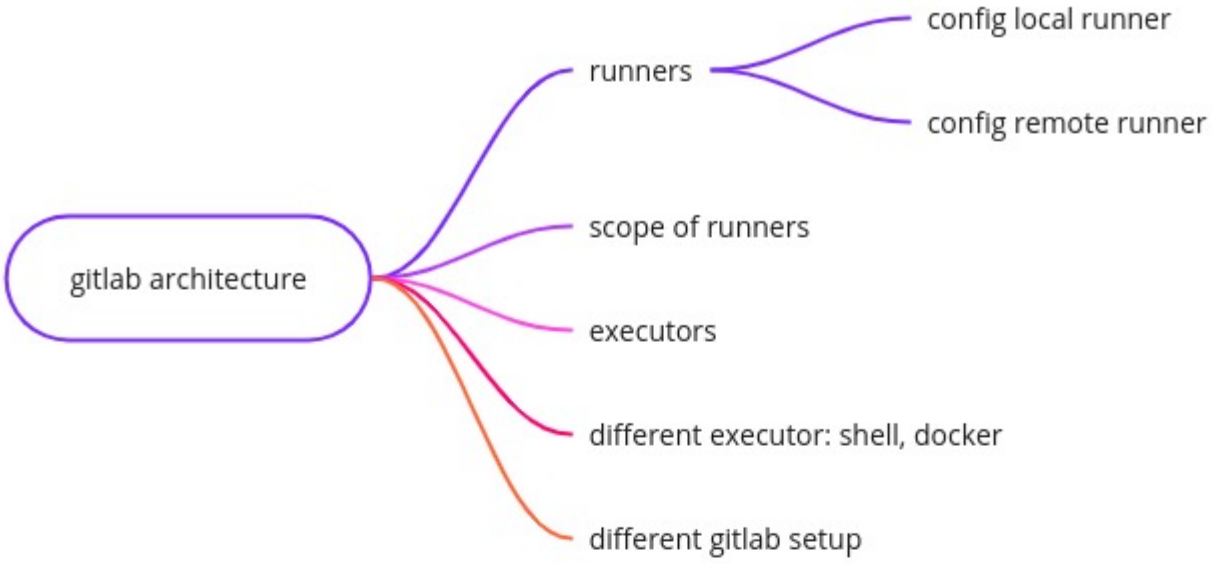
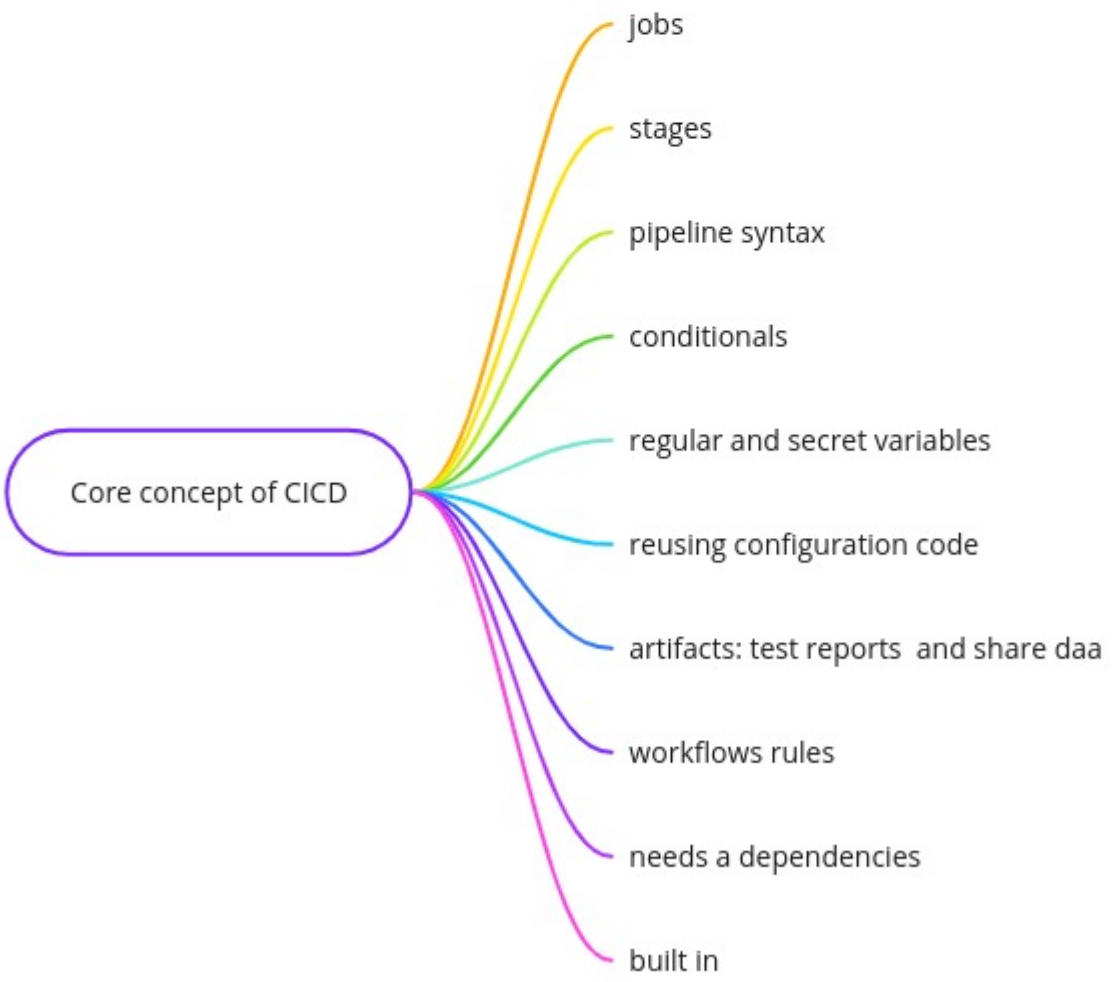
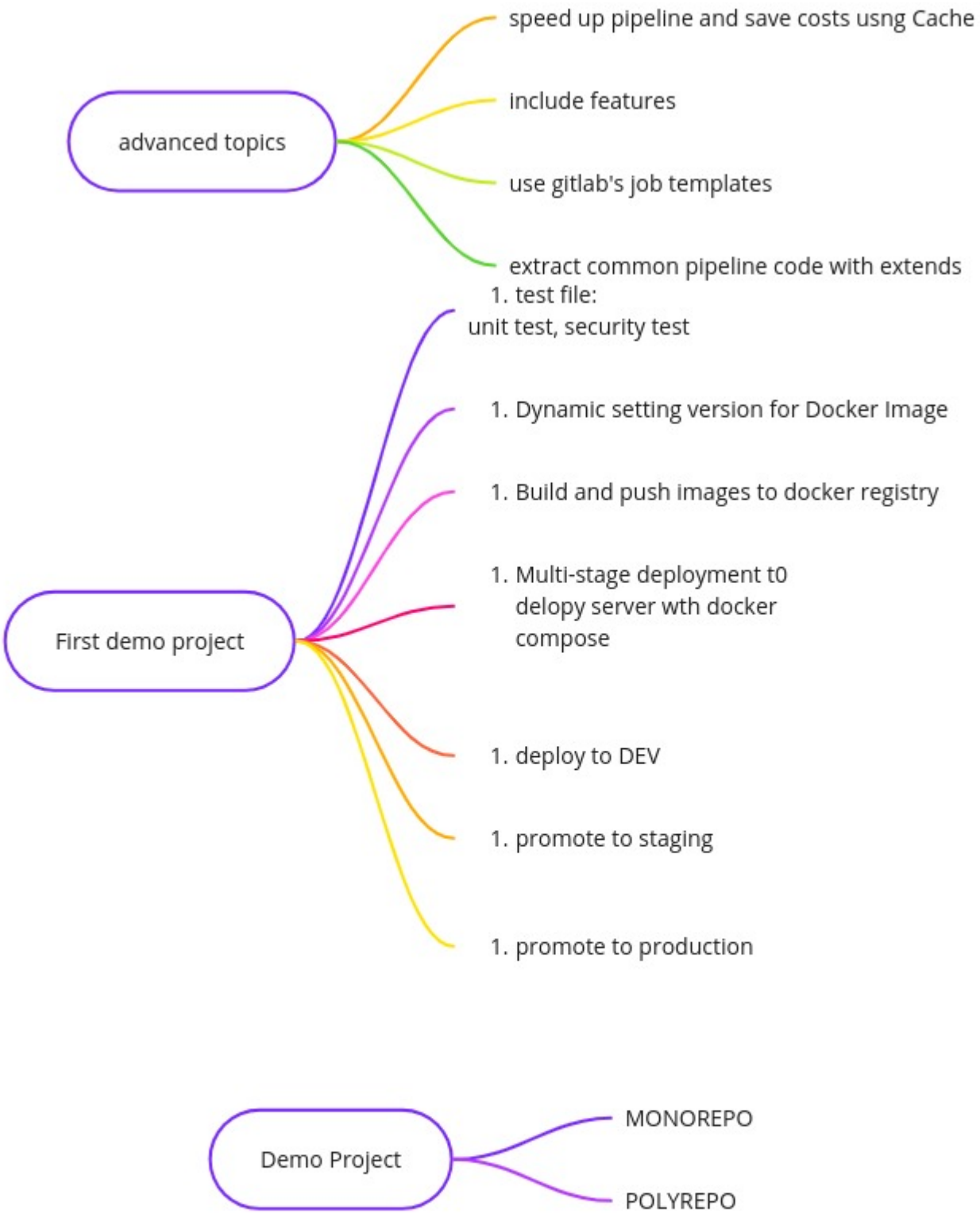
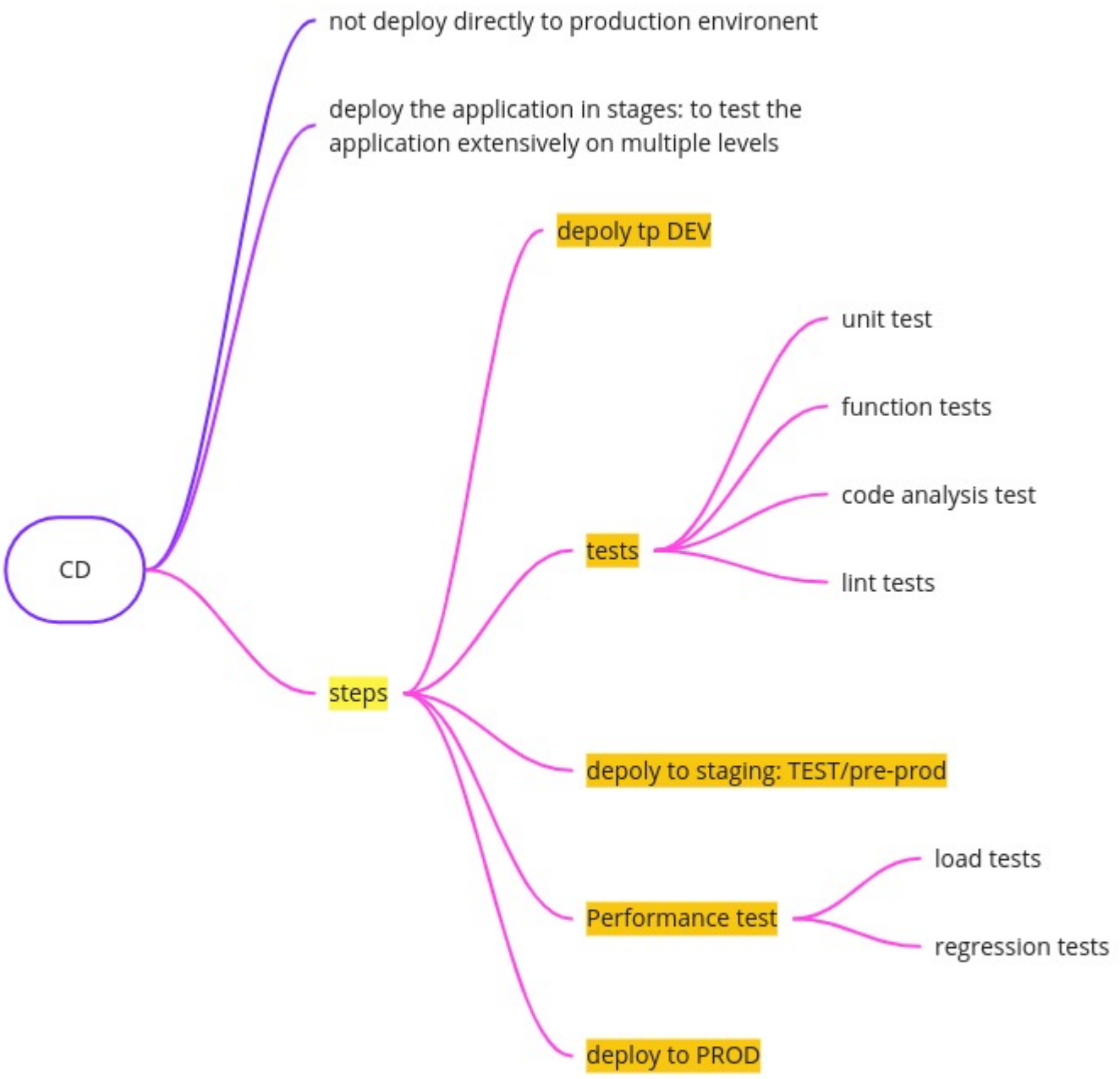

gitlab architecture
runners
config local runner
config remote runner
scope of runners
executors
different executor: shell, docker
different gitlab setup


Core concept of CICD
jobs
stages
pipeline syntax
conditionals
regular and secret variables
reusing configuration code
artifacts: test reports and share daa
workflows rules
needs a dependencies
built in

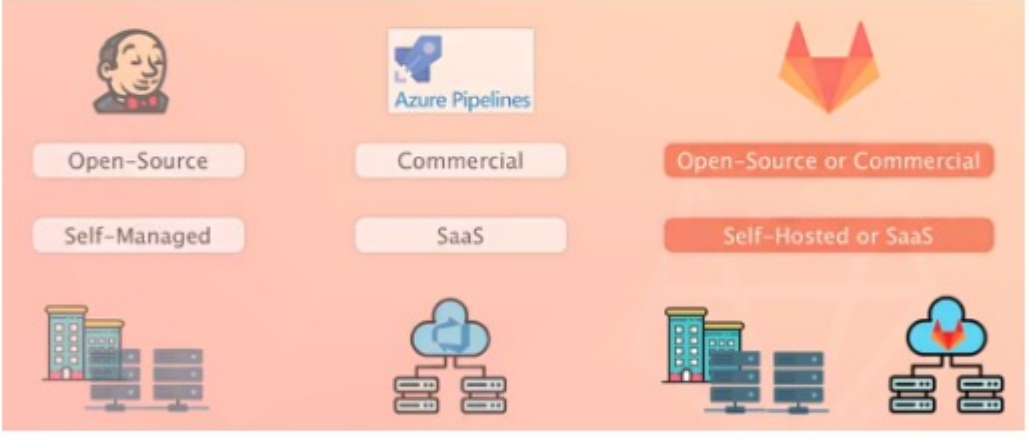

advanced topics
speed up pipeline and save costs usng Cache
include features
use gitlab's job templates
extract common pipeline code with extends
First demo project
1. test file:
unit test, security test
1. Dynamic setting version for Docker Image
1. Build and push images to docker registry
1. Multi-stage deployment t0
delopy server wth docker
compose
1. deploy to DEV
1. promote to staging
1. promote to production
Demo Project
MONOREPO
POLYREPO

CI/CD: frequently deliever changes to custom in an automated way



Gitlab CI/CD platform:

- built-in CI/CD to streamline testing and delievery
- Built-in features:
    - built-in security capailities
    - built-in package registry
    - integrated CD solution
-



Pipeline is scripted:

- infrastructure as code
- create .gitlab-ci.yml file, which contains the whole CI/CD pipeline configuration

Jobs: are the fundamental building block of a .gitlab-ci.yml file.

- what to do
- define arbitray names for your jobs
- must contain at least the script clause
- script specify the commands to execute

.gitlab-ci.yml

home > gu > Documents > DevOps > gitlabCICD > .gitlab-ci.yml

```yaml
run_tests:
# prepare env datasets
  before_script:
    - echo "Preparing tet data..."
  script:
  # script needs a list, using hypen
    - echo "Running tests...."
  after_script:
    - echo "Clean up temporary files..."


build_image:
  script:
    - echo "Building docker image...."
    - echo "Tagging the docker image"


push_image:
  script:
    - echo "Logging into docker registry...."
    - echo " Push  docker image to registry..."
```

- prepare the yaml file
- put the file in the project

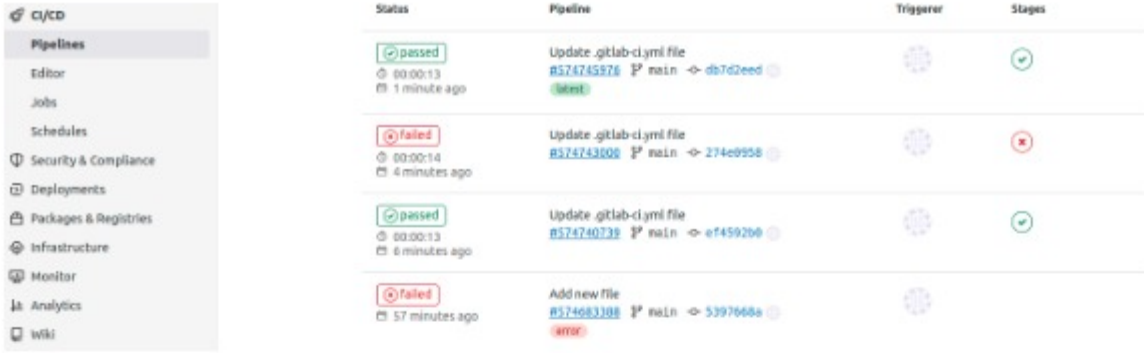
! pipeline logic becomes part of application code: infrastuture as code

- Gitlab detect the .yml file and build.
- In the section CI/CD:
  - pipeline: where your code
  - editor: if you want to change commands in the .yml file
  - jobs: the job names. the workflow of jobs

On every commit, GitLab triggers the pipeline automatically

Stages:

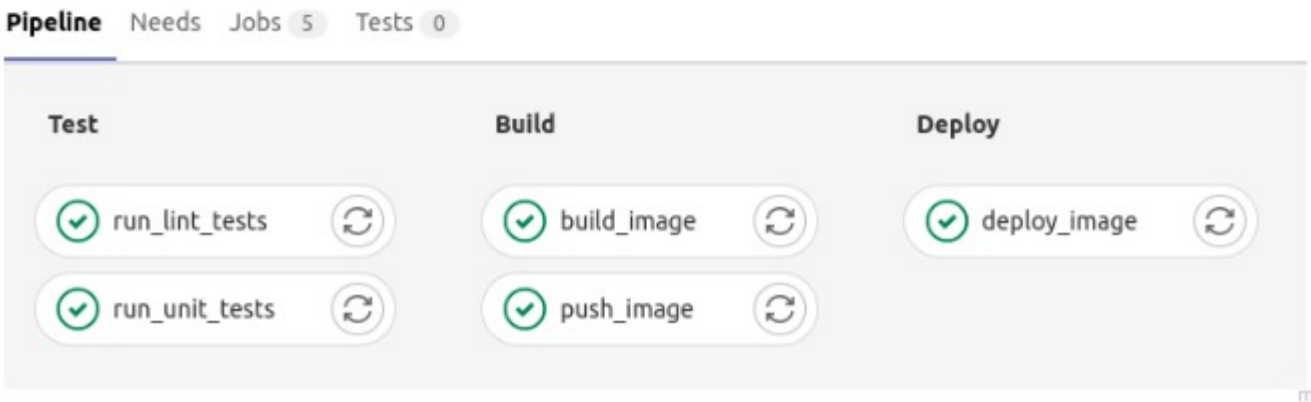
- You can group multiple jobs into stages that run a defined order
- jobs should run in order
- Only if all jobs in a stage succeed, the pipeline moves on to the next stage
- If any job in a stage fails, the next stage is not executed and the pipeline ends
- logically group jobs that belong together
- only wehen all jobs are successful, next stage will be start



In a stage, if there is a two jobs, iif the first one failed, the second one should also stop.

Use needs to specific the dependencies:

```yaml
run_unit_tests:
  stage: test
# prepare env datasets
  before_script:
    - echo "Preparing tet data..."
  script:
  # script neCeds a list, using hypen
    - echo "Running tests...."
  after_script:
    - echo "Clean up temporary files..."

run_lint_tests:
  stage: test
  needs:
    - run_unit_tests
```



If there are too much commands, you lose your overview, what you can do is extract the commands in a bash file. Then you have a logic in a seperated file

prepare-tests.sh 26 bytes

```
pwd
ls
mkdir test-data
ls
```

Each pipeline is run on a fresh environment

```yaml
stages:
  - test
  - build
  - deploy

run_unit_tests:
  stage: test
# prepare env datasets
  before_script:
    - echo "Preparing tet data..."
  script:
  # script neCeds a list, using hypen
    - echo "Running tests...."
    - chmod +x prepare-tests.sh
    - ./prepare-tests.sh
  after_script:
    - echo "Clean up temporary files..."
    - rm -r test-data

run_lint_tests:
  stage: test
# prepare env datasets
  before_script:
    - echo "Preparing tet data..."
  script:
  # script neCeds a list, using hypen
    - echo "Running tests...."
  after_script:
    - echo "Clean up temporary files..."


build_image:
  stage: build
  script:
    - echo "Building docker image...."
    - echo "Tagging the docker image"


push_image:
  stage: build
  needs:
    - build_image
  script:
    - echo "Logging into docker registry...."
    - echo " Push  docker image to registry..."

deploy_image:
  stage: deploy
```

Situation 1

Pipeline is triggered automatically for all branches. (e.g. add a new branch)

We don't want to release new app versions from feature or bugfixes branches.
Instead we just want to run the tests t validate code before merging it into the main branch
=> tell gitlab pipeline logic just run for main branch

keywords: only / except

```
    - rm -r test-data

run_lint_tests:
  stage: test
# prepare env datasets
  before_script:
    - echo "Preparing tet data..."
  script:
  # script neCeds a list, using hypen
    - echo "Running tests...."
  after_script:
    - echo "Clean up temporary files..."


build_image:
  only:
    - main
  stage: build
  script:
```

方法1

New branches still execute test, but not build anymore

! build_image 后面的所有的jobs都要加上only: -main

Situation 2

Completely **ingore** any branches except main branch:

- method1 add only: -main in every single jobs
- method2: add workflow rules

How does gitlab know weather we are in main or branch?
-> In every pipeline GitLab provides predefined **environment variables**, is available in GitLab docs

方法2

```
workflow:
  rules:
    - if: $CI_COMMIT_BRANCH != "main"
      when: never
    - when: always

stages:
  - test
  - build
```

Situation3
if new merge request, than also run the pipeline, but just run the test:

- add $CI_PIPELINE_SOURCE
- add only: -main to other jobs

其他的job还是要加only main

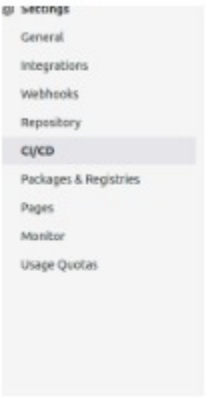
```
workflow:
  rules:
    - if: $CI_COMMIT_BRANCH != "main" && $CI_PIPELINE_SOURCE != "merge_request_event"
      when: never
    - when: always

run_lint_tests:
  stage: test
# prepare env datasets
  before_script:
    - echo "Preparing tet data..."
  script:
  # script neCeds a list, using hypen
    - echo "Running tests...."
  after_script:
    - echo "Clean up temporary files..."


build_image:
  only:
    - main
  stage: build
  script:
```

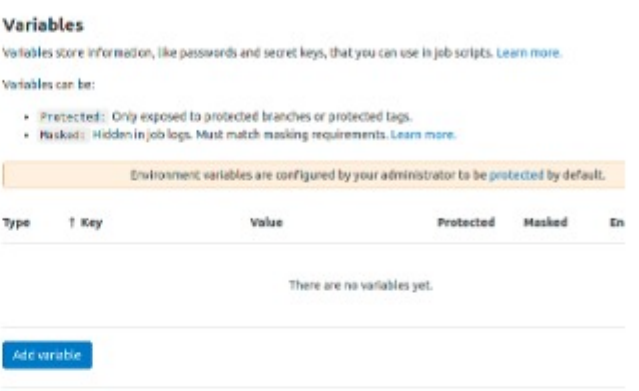
GitLab Predefined variables:

https://docs.gitlab.com/ee/ci/variables/predefined_variables.html

use cases: run pipeline for every mircoservice

- value is passed in on pipleline execution, instead of hardcoding it in the .gitlab-ci-yml file.
- making the config file more re-usable and fleixble

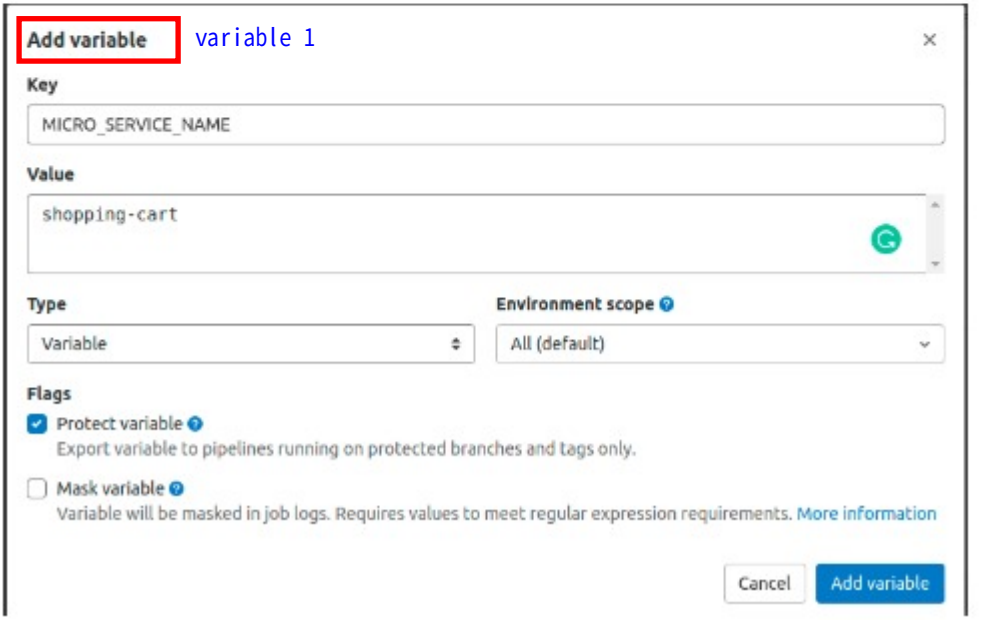




Add project varibles for CI/CD in settings/cicd/variable

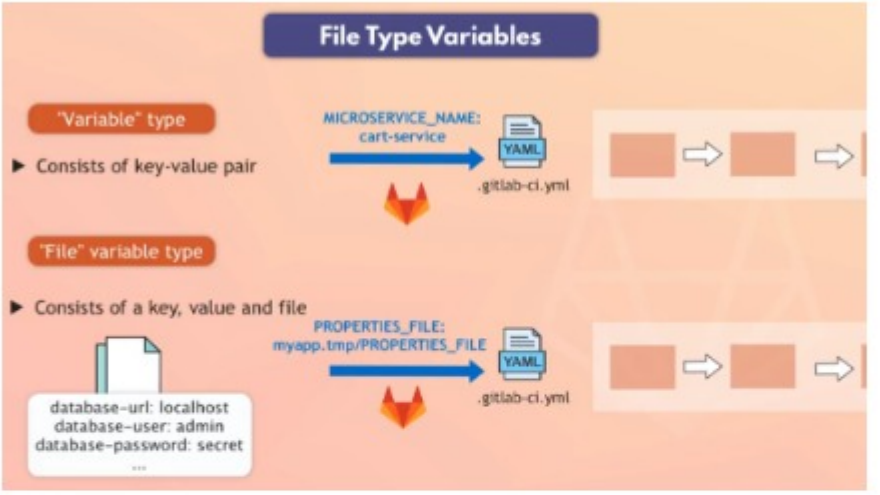
Project Variables:

1. Type: Variable
   - stored outside the git repo (not in the .yml file.
   - ideal for tokens and passwords, which should not be included in the repo for security reason



```
before_script:
  - echo "Preparing tet data..."
script:
# script neCeds a list, using hypen
  - echo "Running tests for micro service $MICRO_SERVICE_NAME"
  - chmod +x prepare-tests.sh
  - ./prepare-tests.sh
after_script:
  - echo "Clean up temporary files..."
```

1. type: file variable variable 2




```
$ echo " deploy new docker image to $DEPLOYMENT_ENVIRONMENT using the following configuration file-$PROPERTIES_FILE..."
deploy new docker image to test 2.myapp.com using the following configuration file /builds/Yingtremendous/mynodeapp cicd project.tmp/PROPERTIES_FIL
...
```

```
deploy_image:
  only:
    - main
  stage: deploy
  script:
    - echo " deploy new docker image to $DEPLOYMENT_ENVIRONMENT using the following configuration file-$PROPERTIES_FILE..."
    - cat $PROPERTIES_FILE
```

Define variable in -gitlab-ci.yml file variable 3

Another use case of variables: Storing values you want to re-use multiple times, reducing code duplication

Define variables locally

locally

```
build_image:
  variables:
    image_repository: docker.io/my-docker-id/myapp
    image_tag: v1.0
  only:
    - main
  stage: build
  script:
    - echo "Building docker image...."
    - echo "Tagging the docker image $image_repository:$image_tag
    -
```

Define variables globally

```
    - test
    - build
    - deploy

variables:
  image_repository: docker.io/my-docker-id/myapp
  image_tag: v1.0

run_unit_tests:
  stage: test
```

globally

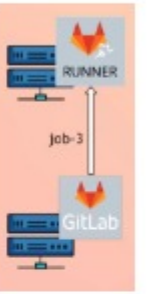
Where are the tasks executed?

