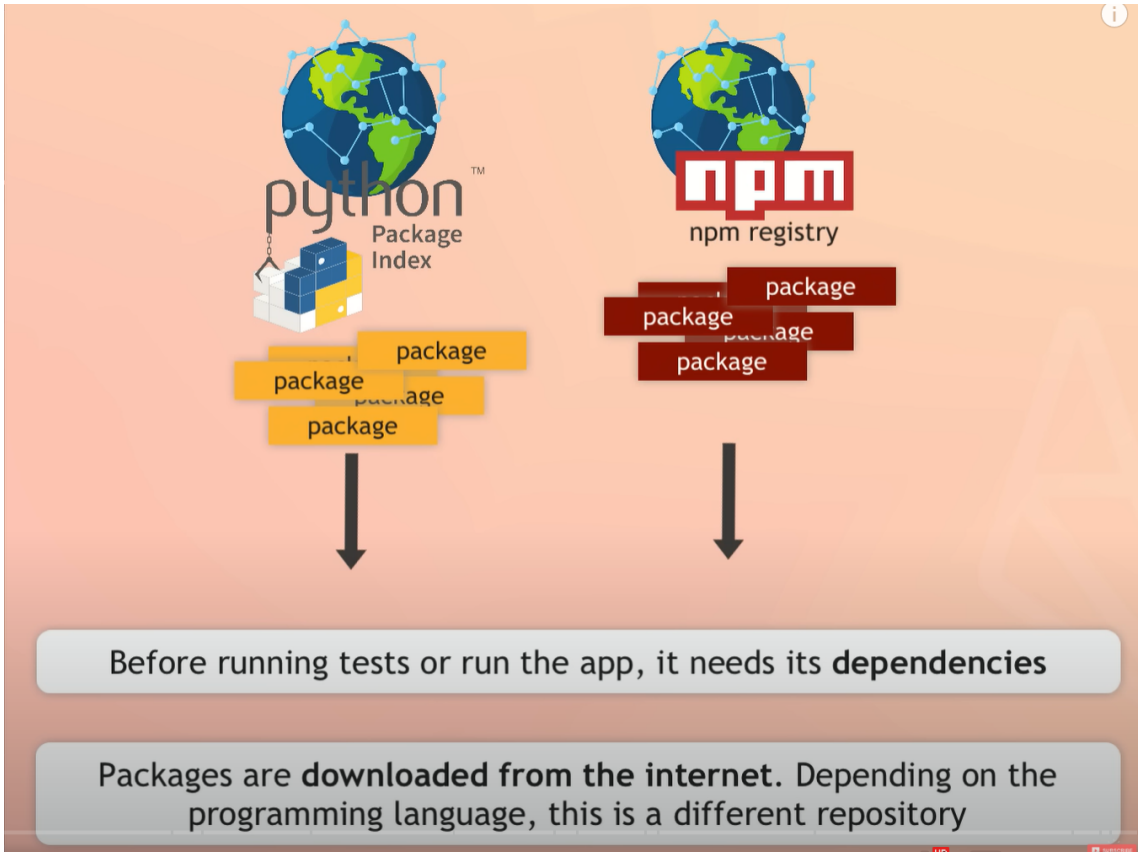

Many other CI/CD Tools available
GitLab
Azure Pipelines
Travis CI
AWS CodePipeline

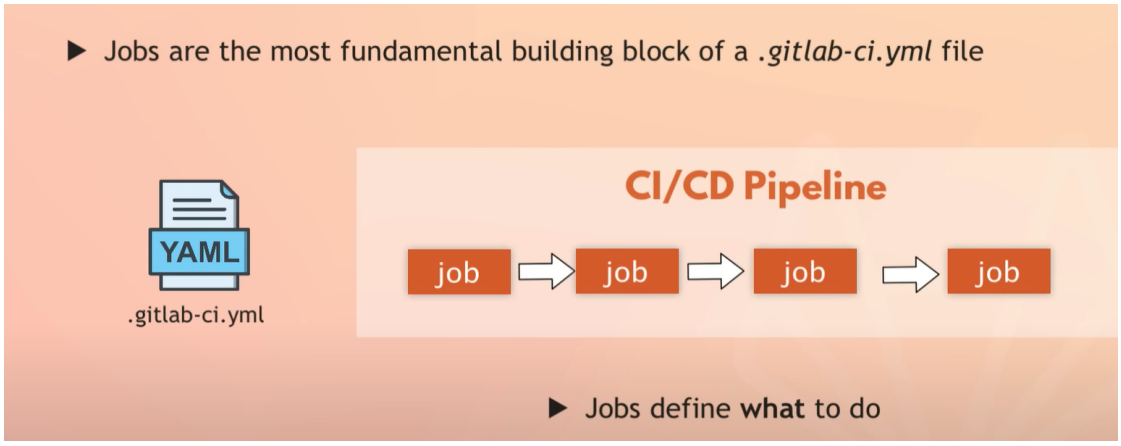

Benefits of using GitLab CI/CD
Seamless integration into code repository
Using CI/CD without overhead of setting it up yourself
Pipeline configuration as part of your application code
Self-Hosted or SaaS (managed)
Only CI/CD tool
Self-hosting is the only option

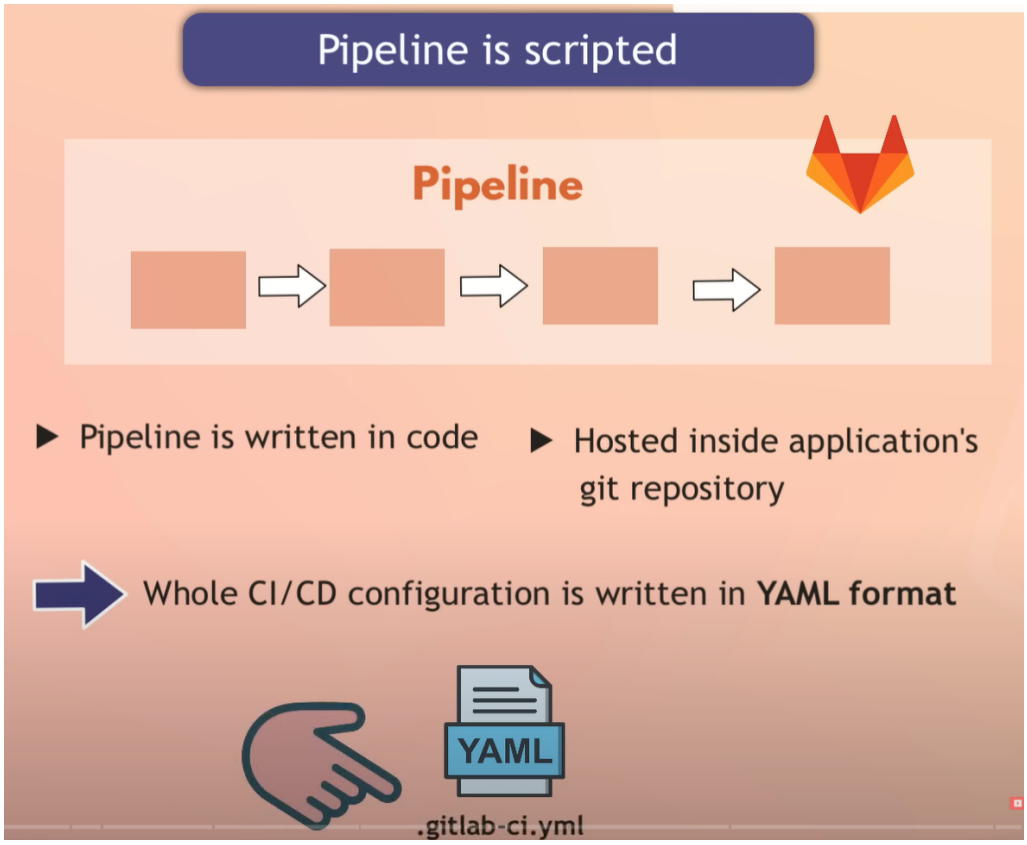




Makefile = a special file, containing shell commands that you create and name


python
Package Index
package
package
package
npm
npm registry
package
package
package
Before running tests or run the app, it needs its **dependencies**
Packages are **downloaded from the internet**. Depending on the programming language, this is a different repository


Jobs are the most fundamental building block of a *.gitlab-ci.yml* file
YAML
.gitlab-ci.yml
CI/CD Pipeline
job
job
job
job
Jobs define **what** to do



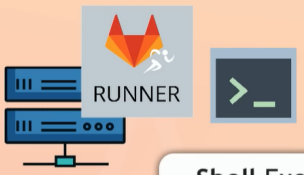
Jobs

- 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"
```

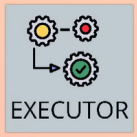
Different Types of Executors

- 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

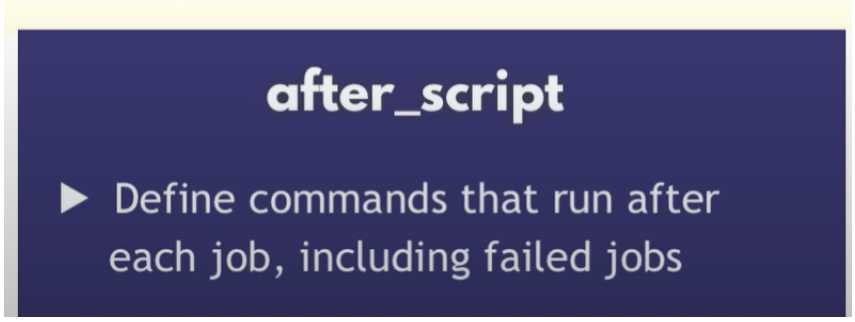


Different Types of Executors

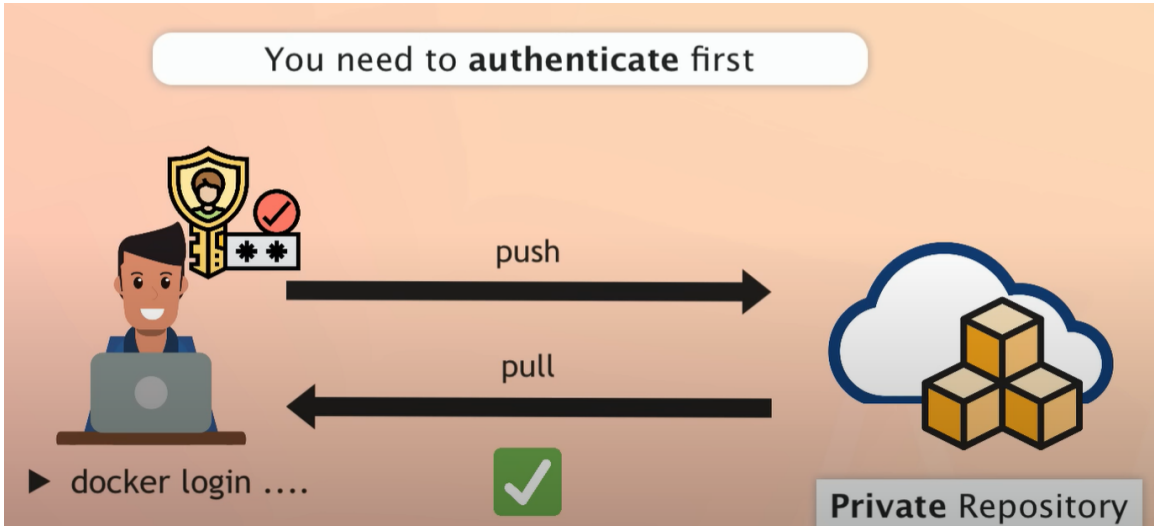
- Executor determines the environment each job runs in

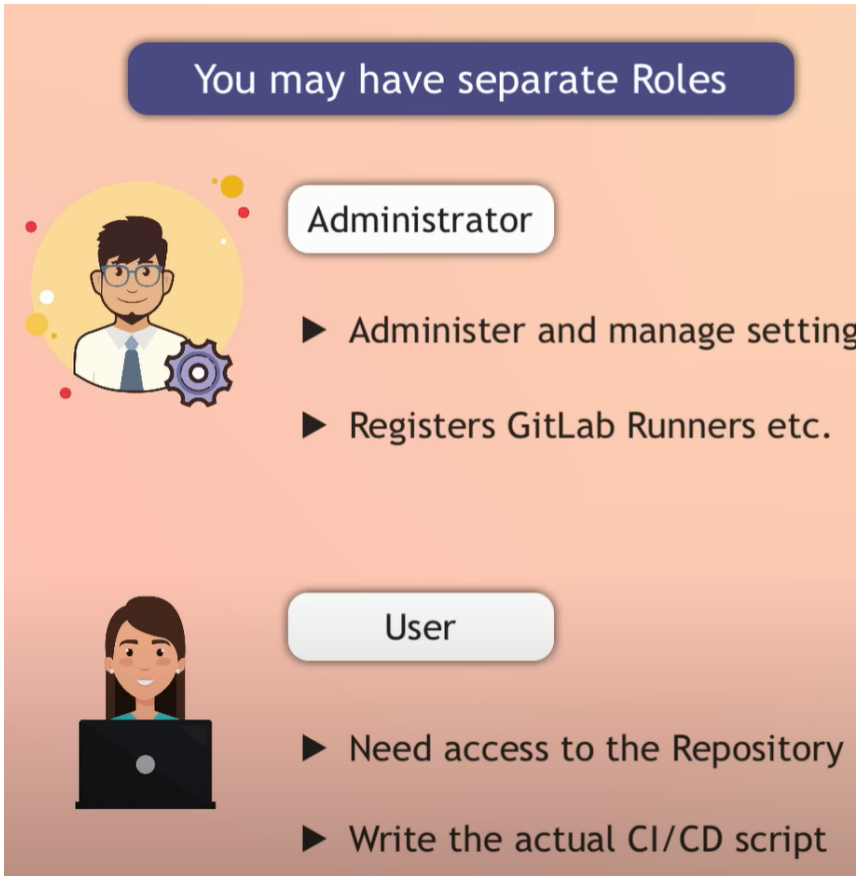
- Commands are executed inside a container
- ✅ Only Docker itself needs to be installed
- ✅ Each job runs in a separate & isolated container

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

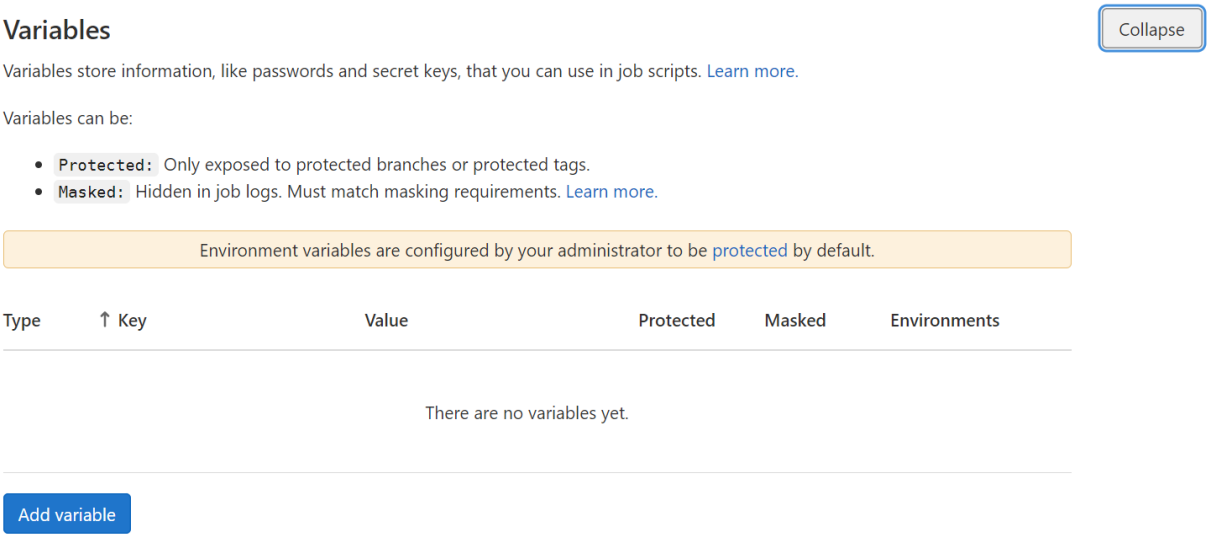


++Pipeline automatically runs after each commit with code change++





Settings => C/ICD : here you can add variables that can be used in the yaml file

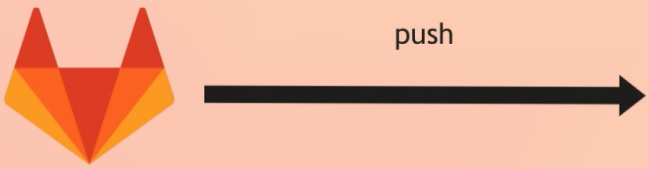


Masked variables

- 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"

Another Use Case of Variables

- Storing values you want to re-use multiple times. Reducing code duplication

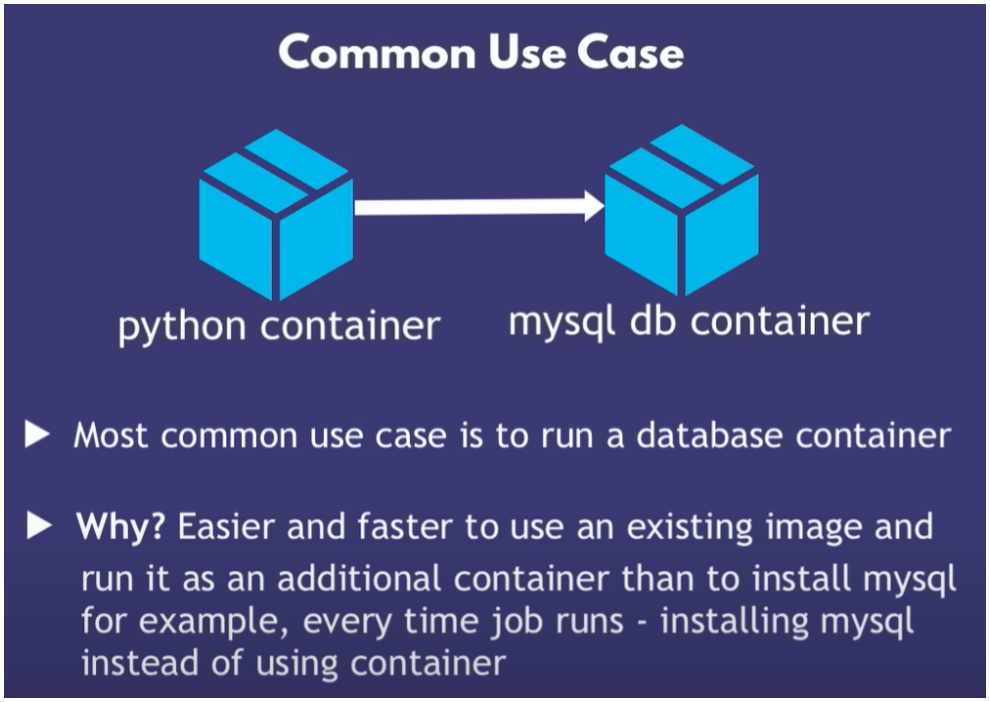


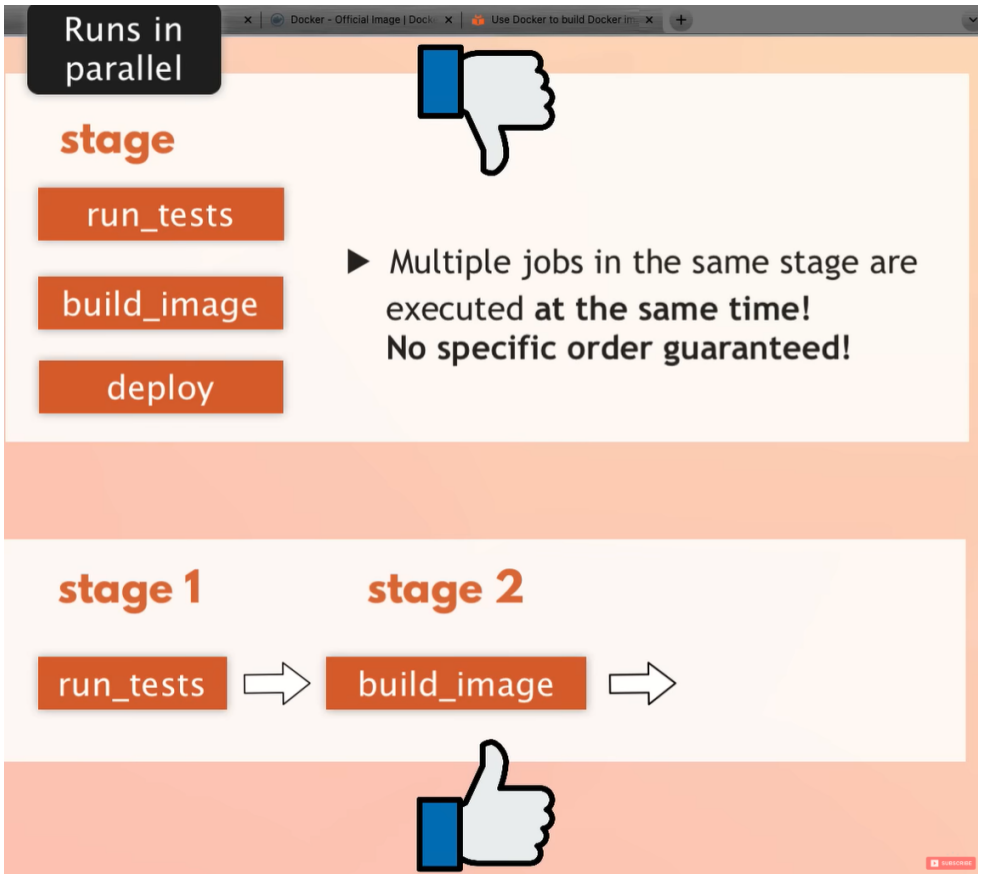
- docker build -t ....

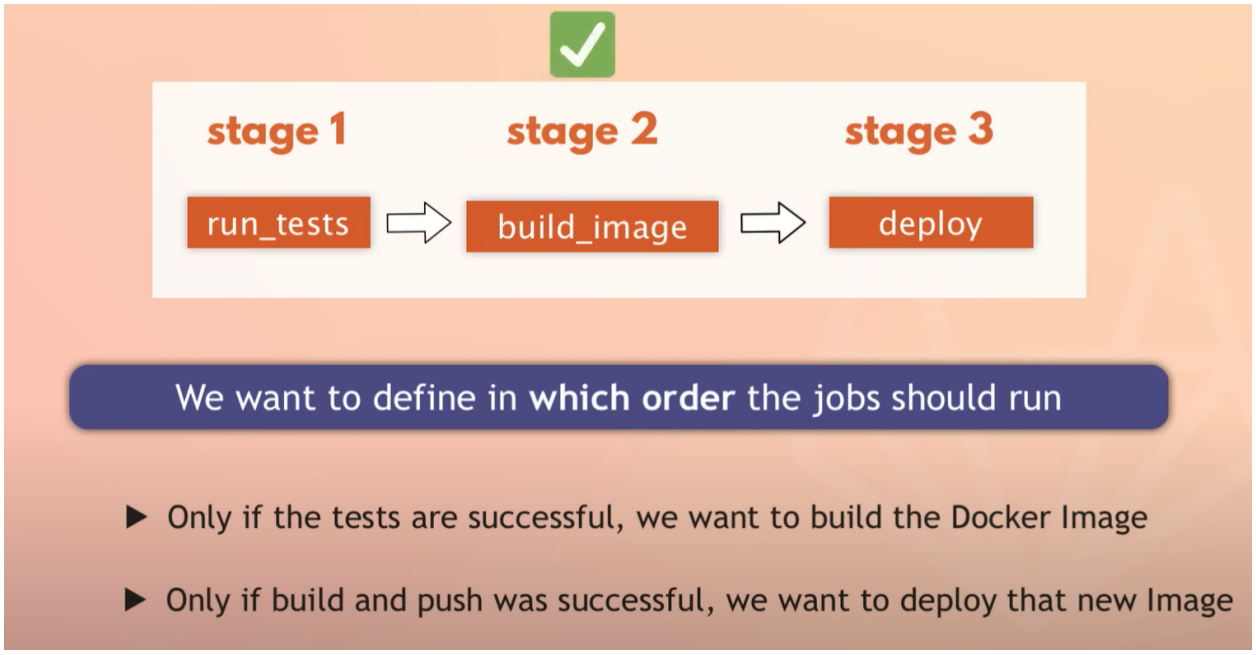
hub.docker.com/nanajanashia/demo-app

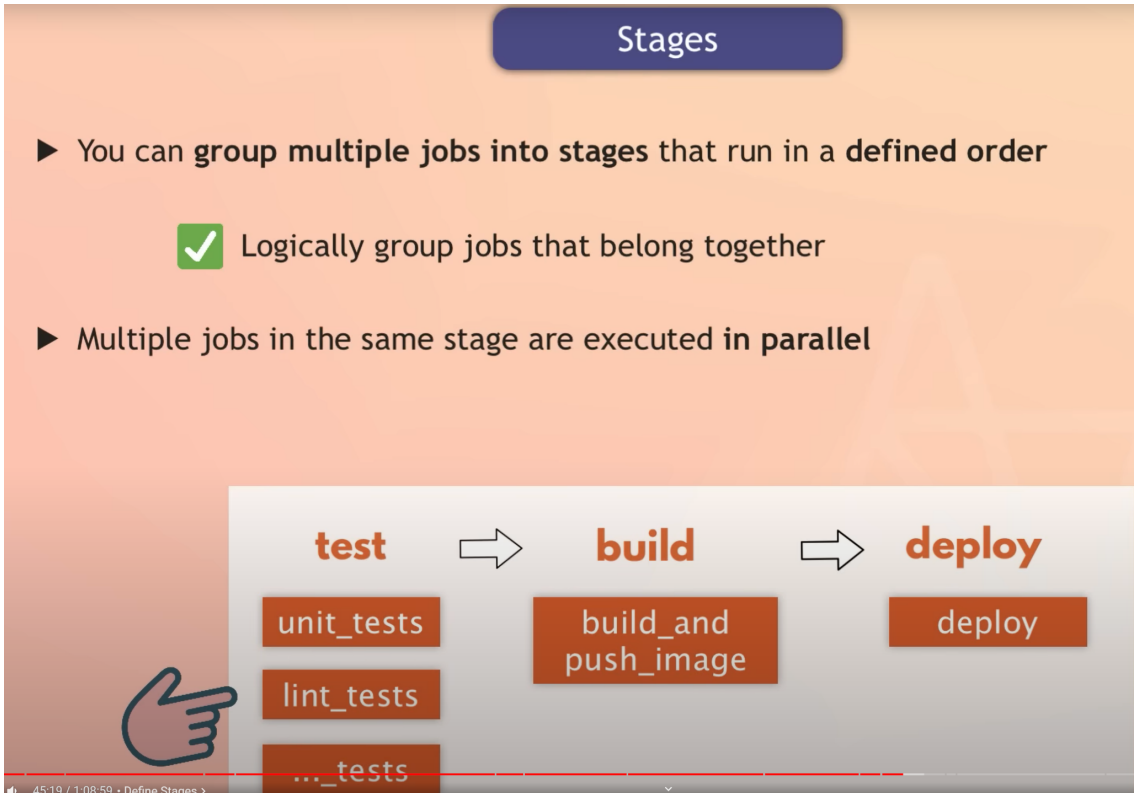
- The repository **location is included** in the **image name**
- **hub.docker.com is the default.** If you use another registry, you need to specify

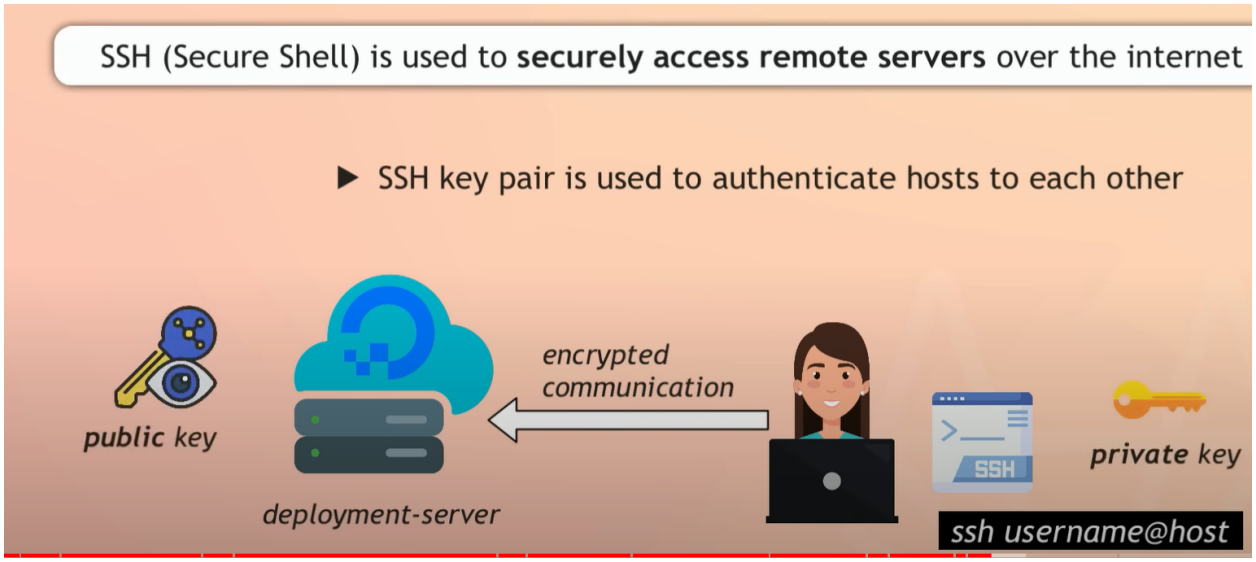