Gitlab architecture:
Gitlab server / gitlab instance / gitlab installation (same name)

- main component
- pipeline configuration
- manages the pipeline execution
- stores the pipeline results

Jobs are executed by gitlab runners:

- gitlab runners are agents that run your ci/cd jobs
- gitlab server assigns these jobs to available runners

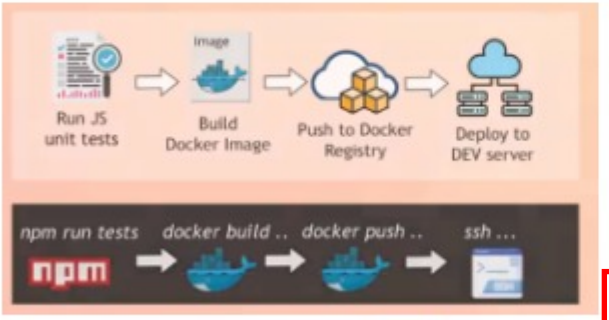


SaaS: managed by gitlab
or self-manager

RUNNER:
Program that you should install on a machine, that's separate from the one that hosts the gitlab server

Shared runners:
. the provided runners by Gitlab are shared runners

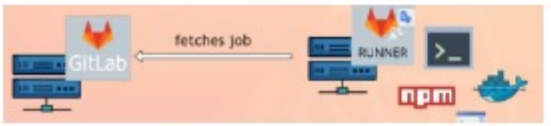
- shared runners are available to all projects in a gitlab instance
- shared runners on gitlab.com are available to all users on the plattform



commands executed on the shell of the server, where GitLab Runner is installed

executor 1

Shell == executor



All required programs for executing the jobs, need to be installed manually
if we want to run docker command, docker **needs tobe installed first**

Problems:

- a lot of runners, in each runners need to install same software
- several microservices, each microservice use different software version (manage different versions)
    - no clean build env for every job execution

=> Executing commands on OS directly comes with this difficuties

----->Limitaion of shell what else: Alternative Executors: Shell, **Docker**, VM
Wenn registering a runner, you can choose your executor

executor 2

Docker executor:

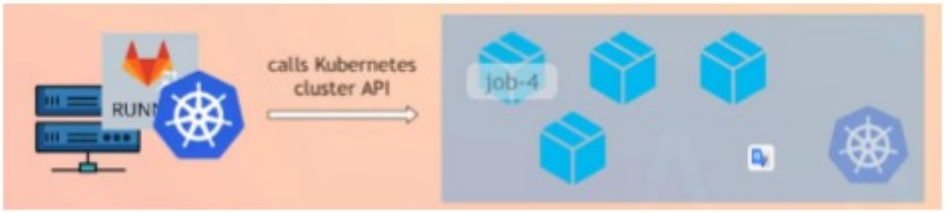
- commands are executed inside a container
- jobs run on user provided Docker images -> Each job runs in **a seperate and isolated container**

SO :
-> All tools need can be put in the docker image, no version conflicts. clean state of each job

executor 3

Kubernete Executor

Allows you to use an existing kuberneters cluster for your builds: utilize your high availability setup.
Kubernetes executor will create a new **pod** for each gitlab job



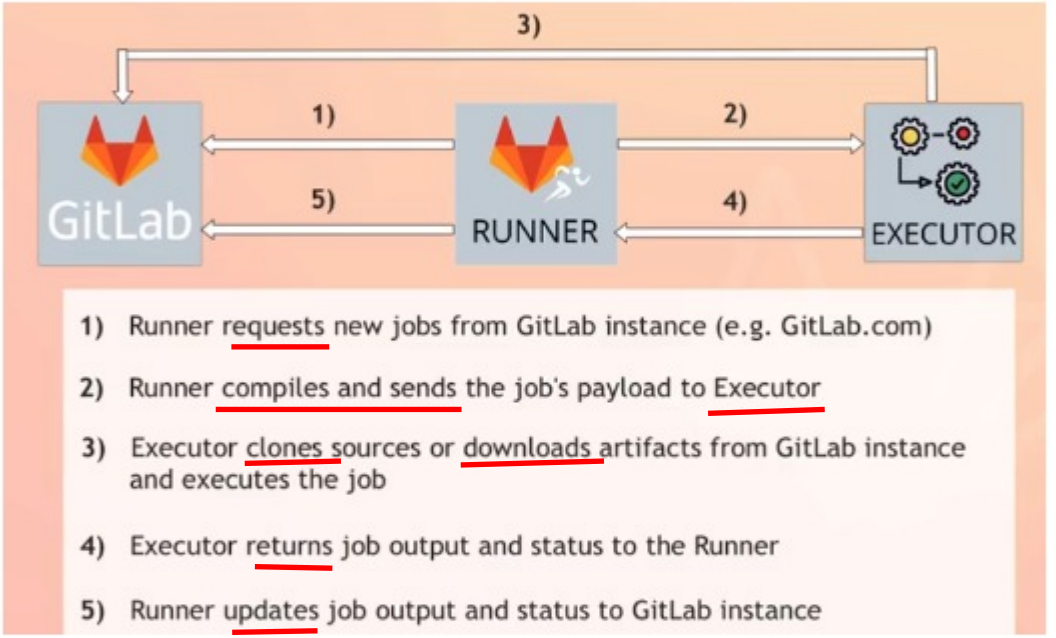
Docker machine executor: [Managing layer over Docker]
special version of docker executor: support auto-scaling
let's you create docker hosts on your computer, on cloud providers on deman, dynamically
creates servers, installs Docker on them and configures the client to talk to them
Docker machine executor works like the normal docker executor

How to select:
Best option: docker executor

How to configure?

- when you registry a runner, you **must** choose an executor
- 1 executor per runner

Multiple executors on same server ? --> Registry **multiple runners on same host**



Default shared runner

- your job is runned by one share runners, which maintained by gitlab. Docker machine executor are used for them

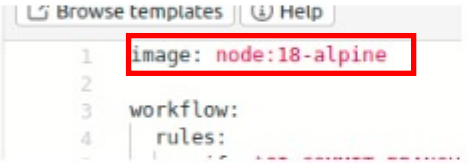
CI/CD jobs can get executed on different runners and machines

A list of shared runners can be checked in settings/cicd/runners

Override docker image:

- can do that in pipleline logic
- easy

Gloabally specify image 方法1



Specify image in job level 方法2

Combine gobal and specified image 方法3

No docker executor: image configure will be ignored in the shell executor

Specific Runners:
for security reasons / Jobs with specific requirements/ projects with a lot of CI activity
specific runners are self-managed
How: Setup machine -> instal gitlab runner programm ->connect to gitlab server

Gitlab runner in every machine
Registering a runner: Binding the runner to a specific Gitlab instance

Configure your own runners ->install tools on them -> use them in pipeline
Two different methods:

1. local
2. remote:AWS- create an EC2 instance on AWS, runner on that EC2

Install gitlab runner:

- download
  - curl -LJO "https://gitlab-runner-downloads.s3.amazonaws.com/latest/deb/gitlab-runner_amd64.deb"
  - install:
    - sudo dpkg -i git....deb
  - check the version
    - gitlab-runner --version
  - check the statuds
    - gitlab-runner status

```
(base) gu@gu-GE60-2PC:~$ gitlab-runner --version
Version:      15.1.0
Git revision: 76984217
Git branch:   15-1-stable
GO version:   go1.17.7
Built:        2022-06-20T10:10:54+0000
OS/Arch:      linux/amd64
```

```
(base) gu@gu-GE60-2PC:~$ sudo gitlab-runner status
Runtime platform                                    arch=amd64 os=linux pid=1370
29 revision=76984217 version=15.1.0
gitlab-runner: Service is running
```

  - connect the runner to gitlab installation specific runners:
    - gitlab-runner register

```
(base) gu@gu-GE60-2PC:~$ gitlab-runner register
Runtime platform                                    arch=amd64 os=linux pid=138437 revision=76984217 version=15.1.0
WARNING: Running in user-mode.
WARNING: The user-mode requires you to manually start builds processing:
WARNING: $ gitlab-runner run
WARNING: Use sudo for system-mode:
WARNING: $ sudo gitlab-runner...

Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com/
Enter the registration token:
GR1348941TPCAFf4ESHT62rt677M8
Enter a description for the runner:
[gu-GE60-2PC]: local-runner
Enter tags for the runner (comma-separated):
linux,local
Enter optional maintenance note for the runner:
This runner uses shell executor
Registering runner... succeeded                     runner=GR1348941TPCAFf4E
Enter an executor: custom, docker-ssh, parallels, ssh, docker, shell, virtualbox, docker+machine, docker-ssh+machine, kubernetes:
shell
Runner registered successfully. Feel free to start it, but if it's running already the config should be automatically reloaded!
```

job run in the gitlab-runner

Runners

Runners are processes that pick up and execute CI/CD jobs for GitLa

Register as many runners as you want. You can register runners as s
Runners are either:

- active - Available to run jobs.
- paused - Not available to run jobs.

Specific runners

These runners are specific to this project.

Set up a specific runner for a project

1. Install GitLab Runner and ensure it's running.
2. Register the runner with this URL:
   `https://gitlab.com/`

   And this registration token:
   `GR1348941TPCAFf4ESHT62rt677M8`

Reset registration token

Show runner installation instructions

Available specific runners

#16216999 (86v619NM) Remove runner

local-runner

linux local

AWS:

- default root user und unlimited privileges
- -> create an admin user
- IAM dashboard to create an adminuser, which can manage the users

Create admin

Benutzer hinzufügen

1 2 3 4 5

Benutzerdetails festlegen

Sie können mehrere Benutzer mit demselben Zugriffstyp und denselben Berechtigungen gleichzeitig hinzufügen. Weitere Informationen

Benutzername* admin

Weiteren Benutzer hinzufügen

AWS-Zugriffstyp auswählen

Wählen Sie aus, wie diese Benutzer in erster Linie auf AWS zugreifen. Wenn Sie nur programmgesteuerten Zugriff auswählen, wird NICHT verhindert, dass Benutzer über eine angenommene Rolle auf die Konsole zugreifen. Zugriffsschlüssel und automatisch generierte Passwörter werden im letzten Schritt bereitgestellt. Weitere Informationen

AWS-Anmeldeinformationstyp auswählen*

- ☑ **Zugriffsschlüssel – Programmgesteuerter Zugriff**
  Aktiviert einen **Zugriffsschlüssel-ID** und **Geheimer Zugriffsschlüssel** für AWS API, CLI, SDK und andere Entwicklungstools
- ☑ **Passwort – Zugriff auf die AWS-Managementkonsole**
  Aktiviert ein **Passwort** , mit dem Benutzer sich in der AWS-Managementkonsole anmelden können

Konsolenpasswort*

- ◉ Automatisch generiertes Passwort
- ○ Benutzerdefiniertes Passwort

Zurücksetzen des Passworts erfordern

- ☑ Benutzer muss bei der nächsten Anmeldung ein neues Passwort erstellen. Benutzern wird automatisch die Richtlinie IAMUserChangePassword zugewiesen, damit sie ihr eigenes Passwort ändern können.

Benutzer hinzufügen

1

Berechtigungen festlegen

Benutzer zur Gruppe hinzufügen | Berechtigungen von vorhandenem Benutzer kopieren | Vorhandene Richtlinien direkt anfügen

Richtlinie erstellen

Filterrichtlinien | Suchen | 755 Erg

	Richtlinienname	Typ	Verwendet
☑	AdministratorAccess	Auftragsfunktion	*Keine*
☐	AdministratorAccess-Amplify	AWS-verwaltet	*Keine*
☐	AdministratorAccess-AWSElasticBeanstalk	AWS-verwaltet	*Keine*

Benutzer hinzufügen

1 2 3 4 5

Prüfen

Überprüfen Sie Ihre Auswahl. Nachdem Sie den Benutzer erstellt haben, können Sie das automatisch generierte Passwort und den Zugriffsschlüssel anzeigen und herunterladen.

Benutzerdetails

Benutzername	admin
AWS-Zugriffstyp	Programmgesteuerter Zugriff und Zugriff über die AWS-Managementkonsole
Typ des Konsolenpassworts	Automatisch generiert
Zurücksetzen des Passworts erfordern	Ja
Berechtigungsgrenze	Die Berechtigungsgrenze wurde nicht festgelegt.

Berechtigungszusammenfassung

Die folgenden Richtlinien werden mit dem oben angezeigten Benutzer angefügt.

Typ	Name
Verwaltete Richtlinie	AdministratorAccess
Verwaltete Richtlinie	IAMUserChangePassword

Tags

Es wurden keine Tags hinzugefügt.

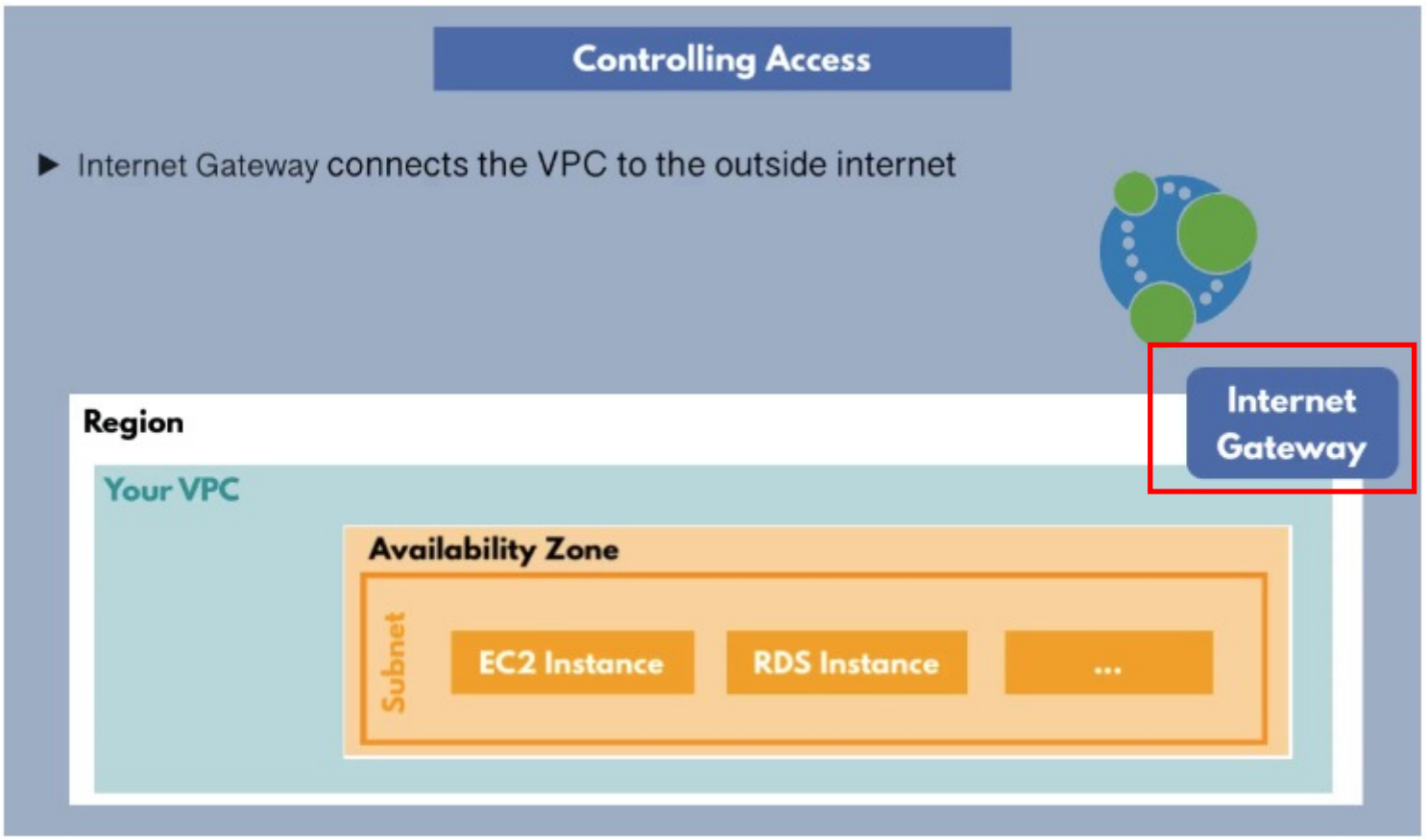
VPC (virtual private cloud):

- in the networkinig and content delivery.
- VPC for each region
- Region has multiple availability zones
- AZ: 1or more discrete data
- private network is isolated from others

- virtual **representation** of network infrastructure
- setup of serves network configuration moved to cloud
- your components have to run in an VPC
-

Bigger coporation: -> multiple VPSs

Subnet: subnet for each availabilty zone ; Private network inside a network
internal IP addresses they are communicate with each other (in the vpc level)
For each subnets, it has also ip

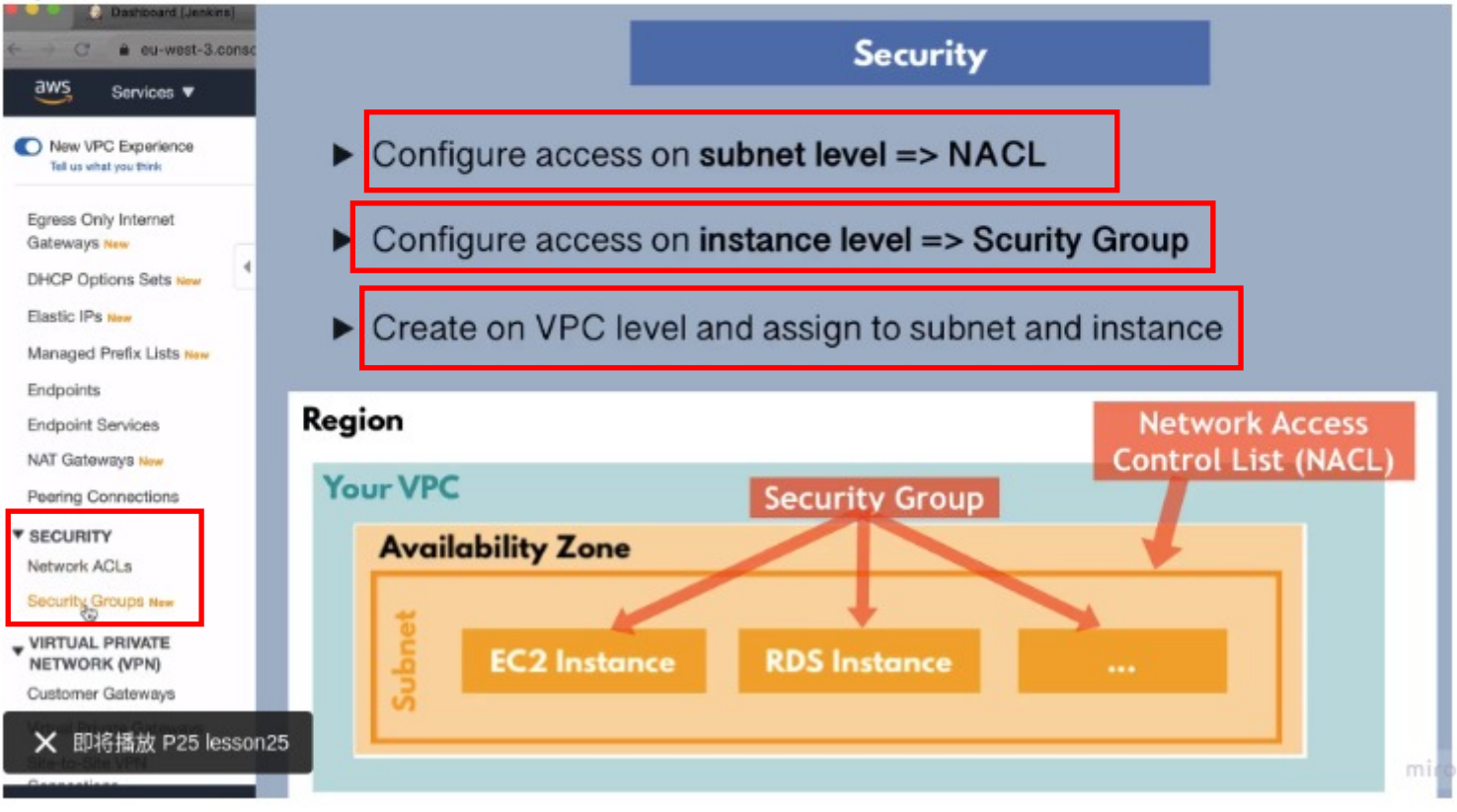


Controlling access

- secure your components
- controll access to your VPC
- controll access to your individual server instance

extenal access:

- configure: extenal acess auch as SSH.
- configuaration access on subnet level ==> NACL
- configure firewall rules in the virtual server level using security groups
-



Instance -> Verbinden

click the verbinden in the instance using the key.pem to connect

```
(base) gu@gu-GE60-2PC:~$ ssh -i ~/Downloads/gitlab-runner-key.pem ubuntu@ec2-3-
127-152-163.eu-central-1.compute.amazonaws.com
```

Because of the warning 设置权限

```
(base) gu@gu-GE60-2PC:~/Downloads$ chmod 400 ~/Downloads/gitlab-runner-key.pem
(base) gu@gu-GE60-2PC:~/Downloads$ ssh -i ~/Downloads/gitlab-runner-key.pem ubunt
Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.13.0-1029-aws x86_64)

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

  System information as of Fri Jul  1 16:32:27 UTC 2022

  System load:  0.0               Processes:             102
  Usage of /:   19.2% of 7.58GB   Users logged in:       0
  Memory usage: 20%               IPv4 address for eth0: 172.31.46.28
  Swap usage:   0%

1 update can be applied immediately.
To see these additional updates run: apt list --upgradable


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-46-28:~$
```

To install gitliba-runner on linux

- 1. apt-get update

```
ubuntu@ip-172-31-46-28:~$ apt-get update
Reading package lists... Done
E: Could not open lock file /var/lib/apt/lists/lock - open (13: Permission denied)
E: Unable to lock directory /var/lib/apt/lists/
W: Problem unlinking the file /var/cache/apt/pkgcache.bin - RemoveCaches (13: Permission denied)
W: Problem unlinking the file /var/cache/apt/srcpkgcache.bin - RemoveCaches (13: Permission denied)
ubuntu@ip-172-31-46-28:~$
```

Then download gitlab runner on ubuntn packages, then install gitlab runner-

```
ubuntu@ip-172-31-46-28:~$ curl -L "https://packages.gitlab.com/install/repositori
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  6885  100  6885    0     0  31728      0 --:--:-- --:--:-- --:--:-- 31728
Detected operating system as Ubuntu/focal.
Checking for curl...
Detected curl...
Checking for gpg...
Detected gpg...
Running apt-get update... done.
Installing apt-transport-https... done.
Installing /etc/apt/sources.list.d/runner_gitlab-runner.list...done.
Importing packagecloud gpg key... done.
Running apt-get update... done.

The repository is setup! You can now install packages.
```

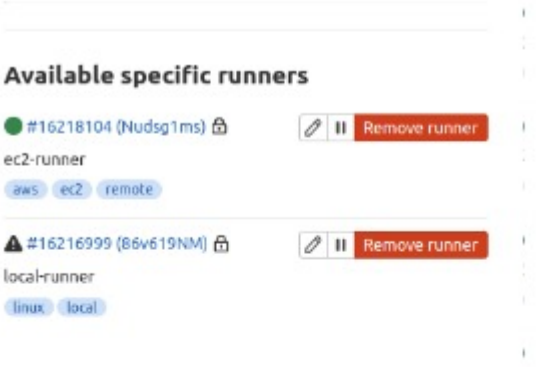
```
ubuntu@ip-172-31-46-28:~$ sudo apt-get install gitlab-runner
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  docker-engine
```

Check the version

```
ubuntu@ip-172-31-46-28:~$ sudo gitlab-runner --version
Version:      15.1.0
Git revision: 76984217
Git branch:   15-1-stable
GO version:   go1.17.7
Built:        2022-06-20T10:10:31+0000
OS/Arch:      linux/amd64
ubuntu@ip-172-31-46-28:~$
```

Register a runner

```
ubuntu@ip-172-31-46-28:~$ sudo gitlab-runner register
Runtime platform                                    arch=amd64 os=linux
Running in system-mode.

Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com
Enter the registration token:
GR1348941TPCAFf4ESHT62rt677M8
Enter a description for the runner:
[ip-172-31-46-28]: ec2-runner
Enter tags for the runner (comma-separated):
remote,aws, ec2
Enter optional maintenance note for the runner:
the runner is using shell executor
Registering runner... succeeded                     runner=GR1348941TPC
Enter an executor: docker+machine, docker, docker-ssh, parallels, shell
shell
Runner registered successfully. Feel free to start it, but if it's runn
ubuntu@ip-172-31-46-28:~$
```



AWS SUMMARY：

- IAM： register a admin , choose the both security type, choose from attached permission policies the AdminstrationAccess police. Download the csv using the IAM user. The ID and key are in the csv.
- Then we wil create a EC2 instance. VPC, Subnet, and security will be explained.
- Then using SSH method to get into your instance. Then install gitlab-runner in the instance

Using **tags** you can select a specific runner from the list of all runners that are available for the project
add tags for each jobs

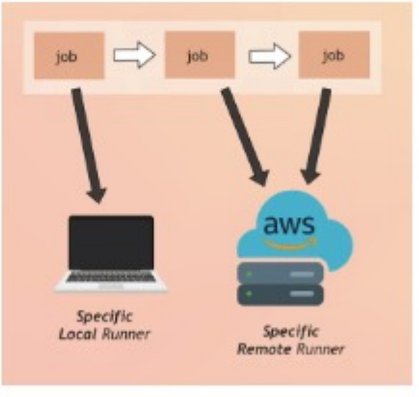
```yaml
run_lint_tests:
  tags:
    - ec2
    - remote
    - aws
  stage: test
```

```
Running with gitlab-runner 15.1.0 (76984217)
  on ec2-runner Nudsg1ms
Preparing the "shell" executor
Using Shell executor...
Preparing environment
Running on ip-172-31-46-28...
Getting source from Git repository
Fetching changes with git depth set to 20...
Initialized empty Git repository in /home/gitlab-runner/builds/N
s/mynodeapp-cicd-project/.git/
Created fresh repository.
Checking out 7bfc480f as main...
Skipping Git submodules setup
Executing "step_script" stage of the job script
$ echo "Preparing tet data..."
Preparing tet data...
$ echo "Running tests for micro service $MICRO_SERVICE_NAME"
Running tests for micro service shopping-cart
$ npm version
bash: line 133: npm: command not found
Running after_script
Running after script...
$ echo "Clean up temporary files..."
Clean up temporary files...
$ rm -r test-data
rm: cannot remove 'test-data': No such file or directory
Cleaning up project directory and file based variables
```

connect ec2

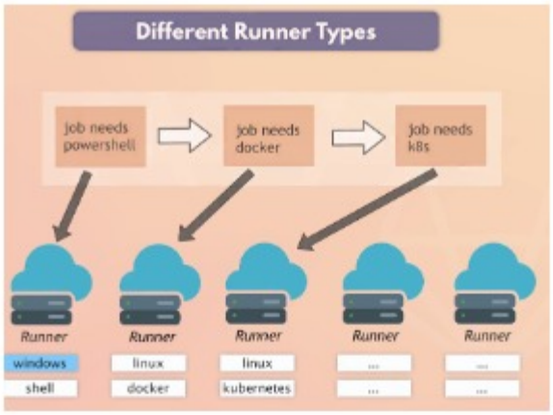
```
(base) gu@gu-GE60-2PC:~$ ssh -i ~/Documents/DevOps/gitlabCICD/gitlab-runner-key.
pem ubuntu@ec2-3-127-152-163.eu-central-1.compute.amazonaws.com
Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.13.0-1029-aws x86_64)
```

sudo apt upgrade
sudo apt install npm
sudo apt install nodejs

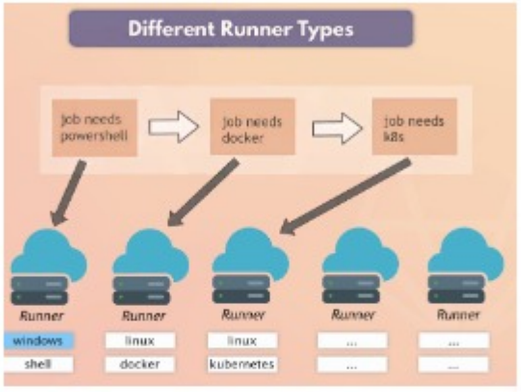


Distribute jobs in different runners

```yaml
run_lint_tests:
  tags:
    - ec2
    - remote
    - aws
  stage: test
# prepare env datasets
  before_script:
    - echo "Preparing tet data..."
  script:
  # script neCeds a list, using hypen
    - echo "Running tests...."
  after_script:
    - echo "Clean up temporary files..."


build_image:
  tags:
    - linux
    - local
  only:
    - main
  stage: build
  script:
    - echo "Building docker image...."
    - echo "Tagging the docker image $image_repository:$image_tag"
```

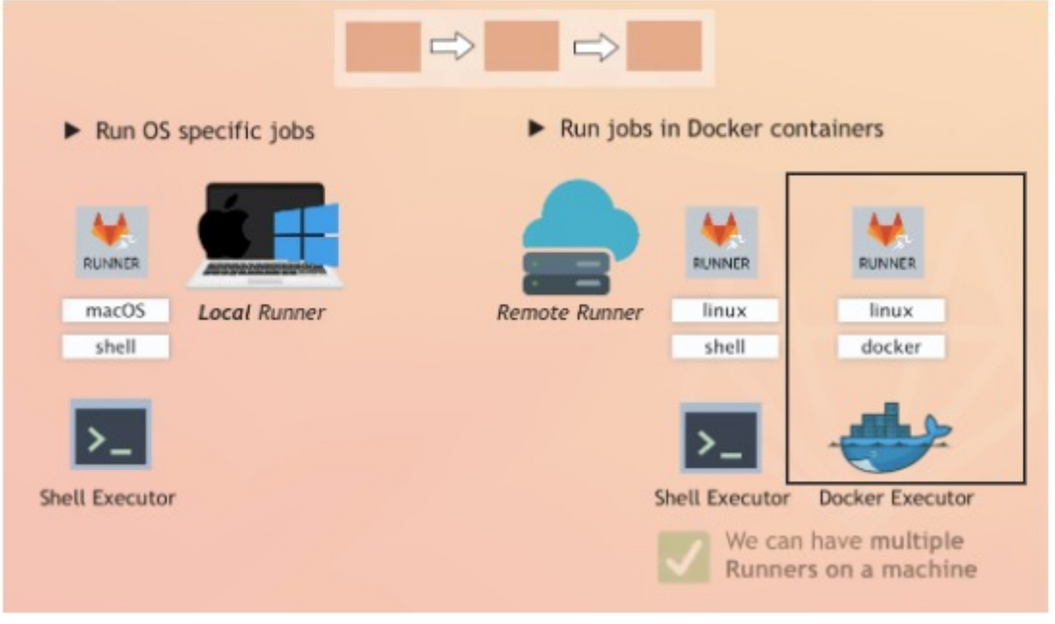


Usually you have two or more runners of the same type with same tags and job that refrences those tags will be excuted on on of the runners



Distribute load of job

not want to install node and npm on runners OS DIRECTLY - > Docker



Register new runner in the remote server
-install docker

```
ubuntu@ip-172-31-46-28:~$ sudo apt install docker.io
Reading package lists... Done
```

```
ubuntu@ip-172-31-46-28:~$ sudo docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
ubuntu@ip-172-31-46-28:~$
```

the docker is runnering

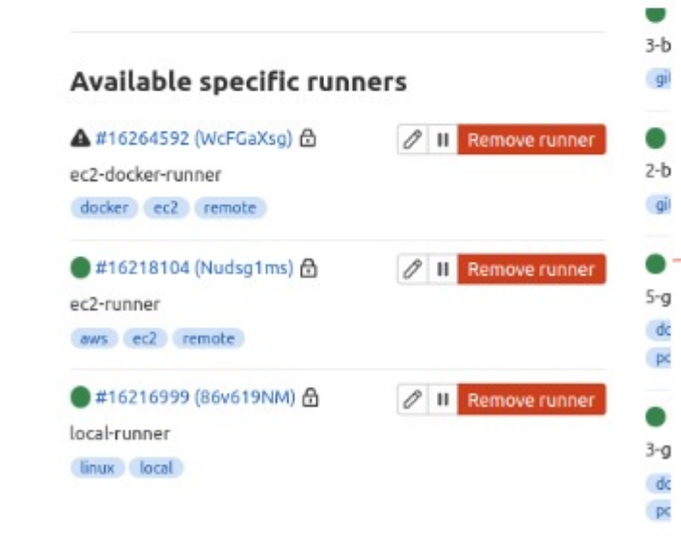
Add ubuntu user to docker group:

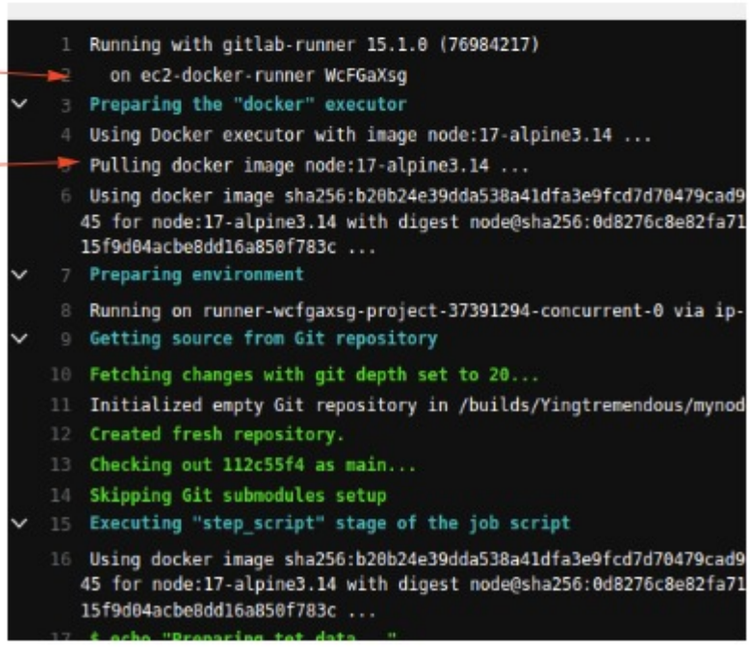
```
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORT
ubuntu@ip-172-31-46-28:~$ sudo usermod -aG docker $USER
ubuntu@ip-172-31-46-28:~$
```

log out and log in again then we can directly use docker command

register a new runner with docker register

```
ubuntu@ip-172-31-46-28:~$ sudo gitlab-runner register
Runtime platform                                    arch=amd64 os=linux pid=5222
9 revision=76984217 version=15.1.0
Running in system-mode.

Enter the GitLab instance URL (for example, https://gitlab.com/):
https://gitlab.com/
Enter the registration token:
GR1348941TPCAFf4ESHT62rt677M8
Enter a description for the runner:
[ip-172-31-46-28]: ec2-docker-runner
Enter tags for the runner (comma-separated):
ec2, docker, remote
Enter optional maintenance note for the runner:

Registering runner... succeeded                     runner=GR1348941TPCAFf4E
Enter an executor: docker+machine, docker-ssh+machine, kubernetes, docker, docke
r-ssh, shell, virtualbox, custom, parallels, ssh:
docker
Enter the default Docker image (for example, ruby:2.7):
alpine:3.15.1
Runner registered successfully. Feel free to start it, but if it's running alrea
dy the config should be automatically reloaded!
ubuntu@ip-172-31-46-28:~$
```

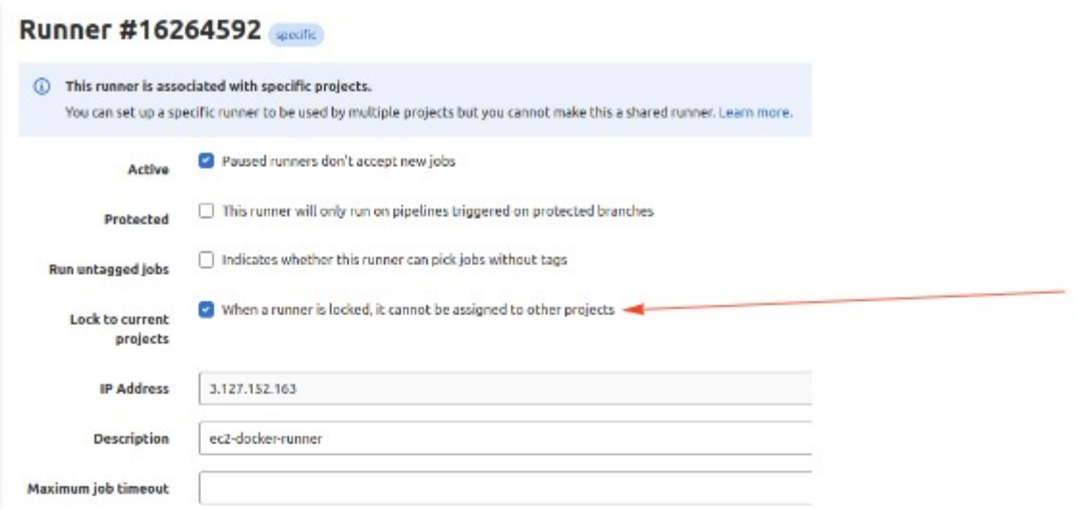





```
run_unit_tests:
  image: node:17-alinpe3.14
  tags:
    - ec2
    - docker
    - remote
  stage: test
# prepare env datasets
  before_script:
    - echo "Preparing tet data..."
```

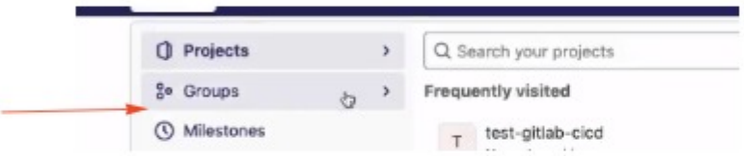
using docker executor and
need image for node

runners assigned to that specific project
You can unlock and assign runner to other projects as well



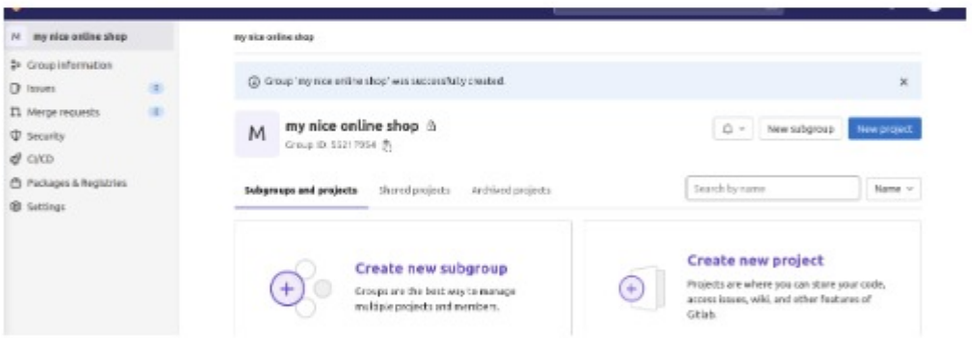
Use case: online shop, a lot of mircoservices (shipping-app, billing-app...), separate git project for each mircoservice . each one has its pipeline
-> concept: group runner

In gitlab we have project, we have also groups



You can use groups to manage one or more related projects at the same time.

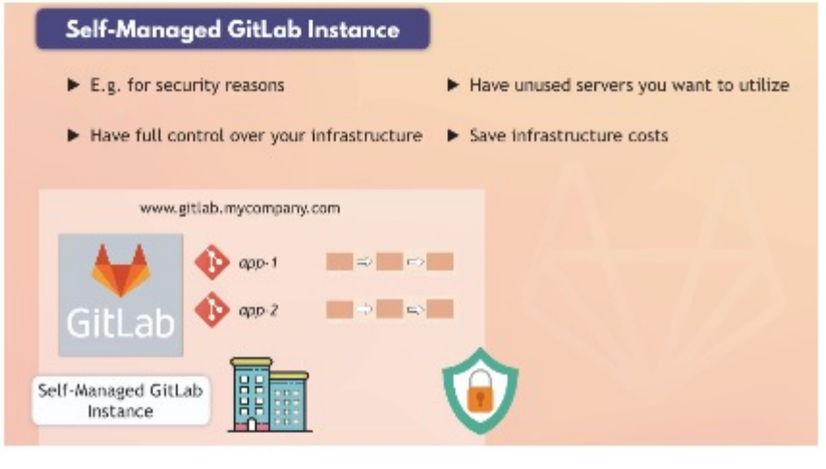
- e.g. you can groups to manage permission: if someone has access tot he group, they get access to all the projects in the group



Share runners among the projects in a group, .> group runner

How to create group runner?
In the group / settings/ CICD/ you have here Group runners

Self managed gitlab instance: repo, runner ....(**sepcific** runner, **group** runner, **shared** runner) -> You can install and register a shared runner which is available to every project in the gitlab instance



Gitlab runner versions

? sync and compatible ?

2 different GitLab Setups

SaaS - GitLab's Instance

www.gitlab.com

- **All GitLab users** can host git repos and configure pipelines

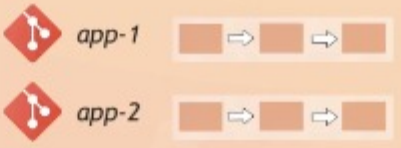
Self-Managed GitLab Instance

www.gitlab.mycompany.com

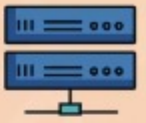
- Only company has access to it

3 Types of Runners

GitLab Instance



GitLab Runners



- Scope of the Runners: Shared | Specific | Group
  - **Shared**: Available to all groups and projects in GitLab instance
  - **Specific**: Associated with specific projects
  - **Group**: Available to all projects in a group

Executors

Unit test: Testing individual code components rather than the code as a whole
SaST: Static application securtity testing: analyzes code itself without actually executing the code
It scans your code for any securtiy vulnerabilities

How to increment and set images version of application dynamically

Run unit test
gitlab->cicd->edit delete all except workflow then write like the followr

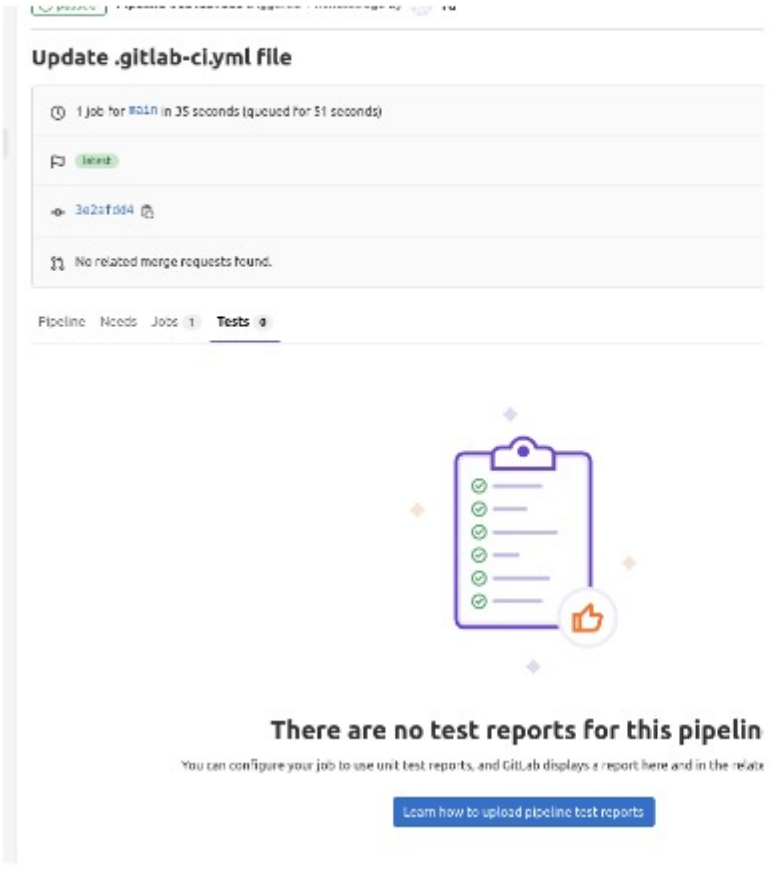
Browse templates | Help

```
workflow:
  rules:
    - if: $CI_COMMIT_BRANCH != "main" && $CI_PIPELINE_SOURCE != "merge_request_event"
      when: never
    - when: always


run_unit_tests:
  image: node:17-alpine3.14
  tags:
    - ec2
    - docker
    - remote
  before_script:
    - cd app
    - npm install
  script:
    - npm test
```

```
Running with gitlab-runner 15.1.0 (76984217)
  on ec2-docker-runner WcFGaXsg
Preparing the "docker" executor                                    00:02
Using Docker executor with image node:17-alpine3.14 ...
Pulling docker image node:17-alpine3.14 ...
Using docker image sha256:b20b24e39dda538a41dfa3e9fcd7d70479cad96e3aa7324a0fc7fd1eacd8de
45 for node:17-alpine3.14 with digest node@sha256:0d8276c8e82fa717a9a88b8734bbad60ac29a0f
15f9d04acbe8dd16a850f783c ...
Preparing environment                                              00:01
Running on runner-wcfgaxsg-project-37391294-concurrent-0 via ip-172-31-46-28...
Getting source from Git repository                                 00:01
Fetching changes with git depth set to 20...
Reinitialized existing Git repository in /builds/Yingtremendous/mynodeapp-cicd-project/.
git/
Checking out 3e2afdd4 as main...
Skipping Git submodules setup
Executing "step_script" stage of the job script                    00:28
Using docker image sha256:b20b24e39dda538a41dfa3e9fcd7d70479cad96e3aa7324a0fc7fd1eacd8de
45 for node:17-alpine3.14 with digest node@sha256:0d8276c8e82fa717a9a88b8734bbad60ac29a0f
15f9d04acbe8dd16a850f783c ...
$ cd app
$ npm install
npm WARN deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#deprecated
npm WARN deprecated urix@0.1.0: Please see https://github.com/lydell/urix#deprecated
added 557 packages, and audited 558 packages in 24s
24 packages are looking for funding
  run `npm fund` for details
4 vulnerabilities (2 moderate, 1 high, 1 critical)
To address all issues, run:
  npm audit fix
Run `npm audit` for details.
$ npm test
> bootcamp-node-project@1.0 test
> jest --ci --reporters=default --reporters=jest-junit
Browserslist: caniuse-lite is outdated. Please run:
npx browserslist@latest --update-db
Why you should do it regularly:
https://github.com/browserslist/browserslist#browsers-data-updating
PASS ./server.test.js
  ✓ main index.html file exists (4 ms)
  ✓ Dockerfile exists
  ✓ .gitignore file exists (1 ms)
Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        1.707 s
Ran all test suites.
Cleaning up project directory and file based variables              00:01
Job succeeded
```

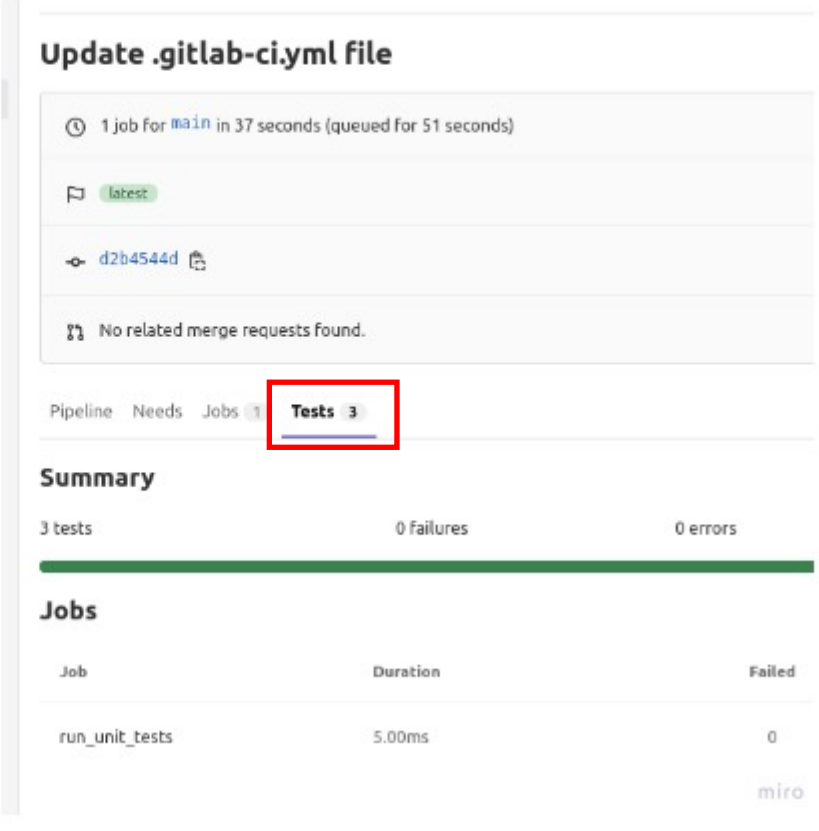
Configure test results

No test results and no artifacts



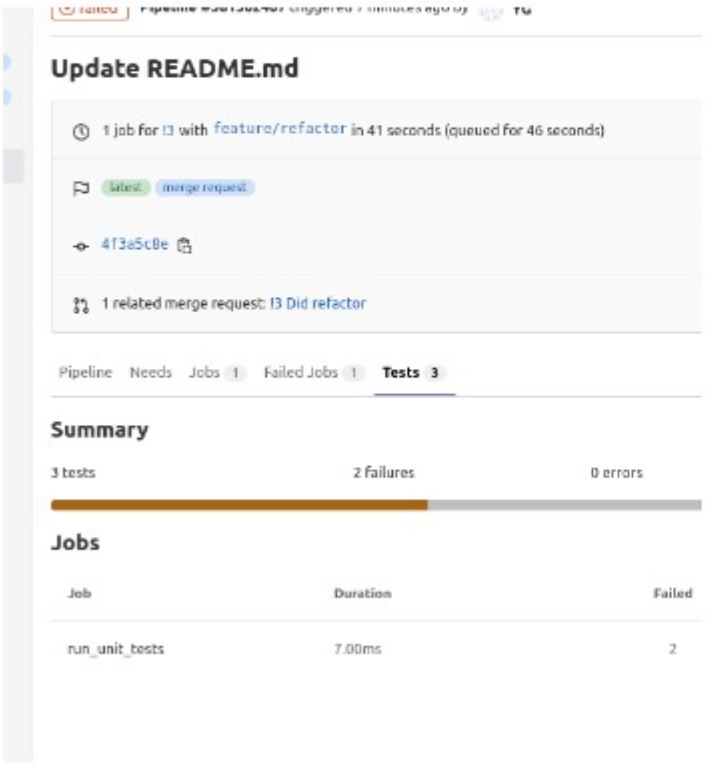
Job can output an archive of files and directories. -> It called a job artifact
JUnit was orginally developed in JAVA. But can also be for other languages like JS python Ruby

```
    - npm install
  script:
    - npm test
  artifacts:
    when: always
    reports:
      junit: app/junit.xml
```



Orgnize your your test results use paths properties

Test reports in the development process, you can check if all tests and trace them



Delete the docker file

build application in the docke and push to docker registry

Every GitLab project can have its own space to store its Docker image

Packages and registries

Packages & Registries

- **Package Registry** = Use GitLab as a private or public registry for variety of supported package managers

Maven | npm | NuGet | PyPI | Ruby gems | ...

- **Container Registry** = Registry to store Docker images

Public vs. Private Repositories

Public

- Docker images stored in a public registry, can be used by everyone
- E.g. on DockerHub images like Ubuntu, mongodb etc are publicly available

Private

- Can only be accessed by logging into the Docker repository
- E.g. for company internal Docker images

Private registry -> to access image push or pull -> you need to **authnticate** first

- **Infrastructure Registry** = Private registry for IaC packages

For Tarranfrom

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

You can add an image to this registry with the following commands:

```
docker build -t registry.gitlab.com/yingtremendous/mynodeapp-cicd-project .
```

```
docker push registry.gitlab.com/yingtremendous/mynodeapp-cicd-project
```

CI_REGISRTY_USER
CI_REGISRTY_PASSWARD The values of these are only valid for 1 job only

```yaml
build_image:
  stage: build
  tags:
    - ec2
    - shell
    - remote
  script:
    - docker build -t registry.gitlab.com/yingtremendous/mynodeapp-cicd-project:1.0 .

push_image:
  stage: build
  needs:
    - build_image
  tags:
    - ec2
    - shell
    - remote
  before_script:
    - docker login -u $CI_REGISTER_USER -P $CI_REGISTRY_PASSWARD  registry.gitlab.com
  script:
    - docker push registry.gitlab.com/yingtremendous/mynodeapp-cicd-project:1.0
```

必须写的非常的准确

lowerclass u and p for user and passward

Multiple image repositories

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

Give images name

```
30
31  build_image:
32    stage: build
33    tags:
34      - ec2
35      - shell
36      - remote
37    script:
38      - docker build -t $CI_REGISTRY_IMAGE/microservice/payment:1.0 .
39
40  push_image:
41    stage: build
42    needs:
43      - build_image
44    tags:
45      - ec2
46      - shell
47      - remote
48    before_script:
49      - echo "$CI_REGISTER_USER $CI_REGISTRY_PASSWARD"
50      - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD  registry.gitlab.com
51    script:
52      - docker push $CI_REGISTRY_IMAGE/microservice/payment:1.0
```

Parametize the name of microservices

```
variables:
  IMAGE_NAME: $CI_REGISTRY_IMAGE/microservice/$MICRO_SERVICE
  IMAGE_TAG: "1.0"

    - remote
  script:
    - docker build -t $IMAGE_NAME:IMAGE_TAG .

push_image:
```

Deploy images to develop serve

不要忘记改变权限

1. You need to have a development server
    a. create and configure a Server to Deploy to

```
(base) gu@gu-GE60-2PC:~/Documents$ cd ../Downloads/
(base) gu@gu-GE60-2PC:~/Downloads$ chmod 400 dev-server.pem
(base) gu@gu-GE60-2PC:~/Downloads$ ssh -i dev-server.pem ubuntu@54.93.229.194
```

sudo apt update
sudo apt install docker.io
sudo usermod -aG docker ubuntu

Provide private key of the server: settings /cicd/ create a variable and set it as a file

Add variable

Key

SSH_PRIVATE_KEY

Value

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA61ccpCuSLmrjvpHyNFgAgY0SEW2XCDJLhZy0PigNkVeZc0Ho
gDvdfpKpH1f5EPjQQn6Bz3znKK+fNbZO17k6jlEUArIU99jDFgXECEKPIytdH6aI
tRqIISYBv1vWFORWYb41ADNpqVx82aNO5/WYMOwW7e6pGD6/PWlCfmIgighbjvv6
WEnFCi/iCpKLOYwjULAFc08jWa/oSmcVa6ix1djbYXylKP5+JQmFRHbnRq5uF6vW
Iz07+reXTob8k3RoFEaY11eAR02DA8Br73VRDrWPkywqCAND4x/xG+i/d8Pi6hnz
```

Type

Variable

Environment scope

All (default)

File

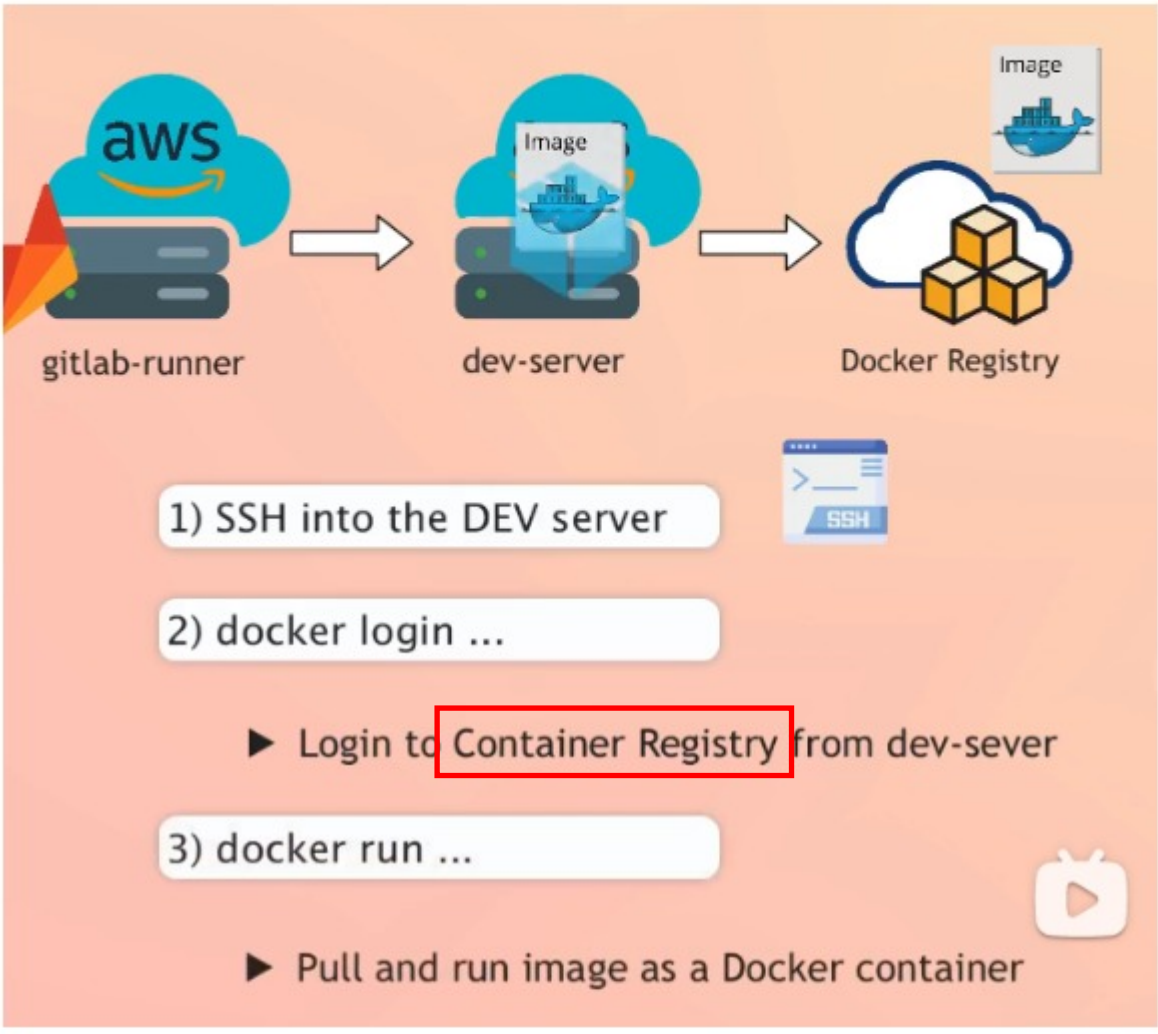
Flags

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

Write the following logic in the job



Disable strict host key checking using -o (Because in the server we don't want to type yes)

On the dev-server we want to execute the Docker commands-> pass the commands as parameters to our ssh command

```
deploy_to_dev:
  stage:
    depoly
  tags:
    - ec2
    - shell
    - remote
  before_script:
    - chmod 400 $SSH_PRIVATE_KEY
  script:
    ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ubuntu@$DEV_SERVER_HOST
    docker login docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD  $CI_REGISTRY &&
    docker run -d -p 3000:3000 $$IMAGE_NAME:IMAGE_TAG
```

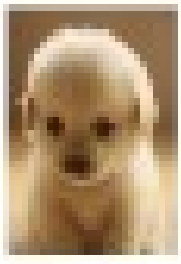
set TCP port 3000

List of projects team is working on

Name: Andrew Hill

Role: DevOps engineer

Projects: AWS migration, DevOps Automation, CI/



Name: Ari Baker

Role: Software developer

Projects: Online Shop, ERP Software

go to depolyment -> Environment

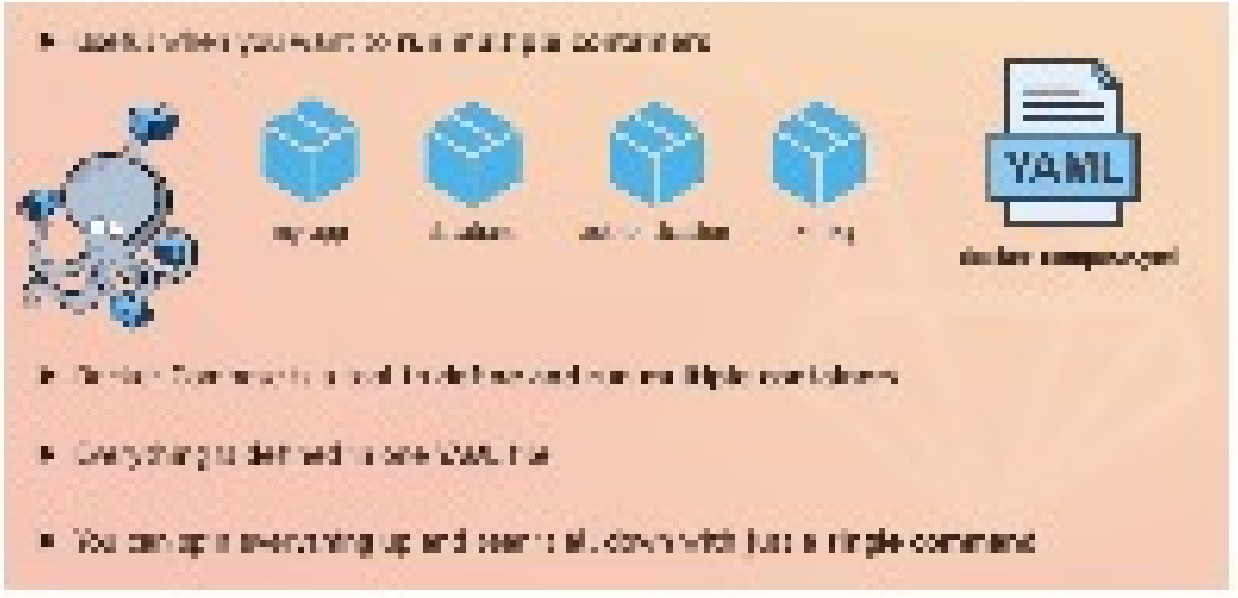
```
    docker run -d -p 3000:3000 gulm
environment:
  name: development
  url: $DEV_ENDPOINT
```

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

CD with Docker-Compose

Docker run will always works for first time, if you change the version from 1.1 to 1.2. It will report that port is already allocated.

If we want to deploy another docker. How to run multiple container in one docker run



create and write a new file docker-compose

docker-compose....