Services


Common Use Case
python container
mysql db container
Most common use case is to run a database container
Why? Easier and faster to use an existing image and run it as an additional container than to install mysql for example, every time job runs - installing mysql instead of using container

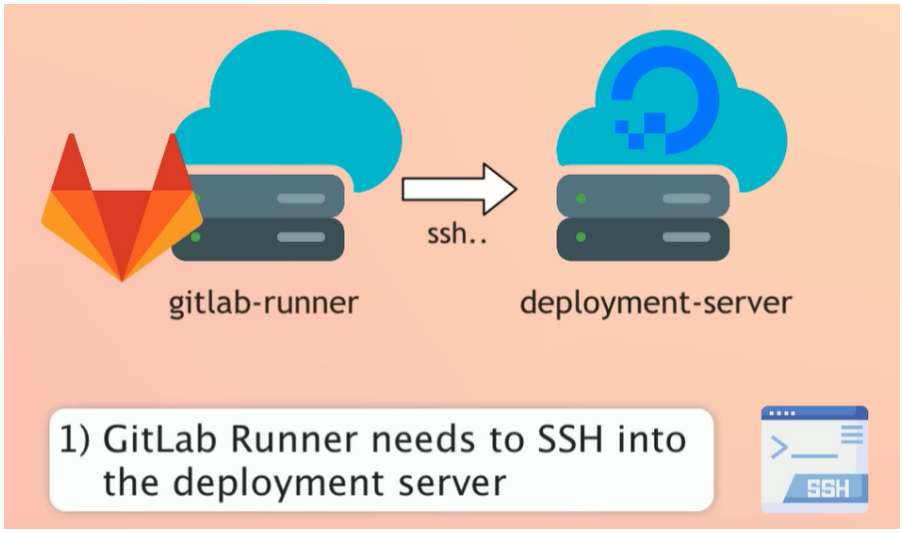

Runs in parallel
stage
run_tests
build_image
deploy
Multiple jobs in the same stage are executed at the same time!
No specific order guaranteed!
stage 1
stage 2
run_tests
build_image


stage 1
stage 2
stage 3
run_tests
build_image
deploy
We want to define in which order the jobs should run
Only if the tests are successful, we want to build the Docker Image
Only if build and push was successful, we want to deploy that new Image

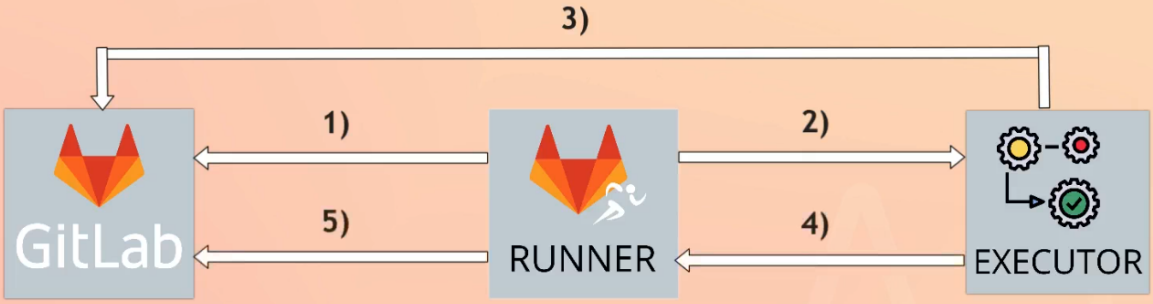

Stages
You can group multiple jobs into stages that run in a defined order
Logically group jobs that belong together
Multiple jobs in the same stage are executed in parallel
test
build
deploy
unit_tests
lint_tests
..._tests
build_and push_image
deploy



If you want to upload and run docker container on a server, server needs to contain the docker application



Execution Flow

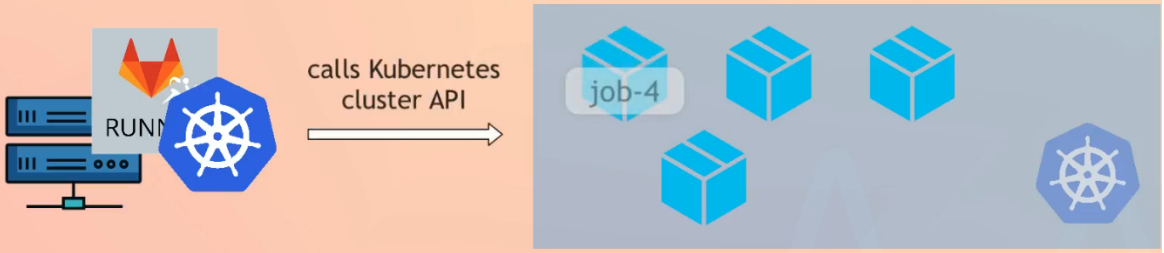


1) Runner requests new jobs from GitLab instance (e.g. GitLab.com)
2) Runner compiles and sends the job's payload to Executor
3) Executor clones sources or downloads artifacts from GitLab instance and executes the job
4) Executor returns job output and status to the Runner
5) Runner updates job output and status to GitLab instance

Docker Executor

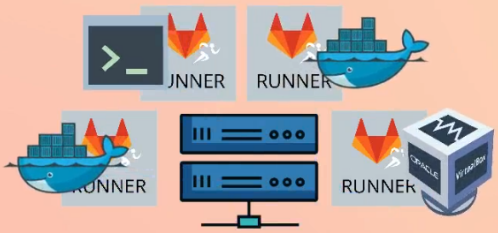
- Commands are executed inside a container
- Jobs run on user provided Docker images

Kubernetes Executor



- Allows you to use an existing Kubernetes cluster for your builds
  - ✅ Utilize your high availability setup
- Kubernetes Executor will create a new Pod for each GitLab job

How to configure the executor for the runner?

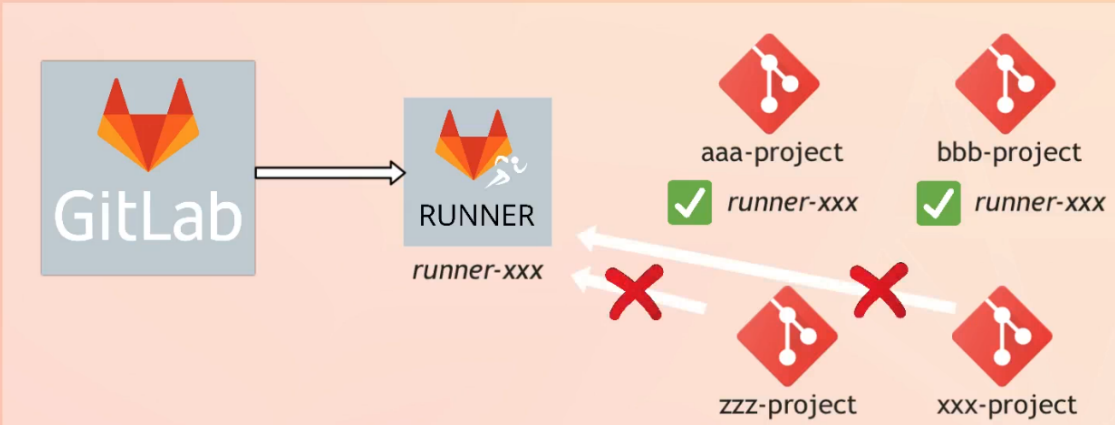


Multiple Executors on same server?

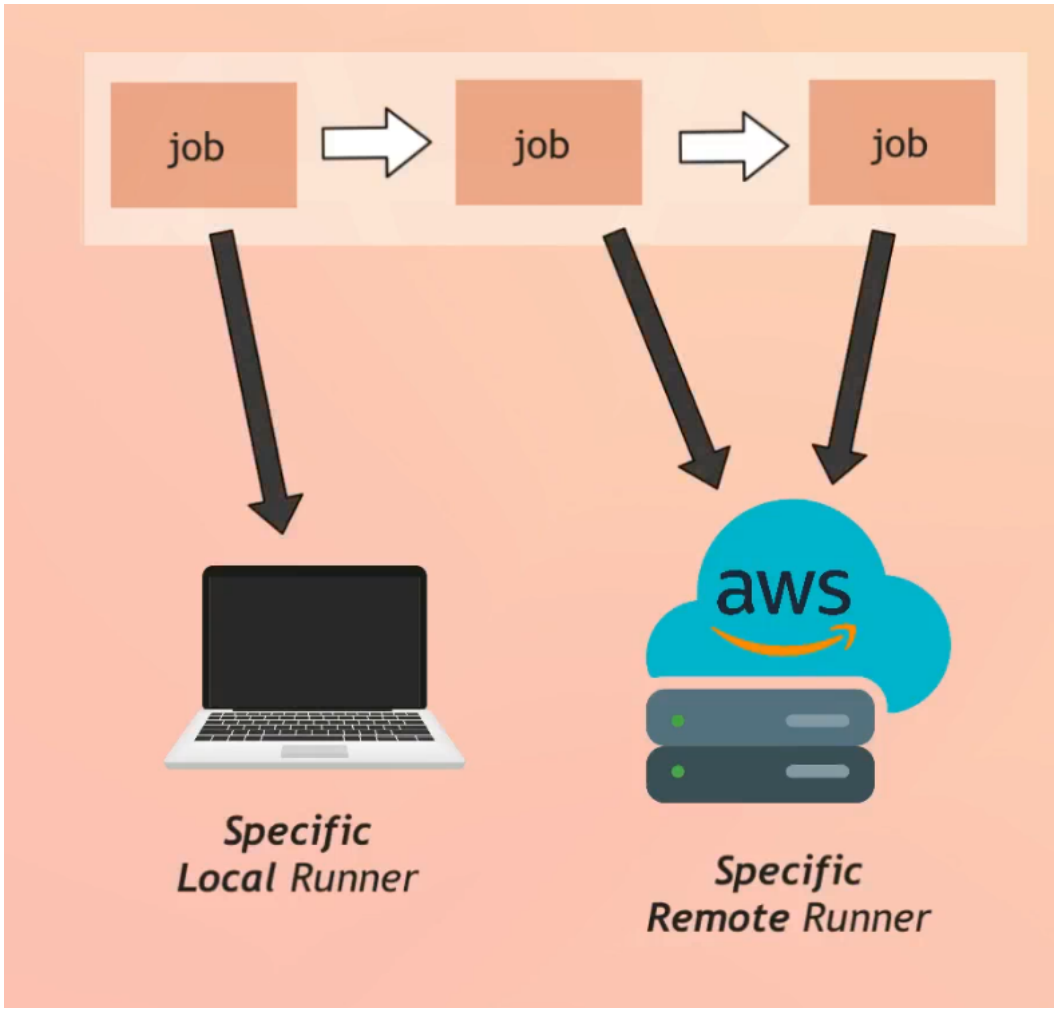
- ✅ Register multiple Runners on same host

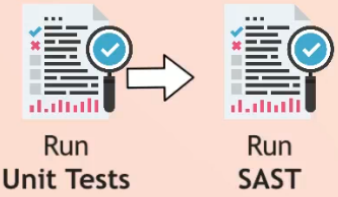
Specific Runners

- Specific Runners are **associated with specific projects**

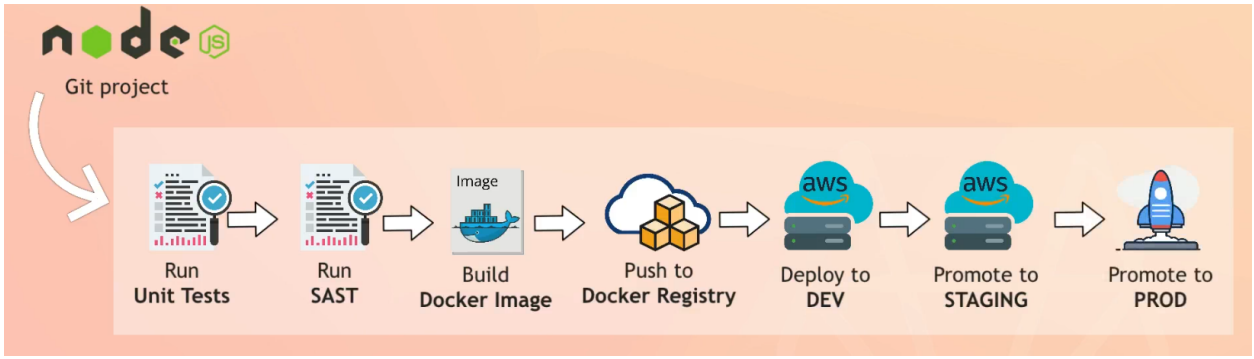


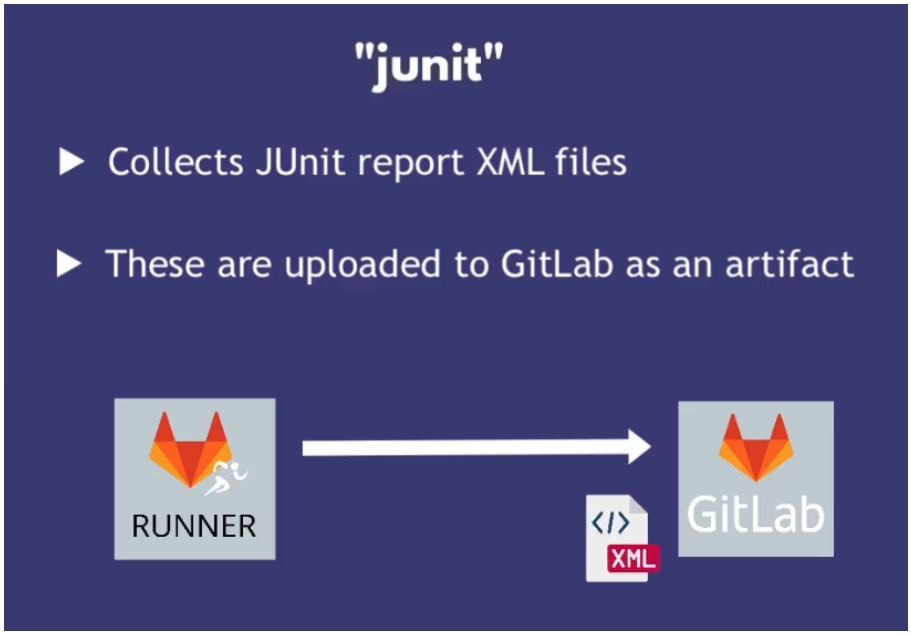
GitLab Installation



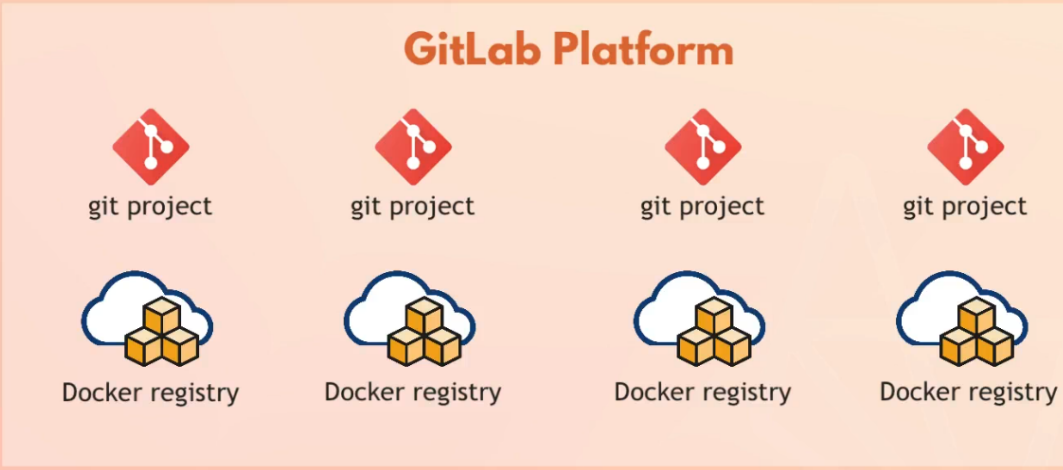


- **Static Application Security Testing (SAST)** =
  SAST analyzes the code itself without actually executing the code
- It scans your code for any security vulnerabilities

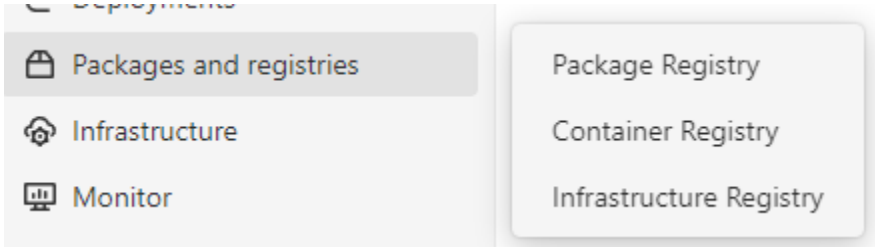

Git project
Run
Unit Tests
Run
SAST
Image
Build
Docker Image
Push to
Docker Registry
aws
Deploy to
DEV
aws
Promote to
STAGING
Promote to
PROD


"junit"
Collects JUnit report XML files
These are uploaded to GitLab as an artifact
RUNNER
XML
GitLab

GitLab Container Registry