```yaml
version: "3.3"
services:
    app:
        images: registry.gitlab.com/yingtremendous/mynodeapp-cicd-project
        ports:
            - 3000:3000
```

Delete docker run in the editor -> use docker compose command

```yaml
deploy to dev:
  stage:
    depoly
  tags:
    - ec2
    - shell
    - remote
  before_script:
    - chmod 400 $SSH_PRIVATE_KEY
  script:
    - ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ./docker-compose.yaml ubuntu@$DEV_SERVER_HOST:/home/
ubuntu
    - ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ubuntu@$DEV_SERVER_HOST "
    docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD  $CI_REGISTRY &&
    docker-compose -f docker-comppose.yaml down &&
    docker-compose -f docker-compse.yaml up -d"
  environment:
    name: development
    url: $DEV_ENDPOINT
```

We using docker-compse in the dev-server. -> install in dev-server

isudo apt install docker-compose

Who code is in the gitlab-runner but not in the dev-server, so we need to copy yaml file to the dev-server.

docker-compose -f docker-compose.yaml is default -> .gitlab-ci.yml optimieru

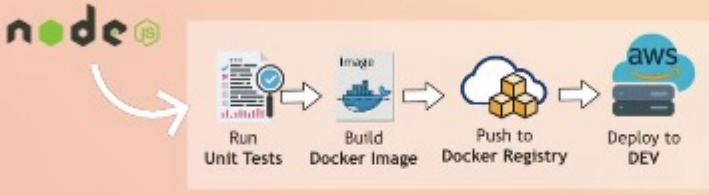
```
  before_script:
    - chmod 400 $SSH_PRIVATE_KEY
  script:
    - scp -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ./docker-compose.yaml ubuntu@$DE\
ubuntu
    - ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ubuntu@$DEV_SERVER_HOST "
      docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD  $CI_REGISTRY &&
      docker-compose down &&
      docker-compose up -d"
  environment:
    name: development
    url: $DEV_ENDPOINT
```

ocker-compose.yaml 123 bytes

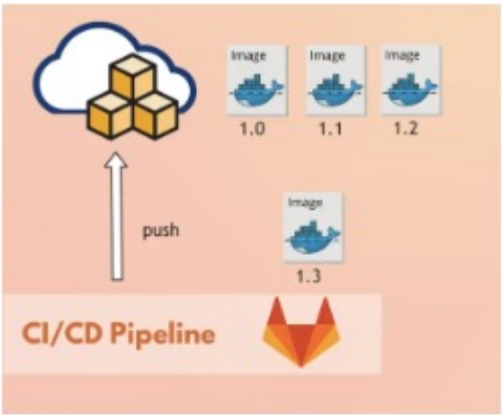
```
version: "3.3"
services:
    app:
        image: ${DC_IMAGE_NAME}:${DC_IMAGE_TAG}
        ports:
            - 3000:3000
```

```
    ---
  - shell
    remote
before_script:
  - chmod 400 $SSH_PRIVATE_KEY
script:
  - scp -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ./docker-compose.yaml ubuntu@$DEV_SERVER_HOST:/home/
ubuntu
  - ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ubuntu@$DEV_SERVER_HOST "
    docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD  $CI_REGISTRY &&
    export DC_IMAGE_NAME=$IMAGE_NAME
    export DC_IMAGE_TAG=$IMAGE_TAG
    docker-compose down &&
    docker-compose up -d"
environment:
  name: development
  url: $DEV_ENDPOINT
```

Extend & Optimize Pipeline to a Real Life Pipeline



- **Dynamic Versioning:**
  Dynamically set an incremented version when we build the Docker image



automate the process of incrementing and setting the version

Application Versioning

1 . 4 . 2

Semantic numbering

Major Minor Patch

- **Major** changes - related to incompatible API changes
  - Replaced Framework
  - New feature
- **Minor** changes - new functionality in a backward-compatible manner
  - New, minor feature
  - Bigger Bugfix
- **Patch** changes - related to small bugfixes & changes, which are backward compatible
  - Small changes
  - Bugfix

- Version is defined in the configuration file of the package manager / build tool of the application


package.json


pom.xml


build.gradle

npm yarn Maven Gradle

read the application version from package.json

```
build_image:
  stage: build
  tags:
    - ec2
    - shell
    - remote
  script:
    - cat app/package.json | jq -r .version
    - docker build -t $IMAGE_NAME:IMAGE_TAG .
```

login into remote runner mit executor shell
install jq

```
Last login: Wed Jul  6 16:42:10 2022 from 80.209.207.19
ubuntu@ip-172-31-46-28:~$ sudo apt-get install jq
Reading package lists... Done
```

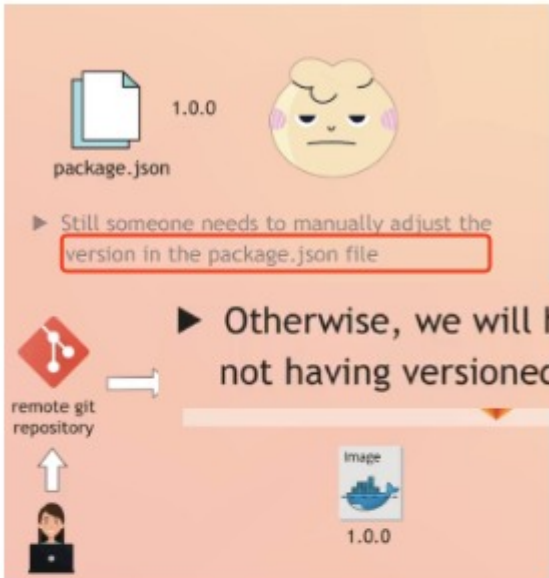
test jq command line

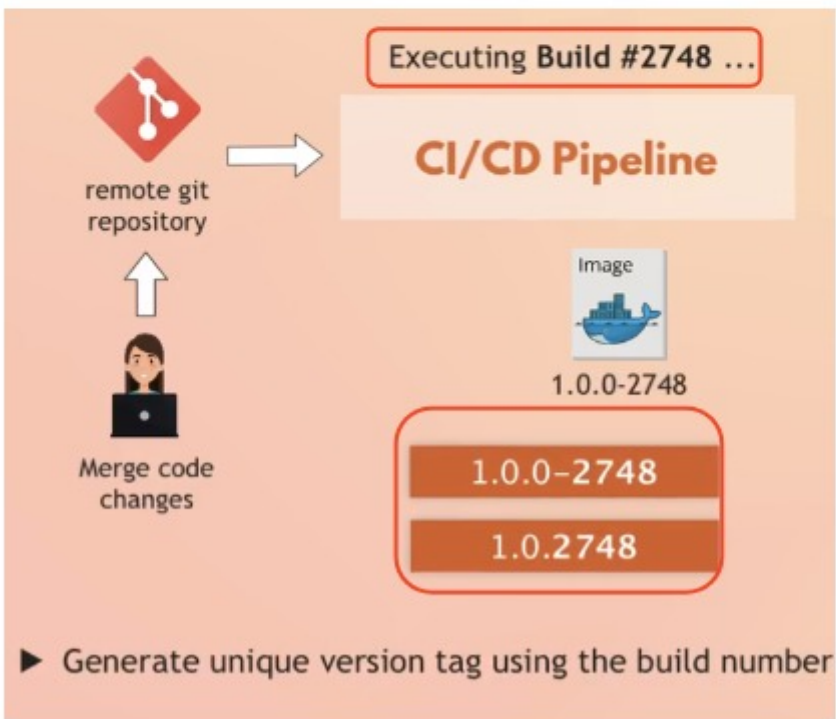
```
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.9) ...
ubuntu@ip-172-31-46-28:~$  vim package.json
ubuntu@ip-172-31-46-28:~$ cat package.json | jq -r .version
1.0
```

export version as a parameter

```
    - remote
  script:
    - export APP_VERSION = $(cat app/package.json | jq -r .version)
    - docker build -t $IMAGE_NAME:IMAGE_TAG .

    - shell
    - remote
  before_script:
    - export APP_VERSION = $(cat app/package.json | jq -r .version)
  script:
    - docker build -t $IMAGE_NAME:$APP_VERSION .
```

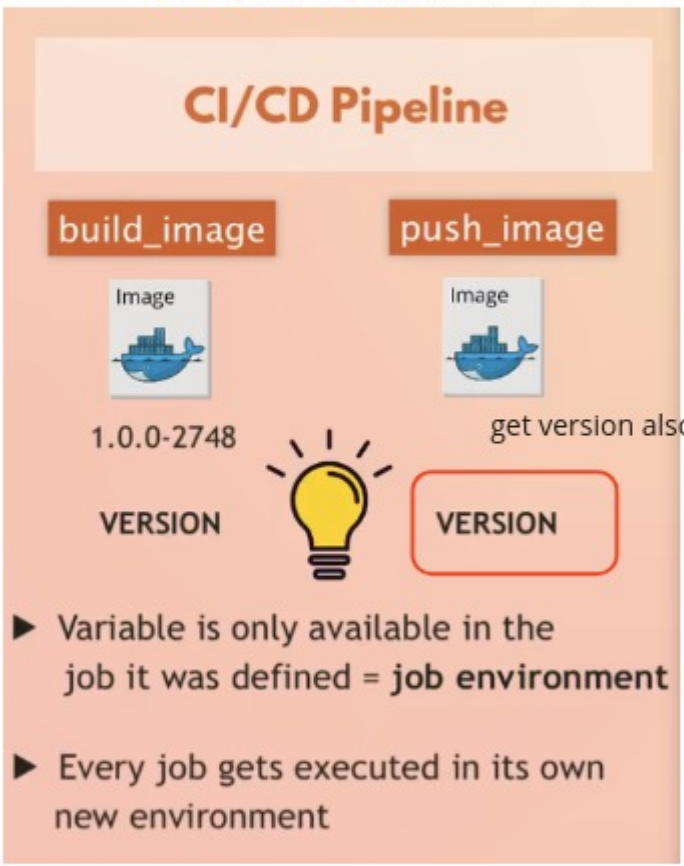
Generate Unique version tag





```
    - remote
before_script:
    - export PACKAGE_JSON_VERSION=$(cat app/package.json | jq -r .ver
    - export VERSION=$PACKAGE_JSON_VERSION.$CI_PIPELINE_IID
script:
    - docker build -t $IMAGE_NAME:$VERSION .
```

Use artifacts in the other jobs



How do we **pass** data **between jobs**?

CI/CD Pipeline

job job job job job

VERSION

Job Artifacts

- You can use **artifacts attribute** in general to create job artifacts
- The artifacts are **sent to GitLab** after the job finishes and available for download in the GitLab UI

Environment Preparation | Cache and Artifacts **Download** | Running Scripts | Cache and Artifacts **Upload** | Cleanup

GitLab job execution phases

Job Artifacts

CI/CD Pipeline

job job job job job

VERSION VERSION

artifacts create a version file, that can be used for other jobs

version-file.txt persists by artifacts paths properties

```
    - remote
  before_script:
    - export PACKAGE_JSON_VERSION=$(cat app/package.json | jq -r .ver
    - export VERSION=$PACKAGE_JSON_VERSION.$CI_PIPELINE_IID
    - echo $VERSION > version-file.txt
  script:
    - docker build -t $IMAGE_NAME:$VERSION .
  artifacts:
    paths:
      - version-file.txt
```

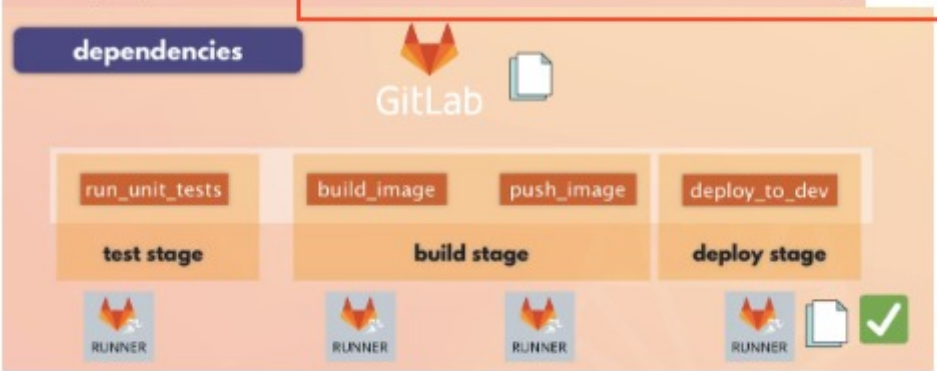
use version file in push stage

```
push_image:
  stage: build
  needs:
    - build_image
  tags:
    - ec2
    - shell
    - remote
  before_script:
    - export VERSION=$(cat version-file.txt)
    - echo "$CI_REGISTER_USER $CI_REGISTRY_PASSWAR
    - docker login -u $CI_REGISTRY_USER -p $CI_REG
  script:
```

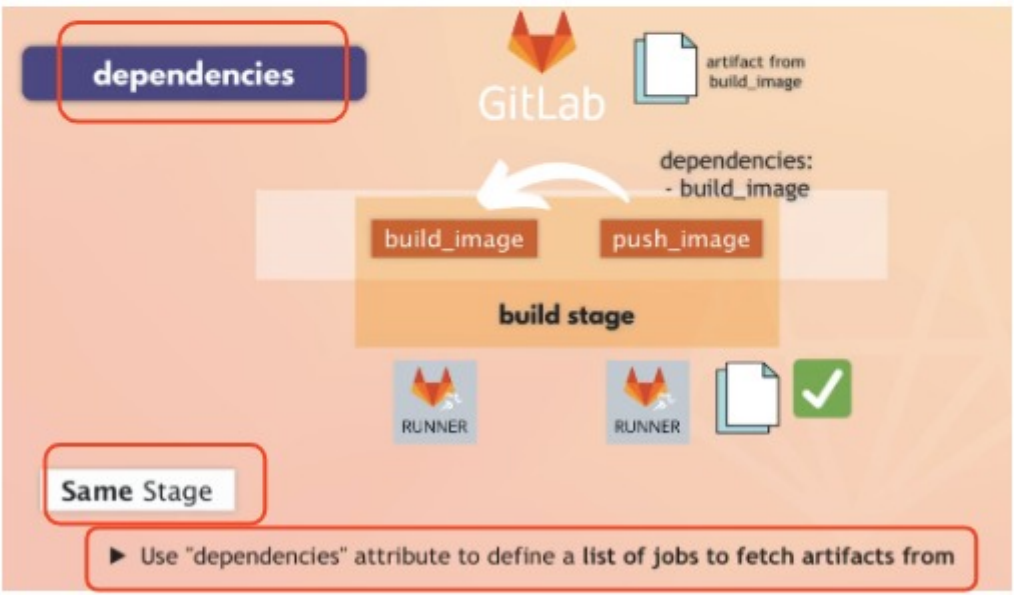
- By default: **jobs in later stages automatically download** all the artifacts created by jobs in earlier stages

different stages

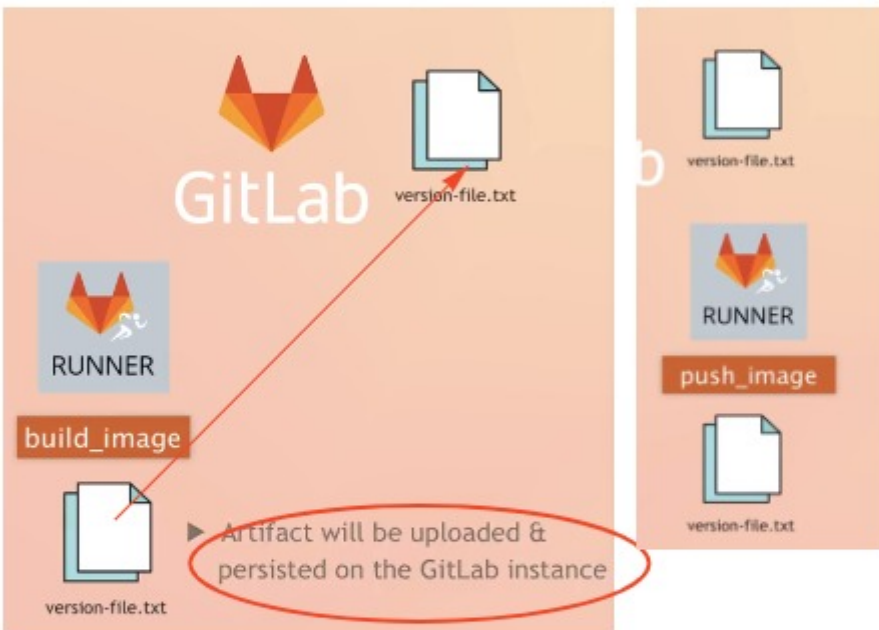
- Deploy_to_dev will have our version text file automatically



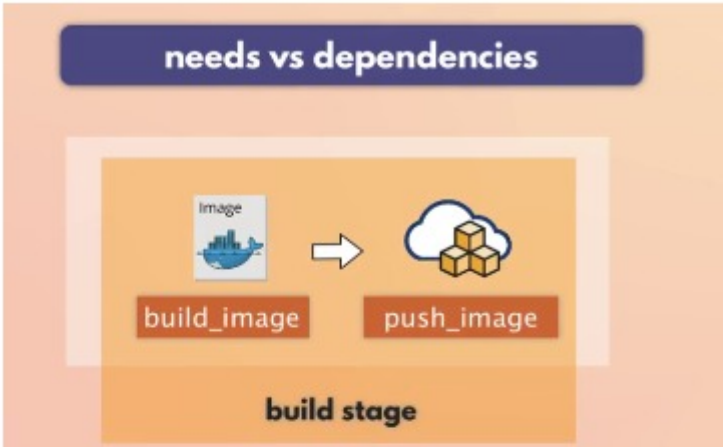
for same stage



```yaml
push_image:
  stage: build
  needs:
    - build_image
  dependencies:
    - build_image
  tags:
    - ec2
```



Difference between need and dependencies



needs

- Tells GitLab push_image needs to wait with execution until build_image completed

dependencies

- Tells GitLab push_image needs artifact from build_image job

dependencies must also in needs

- With "*needs*" only artifacts from the jobs listed in the "*needs*" will be downloaded
- **Why?** Because jobs with "*needs*" can start before earlier stages complete
- With "*needs*", artifacts from the job listed will be downloaded by default
- No need to define "*dependencies*" separately

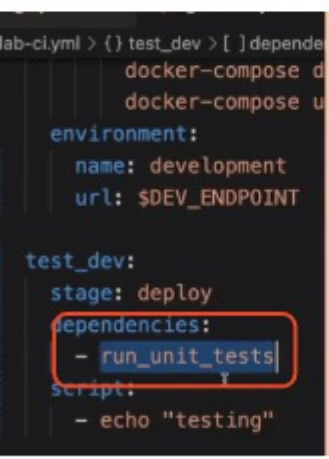
not need dependencies if you have needs

Prevent artifact download

- With empty array, you configure the job to **not download any artifacts**

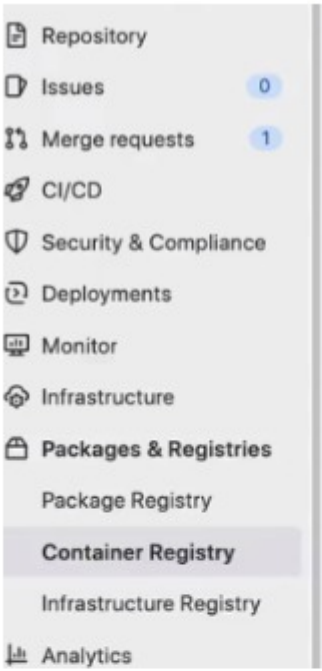
- ✅ Makes sense if you produce many artifacts, and don't need it in a job
- ✅ Speed up execution

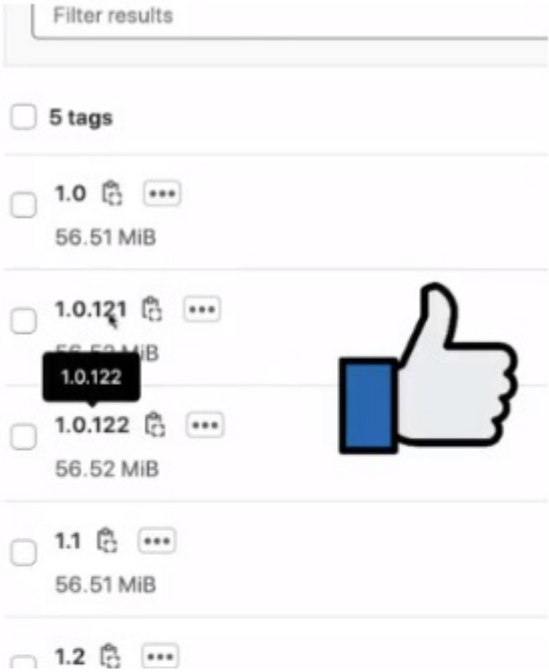
```yaml
test_dev:
  stage: deploy
  dependencies: []
  script:
  - echo "testing"
```



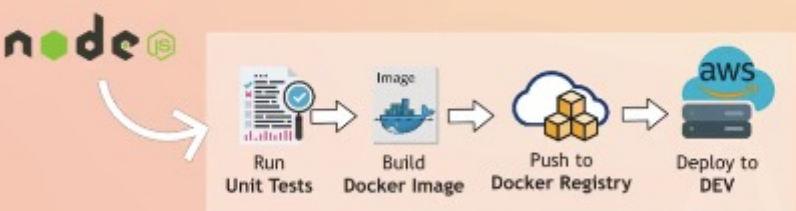
Restrict artifact download

- ✅ Download only the artifacts, you actually need in the job





Extend & Optimize Pipeline to a Real Life Pipeline



- **Dynamic Versioning:**
  Dynamically set an incremented version when we build the Docker image

Pass Env Vars as Dotenv Artifact

share artifacts between jobs -> share variables
-> dedicated feature in gitlab to pass
environment variables between the jobs

Adjustment

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

```
  - remote
before_script:
  - export PACKAGE_JSON_VERSION=$(cat app/package.json | jq -r .ver
  - export VERSION=$PACKAGE_JSON_VERSION.$CI_PIPELINE_IID
  - echo "VERSION=$VERSION" >> build.env
script:
  - docker build -t $IMAGE_NAME:$VERSION .
artifacts:
  reports:
    dotenv: build.env
```

Usage of these env vars

- Collected variables are registered as runtime-created variables of the job
- Jobs in later stages can use the variable in scripts

Variables cannot be used to configure a pipeline, but **only in job scripts**

```yaml
build_image:
  stage: build
  tags:
    - ec2
    - shell
    - remote
  before_script:
    - export PACKAGE_JSON_VERSION=$(cat app/package.json | jq -r .ver
    - export VERSION=$PACKAGE_JSON_VERSION.$CI_PIPELINE_IID
    - echo "VERSION=$VERSION" >> build.env
  script:
    - docker build -t $IMAGE_NAME:$VERSION .
  artifacts:
    reports:
      dotenv: build.env

push_image:
  stage: build
  needs:
    - build_image
  tags:
    - ec2
    - shell
    - remote
  before_script:
    - echo "$CI_REGISTER_USER $CI_REGISTRY_PASSWARD"
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD  $CI
  script:
    - docker push $IMAGE_NAME:$VERSION
```

push just use version variable

optimieren pipeline make faster

Caching in Gitlab ci/cd

Caching in GitLab CI/CD

Pipeline

job_A ⇨ job_B ⇨ job_C ⇨ job_D

- Each job runs in its **own isolated environment**

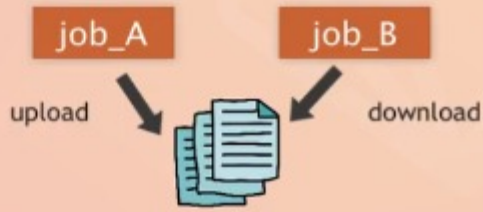
RUNNER job_B

- Starting from a fresh new environment
  - ✓ Jobs don't affect each other
  - ✓ No unexpected side-effects

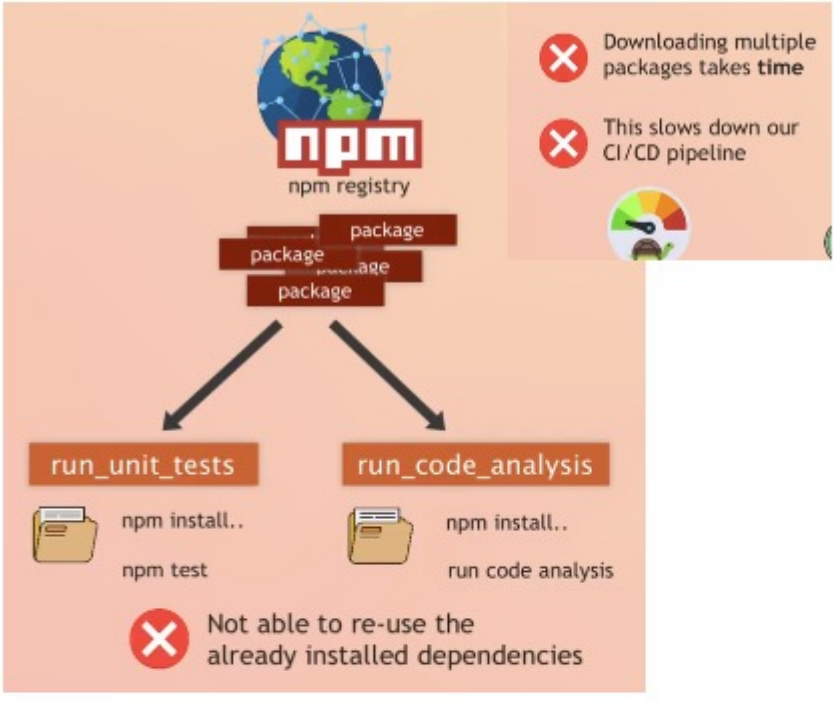
Cases where you need to **share files between jobs**

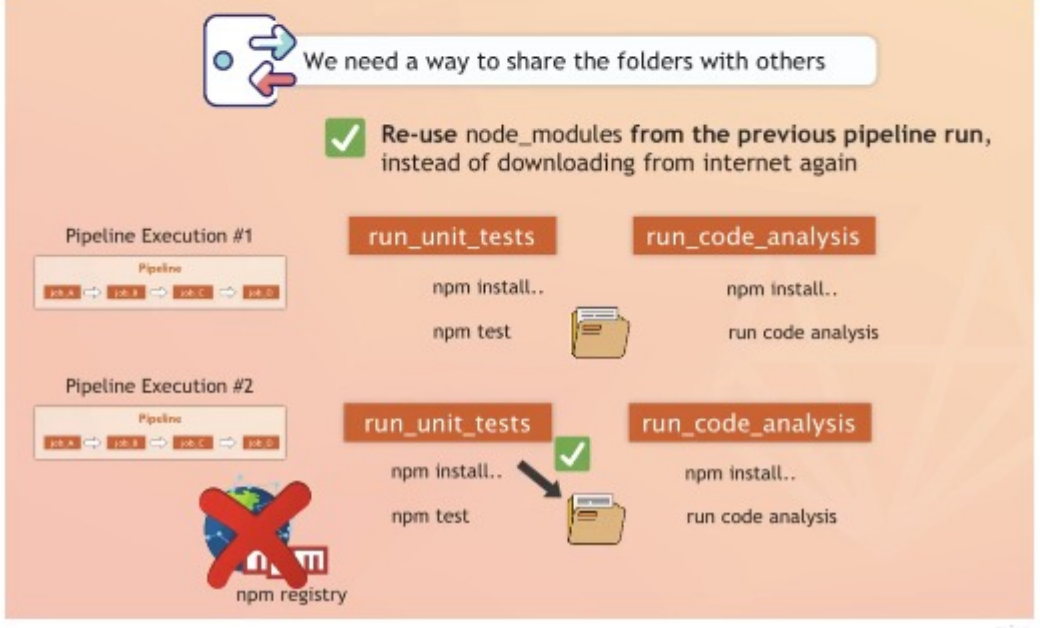
Artifacts

- Access artifacts in GitLab UI after pipeline execution
- Pass artifacts from one job to another (within the pipeline)



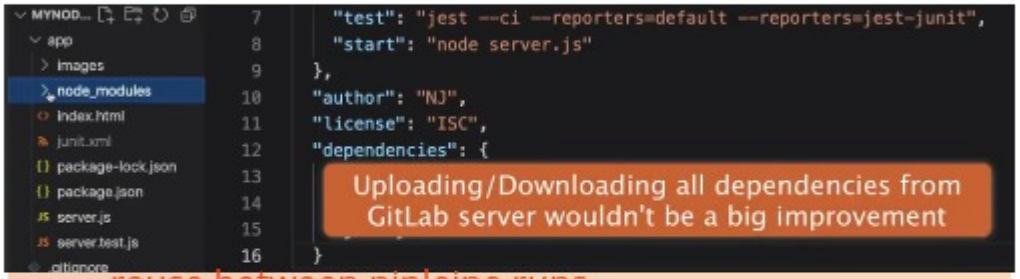
run unit test-> first install npm install
-> saved in node-modules



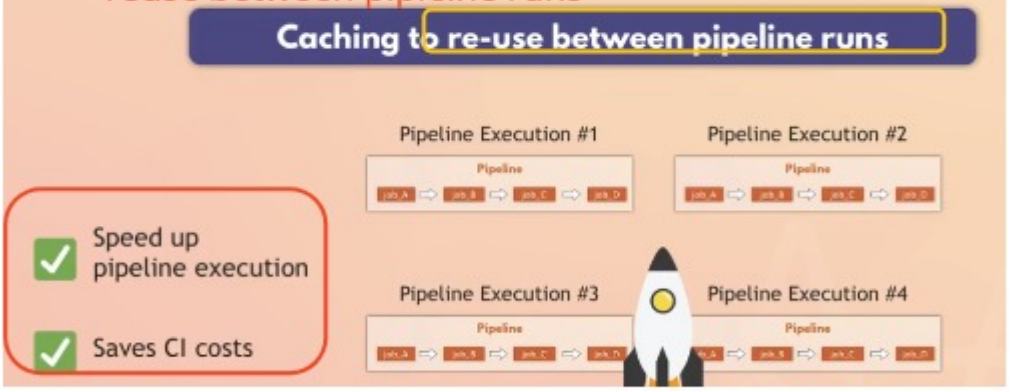


Artifacts

- Job artifacts get **uploaded** from Runner to the GitLab instance
- And then get **downloaded** from the GitLab instance



reuse between pipleine runs



git chache

speed up pipeline execution and save ci cost

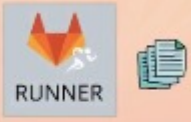
Artifacts vs Cache

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!

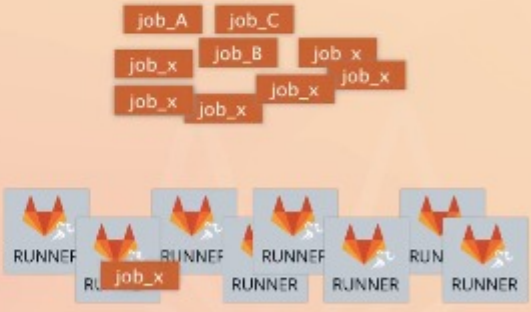


- If jobs run on the **same runner**, they can re-use the local cache on the server

Distributed Cache

Distributed Cache

For caches to work efficiently, use less runners for all your jobs



solution distributed cache

A cache would need to be created on each Runner's server

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

configure cache for pipleline

Configure a Cache

Configure in the .gitlab_ci.yml file

.gitlab-ci.yml

Using **"cache" attribute**

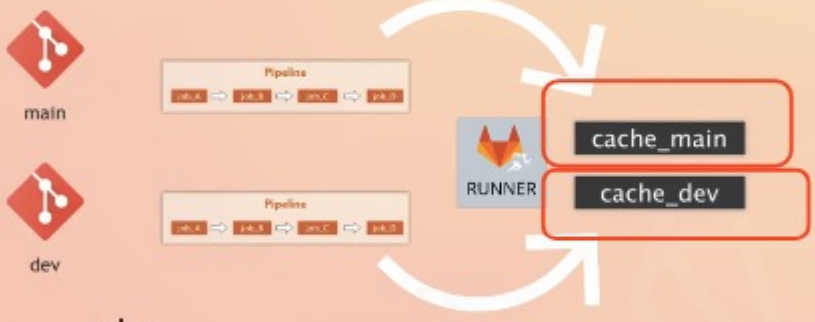
You can define a cache for each job

```
cache-job:
  script:
    - echo "This job uses a cache."
  cache:
    key: my-cache
    paths:
      - .config
```

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**
- All the jobs that run for a specific branch, will share the same cache

- Instead of using a hardcoded string, you can use a **predefined variable** or a combination of string and variable

Cache Path

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

```
run_unit_tests:
  image: node:17-alpine3.14
  stage: test
  cache:
    key:  "$CI_COMMIT_REF_NAME"
    paths:
      - app/node_modules
  tags:
```

2 different purposes

Generate the cache | Download the cache

Cache Policies

pull-push

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

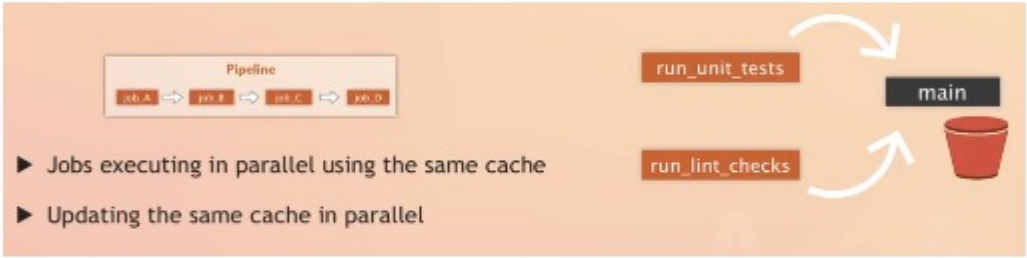
```
paths:
  - app/node_modules
policy: pull-push
```

default

Linting

- A "Linter" is a static code analysis tool
- It checks your source code for programming errors and stylistic errors

all in the test stage_> no needs -> run parallel



policies: readonly

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

Cache Policies

push

- Common to have a job, which just builds the cache
- Will only upload a cache, but never download
- Use "push" policy in that case