- Every GitLab project can have its **own space to store its Docker images**



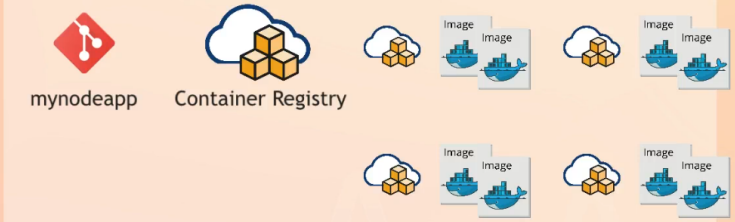
Packages & Registries

- **Package Registry** = Use GitLab as a private or public registry for variety of supported package managers
- **Container Registry** = Registry to store Docker images

Registry URL

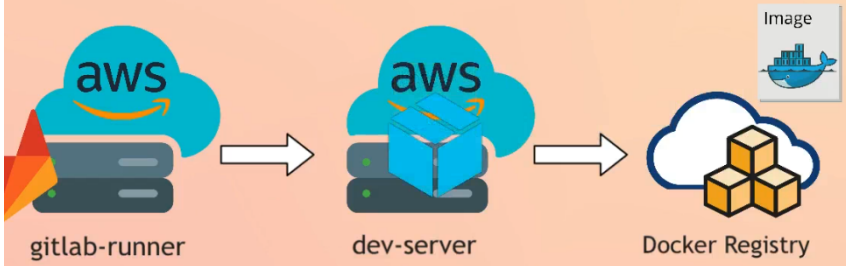
registry.gitlab.com /username /projectname

- Your container registry can have **multiple image repositories**
- Each image repository can store multiple image versions (tags)



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"



1) SSH into the DEV server

2) docker login ...

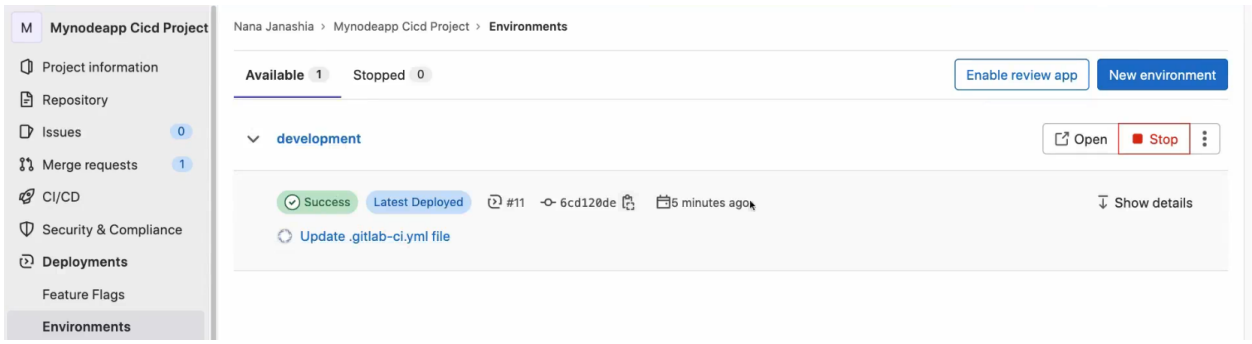
- Login to Container Registry from dev-sever

3) docker run ...

- Pull and run image as a Docker container

"environment"

- If no environment with that name exists, a new one is created

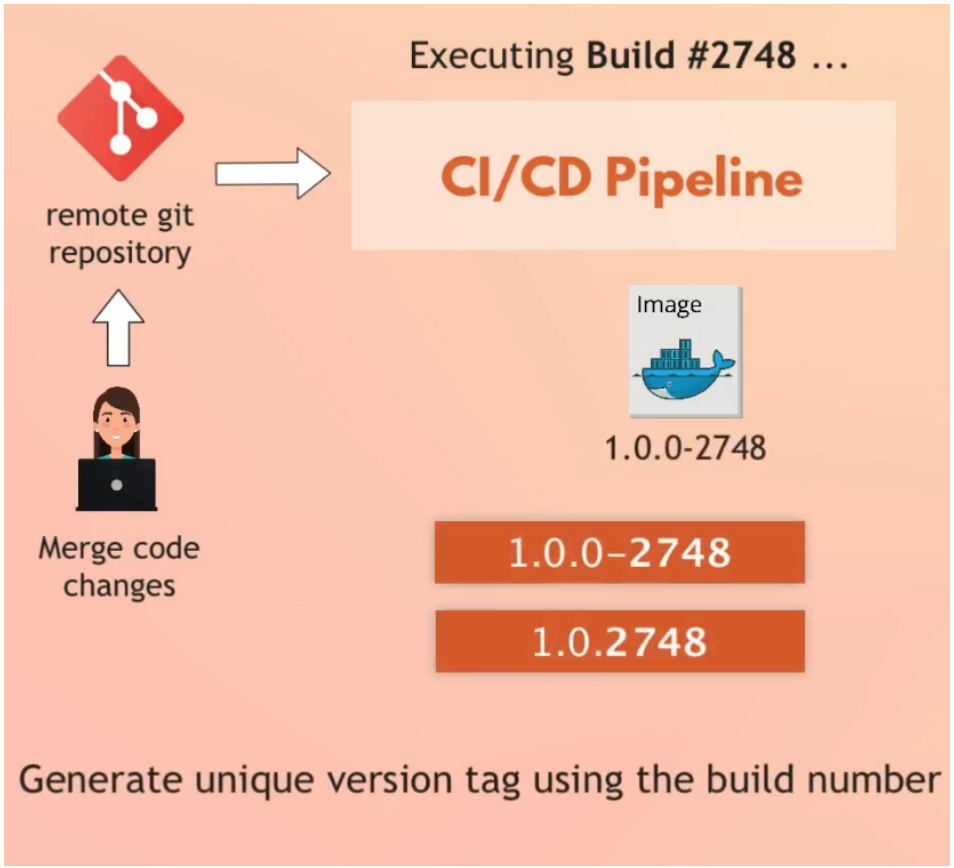


Deployment Environments

- Gitlab provides a full history of deployments of each environment
- You always know what is deployed on your servers

jq

- A command-line tool for parsing JSON



- By default: **jobs in later stages automatically download** all the artifacts created by jobs in earlier stages

What is a Dotenv File

- Dotenv is a lightweight npm package that automatically loads environment variables from a .env file into the process

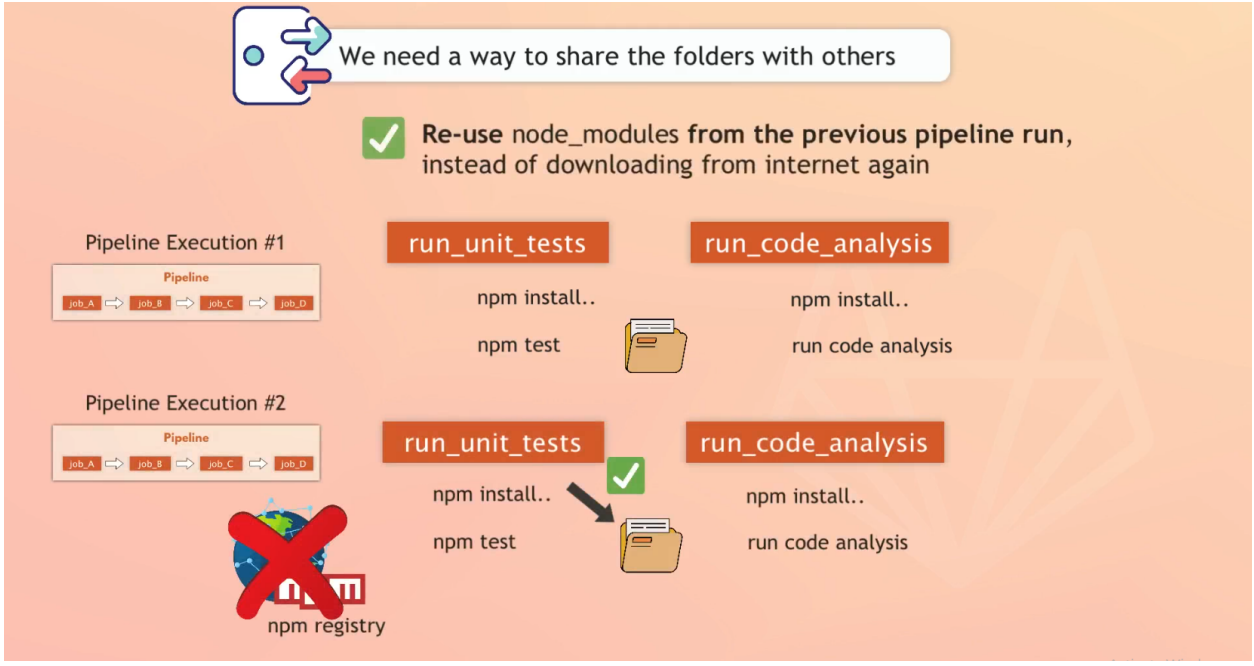
Dotenv Format

- One variable definition per line
- Each line must be of the form:

```
VARIABLE_NAME=ANY_VALUE
```

Dotenv artifact

- Save the *.env* file as an dotenv artifact
- Dotenv report collects the environment variables as artifacts



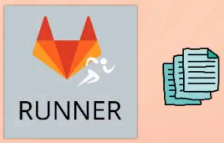
Artifacts vs Cache

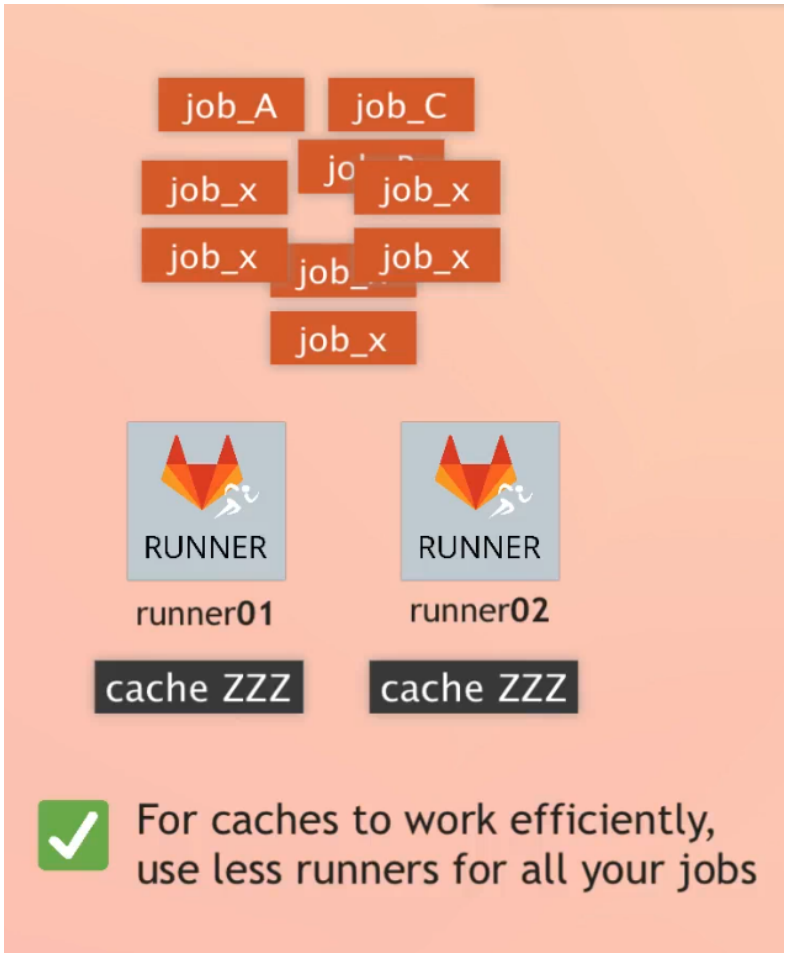
Artifacts

- Job artifacts get **uploaded** and saved on the GitLab server
- Use artifacts to **pass intermediate build results** between stages

Cache

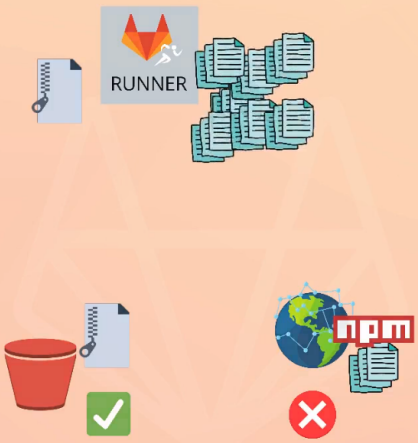
- Use cache for **dependencies**, like packages you **download from the internet**
- Cache is stored on the GitLab Runner!





Distributed Cache

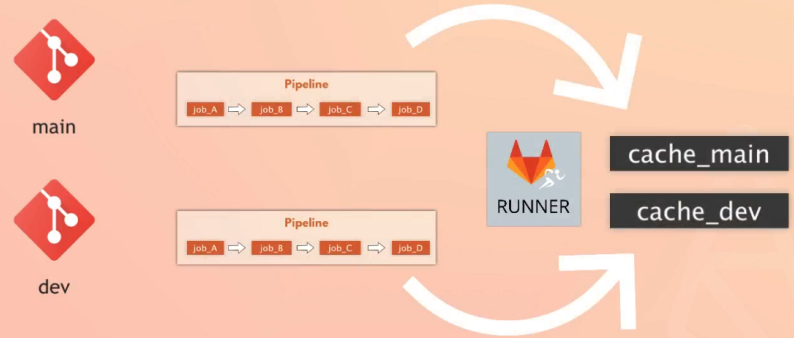
- **Central** cache storage
- Download from remote storage again over the internet
- Still faster to download 1 zip file, instead of each dependency separately



Cache Key

- Give each cache a **unique idenfying key**
- If not set, the default key is "***default***"
- All jobs that use the same cache key use the same cache

Common Naming of Cache Keys



- Use **branch name as the cache key**

Cache Path

- Used to choose which files or directories to cache
- You can use an array of paths relative to the project directory

Cache Policies

pull-push

- Job downloads the cache when the job starts & uploads changes to the cache when the job ends

Cache Policies

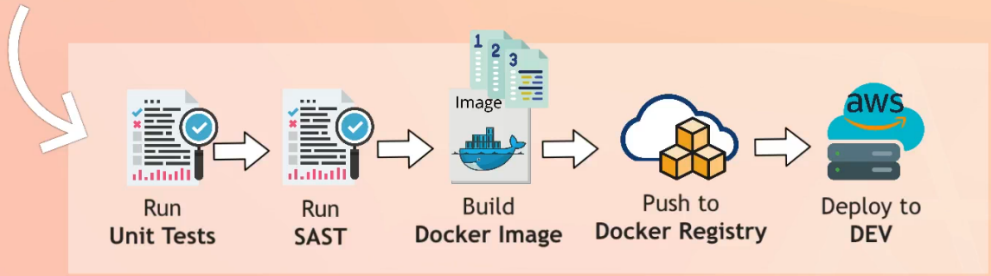
pull

- To only download the cache, but never upload changes
- Use when you have many jobs executing in parallel that use the same cache

✅ Speeds up job execution

✅ Reduces load on the cache server

Include SAST job with job template



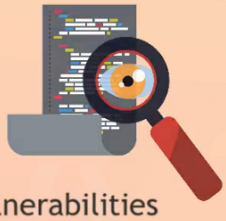
.gitlab-ci.yml

sast.gitlab-ci.yml

Job Templates

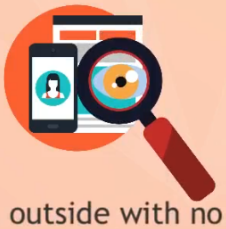
- Can be added to your existing CI/CD workflow
- These are specific jobs provided by GitLab, which you can include

Security Related Tests



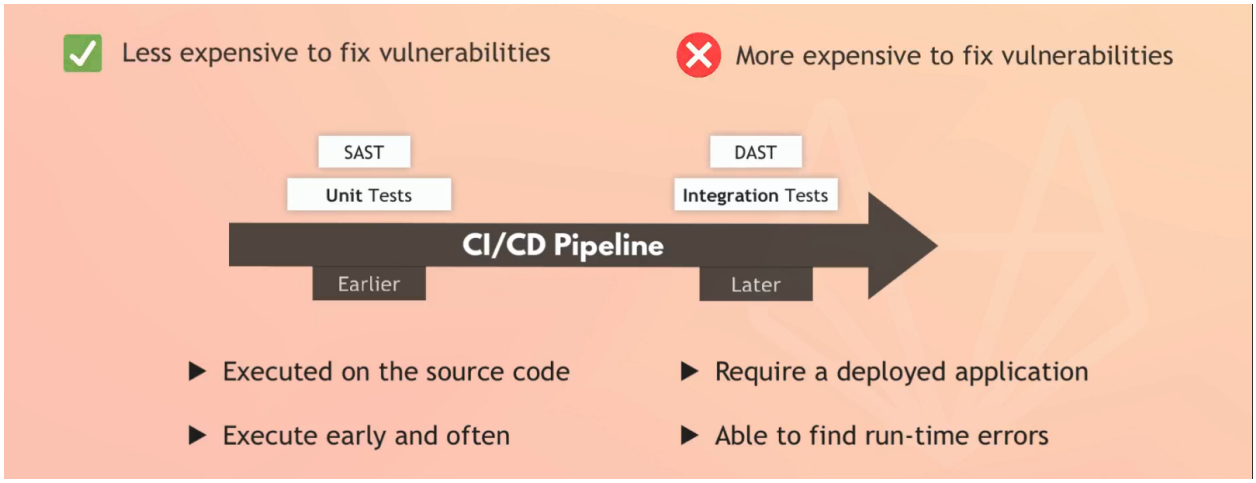
SAST - Static application security testing

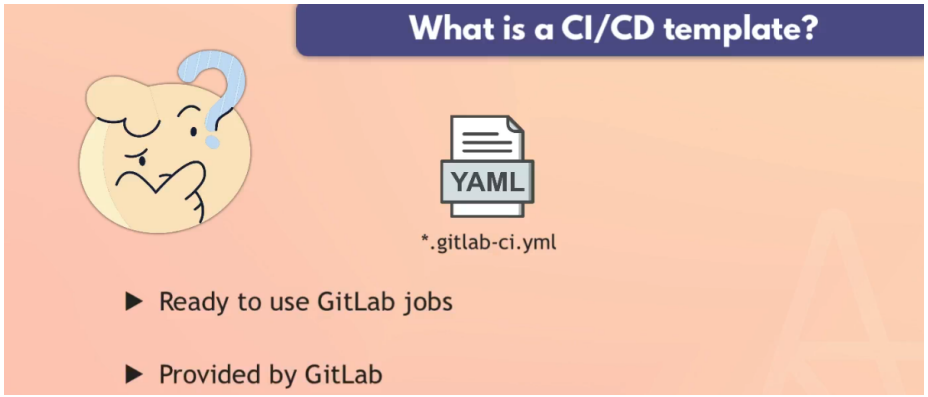
- **Analyzes the source code** to identify any vulnerabilities
- Examples: SQL injection, Cross-site scripting, buffer overflows, ...

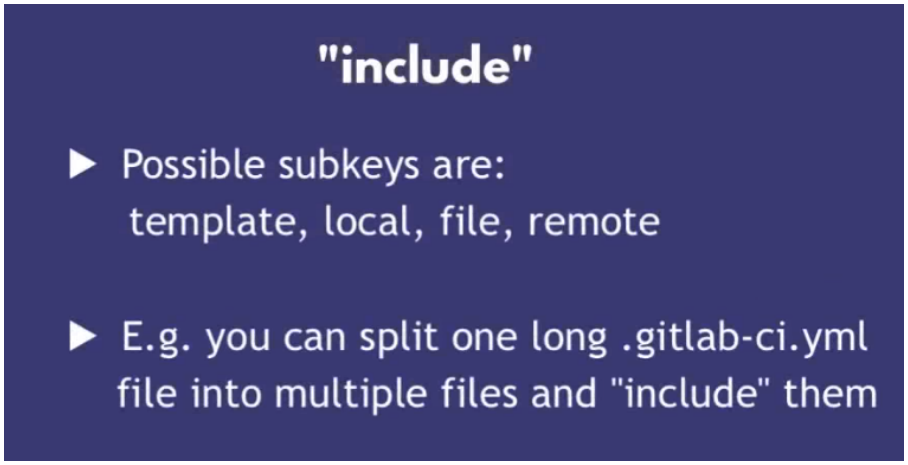


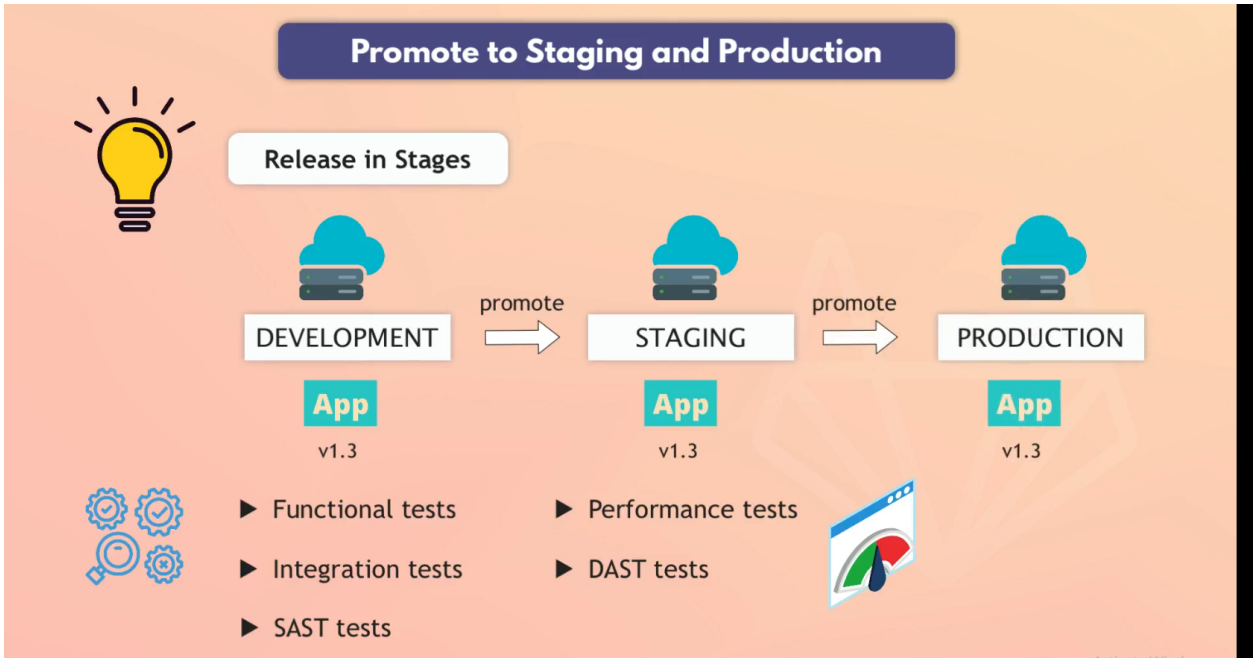
DAST - Dynamic application security testing

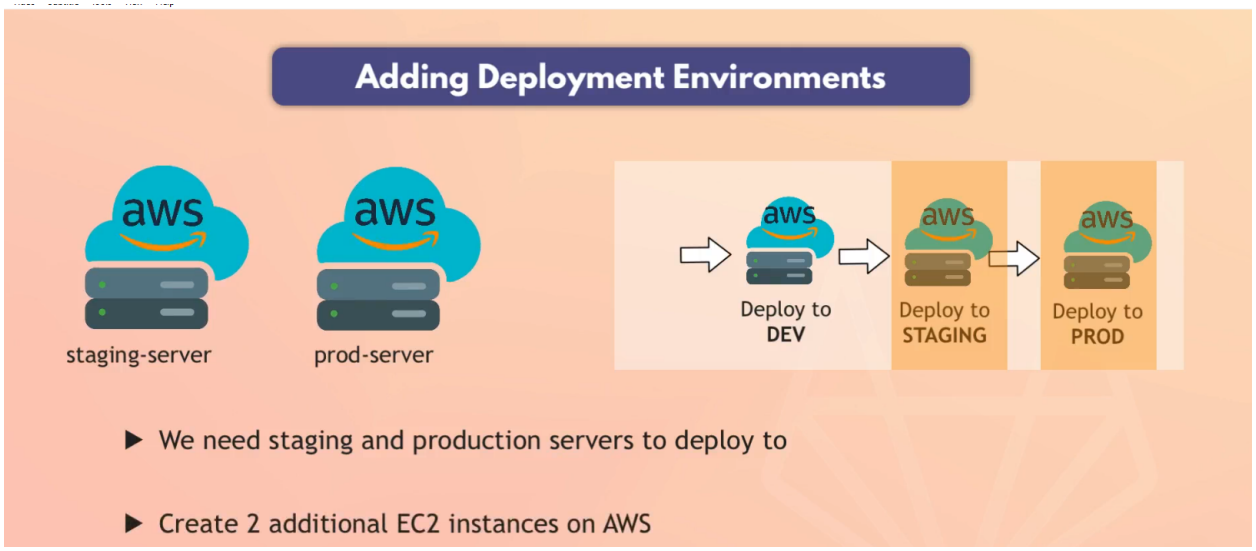
- It analyzes the **running application**
- Represents the hacker approach, testing from outside with no knowledge of the technologies used


Less expensive to fix vulnerabilities
More expensive to fix vulnerabilities
SAST
Unit Tests
DAST
Integration Tests
CI/CD Pipeline
Earlier
Later
Executed on the source code
Execute early and often
Require a deployed application
Able to find run-time errors

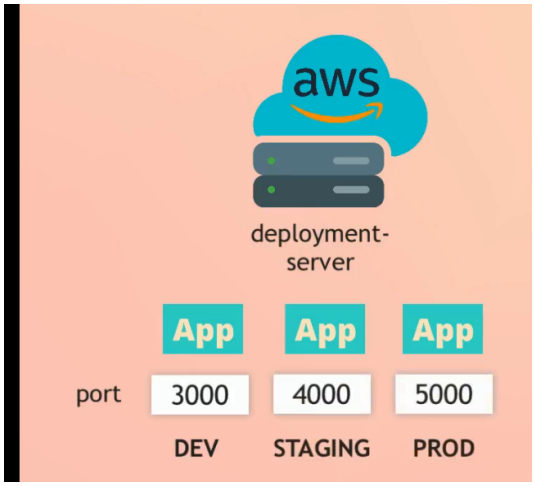

What is a CI/CD template?
YAML
*.gitlab-ci.yml
Ready to use GitLab jobs
Provided by GitLab


"include"
Possible subkeys are:
template, local, file, remote
E.g. you can split one long .gitlab-ci.yml file into multiple files and "include" them





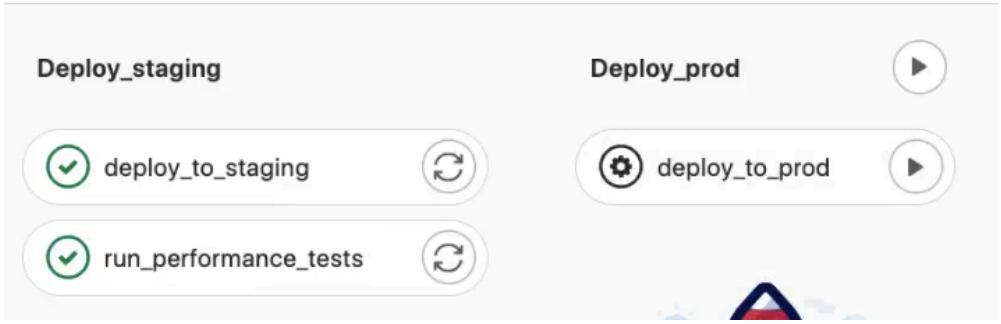
To save money, you can also run 3 instances of the app on one EC2 instance, but with different ports



Then you can use the hidden job (.job_name) as the template to extend on other jobs

when : manual

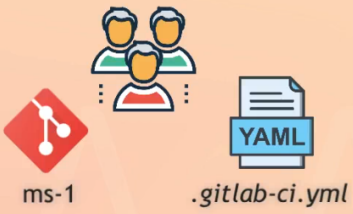
- This job doesn't run, unless a user starts it manually
- Very common for production deployments

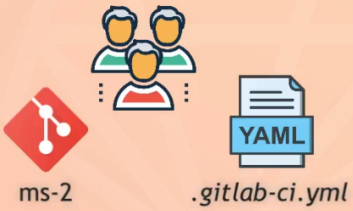


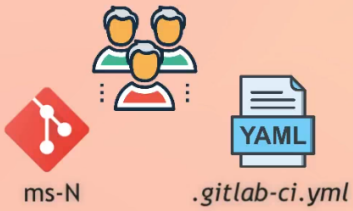
Why Job Templates?

Each team needs to write and maintain own file

- Pipeline configurations may grow apart
- Instead of having a common configuration that all microservices can use

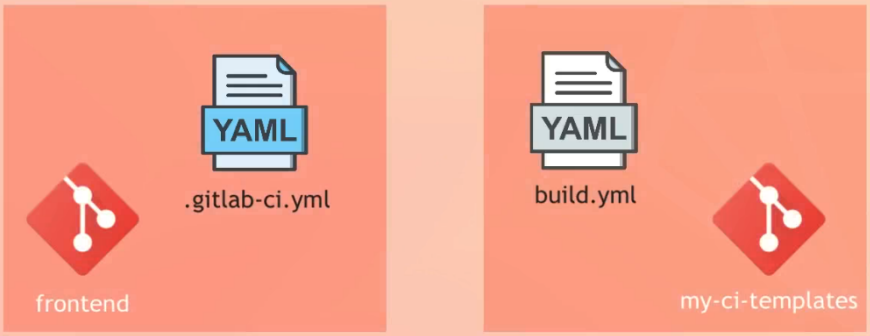






Job Templates

- You can extract common configuration
- Make it generic, so no hardcoded values specific to that project



"include:file" & "include:project"

- To include files from another private project on the same GitLab instance

include

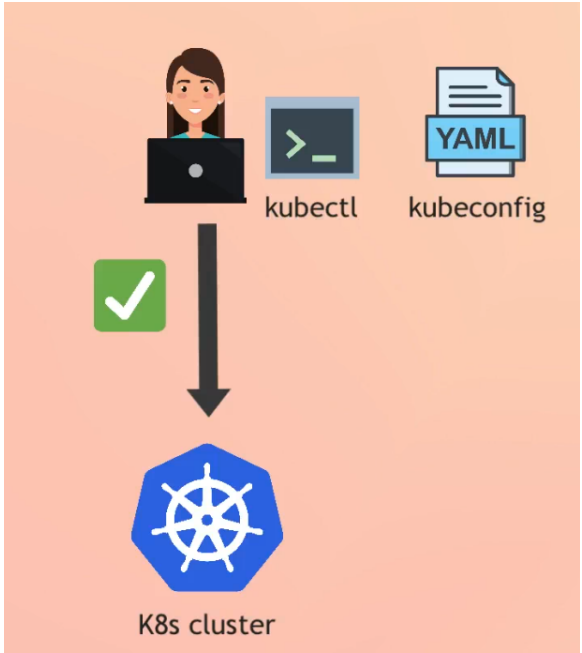
- To include external YAML files

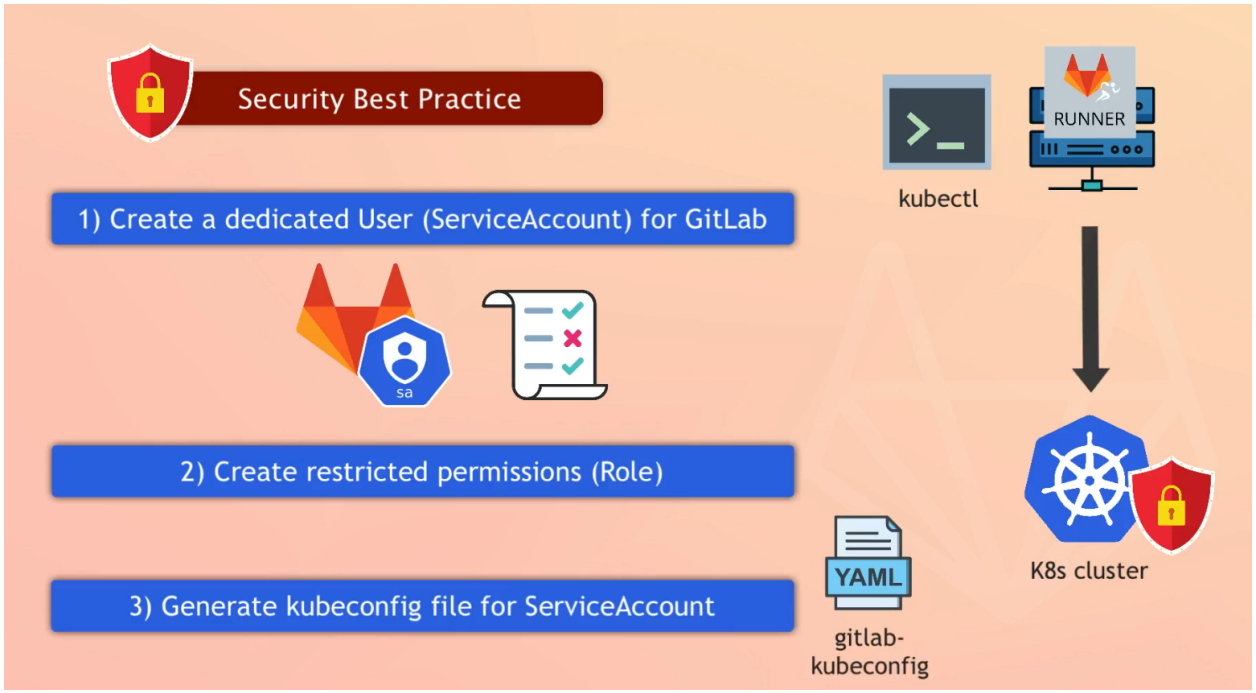
- local	▶ Reference from same repository
- file	▶ Reference from another private project (same GitLab instance)
- remote	▶ Include from a different location (full URL necessary)
- template	▶ Include GitLab's templates