```
    junit: app/junit.xml

run_lint_check:
  image: node:17-alpine3.14
  stage: test
  cache:
    key:  "$CI_COMMIT_REF_NAME"
    paths:
      - app/node_modules
    policy: pull
  tags:
    - ec2
    - docker
    - remote
  before_script:
    - cd app
    - npm install
  script:
    - echo "running lint check"
```

Best Practice

Your job should never depend on a cache to be available

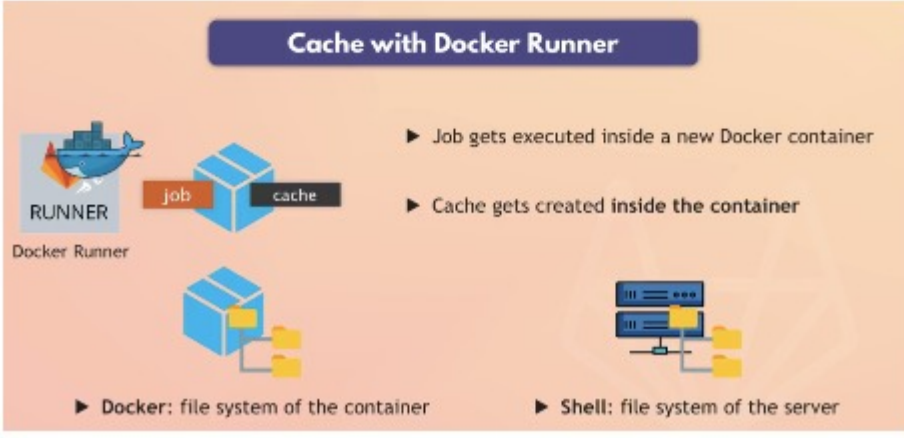
Caching is an optimization, but it isn't guaranteed to always work

Availability of the cache depends on:

- The runner's executor type
- Whether different runners are used ot pass the cache between jobs

configure volume for docker executor

job cached in the docker container with docker executor



Cache with Docker Runner

Docker Runner

- Job gets executed inside a new Docker container
- Cache gets created **inside the container**
- When job finishes, the container is removed

Containers are ephemeral

- Containers are intended to be created and destroyed often
- By default, data inside is **not persisted**

So the cache inside the container is deleted with the container

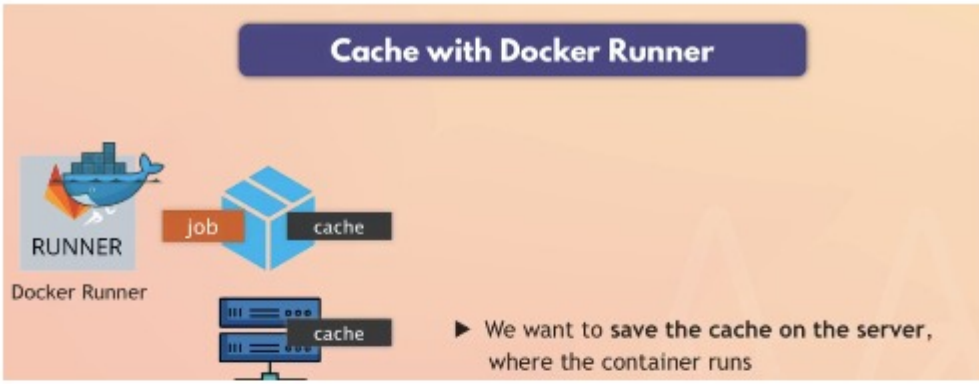
Cache with Docker Runner

Docker Runner

- Files are only stored inside the container, but not on the host (server)

Configure a Docker Volume

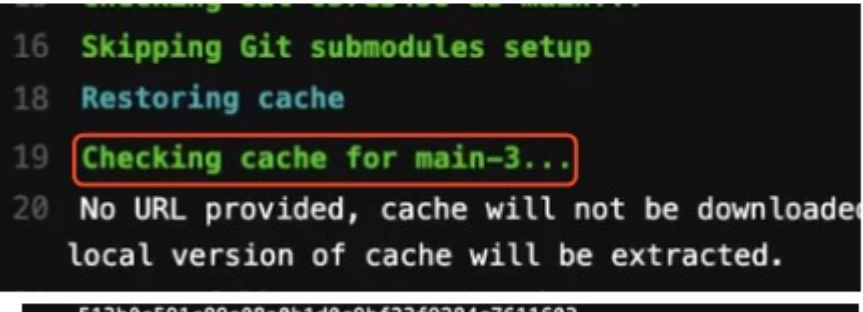
- Volumes persist data generated by containers on the host machine

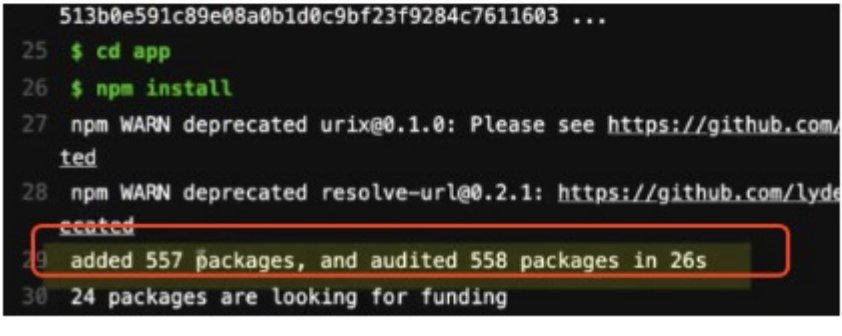


change the configuration of docker runner
directory: /etc/gitlab-runner(config.toml

```
ubuntu@ip-172-31-46-28:/$ sudo cat etc/gitlab-runner/config.toml
concurrent = 1
check_interval = 0

[session_server]
  session_timeout = 1800

[[runners]]
  name = "ec2-runner"
  url = "https://gitlab.com"
  token = "Nudsg1ms2Jxe29YxMPZg"
  executor = "shell"
  [runners.custom_build_dir]
  [runners.cache]
    [runners.cache.s3]
    [runners.cache.gcs]
    [runners.cache.azure]

[[runners]]
  name = "ec2-docker-runner"
  url = "https://gitlab.com/"
  token = "WcFGaXsgL5dj3yrQfycd"
  executor = "docker"
  [runners.custom_build_dir]
  [runners.cache]
    [runners.cache.s3]
    [runners.cache.gcs]
    [runners.cache.azure]
  [runners.docker]
    tls_verify = false
    image = "alpine:3.15.1"
    privileged = false
    disable_entrypoint_overwrite = false
    oom_kill_disable = false
    disable_cache = false
    volumes = ["/cache"]
    shm_size = 0
ubuntu@ip-172-31-46-28:/$
```


[[runners]]
name = "ec2-docker-runner"
url = "https://gitlab.com/"
token = "WcFGaXsgL5dj3yrQfycd"
executor = "docker"
cache_dir = "/cache"
[runners.custom_build_dir]
[runners.cache]
[runners.cache.s3]
[runners.cache.gcs]
[runners.cache.azure]
[runners.docker]
tls_verify = false
image = "alpine:3.15.1"
privileged = false
disable_entrypoint_overwrite = false
oom_kill_disable = false
disable_cache = false
volumes = ["/cache"]
shm_size = 0


16 Skipping Git submodules setup
18 Restoring cache
19 Checking cache for main-3...
20 No URL provided, cache will not be downloaded
local version of cache will be extracted.


513b0e591c89e08a0b1d0c9bf23f9284c7611603 ...
25 $ cd app
26 $ npm install
27 npm WARN deprecated urix@0.1.0: Please see https://github.com/
ted
28 npm WARN deprecated resolve-url@0.2.1: https://github.com/lyde
ecated
29 added 557 packages, and audited 558 packages in 26s
30 24 packages are looking for funding

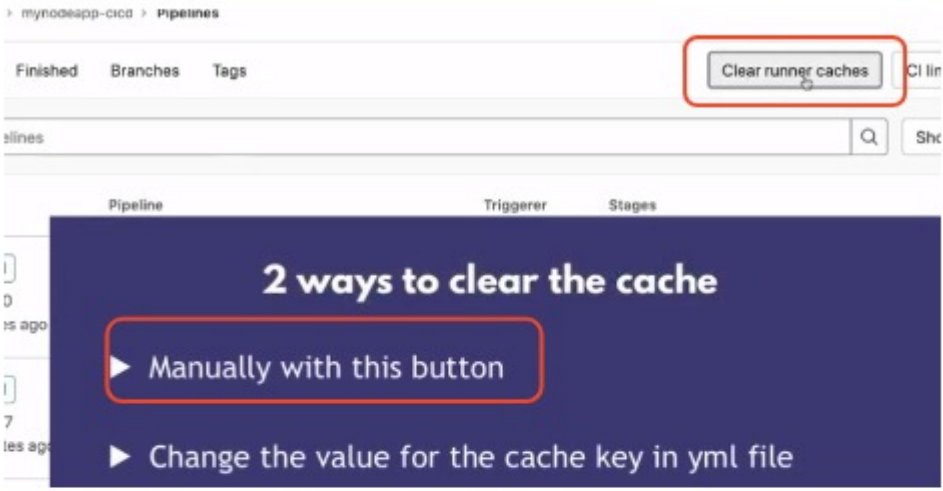
```
51 Ran all test suites.
53 Saving cache for successful job
54 Creating cache main-3...
55 app/node_modules: found 8352 matching files and direc
56 No URL provided, cache will be not uploaded to shared
   stored only locally.
57 Created cache
59 Uploading artifacts for successful job
60 Uploading artifacts...
61 app/junit.xml: found 1 matching files and directories
```

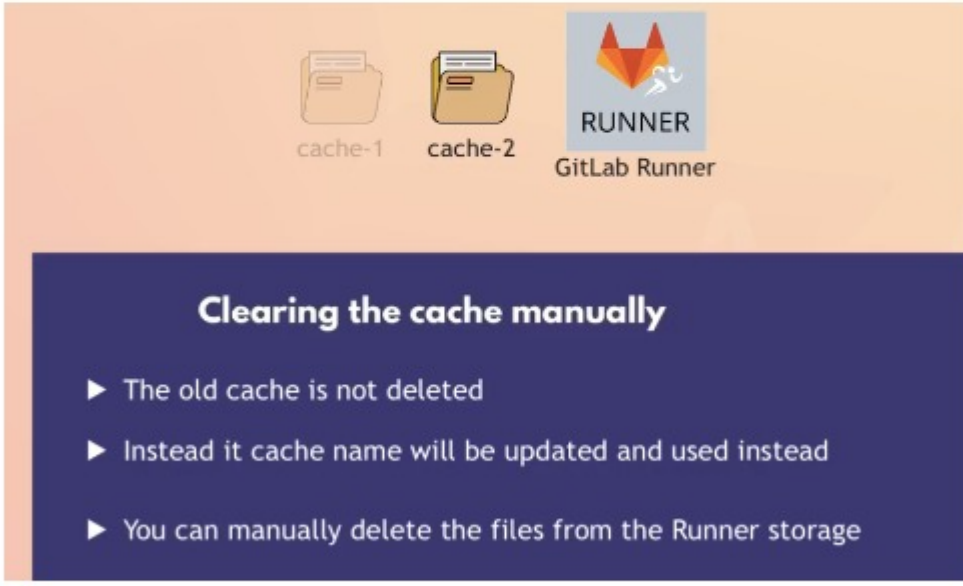
```
Checking cache for main-3...
No URL provided, cache will not be downloaded from shar
local version of cache will be extracted.
Successfully extracted cache
Executing "step_script" stage of the job script
Using docker image sha256:af611d49779a0d9371d6a4101d750
18a50d699 for node:17-alpine3.14 with digest node@sha256
513b0e591c89e08a0b1d0c9bf23f9284c7611603 ...
$ cd app
$ npm install
up to date, audited 558 packages in 3s
24 packages are looking for funding
```

in lint not download anymore

```
34 Run `npm audit` for details.
35 $ echo "Running lint checks"
36 Running lint checks
38 Saving cache for successful job
39 Not uploading cache main-3 due to policy
41 Cleaning up project directory and file based variables
43 Job succeeded
```

clearing cachce





Internal cache name

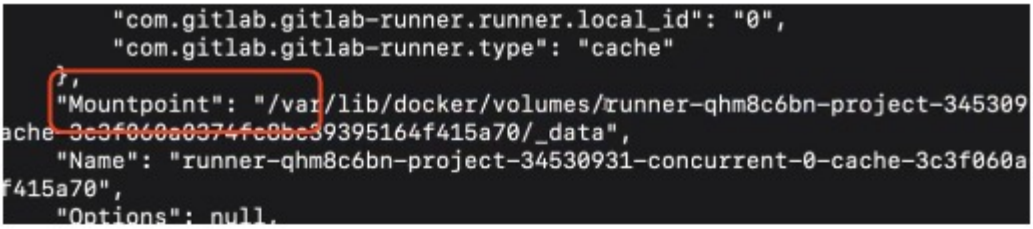
- Each time you clear the cache manually, the internal cache name is updated
- Old cache is not deleted!
- Format "*cache-index*" , index is incremented by one

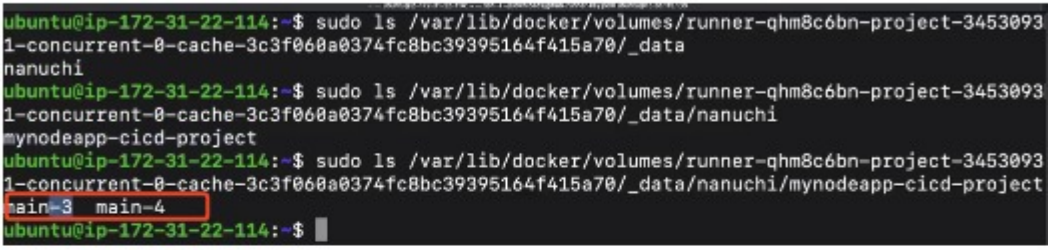
```
Checking out 23c1f581 as main...
Removing app/node_modules/
Skipping Git submodules setup
Restoring cache
Checking cache for main-2-protected...
No URL provided, cache will not be downloaded from s
version of cache will be extracted.
Successfully extracted cache
Executing "step_script" stage of the job script
Using docker image sha256:b20b24e39dda538a41dfa3e9fc
```

where is the cache physically stored

```
ubuntu@ip-172-31-22-114:~$ docker volume ls
DRIVER    VOLUME NAME
local     runner-qhm8c6bn-project-34530931-concurrent-0-cache-3c3f060a0374fc8bc39395164
5a70
local     runner-qhm8c6bn-project-34530931-concurrent-0-cache-c33bcaa1fd2c77edfc3893b41
cea8
```

```
runner-qhm8c6bn-project-34530931-concurrent-0-cache-c33bca

-172-31-22-114:~$ docker volume inspect runner-qhm8c6bn-pro
e-3c3f060a0374fc8bc39395164f415a70
```

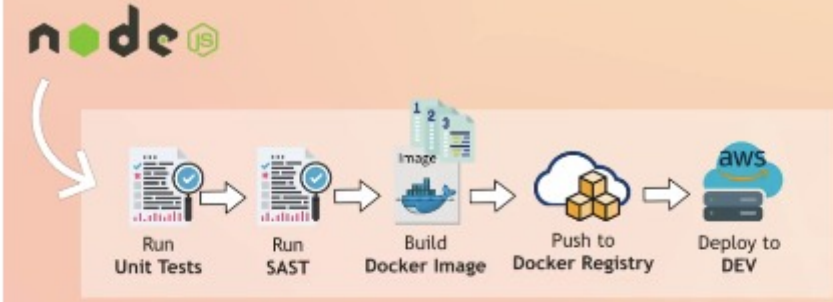




- All caches defined for a job are archived in a single cache.zip file

read gitlab ci/cd cache and artifact

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

Auto DevOps

euough automated test

unit test:

- most basic
- validate that individual parts of the app funtion
- so only specific units, instead of the entire programm

functional testing

- vaöidate the functionalit of the sofeware

integration testing

- validate that individual parts of code work together
- frontend and backend communication
- or microservice application works properly with all those self-contained services

Functionality Related Tests

Unit Testing | UI Testing | Functional Testing

Integration Testing | Regression Testing | API Testing

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

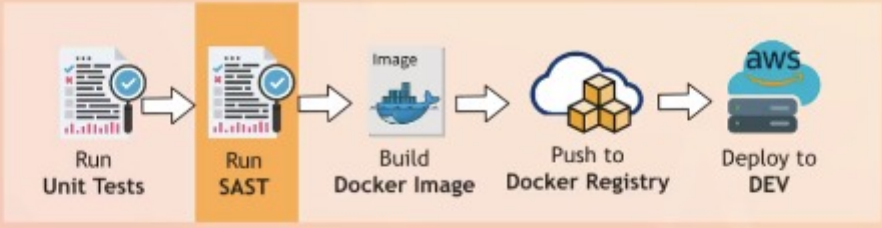
When to run which tests?

✅ Less expensive to fix vulnerabilities

❌ More expensive to fix vulnerabilities

- Executed on the source code
- Execute early and often
- Require a deployed application
- Able to find run-time errors

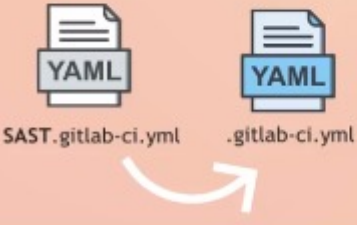
Focus on running SAST Tests



- ✓ Use **existing** GitLab jobs
- ✓ Which have everything pre-configured
- ✓ To include it in your own CI/CD pipeline

Learn a new GitLab functionality & concept

How does this work?



- ▶ Job templates for **different tasks** and for **different programming languages & technolgoies**

Out-of-the-box functionality!

What is a CI/CD template?

*.gitlab-ci.yml

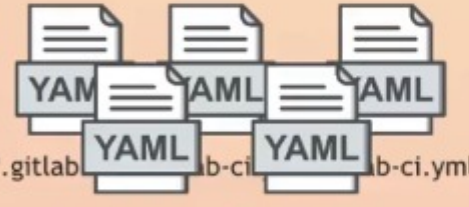
.gitlab-ci.yml

Generic

- Goal is to re-use a template
- Every change must be backwards compatible, because templates are included in other pipelines
- You can pass parameters to customize the behavior

What is a CI/CD template?

GitLab



*.gitlab-ci.yml *.gitlab-ci.yml

- GitLab engineers wrote these job templates
- You can check out the available templates on gitlab.com

gitlab.com/gitlab-org/gitlab-foss/tree/master/lib/gitlab/ci/templates

python ML Y News TUM AI Handelsblatt... GitLab CICD 从... vue Stanford MLS... PONS

Next Menu Search GitLab

GitLab FOSS

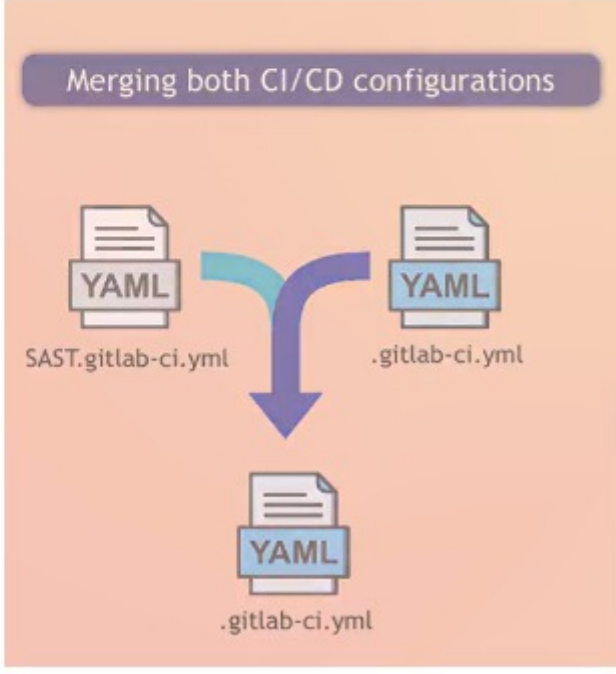
- Project information
- Repository
  - Files
  - Commits
  - Branches
  - Tags
  - Contributors
  - Graph
  - Compare
  - Locked Files
- Issues 0
- Merge requests 1
- Deployments
- Packages & Registries
- Monitor
- Analytics
- Snippets

GitLab.org > GitLab FOSS > Repository

master | gitlab-foss / lib / gitlab / ci / templates / + | Lock | History | Find file | Web IDE

Add latest changes from gitlab-org/gitlab@master
GitLab Bot authored 3 hours ago

Name	Last commit
..	
AWS	Add latest changes from gitlab-org/gitlab@master
Jobs	Add latest changes from gitlab-org/gitlab@master
Pages	Add latest changes from gitlab-org/gitlab@master
Security	Add latest changes from gitlab-org/gitlab@master
Terraform	Add latest changes from gitlab-org/gitlab@master
Verify	Add latest changes from gitlab-org/gitlab@master
Workflows	Add latest changes from gitlab-org/gitlab@master



ter gitlab-foss / lib / gitlab / ci / templates / Jobs / SAST.gitlab-ci.yml Find file

Add latest changes from gitlab-org/gitlab@master

GitLab Bot authored 2 months ago

SAST.gitlab-ci.yml 7.55 KiB Edit Lock

```yaml
# Read more about this feature here: https://docs.gitlab.com/ee/user/application_security/sast/
#
# Configure SAST with CI/CD variables (https://docs.gitlab.com/ee/ci/variables/index.html).
# List of available variables: https://docs.gitlab.com/ee/user/application_security/sast/index.ht

variables:
  # Setting this variable will affect all Security templates
  # (SAST, Dependency Scanning, ...)
  SECURE_ANALYZERS_PREFIX: "registry.gitlab.com/security-products"
  SAST_IMAGE_SUFFIX: ""

  SAST_EXCLUDED_ANALYZERS: ""
  SAST_EXCLUDED_PATHS: "spec, test, tests, tmp"
  SCAN_KUBERNETES_MANIFESTS: "false"

sast:
  stage: test
  artifacts:
    reports:
      sast: gl-sast-report.json
  rules:
    - when: never
  variables:
    SEARCH_MAX_DEPTH: 4
  script:
    - echo "$CI_JOB_NAME is used for configuration only, and its script should not be executed"
    - exit 1

.sast-analyzer:
  extends: sast
```

include sast template into our pipeline code

use feature include

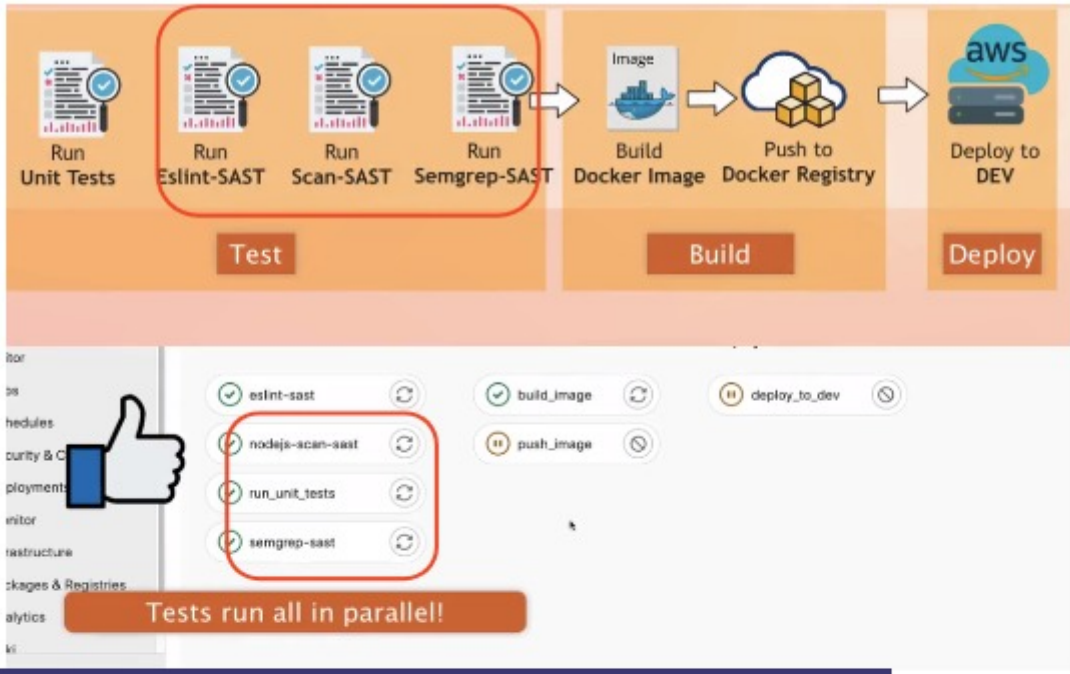
"include"

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

"include"

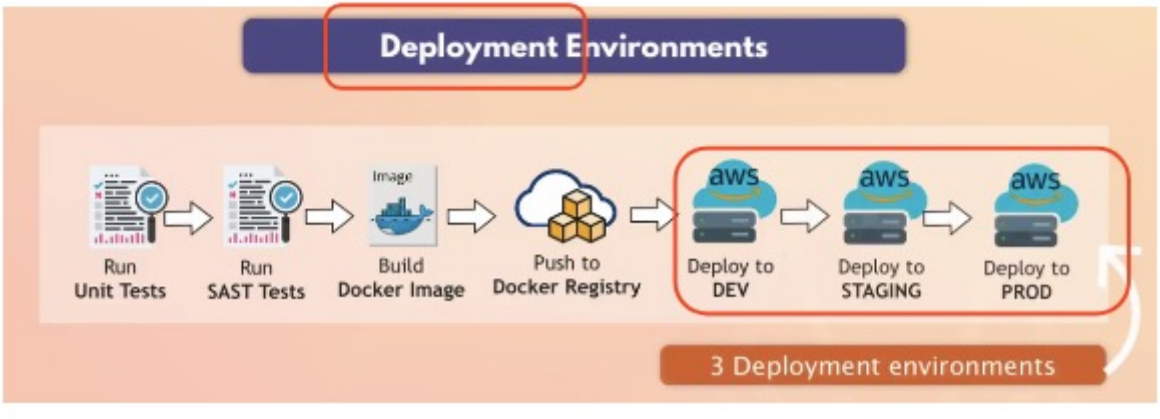
- You can nest up to 100 includes
- Position of include keyword is irrelevant

```
      docker-compose down &&
      docker-compose up -d"
  environment:
    name: development
    url: $DEV_ENDPOINT

include:
  - template: Jobs/SAST.gitlab-ci.yml

    paths:
      - app/junit.xml
    reports:
      junit: app/junit.xml

sast:
  stage: test

build image:
```



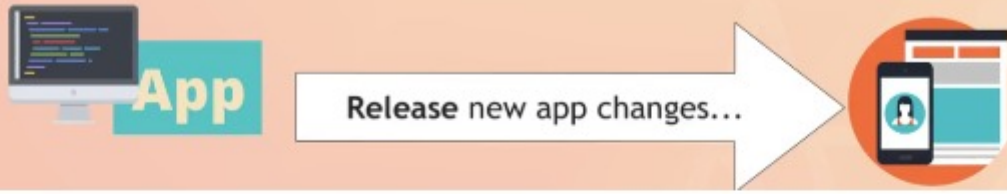
Pipeline Templates

- Provides an end-to-end CI/CD workflow
- It's not included in another main pipeline configuration, but rather used by itself



Deployment Environments

- You need to make sure that new changes **don't break existing logic**



Promote to Staging and Production

Release in Stages


DEVELOPMENT

promote

STAGING

promote

App

v1.3

App

v1.3

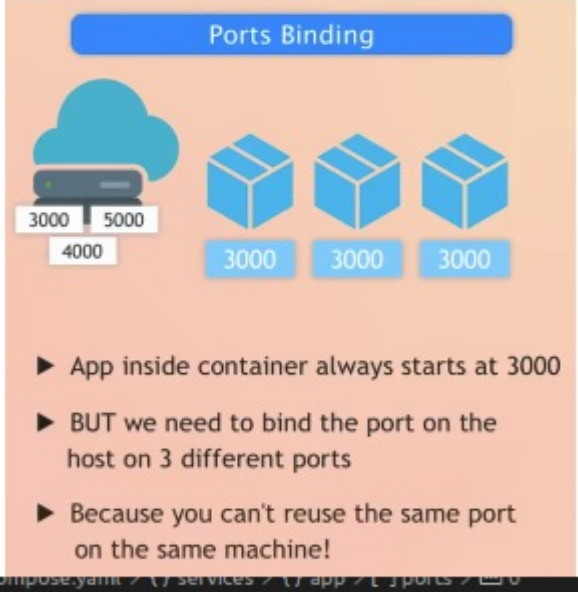
- Functional tests
- Integration tests
- SAST tests

- Performance tests
- DAST tests

not attackt

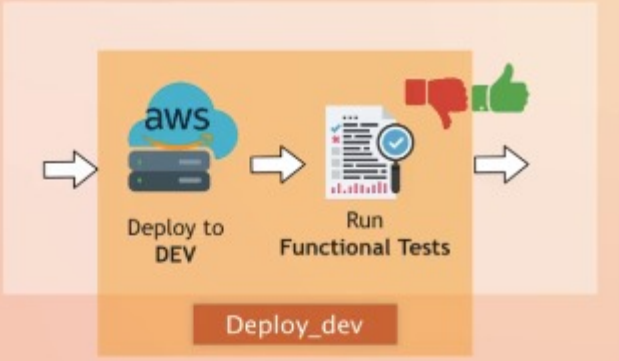
dev-staging-prod
we need staging server, prod server -> our case: use dev server for all three depolyment. Using different port

make host port configurable



all.json

```
version: "3.3"
services:
    app:
        image: ${DC_IMAGE_NAME}:${DC_IMAGE_TAG}
        ports:
            - ${APP_PORT}:3000
```



- After deploying the app to the dev environment
- We run functional tests, before deploying to the staging environment

```yaml
run_functional_tests:
  stage: depoly_dev
  needs:
    - deploy_to_dev
  script:
    - echo "running test"

deploy_to_staging:
  stage:
    depoly_staging
  tags:
    - ec2
    - shell
    - remote
  before_script:
    - chmod 400 $SSH_PRIVATE_KEY
  script:
    - scp -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ./docker-co
    - ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ubuntu@$STA
      docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD  $CI
      export DC_IMAGE_NAME=$IMAGE_NAME &&
      export DC_IMAGE_TAG=$VERSION &&
      export DC_APP_PORT=4000
      docker-compose down &&
      docker-compose up -d"
  environment:
    name: staging
    url: $STAGING_ENDPOINT
```

configure custom container names

docker container consists three different parts:

- ubuntu_app_1

```
ubuntu@ip-172-31-36-31:~$ docker ps
CONTAINER ID   IMAGE                                                                                  CO
          CREATED         STATUS         PORTS
S
629c569218f3   registry.gitlab.com/nanuchi/mynodeapp-cicd-project:1.0.144   "d
oint.s…"   3 minutes ago   Up 3 minutes   0.0.0.0:3000->3000/tcp, :::3000->300
tu_app_1
ubuntu@ip-172-31-36-31:~$ ls
docker-compose.yaml
ubuntu@ip-172-31-36-31:~$ pwd
/home/ubuntu
ubuntu@ip-172-31-36-31:~$
```

<project name>_<service name>_<index>

By default, project name = current directory name

change name to

environment variable of docker: compose project name

Using same docker compose file we can create containers with different names

```
  - chmod 400 $SSH_PRIVATE_KEY
script:
  - scp -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY
  - ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY
    docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_P
    export COMPOSE_PROJECT_NAME=staging
    export DC_IMAGE_NAME=$IMAGE_NAME &&
    export DC_IMAGE_TAG=$VERSION &&
    export DC_APP_PORT=4000
    docker-compose down &&
    docker-compose up -d"
environment:
```

staging_app_1

```
ubuntu@ip-172-31-43-110:~$ docker ps
CONTAINER ID   IMAGE                                                                  COMMAND                  CRE
ATED              STATUS          PORTS                                                NAMES
91b3ee524944   registry.gitlab.com/yingtremendous/mynodeapp-cicd-project:1.0.91   "docker-entrypoint.s…"   2 s
econds ago         Up 1 second     0.0.0.0:4000->3000/tcp, :::4000->3000/tcp            staging_app_1
97ea21155915   registry.gitlab.com/yingtremendous/mynodeapp-cicd-project:1.0.91   "docker-entrypoint.s…"   Abo
ut a minute ago    Up 59 seconds   0.0.0.0:3000->3000/tcp, :::3000->3000/tcp            dev_app_1
ubuntu@ip-172-31-43-110:~$
```

ec2-54-93-229-194.eu-central-1.compute.amazonaws.com:4000

ML Y News TUM AI Handelsblatt... GitLab CICD M... vue Star

List of projects team is working on

test in 3000 or 4000 ports

Name: **Andrea Hill**

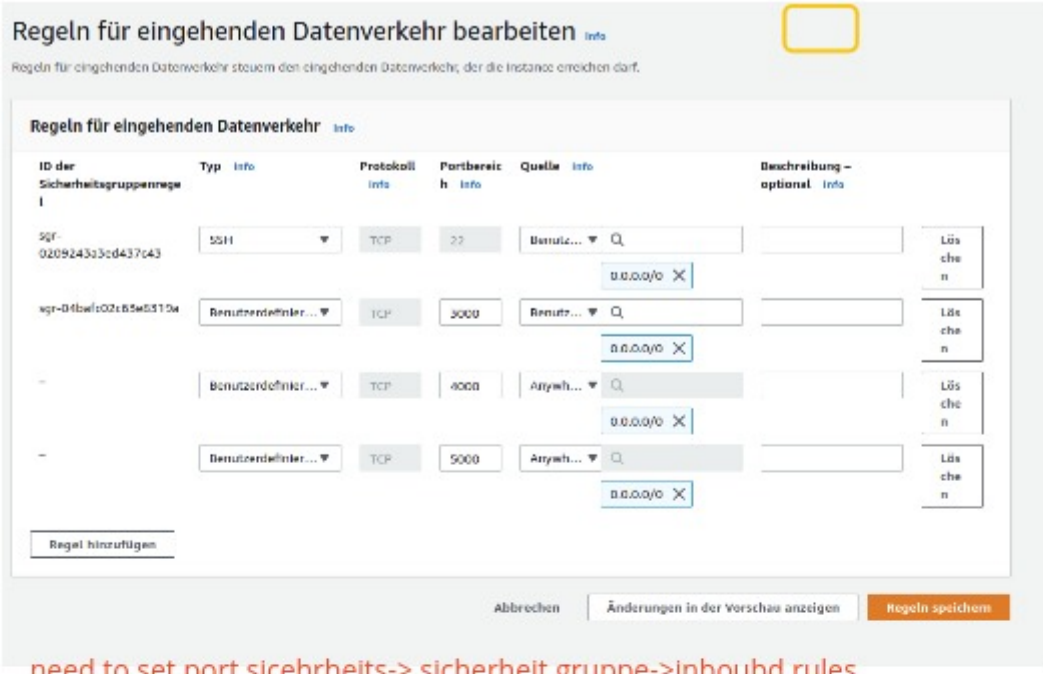
Role: **DevOps engineer**

Projects: **AWS migration, Backup Automation, GitLab Auto DevOps**

Name: **Ari Baker**

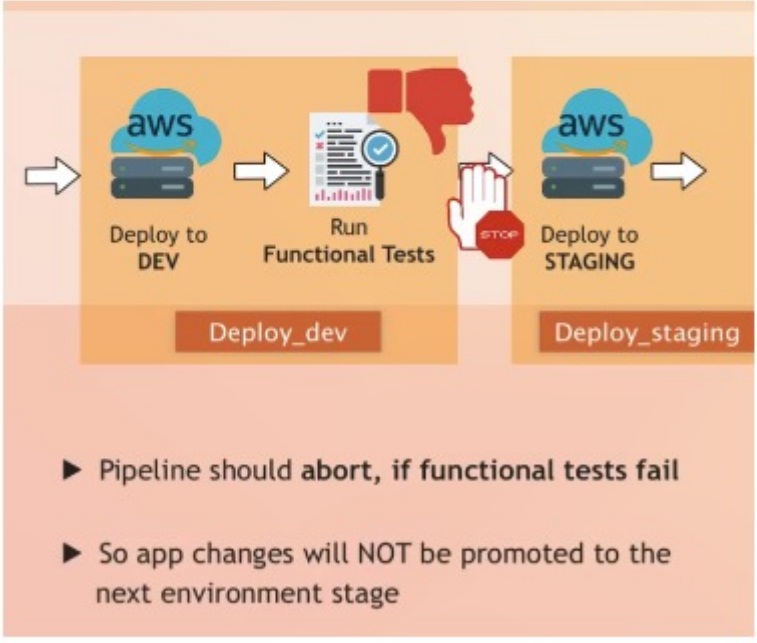
Role: **Software hahah**

Projects: **Online Shop, ERP Software**



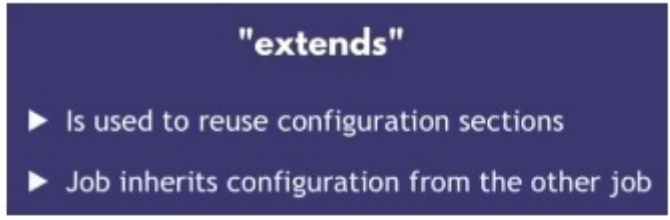
need to set port sicehrheits-> sicherheit gruppe->inboubd rules

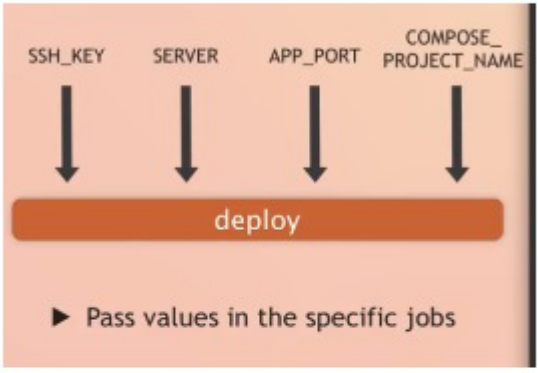
Fail functional tets to skip promoting to staging



deploy to production:

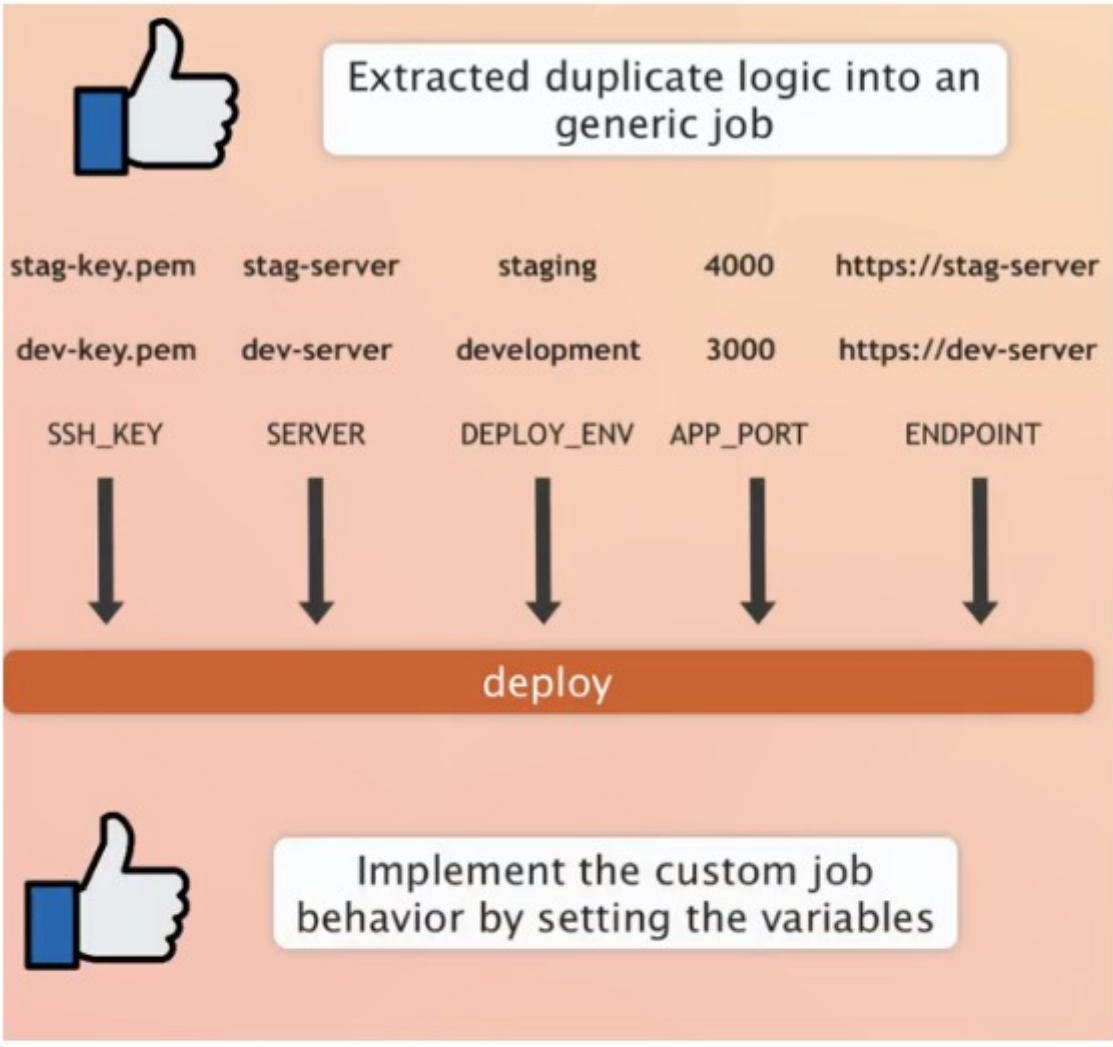
- depoly_dev and deploy_staging have similar strcuture
- avoid code duplication. reuse configuration sections
- ? how to reuse code in gitlab cicd.
- concept: extends





Use base code. using keyword extends in the specific jobs

```yaml
deploy:
  tags:
    - ec2
    - shell
    - remote
  before_script:
    - chmod 400 $SSH_PRIVATE_KEY
  variables:
    SSH_PRIVATE_KEY: ""
    SERVER_HOST: ""
    DEPOLY_ENV: " "
    APP_PORT: ""
    ENDPOINT: "
  script:
    - scp -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ./docker-co
    - ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ubuntu@$SER
      docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD  $CI
      export COMPOSE_PROJECT_NAME=$DEPOLY_ENV &&
      export DC_IMAGE_NAME=$IMAGE_NAME &&
      export DC_IMAGE_TAG=$VERSION &&
      export DC_APP_PORT=$APP_PORT &&
      docker-compose down &&
      docker-compose up -d"
  environment:
    name: $DEPOLY_ENV
    url: $ENDPOINT
```

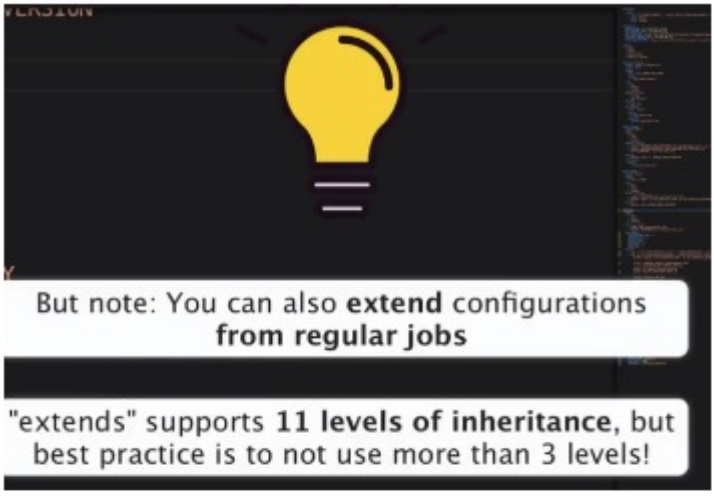


make the base job not run-> hidden job using .

```
.deploy:
  tags:
    - ec2
    - shell
```

Hide jobs

- If you start the job with a dot it's not processed by GitLab CI/CD
- You can also comment out jobs with "#" to hide a job



But note: You can also **extend** configurations **from regular jobs**

"extends" supports **11 levels of inheritance**, but best practice is to not use more than 3 levels!

when : manual

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

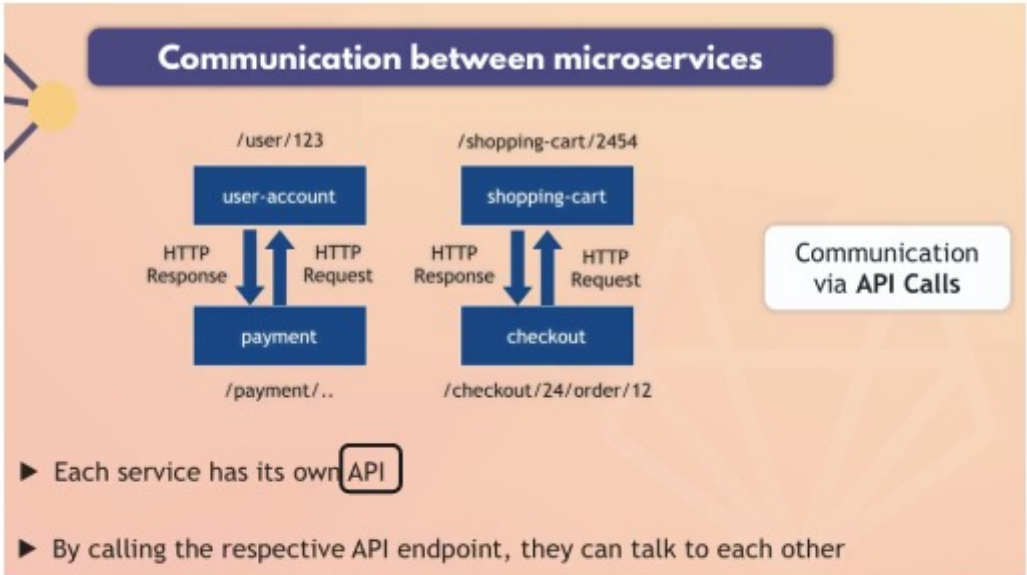
pass file variable: the content are passed into not the file paths

```yaml
variables:
  [illegible]_REGISTRY_IMAGE
  IMAGE_TAG: "1.0"
  DEV_SERVER_HOST: 54.93.229.194
  DEV_ENDPOINT: http://ec2-54-93-229-194.eu-central-1.compute.amazonaws.com:3000
  STAGING_SERVER_HOST: 54.93.229.194
  STAGING_ENDPOINT: http://ec2-54-93-229-194.eu-central-1.compute.amazonaws.com:4000
  prod_SERVER_HOST: 54.93.229.194
  prod_ENDPOINT: http://ec2-54-93-229-194.eu-central-1.compute.amazonaws.com:3000
```

```yaml
.deploy:
  tags:
    - ec2
    - shell
    - remote
  before_script:
    - echo $SSH_KEY | sed -e "s/-----BEGIN RSA PRIVATE KEY-----/&\n/" -e "s/-----END RSA PR
    - chmod 400 deploy-key.pem
    - export VERSION=$(cat version-file.txt)
  variables:
    SSH_KEY: ""
    SERVER_HOST: ""
    DEPOLY_ENV: ""
    APP_PORT: ""
    ENDPOINT: ""
  script:
    - scp -o StrictHostKeyChecking=no -i deploy-key.pem ./docker-compose.yaml ubuntu@$SERVER
    - ssh -o StrictHostKeyChecking=no -i deploy-key.pem ubuntu@$SERVER_HOST "
      docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY &&
      export COMPOSE_PROJECT_NAME=$DEPOLY_ENV &&
      export DC_IMAGE_NAME=$IMAGE_NAME &&
      export DC_IMAGE_TAG=$VERSION &&
      export DC_APP_PORT=$APP_PORT &&
      docker-compose down &&
      docker-compose up -d"
  environment:
    name: $DEPOLY_ENV
    url: $ENDPOINT
```

```yaml
deploy_to_staging:
  extends: .deploy
  stage: depoly_staging
  variables:
    SSH_KEY: $SSH_PRIVATE_KEY
    SERVER_HOST: $STAGING_SERVER_HOST
    DEPOLY_ENV: staging
    APP_PORT: 4000
    ENDPOINT: $STAGING_ENDPOINT

run_performance_tests:
  stage: depoly_staging
  needs:
    - deploy_to_staging
  script:
    - echo "running performance test"

deploy_to_prod:
  extends: .deploy
  stage: depoly_prod
  variables:
    SSH_KEY: $SSH_PRIVATE_KEY
    SERVER_HOST: $prod_SERVER_HOST
    DEPOLY_ENV: production
    APP_PORT: 5000
    ENDPOINT: $prod_ENDPOINT
  when: manual
```

Monolith-> microservices




```
ports:
  - containerPort: 8080
env:
  - name: PORT
    value: "8080"
  - name: SHIPPING_SERVICE_ADDR
    value:  "shippingservice:50051"
  - name: PRODUCT_CATALOG_SERVICE_ADDR
    value: "productcatalogservice:3550"
  - name: CART_SERVICE_ADDR
```

Communication between microservices

user-account

shopping-cart

payment

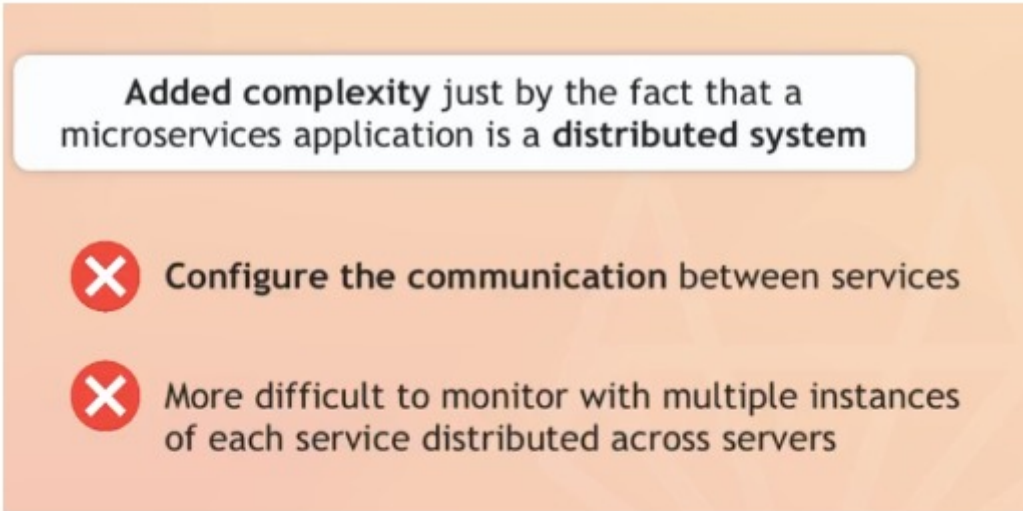
checkout

- Each microservice team can choose its own tech stack
- Each team can **develop the service independently**, without affecting others

Downsides of Microservices Architecture

Added complexity just by the fact that a microservices application is a **distributed system**

Configure the communication between services



CICD pipeline for microservcies
how to configure CICD pipeline for microservice
Application components are developed and deployed separately
how do we manage the code in the gitlab?

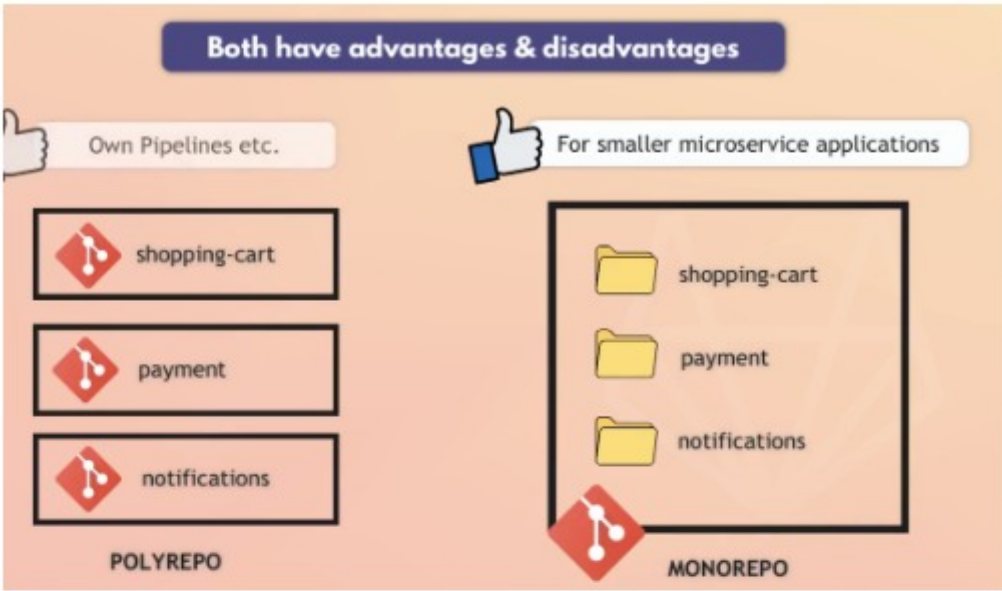
- monorepo : 1 git repo that contains many projects
    - how to structure multiple applications in one repo?
        - 1 code base with folders
        - a directory for each service/project
        - -> make the code manage and development easier
        - -> clone and work only with 1 repo
        - -> changes can be tracked together tested together and released together
        - ->share code and configuration
    - challenges:
        - tight coupling of projects
        - easier to break this criterion and develop more tightly coupled code
        - big source code, means git interactions becomes slow
        - additon logic necesary to make sure only service is built and deployed which had changes
        - example: payment changed, pipeline detect just payment and build
        - all projects and teams are affected if there is some issue
        -
- polyrepo

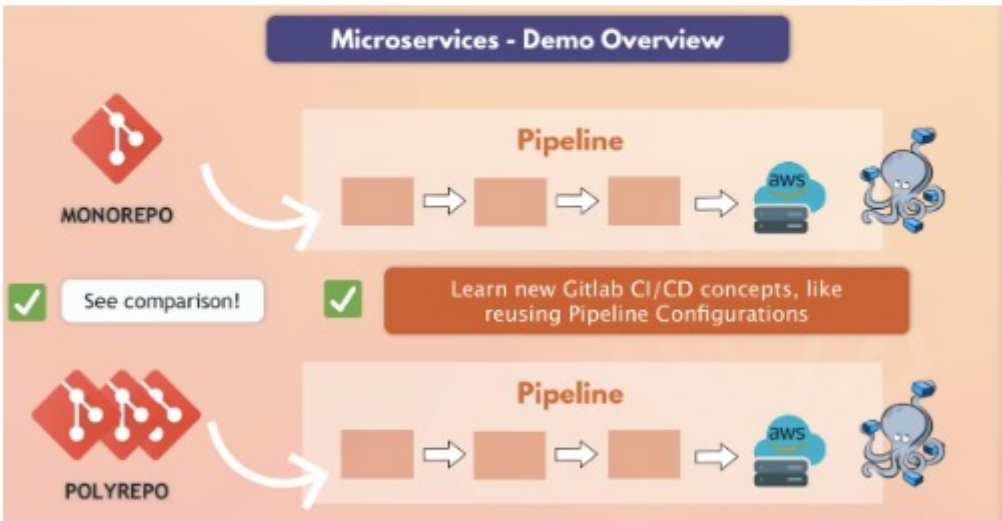
Polyrepo:

- 1 repo for each service
  - code is completely isolated
  - clone and work on them separately
  - gitlab groups
    - group runners
  - helps to keep an overview
  - create shared secrets cicd variables, runners
- CI/CD for polyrepo
  - own pipeline for each repo
- challenges:

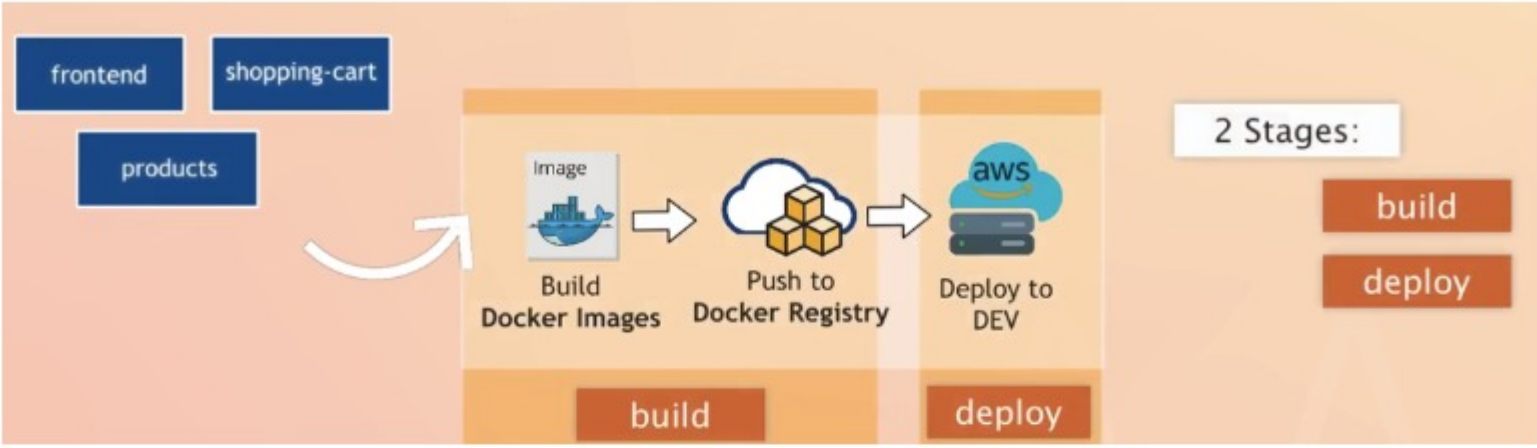
Downsides of Polyrepo

- Cross-cutting changes is more difficult
- Changes spreaded across projects must be submitted as separate Merge requests instead of having a single, atomic MR
- Switching between projects tedious
- Searching, testing and debugging is more difficult
- Sharing resources more difficult

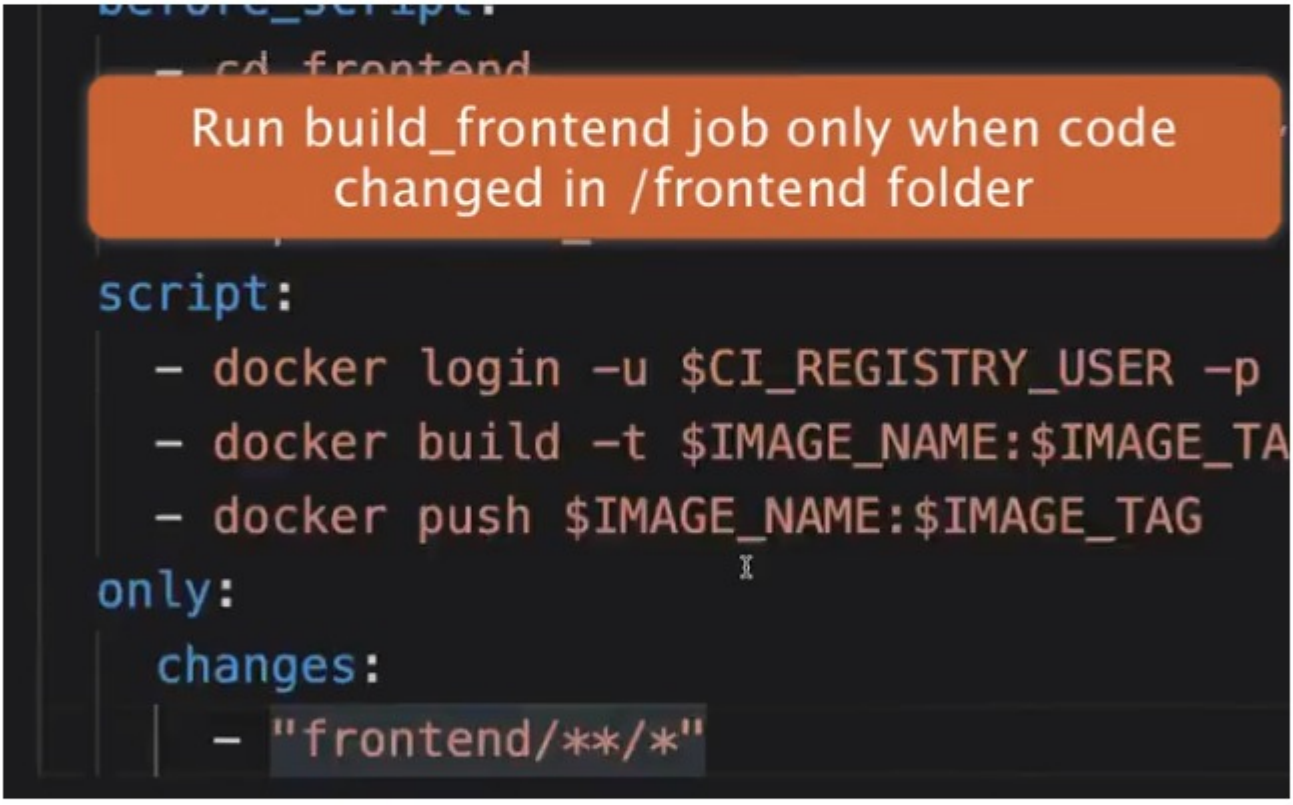

Both have advantages & disadvantages
Own Pipelines etc.
For smaller microservice applications
shopping-cart
payment
notifications
POLYREPO
shopping-cart
payment
notifications
MONOREPO


Microservices - Demo Overview
MONOREPO
Pipeline
aws
See comparison!
Learn new Gitlab CI/CD concepts, like reusing Pipeline Configurations
POLYREPO
Pipeline
aws

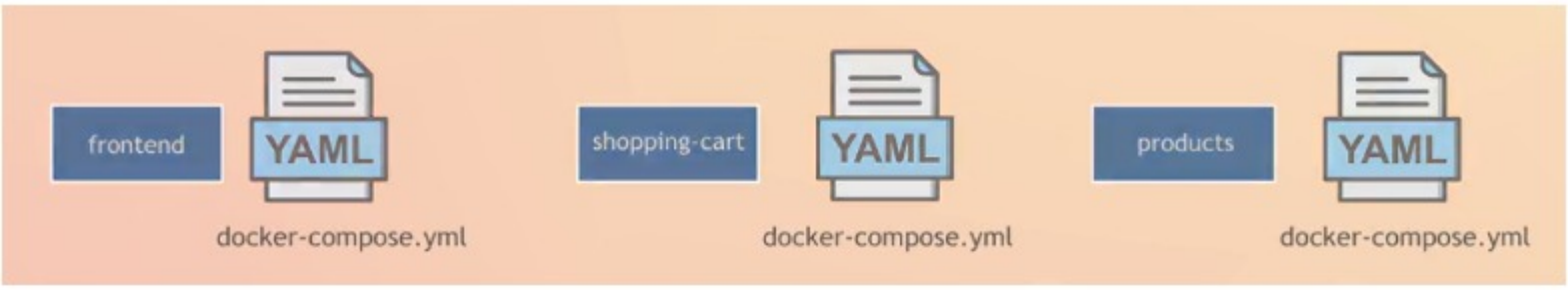
1. use runner from other project
2. prepare deployment environment depolyment server
    a. create a new EC2 instance
    b. install docker docker compose, change user group
    c. save ssh private key

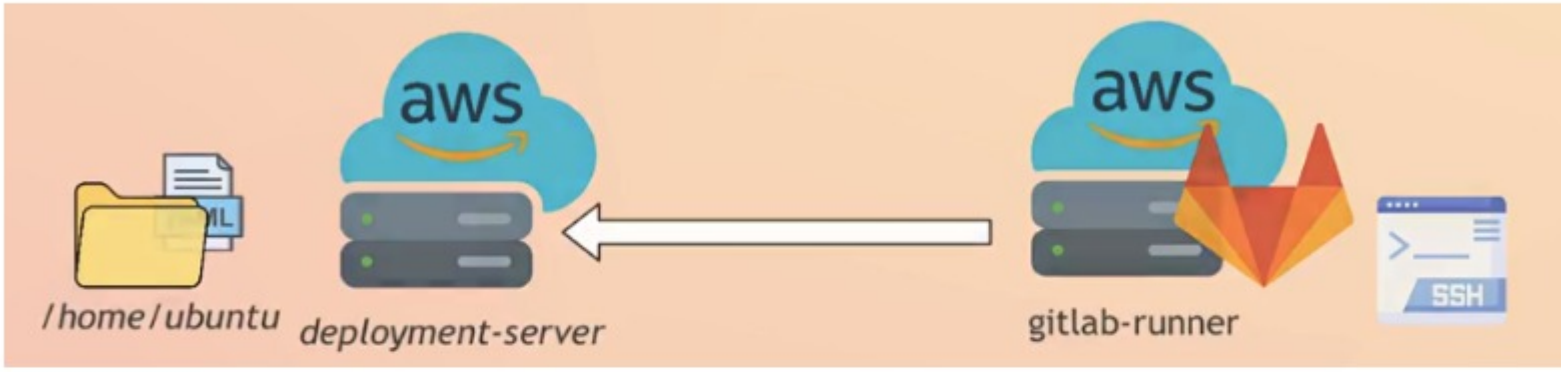


add build job for each service
configuration: so that service separate build-> keyword: only

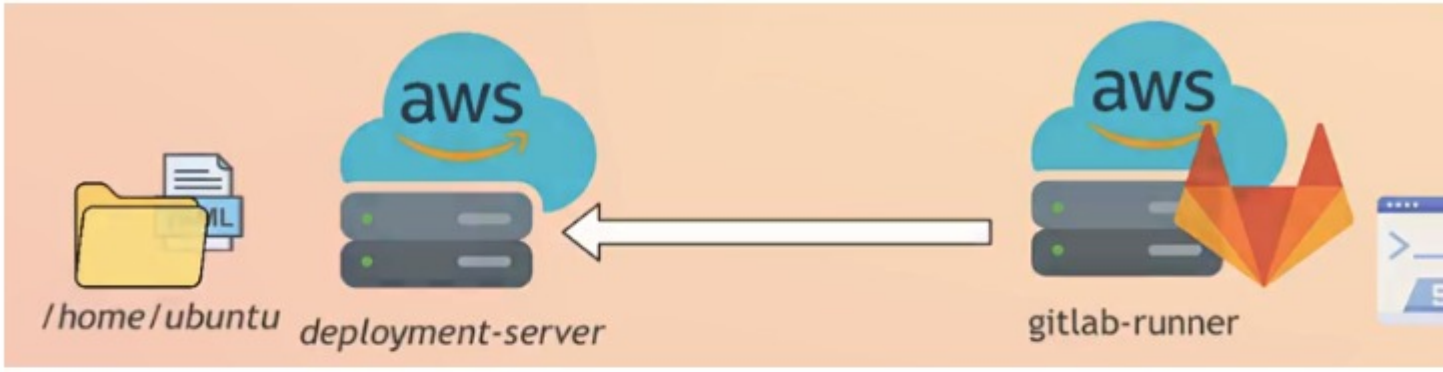