```yaml
include:
  - project: mymicroservice-cicd/ci-templates
    ref: main
    file:
      - build.yml
      - deploy.yml
  - remote: 'https://gitlab.com/awesome-project/raw/master/.before-script-template.yml'
  - template: Auto-DevOps.gitlab-ci.yml
  - local: .install.yml
  - local: .install.yml
```

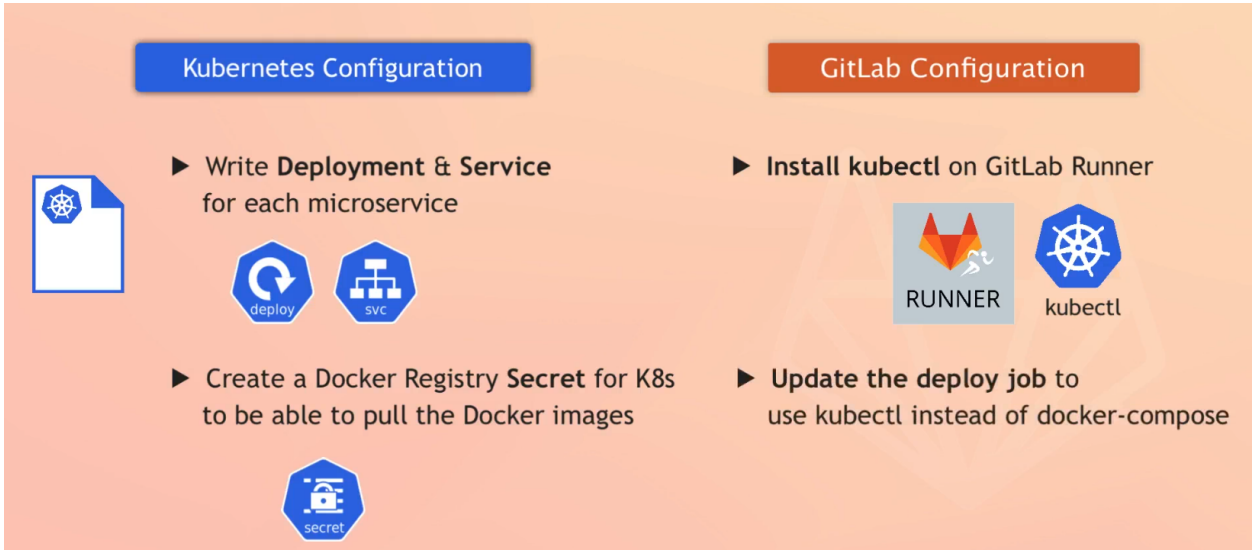
ServiceAccount =
Represent non-human user

- Permissions to define what the ServiceAccount is allowed to do inside the cluster
- Admin user wide privileges

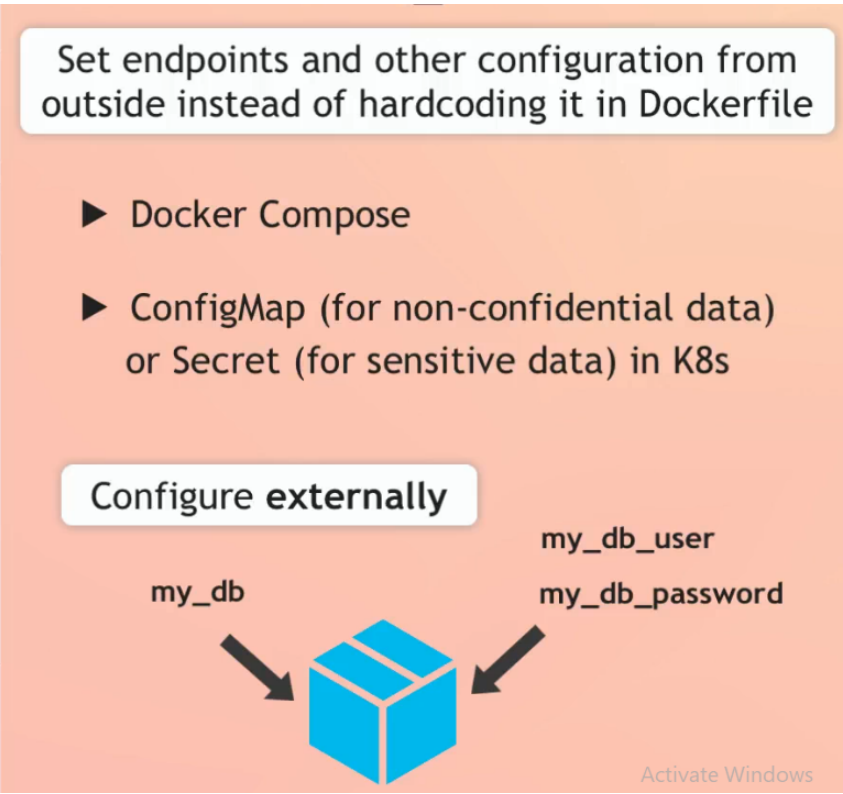


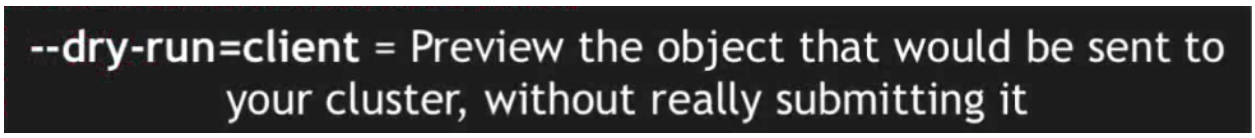

Security Best Practice
1) Create a dedicated User (ServiceAccount) for GitLab
sa
2) Create restricted permissions (Role)
3) Generate kubeconfig file for ServiceAccount
kubectl
RUNNER
K8s cluster
YAML
gitlab-
kubeconfig


[\W]$ kubectl create rolebinding cicd-rb
Role
ServiceAccount
Namespace


Kubernetes Configuration
GitLab Configuration
Write Deployment & Service for each microservice
deploy
svc
Create a Docker Registry Secret for K8s to be able to pull the Docker images
secret
Install kubectl on GitLab Runner
RUNNER
kubectl
Update the deploy job to use kubectl instead of docker-compose

In case you have many different services and don't want to have same file in all of them:






```
# we can also define variables inside a yaml file (for non-sensitive data) or
make them in cicd setting
variables:
  IMAGE_NAME: credit_suisse_repo
```

```
  IMAGE_TAG: python-app-1.0

stages:
  - test
  - build
  - deploy

# this is name of the job => run_tests => it will run the tests against the
project code
# each job runs inside a docker container
run_tests:
  stage: test
  # we need to test the python project, so we use a python image for it
  # don't put latest image, since it might get updated and not be compatible
with current modules
  image: python:3.9-slim-buster
  # run this before running the script
  before_script:
    - apt-get update && apt-get install make
  # each job MUST have a script
  script:
    - make test

# this job will create the docker image (or any other artifact based on the
project code)
build_image:
  stage: build
  # since we want to make a docker image, we need a environment that contains
docker inside it
  # docker in docker :)
  image: docker:20.10.16 # docker client
  # extra container inside the job
  services:
    - docker:20.10.16-dind # docker daemon
  variables:
    # we need this certificate so both containers can use it to communicate
    DOCKER_TLS_CERTDIR: "/certs"
  # need to login before pushing the image
  before_script:
    - docker login -u $REGISTERY_USER -p $REGISTERY_PASS
  # create the image and then push it to dockerhub
  script:
    - docker build -t $IMAGE_NAME:$IMAGE_TAG .
    - docker push $IMAGE_NAME:$IMAGE_TAG
```

```
# deployment runs after successful build, to upload the created artifact/image
into the server
deploy:
  stage: deploy
  # we have saved ssh public key as a file in variables, here we set it to be
visible to all users
  before_script:
    - chmod 400 $SSH_KEY
  # login to the server host through ssh key,
  # when inside the host run the commands inside ""
      # login to dockerHub using our user/pass
      # stop and remove any previous containers if they exist and clear port
      # run docker image on port 5000 in detached mode
  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"
```