```yaml
.build:
  stage: build
  tags:
    - ec2
    - remote
    - shell
  variables:
    MICRO_SERVICE: ""
    SERVICE_TAG: ""
  before_script:
    - cd $MICRO_SERVICE
    - export IMAGE_NAME=$CI_REGISTRY_IMAGE/microservice/$MICRO_SERVICE
    - export IMAGE_TAG=$SERVICE_TAG
  script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD  $CI_REGISTRY
    - docker build -t $IMAGE_NAME:$IMAGE_TAG .
    - docker push $IMAGE_NAME:$IMAGE_TAG

build_frontend:
  extends: .build
  variables:
    MICRO_SERVICE: frontend
    SERVICE_TAG: "1.3"
  only:
    changes:
      - "frontend/**/*"
```






```yaml
.deploy:
  stage: deploy
  tags:
    - ec2
    - remote
    - shell
  variables:
    MICRO_SERVICE: ""
    SERVICE_VERSION: ""
    APP_PORT: ""
  before_script:
    - chmod 400 $SSH_PRIVATE_KEY
    - export IMAGE_NAME=$CI_REGISTRY_IMAGE/microservice/$MICRO_SERVICE
    - export IMAGE_NAME=$SERVICE_VERSION
  script:
    - scp -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ./docker-compose.yaml ubuntu@$DEPLOYMENT_
    - ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ubuntu@$DEPLOYMENT_SERVER_HOST "
      docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY &&
      export COMPOSE_PROJECT_NAME=$MICRO_SERVICE &&
      export DC_IMAGE_NAME=$IMAGE_NAME &&
      export DC_IMAGE_TAG=$IMAGE_NAME &&
      export DC_APP_PORT=$APP_PORT &&
      docker-compose down &&
      docker-compose up -d"
  environment:
    name: development
    url: $APP_ENDPOINT

deploy_frontend:
  extends: .deploy
  variables:
    MICRO_SERVICE: frontend
    SERVICE_VERSION: "1.3"
    APP_PORT: 3000
  only:
    changes:
      - "frontend/**/*"
```




```yaml
.deploy:
  stage: deploy
  tags:
    - ec2
    - remote
    - shell
  variables:
    MICRO_SERVICE: ""
    SERVICE_VERSION: ""
    APP_PORT: ""
  before_script:
    - chmod 400 $SSH_PRIVATE_KEY
    - export IMAGE_NAME=$CI_REGISTRY_IMAGE/microservice/$MICRO_SERVICE
    - export IMAGE_NAME=$SERVICE_VERSION
  script:
    - scp -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ./docker-compose.yaml ubuntu@$DEPL
    - ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ubuntu@$DEPLOYMENT_SERVER_HOST "
      docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY &&
      export COMPOSE_PROJECT_NAME=$MICRO_SERVICE &&
      export DC_IMAGE_NAME=$IMAGE_NAME &&
      export DC_IMAGE_TAG=$IMAGE_NAME &&
      export DC_APP_PORT=$APP_PORT &&
      docker-compose down &&
      docker-compose up -d"
  environment:
    name: development
    url: $APP_ENDPOINT

deploy_frontend:
  extends: .deploy
  variables:
    MICRO_SERVICE: frontend
    SERVICE_VERSION: "1.3"
    APP_PORT: 3000
  only:
    changes:
      - "frontend/**/*"
```

configure a common docker network
we deploy service with separate docker-compse file

Networking in Docker Compose

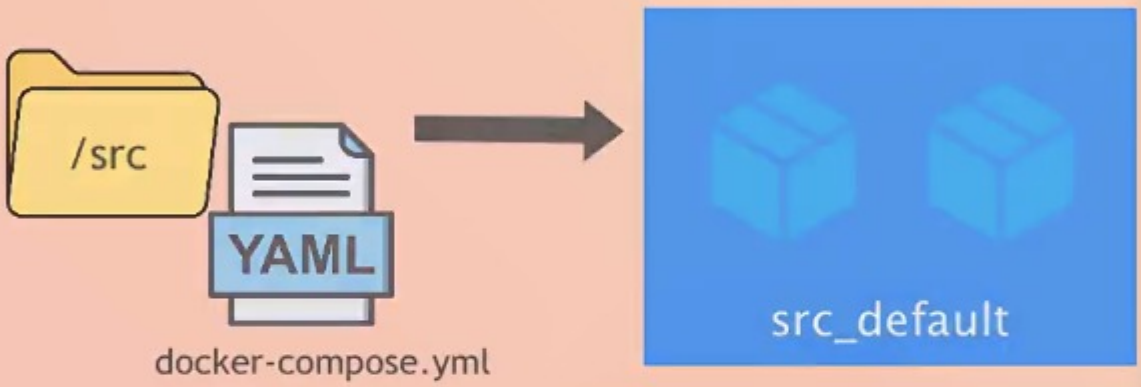
- By default, Docker Compose **sets up a single network** for your app
- All services listed in the Compose file joins the default network

docker network ls

Each network is like an isolated virtual network

Networking in Docker Compose

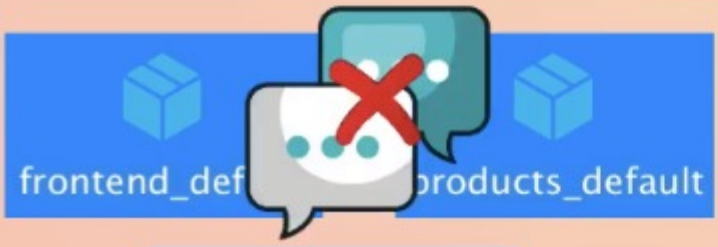
- Compose creates a new Docker network on every docker-compose up
- **By default:** the network name is based on the name of the directory the compose file resides



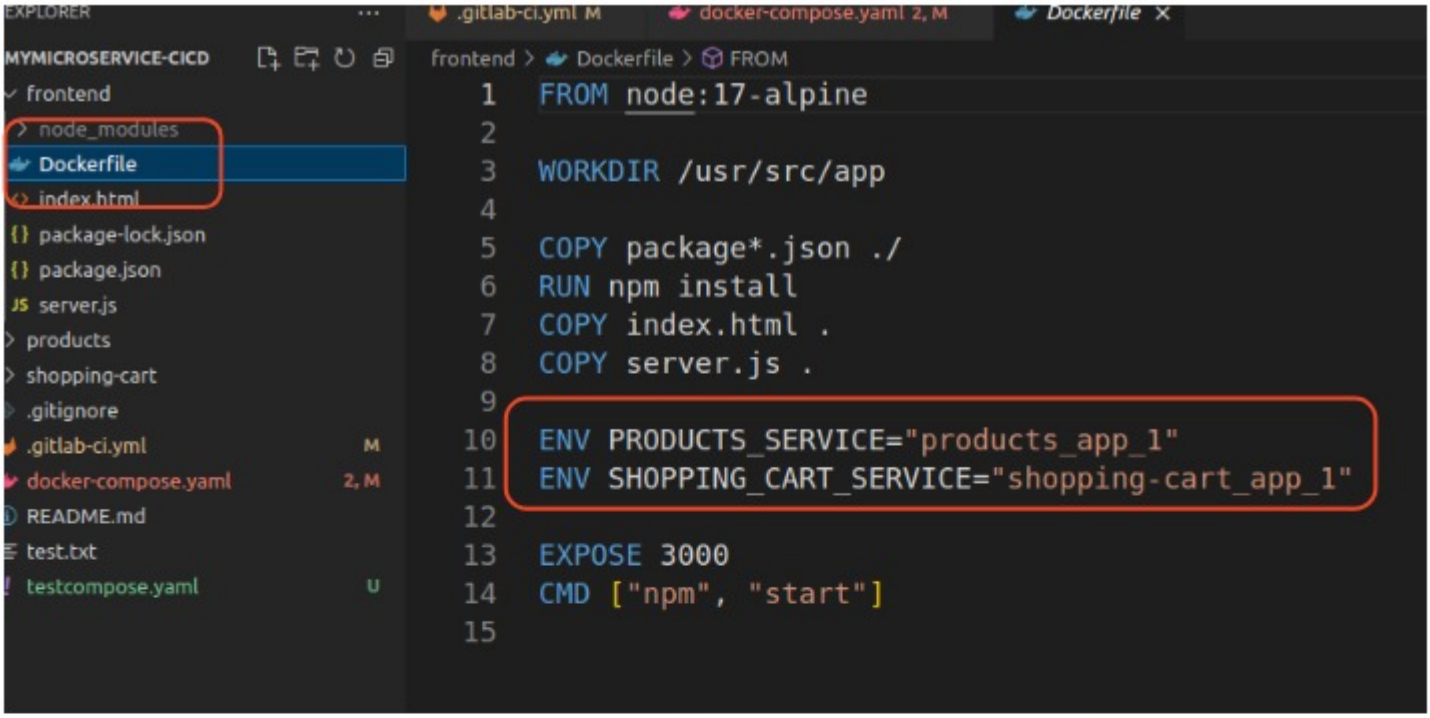
<project-name>_default

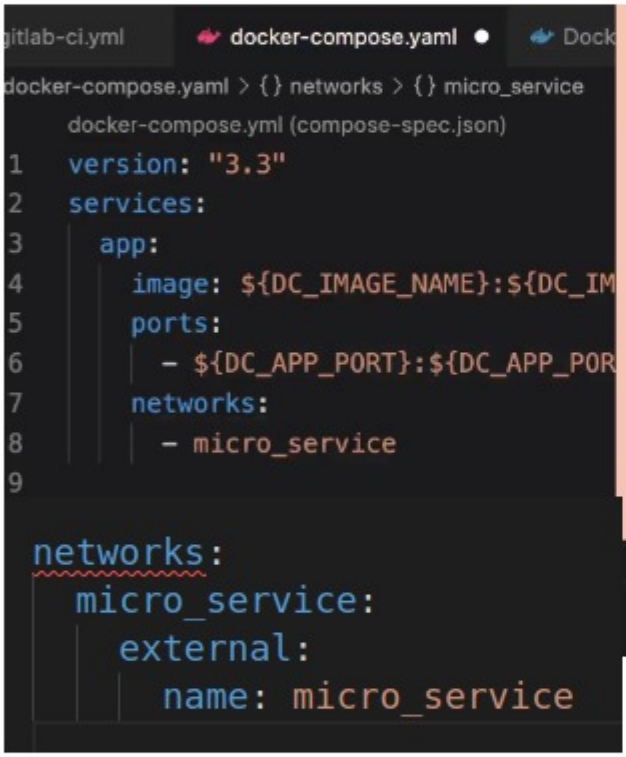
Networking in Docker Compose

Problem:
Containers in different Docker networks **can't communicate** with each other



create one network all microservices inside





Use a pre-existing network

- Instead of creating an own network, Compose looks for a the defined network
- And **joins the containers in that pre-existing network**

```
script:
  - scp -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ./d
  - ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ubu
    docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSW
    export COMPOSE_PROJECT_NAME=$MICRO_SERVICE &&
    export DC_IMAGE_NAME=$IMAGE_NAME &&
    export DC_IMAGE_TAG=$IMAGE_NAME &&
    export DC_APP_PORT=$APP_PORT &&
    docker network create micro_service || true &&
    docker-compose down &&
    docker-compose up -d"
```

create ssh private key

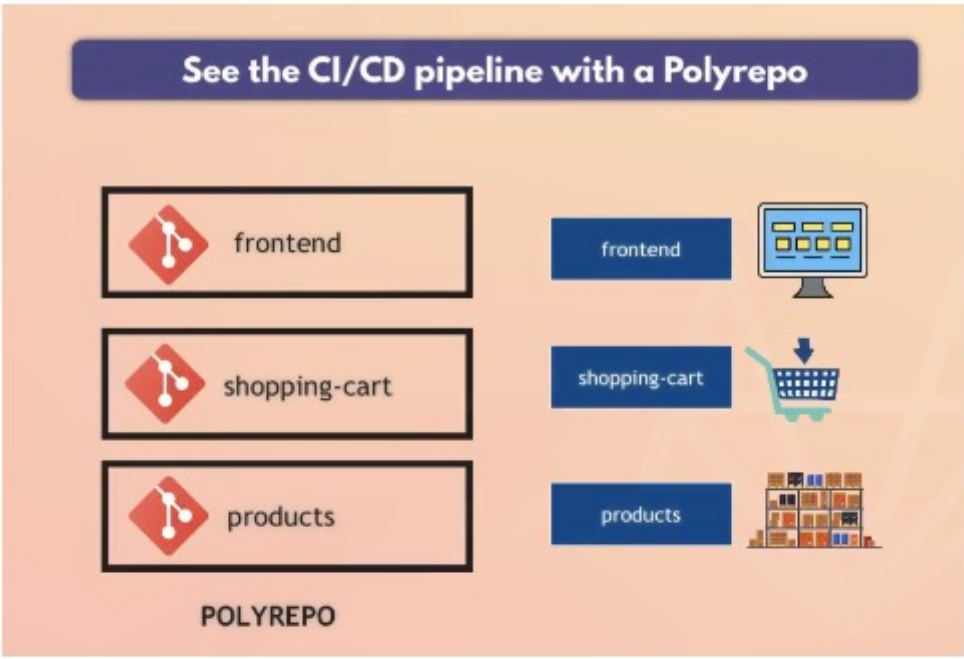
```
FROM node:17-alpine

WORKDIR /usr/src/app

COPY package*.json ./
RUN npm install
COPY index.html .
COPY server.js .

ENV PRODUCTS_SERVICE="products_app_1"
ENV SHOPPING_CART_SERVICE="shopping-cart_app_1"

EXPOSE 3000
CMD ["npm", "start"]

```

```yaml
workflow:
  rules:
    - if: $CI_COMMIT_BRANCH != "main" && $CI_PIPELINE_SOURCE != "merge_request_event"
      when: never
    - when: always

stages:
  - build
  - deploy

variables:
  DEPLOYMENT_SERVER_HOST: 18.197.15.194
  APP_ENDPOINT: ec2-18-197-15-194.eu-central-1.compute.amazonaws.com

.build:
  stage: build
  tags:
    - ec2
    - remote
    - shell
  variables:
    MICRO_SERVICE: ""
    SERVICE_TAG: ""
  before_script:
    - cd $MICRO_SERVICE
    - export IMAGE_NAME=$CI_REGISTRY_IMAGE/microservice/$MICRO_SERVICE
    - export IMAGE_TAG=$SERVICE_TAG
  script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD  $CI_REGISTRY
    - docker build -t $IMAGE_NAME:$IMAGE_TAG .
    - docker push $IMAGE_NAME:$IMAGE_TAG

build_frontend:
  extends: .build
  variables:
    MICRO_SERVICE: frontend
    SERVICE_TAG: "1.3"
  only:
    changes:
      - "frontend/**/*"

build_product:
  extends: .build
  variables:
    MICRO_SERVICE: product
    SERVICE_TAG: "1.8"
  only:
    changes:
      - "product/**/*"

build_shopping_cart:
  extends: .build
  variables:
    MICRO_SERVICE: shopping-cart
    SERVICE_TAG: "2.1"
  only:
    changes:
      - "shopping-cart/**/*"
```

```yaml
.deploy:
  stage: deploy
  tags:
    - ec2
    - remote
    - shell
  variables:
    MICRO_SERVICE: ""
    SERVICE_VERSION: ""
    APP_PORT: ""
  before_script:
    - chmod 400 $SSH_PRIVATE_KEY
    - export IMAGE_NAME=$CI_REGISTRY_IMAGE/microservice/$MICRO_SERVICE
    - export IMAGE_TAG=$SERVICE_VERSION
  script:
    - scp -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ./docker-compose.yaml
ubuntu@$DEPLOYMENT_SERVER_HOST:/home/ubuntu
    - ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY
ubuntu@$DEPLOYMENT_SERVER_HOST "
      docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY &&
      export COMPOSE_PROJECT_NAME=$MICRO_SERVICE &&
      export DC_IMAGE_NAME=$IMAGE_NAME &&
      export DC_IMAGE_TAG=$IMAGE_TAG &&
      export DC_APP_PORT=$APP_PORT &&
      docker network create micro_service || true &&
      docker-compose down &&
      docker-compose up -d"
  environment:
    name: development
    url: $APP_ENDPOINT

deploy_frontend:
  extends: .deploy
  variables:
    MICRO_SERVICE: frontend
    SERVICE_VERSION: "1.3"
    APP_PORT: 3000
  only:
    changes:
      - "frontend/**/*"

deploy_product:
  extends: .deploy
  variables:
    MICRO_SERVICE: product
    SERVICE_VERSION: "1.8"
    APP_PORT: 3001
  only:
    changes:
      - "product/**/*"

deploy_shopping_cart:
  extends: .deploy
  variables:
    MICRO_SERVICE: shopping-cart
    SERVICE_VERSION: "2.1"
    APP_PORT: 3002
  only:
```

```
    changes:
      - "shopping-cart/**/*"
```

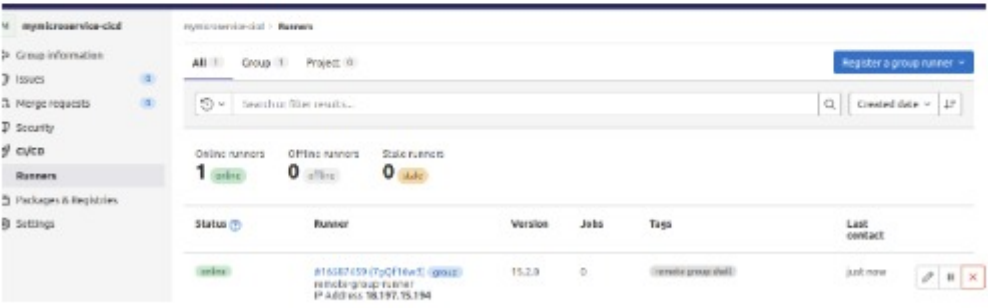
```yaml
version: "3.3"
services:
  app:
    image: ${DC_IMAGE_NAME}:${DC_IMAGE_TAG}
    ports:
      - ${DC_APP_PORT}:${DC_APP_PORT}
    networks:
      - micro_service

networks:
  micro_service:
    external:
      name: micro_service

```



create gruppe for microservices
create a group and three projects

1. register a runner for group

Configure Runner for **whole group**, instead of each project



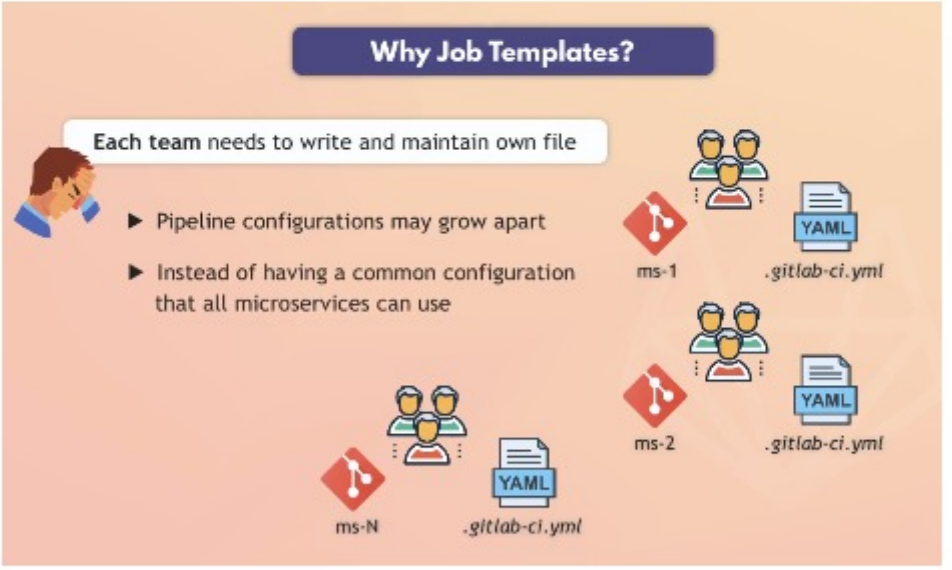
2Prepare deployment environment

We will reuse the deployment server as well

Stop all Docker containers and Images

delete all images, docker and netwokr
create SSH_PRIVATE_KEY

3 first create docker- compose file



template: own or other teams

extract configuration into a a separate YAML file

2 Template Types

- The type impacts the way the template should be written and used

Pipeline Templates

- Provides an end-to-end CI/CD workflow
- Should be used by itself in projects, with no other .gitlab-ci.yml file

Job Templates

- Provides specific jobs

generic reusable customizable

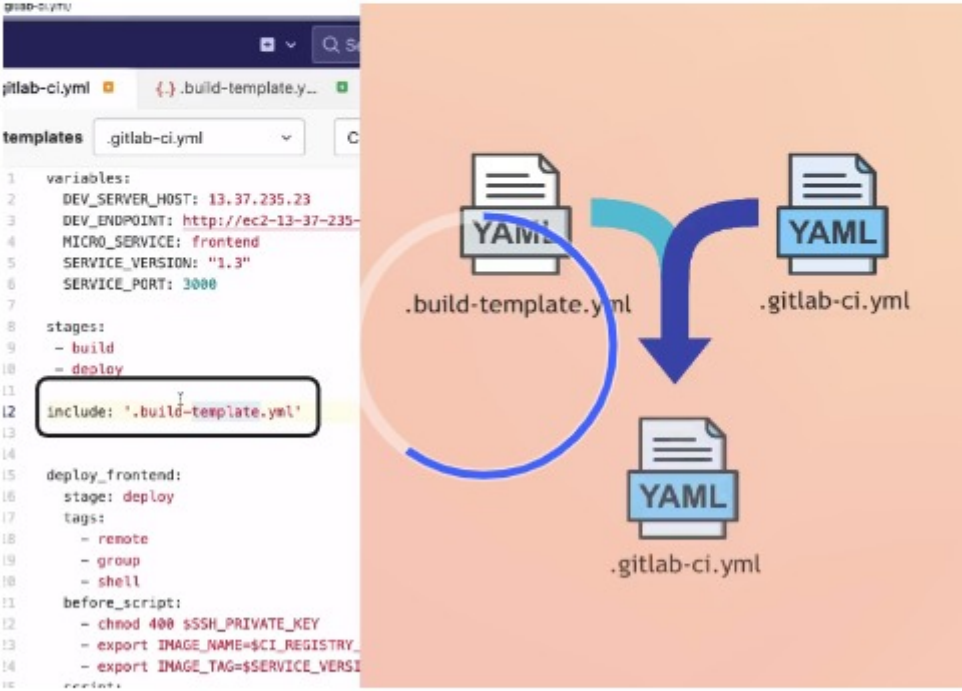
frontend -> .gitlab-ci.yml
create a .build-template.yml file
and pase build job there

Configuration is already generic & customizable, no hardcoded values

```yaml
variables:
  MICRO_SERVICE: ""
  SERVICE_VERSION: ""

build:
  stage: build
  before_script:
    - export IMAGE_NAME=$CI_REGISTRY_IMAGE/microservice/$MICRO_SERVICE
    - export IMAGE_TAG=$SERVICE_VERSION
  script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
    - docker build -t $IMAGE_NAME:$IMAGE_TAG .
    - docker push $IMAGE_NAME:$IMAGE_TAG
```

list variables for good overview

next step include build-templete in the pipeline -> include



Then in the pipleline is very easy:

```
    - run_performance_tests
  variables:
    SSH_KEY: $SSH_PRIVATE_KEY
    SERVER_HOST: $PROD_SERVER_HOST
    DEPLOY_ENV: production
    APP_PORT: 5000
    ENDPOINT: $PROD_ENDPOINT
  when: manual
```

"include"

- Possible subkeys are: template, local, file, remote

```
include:
  - template: Jobs/SAST.gitlab-ci.yml
```

```
include:
  - local: '.build-template.yml'

variables:
  DEV_SERVER_HOST: 13.37.235.23
  DEV_ENDPOINT: http://ec2-13-37-235-23.eu-west-3.compute.amazonaws.com:3000
  MICRO_SERVICE: frontend
  SERVICE_VERSION: "1.3"
  SERVICE_PORT: 3000

stages:
  - build
  - deploy


build:
  tags:
    - remote
    - group
    - shell
```

You can split long pipeline configurations into multple YAML files for better readability etc.

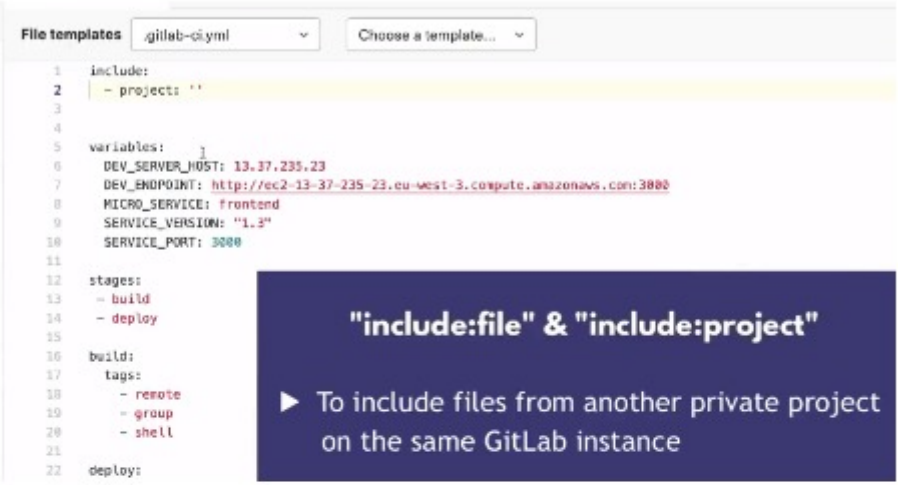
如果template里面也有before_script，将会被build pipleline里面的before_script移除

netx create deploy templat e

```yaml
include:
  - local: '.build-template.yml'
  - local: '.deploy-template.yml'

variables:
  DEV_SERVER_HOST: 13.37.235.23
  DEV_ENDPOINT: http://ec2-13-37-235-23.eu-west-3.compute.amazonaws.com:3000
  MICRO_SERVICE: frontend
  SERVICE_VERSION: "1.3"
  SERVICE_PORT: 3000

stages:
 - build
 - deploy

build:
  tags:
    - remote
    - group
    - shell

deploy:
  tags:
    - remote
    - group
    - shell
```

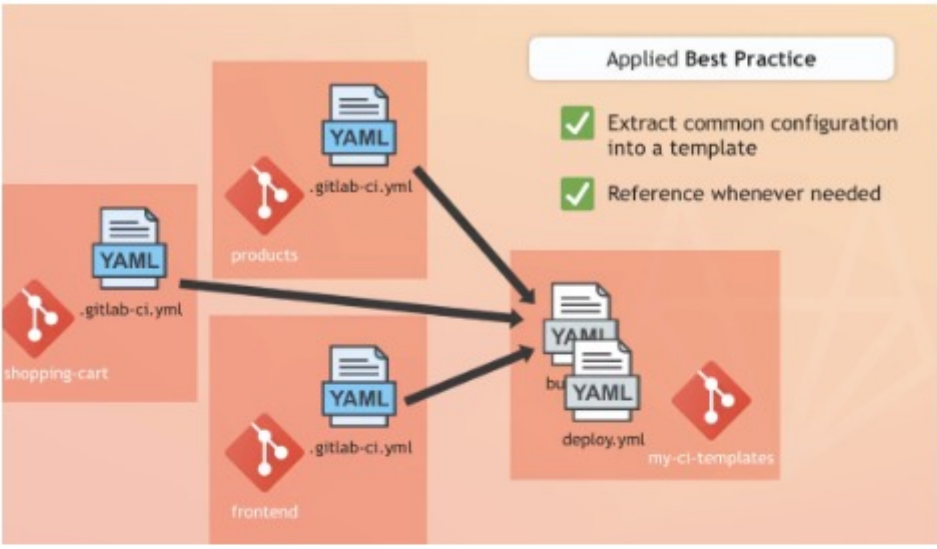
create a project in the group named ci-template project

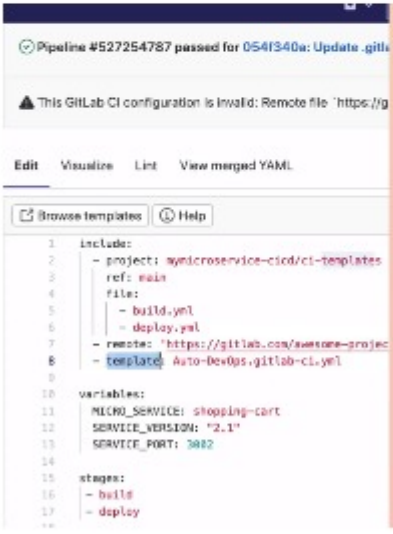
then create two files: build.ymal and depoly.yaml



Use another keyword projects

```
include:
  - project: mymicroservice-cicd/ci-templates
    ref: main
    file:
      - build.yml
      - deploy.yml
```



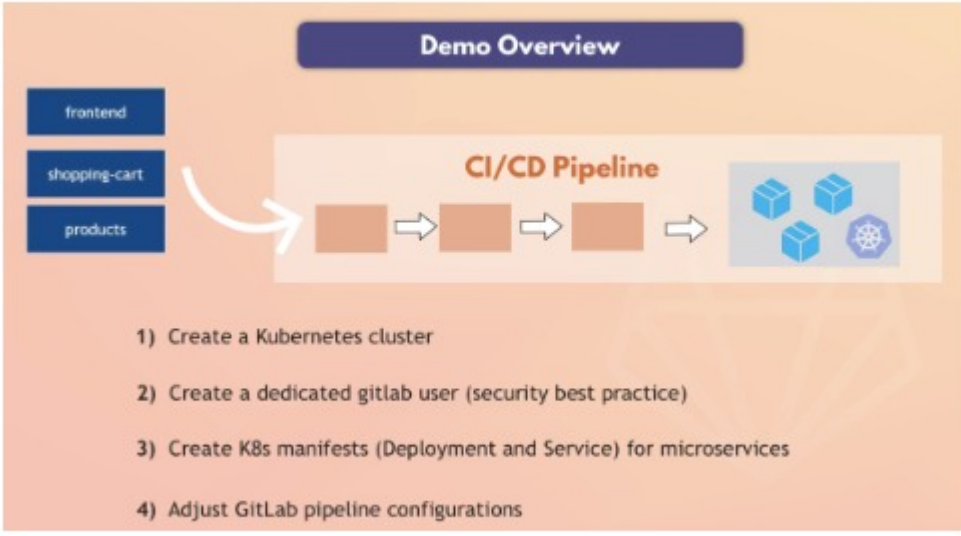


include

- To include external YAML files

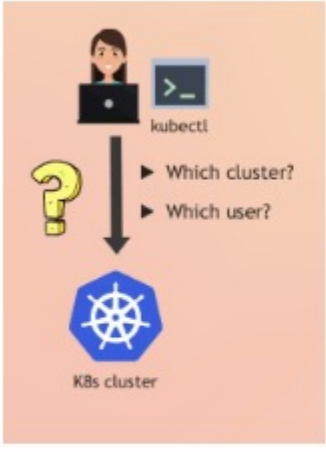
- local	▶ Reference from same repository
- file	▶ Reference from another private project (same GitLab instance)
- remote	▶ Include from a different location (full URL necessary)
- template	▶ Include GitLab's templates

depoly from gitlab cicd pipeline to a k8s cluster



create a managed k8S plattform

tell kubectl



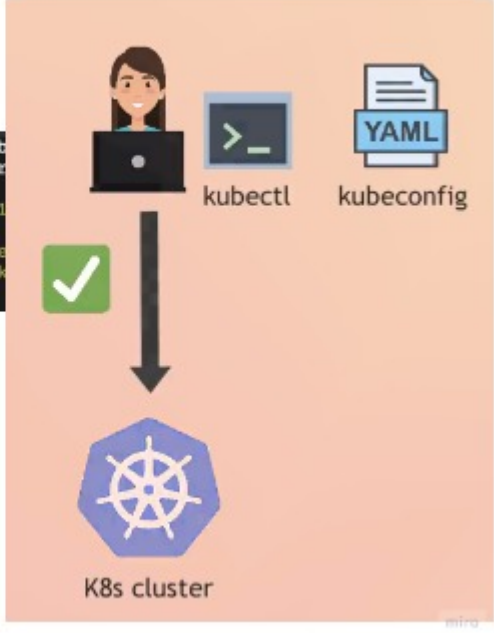
Kubeconfig:

my-micro-service-kubeconfig.yaml

ServiceAccount = Represent non-human user

chmod 400 to configureation file

```
[\W]$ chmod 400 ~/Downloads/my-micro-service-kub
[\W]$ export KUBECONFIG=~/Downloads/my-micro-ser
[\W]$ kubectl cluster-info
Kubernetes control plane is running at https://1
tral-2.linodelke.net:443
KubeDNS is running at https://11607756-08b8-4fae
net:443/api/v1/namespaces/kube-system/services/k
```



- Kubernetes CLI that you can install on every machine


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


Security Best Practice
K8s administrator
Gitlab
Admin User
GitLab User

- Deploy our services in a dedicated namespace

- [x] Restrict GitLab access to that namespace

Namespace = Isolated space for your K8s resources within your cluster

```
[\W]$ kubectl create namespace my-micro-servic
```

```
[\W]$ kubectl get namespace
NAME                 STATUS   AGE
default              Active   14m
kube-node-lease      Active   14m
kube-public          Active   14m
kube-system          Active   14m
my-micro-service     Active   1s
```

```
[\W]$ kubectl create serviceaccount cicd-sa --namespace=my-micro-service
serviceaccount/cicd-sa created
[\W]$
```

```
[\W]$ vim cicd-role.yaml
```

permisson for role

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: my-micro-service
  name: cicd
rules:
- apiGroups: [""] # indicates the core API group
  resources: ["pods", "services"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
- apiGroups: ["extensions", "apps"]
  resources: ["deployments"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

because you will create secret for container docker

add secret after "services"

```
[\W]$ kubectl apply -f cicd-role.yaml
role.rbac.authorization.k8s.io/cicd-role created
[\W]$
```


```
\W]$ kubectl create rolebinding cicd-rb
```

Role | ServiceAccount | Namespace

```
[\W]$ kubectl create rolebinding cicd-rb --role=cicd-role --serviceaccount=my-micro-servic
e:cicd-sa --namespace=my-micro-service
rolebinding.rbac.authorization.k8s.io/cicd-rb created
[\W]$
```

Create kube config file


```
[\W]$ cp ~/Downloads/my-micro-service-kubeconfig.yaml ~/cicd-kubeconfig.yaml
[\W]$ vim cicd-kubeconfig.yaml
```

what to change:
users with different token -> where to get token

```
[\W]$ kubectl get serviceaccount -n my-micro-service
NAME      SECRETS   AGE
cicd-sa   1         9m12s
default   1         10m
```

```
[\W]$ kubectl get serviceaccount cicd-sa -n my-micro-service -o yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: "2022-05-01T10:04:56Z"
  name: cicd-sa
  namespace: my-micro-service
  resourceVersion: "1441"
  uid: 1b316c45-10d4-4419-aa97-ed8cd5e2f88a
secrets:
- name: cicd-sa-token-4fkgn
[\W]$
```

```
[\W]$ kubectl get serviceaccount cicd-sa -n my-micro-service -o yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: "2022-05-01T10:04:56Z"
  name: cicd-sa
  namespace: my-micro-service
  resourceVersion: "1441"
  uid: 1b316c45-10d4-4419-aa97-ed8cd5e2f88a
secrets:
- name: cicd-sa-token-4fkgn
[\W]$
```

```
[\W]$ kubectl get secret cicd-sa-token-4fkgn -n my-micro-service -o yaml
```

decode the token
echo token | base64

```
- name: lke59454-cicd
  user:
    as-user-extra: {}
    token: eyJhbGciOiJSUzI1NiIsImtpZCI6IlFYMDZrdklBWFljYllGbDYzU2h2ZDVvMURCOElXY1RTZFpLV3I
CWUZiV1kifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYW
Njb3VudC9uYW1lc3BhY2UiOiJteS1taWNyby1zZXJ2aWNlIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9z
ZWNyZXQubmFtZSI6ImNpY2Qtc2EtdG9rZW4tNGZrZ24iLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3Nlcn
ZpY2UtYWNjb3VudC5uYW1lIjoiY2ljZC1zYSIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1h
Y2NvdW50LnVpZCI6IjFiMzE2YzQ1LTEwZDQtNDQxOS1hYTk3LWVkOGNkNWUyZjg4YSIsInN1YiI6InN5c3RlbTpzZX
J2aWNlYWNjb3VudDpteS1taWNyby1zZXJ2aWNlOmNpY2Qtc2EifQ.DylkVx_UDQonjJpW4l3flHJDYNCr1xvnjZy3D
Pn5f5okEZ3oHfcyvvSuWc27mtqXTD9lyQAlCtFktRy65LJX-jIglhsFPdApwb8HsaeJkrJ94_VM-iCc2uRRvSo9LHT
4RfXMSsXeuyejn3fmR5Fio3ots9_KYWLMNH1StaR93ZBJ_yFH91U4E0ZLeNf5uhp5gleJAKS7-4QsAXIwbfK8xCnpx
vG-7FPNkZKkJE-nrlKV4zq-gL8VKGqW_vMwI8uQrLCT9ZC5sS45g7PBZXVUHEZwJJneCPAJ9UQJriPnVUHzNblNXio
2nVuTFJeR35Q81_3lONS7jYB6Toa22whsYg

contexts:
- context:
    cluster: lke59454
    namespace: default
    user: lke59454-
  name: lke59454-ctx

current-context: lke59454-ctx
-- INSERT --
```

Context = Used to group access parameters under a name

Each context has 3 parameters: cluster, namespace, user

test:

```
[\W]$ export KUBECONFIG=cicd-kubeconfig.yaml
[\W]$ kubectl get namespace
Error from server (Forbidden): namespaces is forbidden: User "system:serviceaccount:my-mic
ro-service:cicd-sa" cannot list resource "namespaces" in API group "" at the cluster scope
[\W]$ kubectl get pod -n my-micro-service
No resources found in my-micro-service namespace.
```

```
[\W]$ kubectl get pod -n my-micro-service
No resources found in my-micro-service namespace.
[\W]$ kubectl get service -n my-micro-service
No resources found in my-micro-service namespace.
[\W]$ kubectl get pod -n kube-system
Error from server (Forbidden): pods is forbidden: User "system:serviceaccount:my-micro-se
vice:cicd-sa" cannot list resource "pods" in API group "" in the namespace "kube-system"
[\W]$
```

test

SUMMARY:

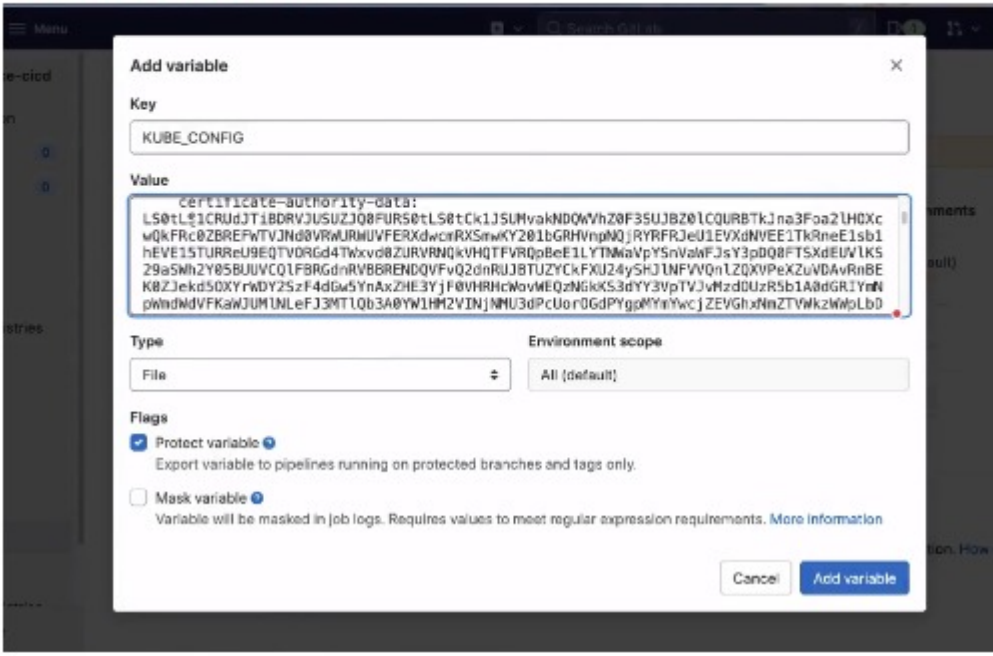
K8S的创建是在Linode上面的，因为简单。最重要的是下载一个config文件。

然后利用kubectl 去使用这个cluster通过 export KUBECONFIG.

但是为了安全，我们不想把所有的权限都给gitlab,所以创建个 namespace ,在这个namespace里面创建 serviceaccount （cicd-sa),接着赋予这个账号role.能做什么不能做什么这个是k8s apiVersion,kind,metadata,rules.有个这个role.yaml 文件就要应用他通过 kubectl apply -f. 接着要把role和serviceaccount, namespace进行绑定 然后设置kubeconfig.yaml 给这个账号。 这个新的config里面的token 要被换掉，最后进行验证

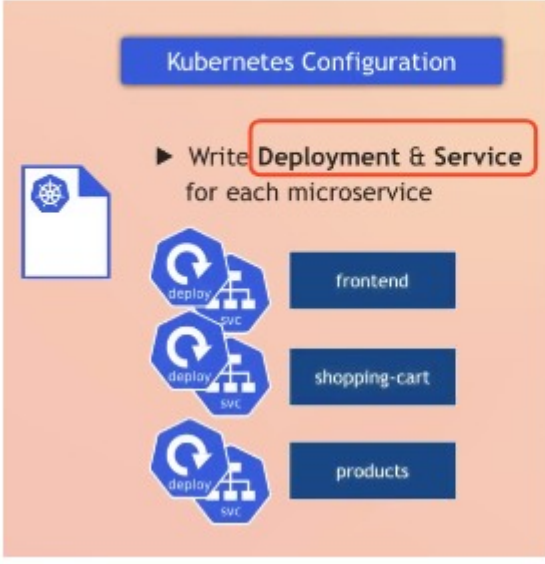
Add kubeconfig file in gitlab -> able to use kubectl command in the pipeline

create a kubeconfig file



configuraitons we need to write to deploy our service tinto the k8s cluster

1



▶ Create a Docker Registry **Secret** for K8s to be able to pull the Docker images

GitLab Configuration

▶ **Install kubectl** on GitLab Runner

GitLab Configuration

▶ **Install kubectl** on GitLab Runner

SUMMARY：

- K8S for each service, you need deploymen and service.
- for the docker inside the pod you need docker regstry secrect
- in the gitlab runner you need to execute the k8s so you need to install kubectl
- we have k8s so we change docker-impsoe with k8s

New file

main / deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  namespace: my-micro-service
spec:
  selector:
    matchLabels:
      app: frontend
  replicas: 1
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image:
        ports:
        - containerPort: 3000
```

Deployment Configuration

- `name` ▶ Can be any name
- `namespace` ▶ The namespace, we want to deploy our services in
- `replicas` ▶ How many replicas, we want to deploy. Could be more than 1 for high availability

Container Configuration inside the Deployment

- `containerPort` ▶ Port on which the container will listen
- `image` ▶ The Docker image we want to deploy

make the code generic

main / deployment.yaml Select a template type

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: $MICRO_SERVICE
  namespace: my-micro-service
spec:
  selector:
    matchLabels:
      app: $MICRO_SERVICE
  replicas: $REPLICAS
  template:
    metadata:
      labels:
        app: $MICRO_SERVICE
    spec:
      containers:
      - name: $MICRO_SERVICE
        image: $IMAGE_NAME:$IMAGE_TAG
        ports:
        - containerPort: $SERVICE_PORT
```

deployment.yaml service.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend
  namespace: my-micro-servic
spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
```

Service Configuration

- **name** ▶ Can be any name
- **selector** ▶ Defines which Pods (created by the Deployment) are connected to the Service

```yaml
apiVersion: v1
kind: Service
metadata:
  name: $MICRO_SERVICE
  namespace: my-micro-service
spec:
  selector:
    app: $MICRO_SERVICE
  ports:
    - protocol: TCP
      port: $SERVICE_PORT
      targetPort: $SERVICE_PORT
```

configurable-> generic

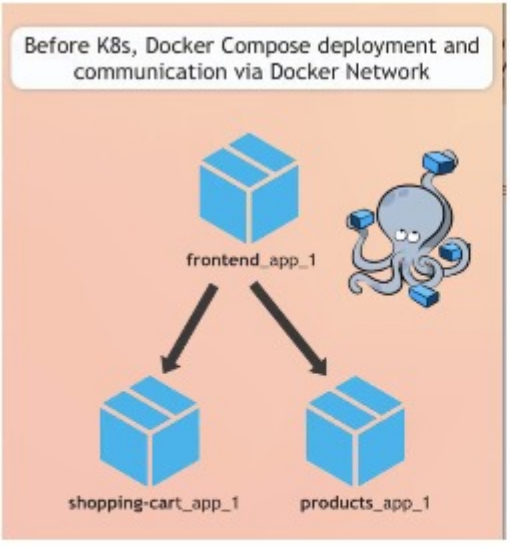
Service Configuration

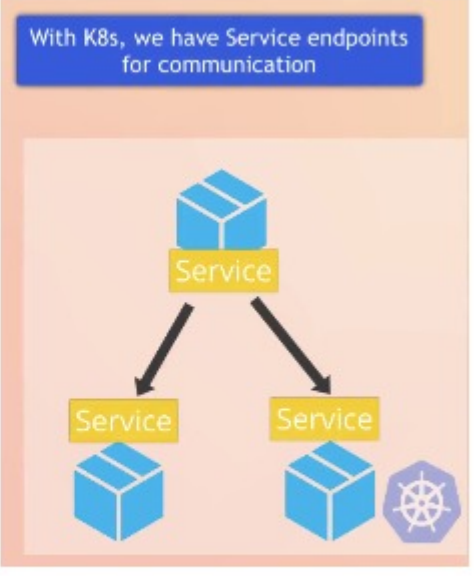
- **port** ▶ Port on which the Service listens
- **targetPort** ▶ Port on which the container listens

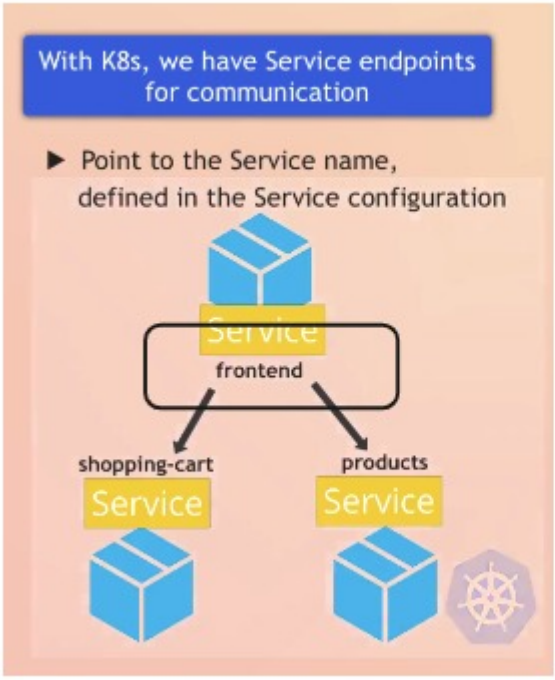
▶ Service forwards the request to the container's targetPort

fox microservice endpoints

In the dockerfile we have ENV with endpoint- > change to k8s service name

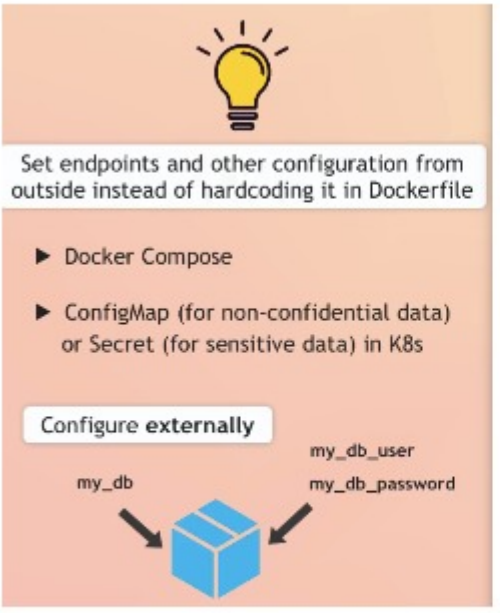







```dockerfile
FROM node:17-alpine

WORKDIR /usr/src/app

COPY package*.json ./
RUN npm install
COPY index.html .
COPY server.js .

ENV PRODUCTS_SERVICE="products"
ENV SHOPPING_CART_SERVICE="shopping-cart_app_1"

EXPOSE 3000
CMD ["npm", "start"]
```

Service = enables communication



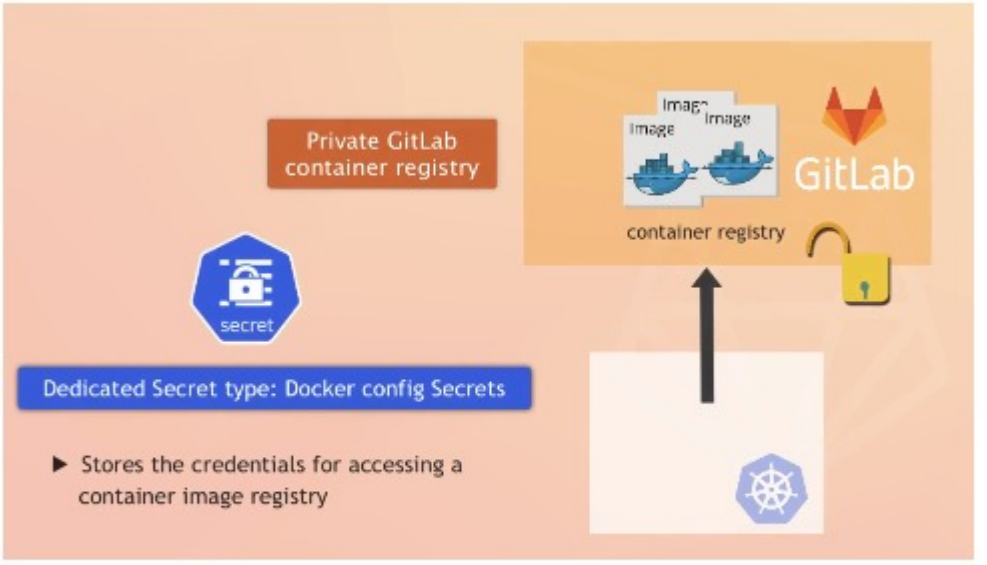
mymicroservice-cicd › ci-templates › **Repository**

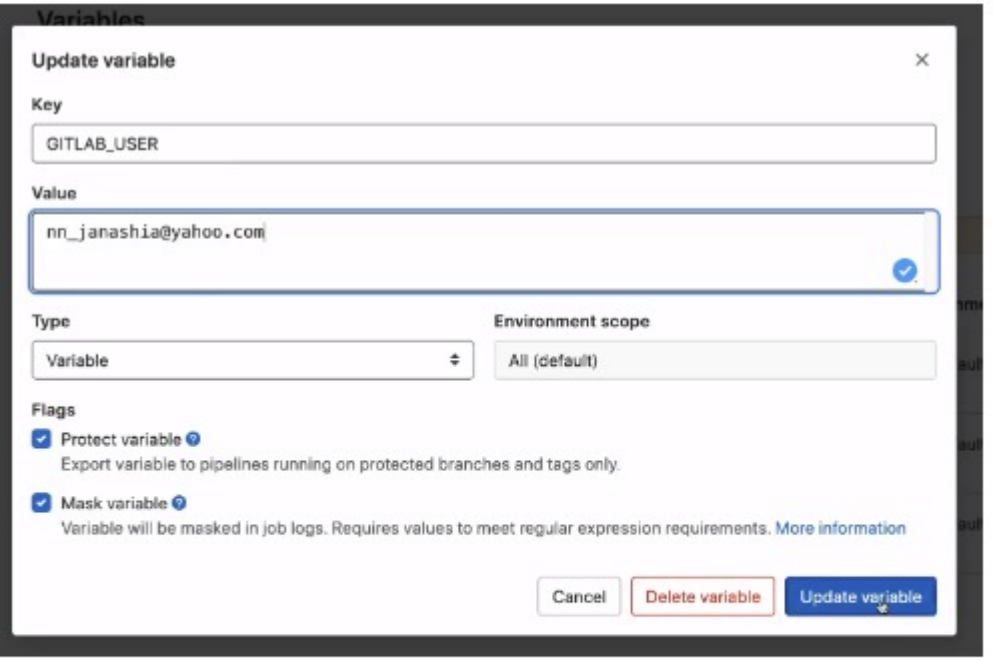
New file

这个文件将会被include到每个service里面，

main / deploy-k8s.yml Select a template type

```
deploy:
    stage: deploy
    before_script:
        - export IMAGE_NAME=$CI_REGISTRY_IMAGE/microservice/$MICRO_SERVICE
        - export IMAGE_TAG=$SERVICE_VERSION
        - export MICRO_SERVICE=$MICRO_SERVICE
        - export SERVICE_PORT=$SERVICE_PORT
        - export REPLICAS=$REPLICAS
        - export KUBECONFIG=$KUBE_CONFIG
    script:
        - envsubst < kubernetes/deployment.yaml | kubectl apply -f -
        - envsubst < kubernetes/service.yaml | kubectl apply -f -
```


Private GitLab container registry
image
image
image
GitLab
container registry
secret
Dedicated Secret type: Docker config Secrets
▶ Stores the credentials for accessing a container image registry


Update variable
Key
GITLAB_USER
Value
nn_janashia@yahoo.com
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
Delete variable
Update variable

```
script:
    - kubectl create secret docker-registry my-registry-key --docker-server=$CI_REGISTRY --docker-username=$GITLAB_USER
--docker-password=$GITLAB_PASSWORD
    - envsubst < kubernetes/deployment.yaml | kubectl apply -f -
    - envsubst < kubernetes/service.yaml | kubectl apply -f -
```

kubectl commands are imperative

Apply YAML files, which are declarative

Imperative = Tell what to do

Declarative = Define desired state

--dry-run=client = Preview the object that would be sent to your cluster, without really submitting it

```
script:
    - kubectl create secret docker-registry my-registry-key --docker-server=$CI_REGISTRY --docker-username=$GITLAB_USER
--docker-password=$GITLAB_PASSWORD -n my-micro-service --dry-run=client -o yaml | kubectl apply -f -
    - envsubst < kubernetes/deployment.yaml | kubectl apply -f -
```

main | kubernetes/deploymer

update deployment file

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: $MICRO_SERVICE
5    namespace: my-micro-service
6  spec:
7    selector:
8      matchLabels:
9        app: $MICRO_SERVICE
10   replicas: $REPLICAS
11   template:
12     metadata:
13       labels:
14         app: $MICRO_SERVICE
15     spec:
16       imagePullSecrets:
17       - name: my-registry-key
18       containers:
19       - name: $MICRO_SERVICE
20         image: $IMAGE_NAME:$IMAGE_TAG
21         ports:
22         - containerPort: $SERVICE_PORT
```

install kubectl in the gitlab kubectl
ssh first to gitlab-runner
install kuebctl

in the gitlab-runner check alos envsubset command. it is a linux command.

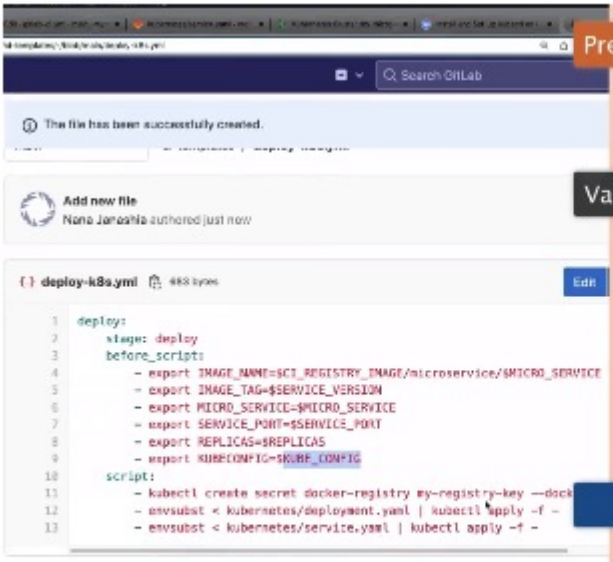
summary任务1 deployment and service for each service.crate it in the kubernets filea nd include later

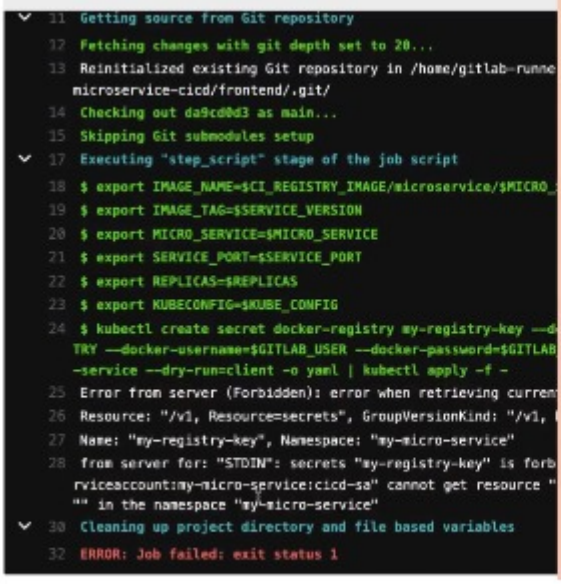
- container image 是重点

Service need to communicate with each other. Docker and k8s different so change endpoint in the dockerfile. (usually endpoint is not in the docker file)
next: 创建一个generic deployment file, inside stage, before_script, script 写好，because they are generic
in what the kubect create docekr registry sercet important, which used in the deployment.yaml image

- in the microservice you can check the container- it will tell you hwo to create the secret for container.
-

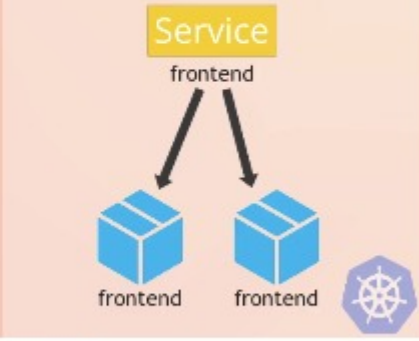
How to change the pipleline code in the service


Search GitLab
The file has been successfully created.
Add new file
Nana Janashia authored just now
deploy-k8s.yml
Edit
deploy:
stage: deploy
before_script:
- export IMAGE_NAME=$CI_REGISTRY_IMAGE/microservice/$MICRO_SERVICE
- export IMAGE_TAG=$SERVICE_VERSION
- export MICRO_SERVICE=$MICRO_SERVICE
- export SERVICE_PORT=$SERVICE_PORT
- export REPLICAS=$REPLICAS
- export KUBECONFIG=$KUBE_CONFIG
script:
- kubectl create secret docker-registry my-registry-key --dock
- envsubst < kubernetes/deployment.yaml | kubectl apply -f -
- envsubst < kubernetes/service.yaml | kubectl apply -f -
Pre-defined GitLab variables
$CI_REGISTRY_NAME
Variables, we need to define
$MICRO_SERVICE
$SERVICE_VERSION
$SERVICE_PORT
$REPLICAS
Global variable
$KUBE_CONFIG

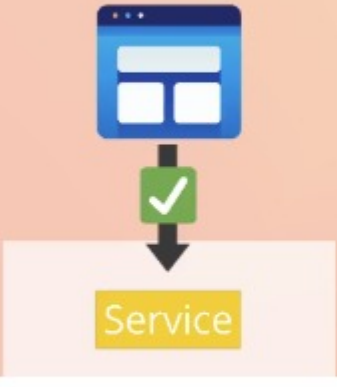

Getting source from Git repository
Fetching changes with git depth set to 20...
Reinitialized existing Git repository in /home/gitlab-runne
microservice-cicd/frontend/.git/
Checking out da9cd0d3 as main...
Skipping Git submodules setup
Executing "step_script" stage of the job script
$ export IMAGE_NAME=$CI_REGISTRY_IMAGE/microservice/$MICRO_
$ export IMAGE_TAG=$SERVICE_VERSION
$ export MICRO_SERVICE=$MICRO_SERVICE
$ export SERVICE_PORT=$SERVICE_PORT
$ export REPLICAS=$REPLICAS
$ export KUBECONFIG=$KUBE_CONFIG
$ kubectl create secret docker-registry my-registry-key --d
TRY --docker-username=$GITLAB_USER --docker-password=$GITLAB
-service --dry-run=client -o yaml | kubectl apply -f -
Error from server (Forbidden): error when retrieving curren
Resource: "/v1, Resource=secrets", GroupVersionKind: "/v1,
Name: "my-registry-key", Namespace: "my-micro-service"
from server for: "STDIN": secrets "my-registry-key" is forb
rviceaccount:my-micro-service:cicd-sa" cannot get resource "
"" in the namespace "my-micro-service"
Cleaning up project directory and file based variables
ERROR: Job failed: exit status 1
sa
role
Role with defined permissions
Not allowed to create a Secret

```
[\W]$ kubectl get service -n my-micro-servi
NAME       TYPE        CLUSTER-IP       EXTE
frontend   ClusterIP   10.128.160.92    <non
[\W]$ kubectl get deployment -n my-micro-se
NAME       READY   UP-TO-DATE   AVAILABLE
frontend   2/2     2            2
[\W]$ kubectl get pod -n my-micro-service
NAME                        READY   STATUS
frontend-6d5cb78bc5-4sm56   1/1     Running
frontend-6d5cb78bc5-j2zjm   1/1     Running
[\W]$
```

2 replicas deployed



Access the K8s cluster with localhost using kubectl proxy



```
[\W]$ kubectl port-forward service/frontend -n my-micro-service 3000:3000
Forwarding from 127.0.0.1:3000 -> 3000
Forwarding from [::1]:3000 -> 3000
```

```
include:
  - project: mymicroservice-cicd/ci-templates
    ref: main
    file:
      - build.yml
      - deploy-k8s.yml
```

build not change because jiust buid image and save in the gitlab. but deployment we need to depoy instde dockerr to k8s. So we need to rewrite the deployment pipeline and change it also in the cinclude file.